AN EVENT-BASED VISION SENSOR SIMULATION FRAMEWORK FOR SPACE DOMAIN AWARENESS APPLICATIONS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Rachel Oliver

August 2024

AN EVENT-BASED VISION SENSOR SIMULATION FRAMEWORK FOR SPACE DOMAIN AWARENESS APPLICATIONS

Rachel Oliver, Ph.D.

Cornell University 2024

Event-based vision sensors (EVS) provide a unique opportunity for Space Domain Awareness (SDA) applications. Inspired by the human eye, these sensors operate on the principle of change detection and each pixel functions independently and asynchronously from the other pixels. Pixels trigger events when a change in light intensity occurs. The sensor records events as a sparse time series output with microsecond level of precision. The space sensing community is interested in this technology due to wide dynamic range achieved by the operating in the log scale, the minimal data produced for a relatively static scene where changes in intensity are infrequent, and the temporal precision that opens opportunities to capture information on fast moving space objects where traditional frame imaging is not an option. As a relatively inexpensive sensor with technical capabilities well suited for tracking, they could augment existing ground-based systems or be a primary sensor on-board a spacecraft. EVS are uniquely suited for space-based operations. With low size, weight, power, and data requirements, they easily fit into tight engineering budgets for space systems. The data may even be well suited for on-board processing due to its sparsity. Despite all these advantages, the sensors are not ready to implement into SDA operations. Creating algorithms to handle the time series data and optimizing the sensor for low-light imaging are areas of active research to improve the utility of these sensors as tools for SDA. To support these efforts, I

develop physics-based end-to-end model for event-based sensing of resident space objects (RSOs). This model adapts previous synthetic event generation methods to operate with photon flux input and precise measures of current. By implementing a model of pixel readout with microsecond-level precision and developing methods to model noise based on dark and induced current levels, I improve the accuracy of the frequency and polarity of the events produced. This accuracy is necessary to extrapolate sensor performance from the model and to generate synthetic events to feed algorithmic development. I also contribute to event-based algorithms through development of an online non-frame based tracking algorithm. In order to validate the sensor model and train the classifying portions of my tracking algorithms, I develop batch-based clustering methods that leverage the temporal dimension which improves the labeling of events between star and noise by 31.8%. Through exploration of different grouping and classifying methods for the tracking algorithm, I attain a maximum of 94.5% group agreement with the batch clustered data and a 97.6% true positive rate and 99.9% true negative rate when classifying satellites on a validation data set. Star classification performance is slightly lower at a 96.7% true positive rate and 96.5% true negative rate. The tracking algorithm's success on this one set of data suggests promising performance from these sensors in future SDA applications.

BIOGRAPHICAL SKETCH

Rachel Oliver is a Major in the United States Space Force. She received a Bachelor of Science in Mechanical Engineering from the United States Military Academy in May 2015 and subsequently commissioned into the United States Air Force as an astronautical engineer. During her first assignment, she earned a Master of Science in Astronautical Engineering from the Air Force Institute of Technology in March 2017. Her master's thesis explores simulated light curve fidelity to determine satellite geometry. From 2017 to 2020, Rachel served at the Air Force Research Laboratories Space Vehicles Directorate. She researched remote thermometry of satellites and managed the thermal device portfolio including innovative technologies such as oscillating heat pipes. Rachel started at Cornell in the fall of 2020 under a National Science Foundation Fellowship. With her military experience in space domain awareness, she focused her studies to develop event-based sensor processing to improve satellite tracking capabilities. During her studies, Rachel transferred to the United States Space Force in February 2021. For the final year of her research she began her new assignment at the Air Force Institute of Technology where she plans to push event-based sensing for space domain awareness further to space-based platforms.

This document is dedicated to all my friends and family who supported me these past 4 years.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to the chair of my committee, Dr. Dmitry Savransky, for his incredible patience and persistence which carried me to the end of this research. For the sheer number of times we discussed optical transmission, you are a saint. I'm also extremely indebted to my defense committee, for their flexibility, understanding, and lessons they taught me along the way. Additionally, this endeavour would not have been possible without the National Science Foundation, who funded my research.

Many thanks to Brian McReynolds, for countless hours discussing event-based sensors. You helped guide my research in the right direction, answering every question I posed. A special thanks should also go to Dr. Dave Monet, who first inspired me to study Space Domain Awareness through his light curve research. It was a pleasure to work with you in tandem this time. Thank you for all your guidance. I am also grateful for the support of Dr. Peter McMahon-Crabtree and Dr. Zach Theis, who generously provided their knowledge, expertise, and support whenever I asked.

I could not have undertaken this journey without my partner, Michael Albert, who fully committed to learn alongside me and has never wavered in supporting me through the good times and the hard times. I would not be where I am today without you. I would also be remiss in not mentioning my parents, sister, and best friend, your support physically and mentally has seen me to the end of this journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Uncertainty is an unavoidable reality of operating a satellite. Around the Earth there are 45,500 space objects that are actively being tracked at the time of writing [88]. For each of these 45,500 space objects, approximately 10,200 of which are currently operational satellites and actively controlled, remote measurements critically provide update information about orbital state and operations. Those measurements inform propagation models, typically two-body equations of motion with perturbations, that help predict satellite states into the future. These models cannot account for every physical force a satellite experiences such as variable environmental conditions like solar radiation pressure or operator choice to impart a force to change the orbit. We make assumptions about these forces to try an improve predictions or simply neglect them. Without perfect knowledge of the external forces acting on a satellite and the command and control of that satellite, the only consistent thing in our knowledge about satellites is a growing deviation between predictions and reality [98]. Therefore, we rely on external update measurements to close that gap. Unfortunately, there is a significant volume of space where satellites exist, so measurement updates are limited in frequency. This constant need for situational updates is why Space Domain Awareness (SDA) is a core competency of the United States Space Force. SDA encompasses more than updating satellite states, however. The competency includes "the effective identification, characterization and understanding of any factor associated with the space domain" [82].

The current challenge in SDA surveillance operations is the increasing com-

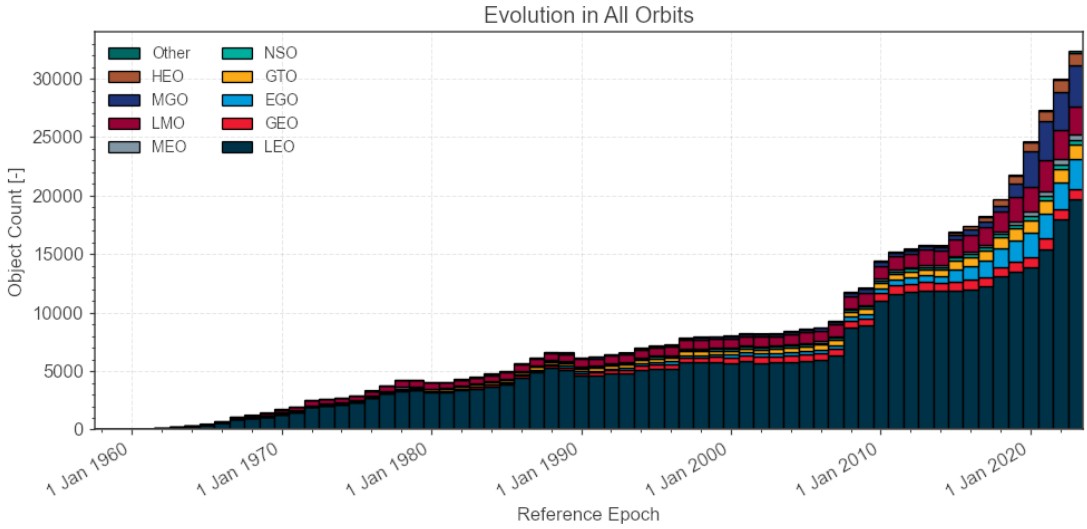


Figure 1.1: The European Space Agency (ESA) annual assessment of the total trackable orbital objects is exponentially growing as launching to space becomes less expensive and more commercial entities can afford large constellations. [72]

plexity in tracking resident space objects (RSO) as the number grows exponentially, shown Figure 1.1 depicts [72]. This growth is a result of lower launch costs decreasing the barrier to utilizing space as a resource and accumulating debris from defunct satellites, launches, and collisions [23]. The accumulation of satellites in orbit is driving growth in the number of conjunction events. Figure 1.2 summarizes the total conjunction events in low Earth orbit (LEO) over the 2022 calendar year [72]. Conjunction events occur when the covariance in a space object's state, its position and velocity, overlaps with another space object's covariance and, therefore, a collision is possible. Measurement updates reduce the covariance, so maintaining low levels of covariance helps identify the most concerning conjunction events where action must be taken. The issue facing SDA operators is limited resources in both personnel and equipment to collect update measurements which cannot keep up with the demand from the

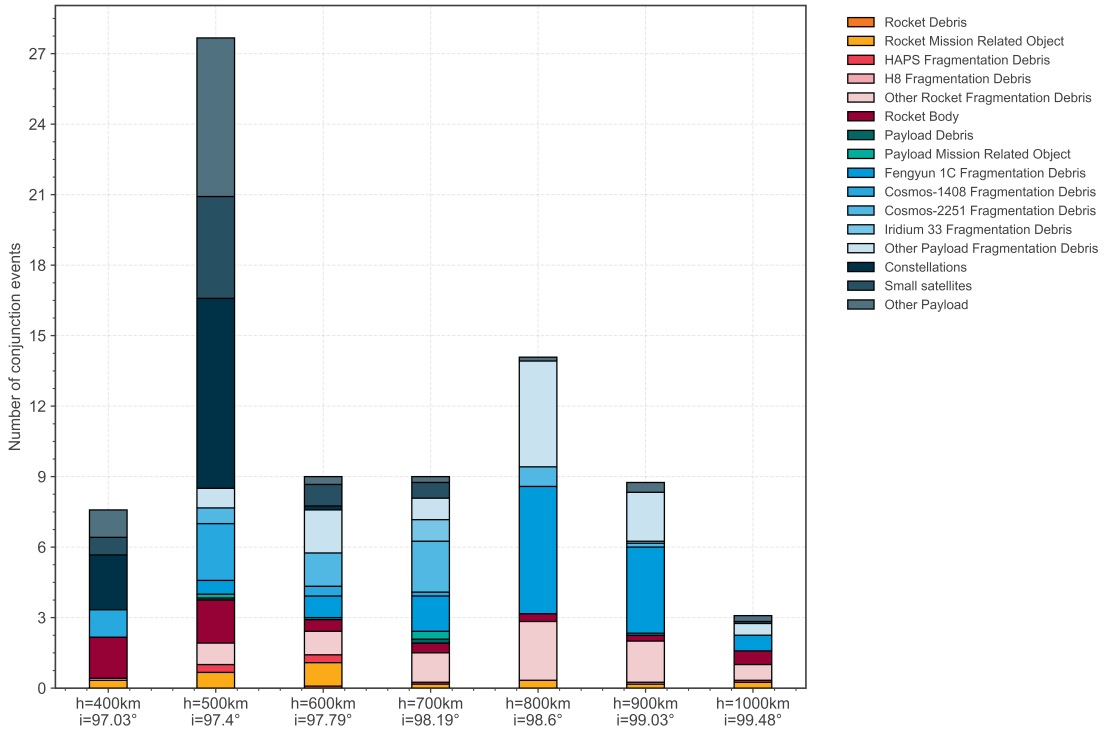


Figure 1.2: The ESA annual report on the space environment highlights the total number of conjunctions events by altitude [72].

increasing number of satellites.

Ideally, future SDA operations will rely on some level of autonomous sensing, conducting nominal updates of the catalogue and monitoring satellite activities where the humans are in the loop to handle the problems identified. Currently, much of the RSO catalog maintenance the United States Space Force conducts uses phased-array radar primarily constructed for missile warning [104]. Despite being well-suited to provide range and range rate of CubeSats and larger satellites for state estimation corrections, the American solid-state, phased-array radar system has a disadvantage of limited geographic positions due to its primary mission to protect the United States. Due to the limited geographic positions, the radar sites are only able to update RSO state information for a portion of the objects. As conjunctions increase in probability due to

the growing number of RSOs, improving coverage via both ground-based and space-based sensors is a priority of the SDA community [23]. Increasing the angular coverage of radar collections is the motivation behind Lockheed Martin's space fence in Australia with a southern hemisphere view. However, the project costs are high, on the order of $1.5 billion for building and more in operating costs [73]. Alternatively, the Space-Based Space Surveillance (SBSS) satellite is a proof of concept electro-optical SDA satellite launched in 2010. Its optic is unimpeded by the atmosphere and weather, and it provides a unique vantage point, but its utility is limited by downlink bandwidth with image analysis occurring on the ground [71]. The most promising technologies to augment the current SDA operations will address at least one of these deficiencies by either lowering physical and logistic costs for ground-based sensing or reducing instrument size, weight, and power (SWaP), and logistic costs for space-based sensing.

Event-based sensor technology has the potential to address both of these deficiencies. Event-based vision sensors (EVS) or neuromorphic sensors have asynchronous pixels that only record data when a logarithmic change in the pixel's photocurrent passes a defined threshold ratio from a previously stored value. The data generated is not contained in a traditional frame, but a time series list with the address-event format

$$e = [t, x, y, p] \tag{1.1}$$

where t is the time stamp of the recording, x and y are the pixel location in the frame, and p is the polarity of the event to indicate if the recorded change was positive or negative, that is, an increase in illumination of the pixel triggers a positive event, and a decrease triggers a negative event. This logarithmic change response has high dynamic range and results in minimal data when observing a static scene. This suits a use case with a reasonably uniform back-

ground, such as the night sky, to minimize data collection to signals resulting from motion of the star field and satellites relative to the observer. Space-based sensing could profoundly benefit from the reduction in data; less data reduces bandwidth requirements of the system and potentially relieves computation power limitations of on-board processing. This is a compounding benefit, where autonomous, on-board computing further processes the collected data prior to down-linking. In addition to data reduction, the lack of a traditional frame yields high temporal sensitivity for changes that occur within the scene. For all paradigms of SDA sensing, higher temporal resolution assists capture of fast moving objects that would typically enter and exit the frame in a traditional exposure time which prevents deductions about the satellite's velocity. In addition to low data and temporal sensitivity, the sensor also benefits from lower energy consumption, ideal for limited power budget systems and latency [35].

Most of the aforementioned benefits have the greatest impact on space-based sensing where resources of power, memory, and bandwidth are limited. However, these sensors also have promise in augmenting ground-based SDA operations. In particular, these sensors are now commercially available with prices on the order of thousands of dollars. When compared with the price of the new radar systems, on the order of billions of dollars, a network of remotely controlled, event-based sensor-equipped telescopes such as the Astrosite network, provide more utility per dollar spent [14].

With these potential benefits in mind, the SDA community began proof-of-concept demonstrations and characterization of commercial-off-the-shelf (COTS) hardware to assess their capability for low-light sensing and noise analysis [63, 81, 65, 40, 39]. Additionally, low-cost proof of concept systems have

been implemented [15, 106, 14]. Despite these efforts, implementation of event-based sensors into the SDA architecture still requires research and development. First, the current COTS hardware is not designed for low-light SDA sensing. The limiting visual magnitude, m_v, of the current hardware given a low-background-noise bias is at best 9.8 m_v in a staring configuration, with the next generation predicted to be at 10.9 m_v [63]. During telescope slewing motion, the additional events from the background variation degrade the limiting magnitude performance [63, 106]. Sensitivity studies have shown that event-based cameras are limited in operation to between 1 and 1.8 visual magnitudes brighter than an integrating camera when viewing the same scene. The second reason event-based sensing is not ready for implementation is that current use of the disaggregated output from the sensors does not take advantage of data minimization. Analyses presented in [15, 63, 106] describe results in the traditional frame versus time format. While this is convenient for comparison against conventional imaging methods, analysis techniques in the native time series data format needs further research and development.

This dissertation contributes steps towards both of the current implementation deficiencies of simplified modeling and lack of processing in the native sparse data format: through development of a physics-informed EVS model with a focus to accurately produce address-event representation data streams and through the exploration of data processing of event-based data tracking algorithm without the need to reassemble standard image frames.

1.1 EVS SDA Simulation

I create a high-fidelity, event-based sensor model to produce accurate address-event representation time series. Having such a model enables efficient hardware exploration: theoretical improvements predicted by the simulation provide insights into parameter settings that may improve sensor performance such as picking a bias threshold setting that limits the noise generation, before investment and analysis. A physics-based model with accurate event signatures also provides consistent simulated data sets that match known truth for algorithm development. My intention for the end-to-end physics-based model is to harness these benefits by effectively replicating what occurs in the analog circuit.

There are previous modeling efforts described in the literature. One effort converts standard video footage into a simulated event-based output [21]. Their simulator provides a processing framework for my simulation, but their version is not tailored to the SDA analysis and makes simplifications to produce a representative event stream output with less emphasis on accurate polarity and event frequencies. These simplifications result in loss of time and polarity information, the signature attributes of sources and noise in the SDA data set. Accurately capturing these features of known truth data sets produces reliable synthetic data to use for model training and algorithmic development. While the model presented in this dissertation builds on their framework, updates are made to simulate a realistic pixel current and allow the current to determine not only events but noise. Chapter 3 describes these methods in detail, but the model can be broken down into two components. First, the model estimates incident power on the focal plane with inputs such as two line element sets,

satellite material and physical properties, observer and sensor properties, and atmospheric conditions. Then the circuitry simulation takes the induced photocurrent and translates that into the binary event output to include noise generation. This second half of the simulation, is what follows the aforementioned effort. However, I alter almost every step in the process. In particular, I develop new methods to simulate shot and high-frequency noise generation. By decoupling the response due to noise and true signals in real EVS SDA data collections and leveraging laboratory data collections of the background noise event rate, I verify the simulation's creation of noise data. Chapters 4 and 5 contain the noise verification and analysis of methods to process the verification data.

1.2 EVS Tracking Algorithm

Traditional processing methods assume the use of traditional imagery and not a time series list of information that contains spatial and temporal data. While one could assemble traditional frames with event-based imagery, that process effectively removes one of the benefits of EVS data, its sparsity. Implementation of EVS is non-trivial to utilize its time series output because the paradigm is relatively new. There are not packaged methods or accepted optimal solutions currently available to achieve effective and computationally efficient algorithms. Therefore, I explore what can be derived by the time series output to develop a tracking methodology that leverages the temporal information. Starting with the data I process for the simulation verification, I begin the development of a probability-informed tracking algorithm with a classic multiple hypothesis tracker (MHT) as my inspiration. At a high level there are 4 steps to an MHT: process new information into clusters, develop hypotheses, confirm

hypotheses, and prune hypotheses and clusters. In this dissertation I explore online cluster development and evaluating hypotheses through Bayesian and machine learning techniques. The methods I explore are outlined in Chapter 6. In Chapter 7 and 8 I pick parameters for these techniques and compare their performance to each other.

CHAPTER 2

BACKGROUND

2.1 EVS Foundations

The predecessor to EVS is the silicon retina. The original electronic version of a retina was constructed in 1970 [33], however Mahowald's 1992 California Institute of Technology dissertation documented the first silicon retina sensors with an asynchronous readout [58]. Mahowald's goal to create an imaging system capable of real-time optical depth perception, stereopsis, motivated her development of the silicon retina. Drawing inspiration from biological systems, Mahowald designed three distinct subsystems, the silicon retina, the silicon optic nerve, and the stereo-correspondence chip. Two of these technologies heavily influence the current EVS technologies, the silicon retina and optic nerve, each sharing several features with their biological counterparts.

The silicon retina mimics its biological counterpart through its structure, parallel processing, and data compression. In biological retinas, there are three primary layers: the photoreceptor, the outer plexiform, and the inner plexiform [56]. The most recognizable of these layers is the photoreceptor layer, constructed of rods and cones, which convert the incident light into an electrical signal. The silicon retina achieves the same process through the use of a photodiode. Biological photoreceptors excite the horizontal and bipolar cells of the outer plexiform layer. The bipolar cells register contrast changes from the spatiotemporal averages between the interconnected horizontal cells. The silicon retinas mimic this process by maintaining a memorized current that is compared to the instantaneous current. The silicon retinas trigger one of two circuits

signifying an ON or OFF event when the difference between the two currents reaches a threshold. The process of memorizing a local current and comparing it to the instantaneous local current on a logarithmic scale is the mechanism that enables the silicon retina to achieve the data compression seen in real biological systems. As Mahowald describes, the "need for data compression arises because ambient light intensity varies over many orders of magnitude, yet the local intensity variation in a single image is usually small" [58]. The logarithmic nature of the silicon retina response makes the sensor sensitive across a large dynamic range and ensures the bandwidth required to capture change is not enormous in any region of photon flux. The final layer of the biological retina, the inner plexiform layer, has many additional types of cells, such as ganglion cells, that do not provide a direct analogy for any part of the silicon retina which is currently simplified to the first layer of synapses. The final similarity to the biological counterpart is the parallel processing. Each cell in the biological and silicon retina is tracking its own memorized current to compare against. The cell independently determines when a significant change occurs yielding an electrical signal for interpretation. Mahowald's truly achieved the parallel processing from her efforts as she moved the readout of the array to an asynchronous fashion with her silicon optical nerve development.

Mahowald's silicon optic nerve was a receiving chip that copies the image from the retina. On the surface that sounds the same as any other optical imaging readout system. However, the "self-timed digital multiplexing technique" she employed with an "address-event representation" stepped away from the scanning of traditional imaging systems to create something closer to the "action-potential representation used by real neurons" [58]. Instead of scanning each row after a set exposure time interval as with traditional imagers,

Mahowald suggested the optic nerve readout should allow for the firing pixel neurons to readout without reading out their surrounding neighbors. After any pixel fires in her design, the event address is read out by two separate arbiters, one for the x pixel location and one for the y pixel location. There is some semblance of the scanning remaining, because each x pixel is addressed after the y "row" is selected to ensure the time multiplexed output points to the pixel that fired, not some random combination of row and columns. When the array is filled with hot neurons, then the readout resembles a traditional scanning formula [9]. This schema of reading out the row and column reduces the number of wires of the readout circuit for a circuit of N pixels from N to $\log_2 N$ wires. Additionally, the time-division-based multiplexing of the arbiter output gives each readout pixel address its own unique timestamp, creating a time series list of an event output.

These two basic components together, the silicon retina and the silicon optical nerve, comprise the foundational technologies on which other event-based neuromorphic sensors have grown. For EVS processing visible wavelength light, there have been massive improvements to the silicon retina to improve quantum efficiency and different arbitration styles developed to leverage the technology for different tasks.

2.2 Commercial EVS

In the last 30 years since Mahowald's thesis, the silicon retina and optic nerve has become a commercially available technology. There are two primary companies manufacturing EVS, Prophesee and iniVation. Table 2.1 summarizes the

currently available sensors with applicable attributes. In this disseration, I work with data primarily from the 3rd generation Prophesee VGA-CD sensor. I describe this data set in Section 3.4.1. I also work with background noise data collected on the subsequent generation of the Prophesee sensor with the IMX646 chip as I outline in Section 3.3.6.

Table 2.1: Commercially Available EVS [18, 76, 78, 77, 48, 30]

Camera Parameter	**DAVIS346**	**DVXplorer**	**DVXplorer Micro**	**DVXplorer Lite**	**VGA-CD**	**IMX 636 (637)**	**IMX 646 (647)**	**Metavision GENX320**
Manufacturer	iniVation	iniVation	iniVation	iniVation	Prophesee	Prophesee/ Sony	Prophesee/ Sony	Prophesee
FPA format (pixels)	346 x 260	640 x 480	640 x 480	320 x 240	640 x 480	1280 x 720 (640 x 512)	1280 x 720 (640 x 512)	320 x 320
Pixel pitch (um)	18.5	9	9	18	15	4.86	4.86	6.3
Max Events per Second	1.20E+07	1.65E+08	4.50E+08	1.00E+08	6.60E+07	1.06E+09	1.06E+09	1.00E+07
Dynamic range (dB)	120†	110†	110†	110†	120⋆	86‡	110‡	120⋆
Temporal Resolution (us)	1	65-200	65-200	65-200	Unknown	<100	<100	100
Latency (us)	<1000	<1000	<1000	<1000	40-200	100-1000	800-9000	<150

† inivation Dynamic Range 50% pixel response with 80% contrast

‡ Sony Dynamic Range 50% pixel response with 100% contrast

⋆ Prophesee unknown Dynamic Range Measure

2.3 EVS Observing Space Objects

Drawn by the unique combination of high dynamic range and temporal sensitivity, the first documented application of EVS to space domain awareness answered the question of whether these sensors, not designed for low-light sensing, could capture satellite movement from the ground [15]. Using the DAVIS240 and ATIS EVS mounted to a Meade LX200 telescope with an 8 inch diameter and focal speed of f/10, the sensors captured observations of the night sky. To process the data they generated traditional image frames and develop an event visualization surface that fits an exponentially decaying function to the last event spike time. The event visualization surface captured the temporal aspect of the 3-dimensional data through coloring based on the most recent events fired. While their techniques leveraged traditional frame methods, the effort demonstrated that the sensors are capable of observing RSOs in LEO and geostationary orbit (GEO) satellites both during the night and day. After this initial demonstration, different research efforts continued to build on the observational techniques to collect stars and satellite data. Żolnowski, for example, examined the effect of slewing at different sidereal rates on the limiting visual magnitude in empirical event-data [106].

Inspired by the observational success, the work of McMahon-Crabtree and Monet took a step towards theorizing the physical limitations of commercial off the shelf (COTS) EVS capabilities for space surveillance applications in 2021 [63]. Their assessment of third-generation COTS hardware is particularly important to the work within this dissertation because the validation data discussed in Section 3.4.1 is from a third-generation sensor using the same optical train. They theorized that the limiting visual magnitude (m_V), the relative

brightness of point source objects in the night sky scaled off of a reference star, is constrained by the on-sky temporal contrast, TC_{on-sky}, and the dark current rate of the detector. The on-sky temporal contrast

$$TC_{on-sky} = \frac{I_{pho}^{obj}}{I_{pho}^{sky} + I_{dark}} \tag{2.1}$$

is the ratio between the induced sensor photocurrent due to any object in the sky I_{pho}^{obj} and the summation of the induced sensor photocurrent due to the sky background, I_{pho}^{sky}, and the sensor dark current, I_{dark}. This metric derives from the contrast operations of an event based sensor. To induce at least one event, the difference in photocurrent from the nominal background, which is a combination of the nominal sky background current and dark current, will have to reach some percentage change, most easily assessed as a ratio. The other constraint

$$(I_{pho}^{obj} + I_{pho}^{sky}) \geq I_{dark} \tag{2.2}$$

ensures the minimum m_V is above the noise floor of the sensor. McMahon-Crabtree and Monet related the object and sky brightness to the m_V with Planck's equation

$$E_\lambda(\lambda, m_V, T_{eff}) = C_1 \lambda^{-4} 10^{-0.4 m_V} \frac{\exp\left[\frac{C_2}{(0.55 T_{eff})} - 1\right]}{\exp\left[\frac{C_2}{(10^6 \lambda T_{eff})} - 1\right]} \tag{2.3}$$

where $C_1 = 9.627x10^{-9}$ m-photons/sec, $C_2 = 1.44x10^4 \mu$m K, wavelength λ is in meters, and they let $T_{eff} = 5920$K for a G0-type star. After converting the current terms to m_V terms with Plank's equation and optical system-specific considerations such as aperture diameter, they compared the derived limit of 9.7 m_V to data collected with a third generation sensor. To collect this data, they pointed their sensor to a fixed azimuth and elevation for observations of approximately 20 seconds. The observable stars in these staring data sets are relatively fixed

to a right ascension and declination in the barycentric coordinate system. They move through the frame as the Earth rotates. While the induced motion is minimal, it is enough to generate events. From this data, they extracted a 9.8 m_V, by identifying the stars in the field of view. This m_V effectively validated their limiting magnitude model. In Section 3.1, I directly utilize this empirical m_V limit to filter the star catalogue for stars appropriate for simulation.

2.4 Foundational Event-Based Simulation

One of the two original primary manufacturers of event-based sensors built a simulation tool, v2e, to market event-based data to external audiences [21]. The primary focus of this simulation tool was to take normal frame rate video and produce a semi-realistic event stream that captured the movement within the scene. The name, v2e, stands for video to event. The tool captured many aspects of the EVS circuitry, so I use it as a foundation for my own event generation simulation.

Following the flow chart in Figure 2.1 the event generator starts with the frames of a video input. These can also be a series of synthetically generated images. If the imagery is in color, the v2e tool first converts the imagery into grayscale to treat the pixel digital number values as luma. This is how v2e estimates pixel current. To approximate the flow of current between the lower temporal frames of traditional video, v2e applies an interpolation between frames to upsample the temporal dimension. This is an important step because v2e records the maximum possible events between frames and evenly subdivides between them. The closer these frames approach the temporal resolution of the

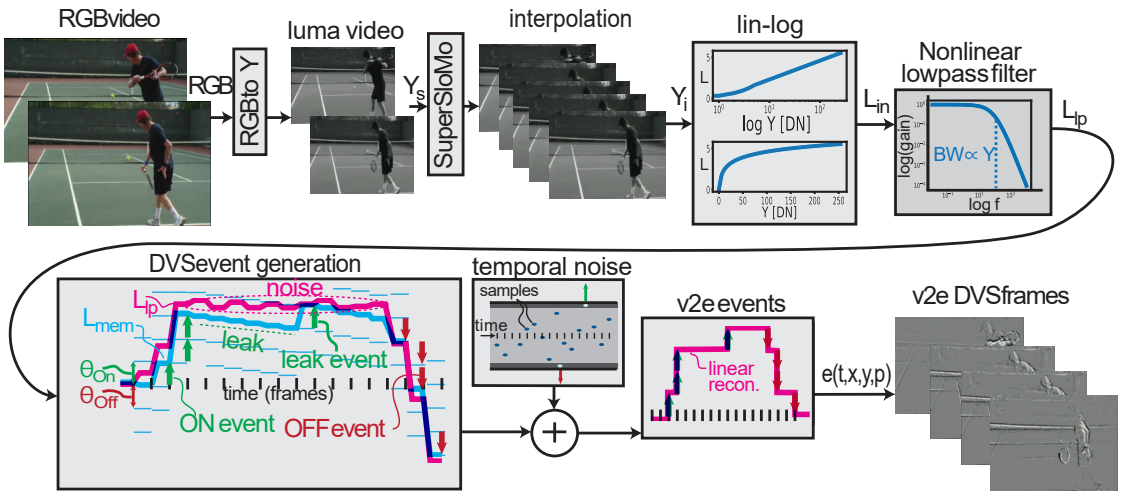


Figure 2.1: The v2e toolbox takes standard frame rate video and approximates an event stream output [21]. The video is manipulated by first converting to gray-scale luma and then interpolating frames to artificially increase the frame rate. Then the current value on each pixel is estimated with a mapping between the digital luma values and current. The estimated current is then converted to a logarithmic scale and passed through a low pass filter. After these steps, the current values are compared over time to generate events and temporal noise is added at a set rate. The event streams are then reconstructed into artificial frames to demonstrate event output in a familiar format.

sensor, the less artifacts exist in the synthetic event times. The problem is interpolation between frames to the microsecond recording resolution is memory intensive.

After preparing the traditional frame data, v2e then simulates components of the circuitry to determine when events trigger. Because the current comparison occurs in the logarithmic scale, v2e first converts the luma intensity value, a digital number, Y_s, into the log scale

$$L_{in} = f(Y_s) = \begin{cases} \frac{Y_s}{5}\ln(5) & Y_s < 5 \\ \ln(Y_s) & Y_s \geq 5. \end{cases} \tag{2.4}$$

This conversion is piecewise to avoid the log function sensitivities close to 0. When working with digital numbers that do not dynamically scale, the linear conversion is reasonable. When operating on currents at the order of fem-

toamps, however, linearization close to 0 prevents event generation. For this reason, I do not use a piecewise conversion in the simulation I describe in Section 3.3.2.

Once in the log scale, v2e captures another physical phenomenon of EVS sensors. Despite their temporal sensitivity, the circuitry does not instantly respond to a change in photocurrent. Therefore, v2e applies a discrete 2 stage low-pass filter

$$\begin{aligned} f_{3dB} &= ((Y_s + 20)/275) \times f_{3dBmax} \\ \epsilon &= \arg\min(2\pi\Delta t f_{3db}, 1) \\ L_1 &\longleftarrow (1-\epsilon)L_1 + \epsilon L_{in} \\ L_{1p} &\longleftarrow (1-\epsilon)L_1 + \epsilon L_1 \end{aligned} \tag{2.5}$$

to estimate the compared current, L_{1p}, at the simulated time. The filter's corner frequency, f_{3dB}, depends on the induced photocurrent and the simulation time step, Δt. Since v2e operates on the digital number of a traditional image, they map the corner frequency value to the non-logarithmic digital number value. Section 3.3.3 covers my low-pass filter construction to handle the current values, as opposed to digital numbers, to scale the corner frequency.

Now that the current is in the log-scale and the circuity response is accounted for, v2e compares the current to a previously memorized current, L_{mem} for each pixel. This comparison is a ratio between the two currents on the analog circuit.

Since the current values are in the log scale this relationship becomes

$$\begin{aligned} \Delta L &= L_{1p} - L_{mem} \\ \theta &= \begin{cases} \theta_{ON} & \Delta L >= 0 \\ \theta_{OFF} & \Delta L < 0 \end{cases} \\ N_e &= floor\left(\frac{\Delta L}{\theta}\right) \\ L_{mem} &\longleftarrow L_{mem} + Ne \times \theta \end{aligned} \tag{2.6}$$

where N_e is the idealized number of ON and OFF events for the simulation step. The number of events is ideal because the simulation does not take into account the readout time or the refractory period which limits the frequency response of the pixel. Instead, v2e takes N_e for each pixel and subdivides the simulation step equally in order to assign readout times. Across the array, the v2e readout can have the same readout time for multiple pixels. This phenomenon does not accurately capture the silicon retina which can only readout one pixel at a time. While the readout can assign multiple events to the same microsecond, the arbiters' scanning of the row and columns still organizes the event readout spatially. Since the events are equally subdivided between the frames, v2e artificially induces consistent frequencies in the event generation. Since I am considering synthetic data for algorithmic development which will utilize the temporal signatures found in the timing between events, I include the readout and refractory period in my modified simulation. I discuss these modifications in Sections 3.26 and 3.3.5.

The last aspect of synthetic event generation v2e covers is noise generation. v2e models 2 noise event sources: temporal noise and leak rate noise. The temporal noise, from the quantal nature of the photons, results in temporal variations of the current which can induce shot noise events. Since shot noise is

inversely proportional to photocurrent, v2e models the temporal noise with a Poisson methodology

$$\begin{aligned} r &= ((F-1) \times Y_s + 1) \times R_n \\ p &= r \times \delta t \\ u < p &: event_{OFF} \\ u > (1-p) &: event_{ON}, \end{aligned} \tag{2.7}$$

which scales an observed shot noise rate, R_n, by the non-logarithmic digital number representation of intensity through use of linear function with the slope, F, to determine the pixel-wise noise rate, r. With this rate, they calculate the probability of a shot noise event, p, and then sample a uniform distribution, u, between 0 and 1. If u if less than p or greater than $(1-p)$ they add an additional event to the pixel at that simulation step. While this methodology utilizes the digital number equivalent of intensity to inform the shot noise induced, I make significant modifications to the simulation to work with the electron rate information my simulation has available.

The second and final noise source v2e models is junction leak noise. This noise occurs as photocurrent leaks between components in the EVS circuitry. Particularly, the memorized current drops over time as a result of junction leakage and parasitic photocurrent in the change detector reset switch [69]. When the memorized current drops, an ON event will occur when the change between the memorized current and the logarithmic induced photocurrent is exceed the logarithmic threshold, even if the induced photocurrent has not changed. v2e models the leak rate events by

$$\begin{aligned} \delta_{leak} &= \Delta t R_{leak} \theta_{ON} \\ L_{mem} &\longleftarrow L_{mem} - \delta_{leak} \end{aligned} \tag{2.8}$$

subtracting a constant current leak value, δ_{leak}, from the memorized current at each frame. The δ_{leak} value calculates the events per frame by multiplying the leak event rate, R_{leak} in events per second, by the time between frames, Δt, and the amount of change required to induce an ON event, the threshold θ_{ON}. While the modeled leakage is a real phenomenon, it only produces ON events. Empirical data shows the background noise event rate on illuminated pixels induces both ON and OFF events. Since future algorithms will need to reject noise, I want my synthetic event generation to capture their realistic characteristics in timing and polarity. Therefore, I explore modifications to capture the circuit noise phenomenon that produces both ON and OFF events. Section 3.3.6 describes both of these noise modeling modifications in detail.

2.5 EVS Space Object Tracking

Since EVS can observe some RSOs, there has been interest in observing the night sky. The next logical step is to determine what can be done with the event information from those observations. There have been previous efforts and theories of how to apply these sensors to satellite and star tracking. My overarching goal in this dissertation is be able to conduct SDA satellite tracking with EVS, so I introduce the state of the art tracking methods currently in use in this Section.

Because star events are in nearly every EVS sky data set, stars are a logical place to start the development of point source tracking algorithms. It only takes good viewing conditions and proper EVS settings to collect more data sets for the algorithm development. The first star algorithms focused on building star maps to conduct astrometry with the event-based data [16]. Building off this

foundation, the first online star tracker collapsed events into traditional image frames to prove they could use event data of 40 ms windows to determine relative rotation of an observer looking at a star field [13]. In the follow-on algorithm, the same research group applied progressive multi-resolution Hough transforms to extract the relative rotation of the observing sensor [2]. This technique still required some level of maintenance of events defining a bank of Hough Transforms taken over multiple time resolutions to capture the rotation. The most recent star-tracking algorithm, applied Kalman filtering to the event data by calculating the log likelihood of a pixel, which has a new event, of being a star. If the likelihood exceeded a threshold, the state of the closest star updated using the new measurement information [67]. While I focus on extracting satellite events from event-based data in this dissertation, I inherently identify star sources during the data rejection process as I discover in Section 7.2.6. Applying the aforementioned star tracking techniques to the isolated star data is an alternative option to satellite tracking after utilizing my data rejection techniques.

Another effort, aligned with my own, developed a tracking algorithm that operates in the ideal event-based domain with the primary goal of determining whether the event from a point source object is a real detection of a RSO [1]. This algorithm functioned primarily on geometric methods, building a 2-dimensional time surface that keeps track of the latest event on a pixel. The surface

$$S(x_k) = \exp \frac{T(x_k) - t_i}{\tau} \tag{2.9}$$

scales its value for each pixel, x_k, for k indexes through the exponentially decaying function driven by difference in time between the last event time on a pixel, $T(x_k)$, and the current time, t_i, that is scaled by the time constant, τ. Using

a 15 by 15 pixel region of interest (ROI) around the most recent event on this 2-dimensional surface, they multiplied these with a set of angular templates that activate when the new event is at the tip of a streak on the 2-dimensional time surface. After passing the surface activation test, their algorithm used a stored look up table to convert the ROI values into an angular activation vector which has the number of events either ON or OFF at different angles through the ROI. If the angular activation vector had values above an event threshold, Γ, they used the measurements above Γ to calculate the mean angle above the threshold and max angle above the threshold. If the angular difference between the mean and max was below an angular threshold, δ, the events were considered to be in agreement angularly and pass the "unimodality" test. The events passing all of the former conditions are considered to be proper detection events of RSOs. The authors then managed track hypotheses informed by the detection events and used the position and rate of previous events to predict the object location over time. Overall, their detection algorithm had a true positive rate of 90% for ON events and 87% for OFF events and a true negative rate of 99% for ON events and 97% for OFF events. I construct the tracking methodology in Chapter 6 differently. I move away from individual event detection thresholding and 2-dimensional time surfaces. Instead, I relying on grouping geometrically close events and use their inherent features in time and space to classify at the pixel and group levels. Both our methods, however, follow the same structure of limiting the event information to what is desirable prior to applying additional algorithms.

Another algorithm aimed at identifying point source objects, developed a Probabilistic Multiple-Hypothesis Tracker (PMHT) to find tracks in sky field EVS data [12]. This author explored the PMHT using both a frame-based

method and Poisson Prior method which processed independent events as they arrive. Their PMHT algorithm aimed to find the best estimate target position and velocity states through the factoring of the state's joint probability term as the product of single measurement terms. The Poisson Prior version treated each trackable object as an independent Poisson measurement source with an unknown measurement rate. Their algorithm estimated the measurement rate along with the target state. The general success of this effort inspired my consideration of a MHT format for my own processing techniques that I introduce in Chapter 6. Distinctly, however, I focus on pre-processing the events to reject data from undesirable sources. Therefore, I apply classifying instead of the pure estimation of the PMHT work. A PMHT may be a good method to apply to the events after they have been reduced to only the most likely satellite events.

Not all the RSOs being tracked are satellites and stars. One similar effort to my own attempted to detect, track, and identify meteor RSOs in event-based data [97]. This research effort went through a very similar process as the progression seen in this dissertation. First, they attempted to simulate meteor trajectories by feeding video from a sky simulation software through the v2e event generation tool. Interestingly, they noted the validity issues in this simulated data for algorithmic development due to its lack of noise events. The simulation that I develop in Section 3.3 directly addresses the issues they found in the v2e method. These researchers did not try to improve the simulation and instead turned to real data collections of meteors. They used a mount tracking at the sidereal rate to limit events from stars. They manually annotate the truth data they collected, but their method was a slightly different approach than the one I develop in Section 3.4.3. Instead of relying on the 3-dimensional data as I do, they compressed the events into 2-dimensions to find the first and

last events in their manually selected streaks. Fitting events to a line between the first and last event and filtering out noise through a projection error threshold, the authors created their ground truth data for meteor detection. Similar to my proposed tracking algorithm in Chapter 6, the authors first focused on other signal suppression. However, they do not utilize classification in this process and, instead, applied their classification to the final track. To reduce other signals, the authors applied a pixel mask over individual pixels that produce events frequently, such as hot pixels and the stars which induce events through atmospheric turbulence. After suppressing some pixels, the authors filtered for events occurring in quick succession to identify the leading edge of events and develop a track through "spatio-temporal nearest neighbors." After identifying a track, the authors recognized a simple duration can distinguish between their two classes of aircraft or meteor with the final completed track. However, as a proof of concept, they trained an SVM with a single attribute, a temporal profile that counts the ON events that the source generates over time. How the profile bins events over time is unclear. They used this profile to train their SVM with approximately 20 example profiles of each class. While their method identified final track profiles for a unresolved object in event-based data, my tracking algorithm's classification I describe in Section 6.2 happens online to assist with filtering data before using the event data in algorithms that could assemble the track, fit an orbit, or update a state estimate of a RSO.

One final innovative approach to utilizing EVS for RSO tracking, AURORAS, theorized performance of an algorithm to fit an RSO orbit through use of the EVS data that simultaneously measures the angular position, velocity, and acceleration [6]. With all three of these components and use of the Laplace's initial orbit determination (IOD) approach, they derived equations to extract

the satellite's state vector describing its current position and velocity with only a single measurement. A single measurement in this case would be enough events to estimate the velocity and acceleration. The key to their approach is a high enough time resolution to extract the angular velocity and acceleration information with one pass. The authors listed multiple sensor options to accomplish this task: single photon avalanche diode (SPAD) arrays, electron multiplied charge coupled devices (EMCCDs), and EVS. The EVS is a good choice for the theorized algorithm because it has sufficient time resolutions, it can be tuned to limit the noise events, it can provide the reference stars required with specific operational constraints, and the change detection limits the data volume to conduct the processing. While the authors validated the utility of the AURORAS algorithm using a SPAD array simulation, they acknowledged that EVS technology is a good fit to utilize AURORAS.

2.6 EVS in the Space Environment

Considering the low data output, power consumption, and overall volume of these sensors, EVS seem ideal for space-based applications where the power is limited and small amounts of data are either easier to process onboard a satellite or uses less of the link budget to downlink information from orbit. Due to this promise, initial steps are being taken to assess the current technologies for spaceflight. Understandably, current versions of the hardware are not designed for space-based applications. The engineering does not take into account radiation hardening or other considerations to survive the space environment because it would add expense to the current iterations of the hardware.

Given the fact that the hardware is currently lacking design considerations for space utilization, a collaborative effort between the University of Pittsburgh, Sorbonne Université, Carnegie Mellon University, and Universidad de Sevilla evaluated the performance of an event-based sensor during irradiation by a wide-spectrum neutron source at Los Alamos Neutron Science Center. The team evaluated the radiation-induced damage to the sensor and the radiative effect on the signal-to-noise ratio of the output at different angles of incidence of the beam source [86].

This experiment was performed on an older generation of the EVS with a 15 μm×15 μm pixel pitch and a fill factor of only 25%. Over the course of 2 days of testing, the sensor was subjected to a range of power of wide-spectrum neutrons between 0.1 MeV to 600 MeV at with a 0° and 90° orientation of the camera with respect to the incident neutron beam. For some tests, the lens cap was left on the sensor to capture primarily noise as the consistently uniform background of the lens cap should not induce events. Another subset of tests removed the lens cap to observe the effect of relative background illumination comparing the event rate output in 0 lux and 500 lux environments. Finally, the study imaged a gyroscope to determine the effect of the radiation to ascertain real moving signals from the noise. For all tests the sensor is set at a fixed distance from the beam source to have control of the effective neutron flux.

When compared to the control measurement with the beam off, there was a 22 times increase in the rate of events generated per second with the lens cap in place tests. The ratio of generated ON events to OFF events was (3:1) assuming equal threshold settings for the on and OFF events of the camera. Across the array, there were more events at higher values of (x,y) pixel locations, but that

was attributed to human error in positioning the sensor in the neutron beam. At the two tested angles of incidence, no statistically significant difference was found using a Mann-Whittney U test [59].

With the lens cap off, the background illumination study found the sensor 1.5 times more sensitive in the 0 lux environment. This response intuitively follows the higher sensitivity, because of the logarithmic scale, the closer the sensor is to the dark current level of the circuit. To estimate the signal to noise with incident radiation, the study observed a moving gyroscope. It then compared the average event rate over the total scene against that of the isolated radiation-induced noise events. The observation with the gyroscope signal was 3.355 times higher than that of the isolated radiation. Since this was a sheer comparison of the total events measured across the array divided by time, this experiment did not take into account the fact that only some pixels experience the signal throughout the gyroscope measurement. Therefore, it may have underestimated the signal to noise ratio local to a changing signal. A secondary analysis of a smaller bounded area, 50 x 50 pixels, where the gyroscope enters and exits the field provided the frequency of the gyroscope oscillation with and without the additional neutron radiation to prove a signal with enough intensity will overcome the neutron radiation noise.

The most intriguing part of the neutron noise study was the analysis of the patterns induced by this type of noise source on an EVS and how it can be replicated. The study divided the noise generated into two categories: clusters and line segments. Unlike shot noise events, which are isolated to an individual pixel, noise generated by an incident neutron can excite multiple pixels. When an incident neutron interacts with the array head on, a burst relatively uniform

in its spread on the focal plane occurs. Alternatively, when the neutron strikes the array at an angle, the spread of the affected pixels becomes more linear. The angle of incidence influenced the prevalence of one category of noise over the other with 7 times more line segments occurring when the EVS is oriented 90° from the incident beam than when directly facing the beam. The linear noise is of particular note due to the possible similarities in tracked object patterns based on the EVS hosting satellite's attitude control methodology. If the observing methodology resembles the ground-based staring or sidereal tracking to survey stars and satellites resulting in data similar to that discussed in Section 3.4.1, then linear neutron noise artifacts may be identified as a trackable object. This should be kept in mind during algorithm development for space-based star and satellite tracking.

The study took the noise analysis a step further by proposing a model to generate neutron noise. The probability of a neutron event anywhere in the array can be expressed as a rate for a given time step. Therefore, the study modeled noise injection at the microsecond level as

$$P(\lambda) = \lambda e^{-\lambda} \tag{2.10}$$

where λ is the rate of events per microsecond. The Poisson distribution produces an integer number of neutron-induced events per simulation step and the starting location of said event is modeled by a uniform distribution of the array. The proposed model then assigns each simulated neutron event as either a cluster or line segment event by drawing from a probability distribution defined as

$$\begin{aligned} P(Cluster) &= |\cos(\theta + \epsilon)| \\ P(LineSegment) &= 1 - P(Cluster) \end{aligned} \tag{2.11}$$

to make it a function of the incident angle of the beam. Once they select a type of noise, they randomly chose pixels excited around the initial pixel of a cluster noise event and they randomly selected the direction of excited pixels between 0° and 360° for the line segment event. Finally, the generation of ON and OFF events was also randomly generated. The study sampled the duration of ON events and OFF events, t_{ON} and t_{OFF} respectively, from Gaussian distributions defined by the empirically collected data. During the ON event window the probability of an ON event

$$P(ONEvent, t) = \exp\left(\frac{-(t - t_{ON})^2}{2\sigma_{ON}}\right) \tag{2.12}$$

resembles a Poisson distribution where the rate is a function of the time from the Gaussian sampled ON window length and the standard deviation of the window, σ_{ON}. After the ON event window there is a period of time before OFF events start recording. The study therefore sampled a Gaussian waiting period, t_{wait}, to define the front bound of the OFF event window. During the duration of the OFF event window, the study defined the probability of an OFF event as another Poisson distribution

$$P(OFFEvent, t) = \frac{1}{\beta}\exp\left(-\frac{1}{\beta}t\right). \tag{2.13}$$

To test this model, the study injected noise using this method onto an existing data set without the incident beam. They matched the observed pattern of event rates with 5% error for the ON rates and 12% error for the OFF rates.

While the simulation in this dissertation only reproduces ground-based collections, I desire to advance the simulation into space-based applications as described in Section 5.3.3 where the neutron noise will be a more prevalent feature that should be taken into account. The current simulation methodology proposed in the neutron study focuses on modifying existing data to add in rep-

resentative neutron-generated noise. While the statistics of the randomness of the neutron striking and angles are valid, I would like to approach the neutron noise generation like the dark shot noise generation in Section 3.3.6 and connect it to the fundamental operation of the camera to create a resulting event stream. Overall, the neutron noise study on the cameras provides a foundation for more accurate space-based EVS simulations and affirmation that COTS EVS are capable of operating and providing useful information in a space environment without modification.

Quickly following confirmation of the likely successful operation of a space-based EVS, the Falcon Neuro project [61], run by the United States Space Force Academy, launched two DAVIS 240C cameras on the International Space Station in December 2021. This launch was only a few months after the neutron noise study was published. The Falcon Neuro hardware may have been delivered to meet the Space Test Program (STP) launch even before the neutron radiation publication. It is not surprising given how relatively inexpensive EVS are compared to other space rated hardware, that the Falcon Neuro project materialized prior to the radiation assessment of the cameras.

The Falcon Neuro project's primary objective leveraged the temporal advantages of EVS to examine lightning and sprites from orbit [61]. Sprites are large-scale electric discharges, typically over mesoscale thunderstorms following positive cloud to ground lightning, that occur in the troposphere creating a cold plasma phenomena [85, 62]. The Falcon Neuro project points one of its two cameras towards the nadir direction, towards the surface of the Earth, and the other camera towards the RAM direction, the direction of motion of the ISS, to capture lightning and sprite events from directly above and from a profile

view. While the 2022 Optical Engineering publication on Falcon Neuro does not include analysis on lightning and sprite events due to the short duration of operations at the time of publication, the team demonstrated reconstruction of imagery using the event-based data imaging the Honduras coastline, effectively checking out the camera's operational capabilities in orbit. Between the Falcon Neuro project and baseline testing of resilience to radiation, COTS EVS technologies show promise for on-orbit use cases.

CHAPTER 3

SIMULATION METHODOLOGY

The intent of this research is to move away from the generation of synthetic event streams based on estimated current values and injection of noise. A simulation that generates photometric information before the camera, in terms of photon flux, that is then converted to photocurrent and stepped through the analog circuitry not only provides an opportunity for verifiable and tunable synthetic event streams, but insight into the event generation. To build a higher-fidelity simulation, I break the end-to-end simulation of the event-based sensor into two phases, the physics-based front-end of the model and the circuit-based back-end.

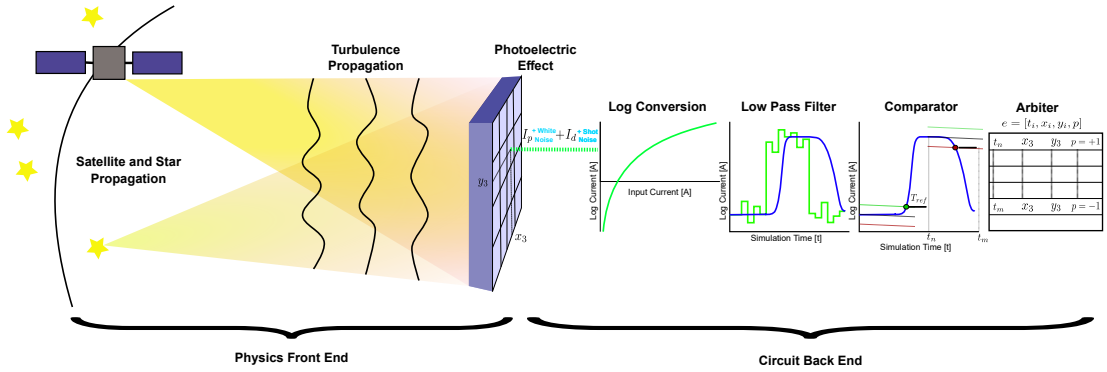


Figure 3.1: The simulation is broken into two components, the physics-based front-end of the model and the circuitry-based back-end of the model. The front-end incorporates dynamics of observable objects, computes the energy sent by those objects, and propagates that energy through a simulated atmosphere. The back-end of the model encompasses the conversion to photocurrent, noise on that current, conversion to a logarithmic scale, low pass filtering the current values, comparison of the current to a memorized one, and finally arbitrating an output event stream.

In the front-end model, I include dynamics and energy propagation in Section 3.2. I model impacts of the camera pointing and the position of signal pro-

ducing objects such as satellites and stars through an orbital propagator model discussed in Section 3.1. Then in Section 3.2.2, I propagate light from each point source through the application of an optical split-step propagation with phase screens built with Zernike modes to model the distortion during atmospheric transmission. Finally, I take a final semi-analytical Fourier transform of the final field occluded with a mask to generate the final field incident on the sensor which I describe in Section 3.2.3.

In the back-end model, my goal is to accurately simulate the camera's behavior in low-light sensing situations to capture the timings between events and their polarities. At a fundamental level, EVS circuitry compares induced current on a log scale to a previously memorized value. Generally, the steps in the simulation follow the processing of the induced photocurrent imposed by the hardware. Therefore, I first convert the current values to a logarithmic scale in Section 3.3.2. Then in Section 3.3.3, I pass the current value through a low pass filter that captures the pixel's temporal response in low lighting conditions. Subsequently, I compare the resulting current value against a previously memorized value to determine if an event is recorded as I describe in Section 3.3.4. Finally, I record the triggered events with the method in Section 3.3.5 to mimic the systematic arbitration process.

3.1 Dynamics

I start each simulation by determining which stars and satellites are in the observer field of view (FOV) at each simulated time step. This process starts with some critical definitions of the observing system. The relevant observer prop-

erties include a viewing location, pointing of the observer, and either an angular definition of the true FOV or parameters to calculate the true FOV. These conditions are situationally dependent. For example, the data used to validate this simulation, discussed in Section 3.4.1, was taken from Albuquerque, New Mexico with a ground-based Az-El telescope mount. Therefore, the viewing location is described as a fixed latitude, longitude, and elevation on an Earth-fixed geodetic coordinate system. For space-based scenarios, the Earth-fixed system should be swapped to a set of coordinates, such as position and velocity of the satellite in the Earth-centered inertial (ECI) frame. For the validating data sets, the pointing is fixed to a right ascension (RA), declination (DEC), and angle of rotation (θ) about the axis defined between the observing location and (RA, DEC) on the celestial sphere for a 30 second collection. Therefore, I set the pointing of the simulation to a fixed (RA,DEC,θ) for the simulations in this dissertation. Dynamics of the observer are included as an option in the user-defined parameters, allowing for modeling of more complex scenarios. Finally, I determine the FOV by either manually setting it through angular definitions or as

$$FOV = 2\tan^{-1}\left(\frac{L}{2f}\right), \tag{3.1}$$

which defines the FOV through the optical system's parameters of focal length, f, and the height or width of the focal plane, L. In the case of the data described in Section 3.4.1, the 85 mm focal length and 15 μm for the width of 640 pixels make the FOV 6.5°.

After all of these user-defined parameters are chosen, I determine what satellite and star objects are within the FOV. The angular definition of the FOV and the (RA, DEC, θ) of the center of the FOV at each time step defines the maximum and minimum (RA, DEC) coordinates on the outside of the FOV, as shown in

Figure 3.2. These (RA, DEC) bounds are sufficient to find the stars and satellites within the FOV. Drawing from the European Space Agency's Hipparcos star catalogue [25], star locations are defined in time as existing at an (RA, DEC) on the barycentric coordinate system sphere. I propagate these (RA, DEC) locations to the time of simulation to capture the proper motion of the stars. I filter the catalogue down to the maximum and minimum (RA, DEC) within the full simulation time to account for potential movement of the observer's pointing direction. Next I query the Gaia G-band magnitude for stars remaining in the catalogue [17, 27, 100] and again filter by a limiting magnitude to reduce the point sources generated. I rely on prior studies on the 3rd generation camera collecting my validation data for a limit of 9.8 m_V [63]. Finally, for each time step, I apply the more restrictive FOV bounds to have a concise list of all stars bright enough for the EVS to detect in each simulation step.

Alongside the tracking of star objects within the FOV, I track the mapping of (x, y) pixel locations to the (RA, DEC) locations of the star and satellite objects. The mapping of a spherical coordinate system onto a flat plane is a non-trivial task with multiple options, which are typically formulated for geographic mapping purposes because our civilization has been depicting the surface of Earth on flat projections for centuries. The RA and DEC in the barycentric coordinate system are akin to latitude and longitude of the geocentric coordinate system as both describe position with angles on a spherical object. Both of these coordinates parameterizations can be equated to the sphere that they lie upon being divided into great and small circles. Great circles are the intersections of a sphere a plane passes through its center [41]. Every longitude and RA correspond to a great circle. Small circles are created by any plane slicing the sphere that does not intersect the center of the sphere. The resulting small circles all

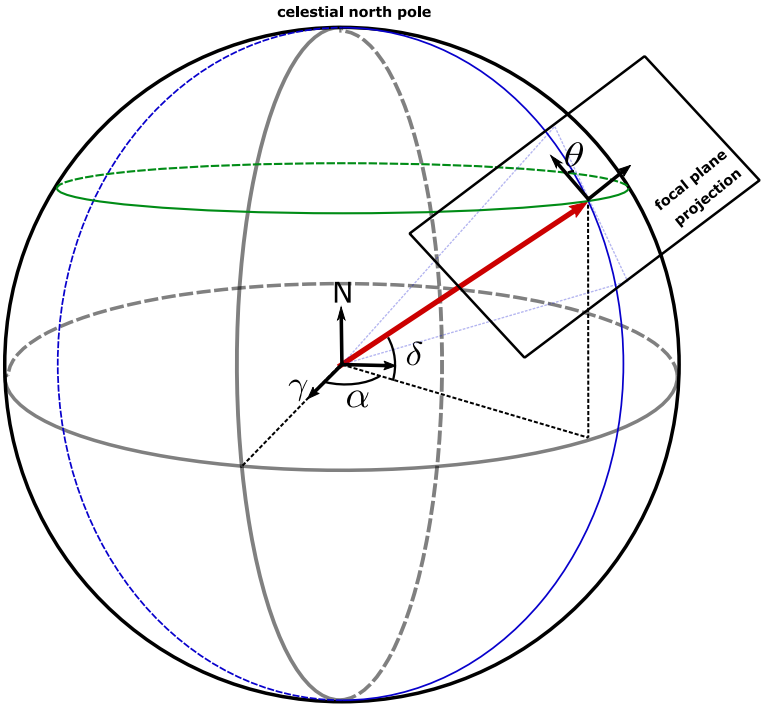


Figure 3.2: Projecting stars of the barycentric coordinate system onto the flat focal plane coordinate system with a gnomonic projection. The blue line, corresponding to a right ascension, is an example of a great circle and the green, corresponding to a declination, is a small circle. The flat projection of the focal plane can be assumed to be tangent to the sphere at the pointing (RA, DEC).

have smaller diameters than the great circles. The lines of latitude and DEC correspond to small circles created by planes parallel to the equatorial plane. When choosing a projection, these circles in spherical geometry are distorted in different ways. A focal plane projection can be viewed as the center pointing (RA, DEC) being tangent to the sphere at that one point as seen in Figure 3.2 and everywhere radiating from that point will be distorted in some way. A gnomonic projection keeps the great circles in spherical geometry as straight lines and the small circles are curved. When working with a small FOV, the divergence of the straight line great circles towards the barycentric equator and curvature from

the small circle projections is minimal.

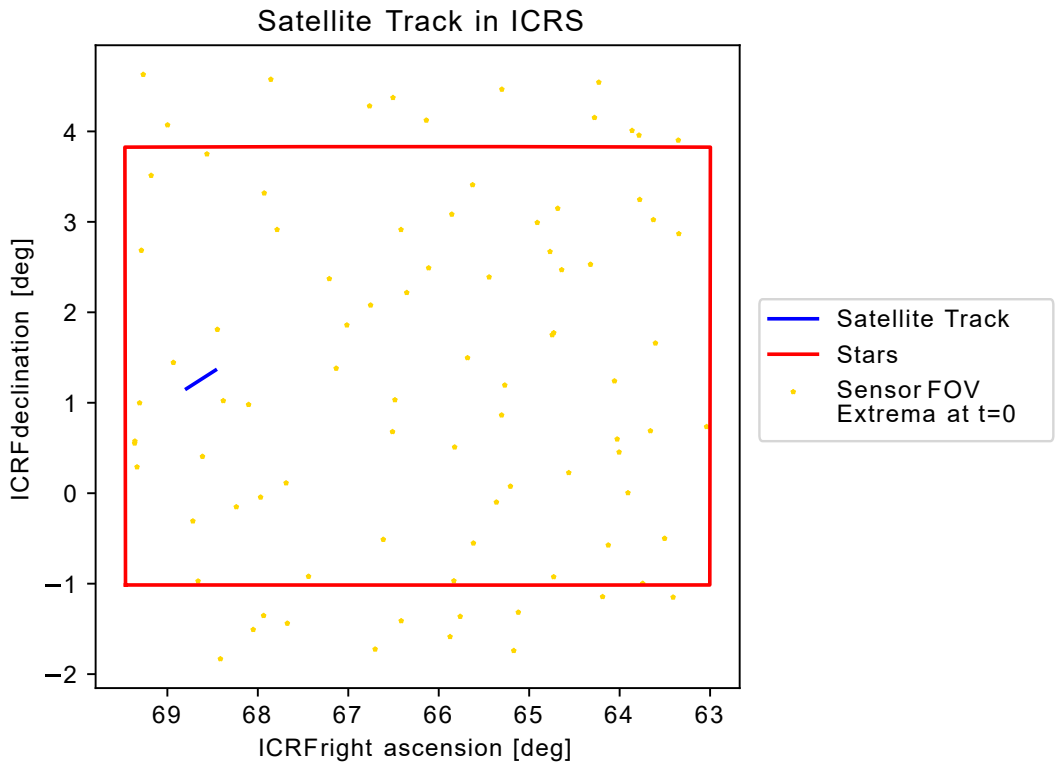


Figure 3.3: Output of satellite selection and propagation includes identification of which satellites and stars are in each simulated time step's FOV. For each of these relevant point sources, (x, y) location on the array via a gnomonic projection is recorded.

Now that I have all the stars to be simulated at each time step, I need to accomplish the same for satellites and determine where in the frame the satellites will be during the simulation. To solve for satellite position over a set time window, I first need relevant initial conditions before propagating a satellite forward in time using the two body problem (2BP) equations of motion. I use two line element sets (TLEs) as my initial conditions, ideally loading TLEs from Celestrak.org or Spacetrack.org within the day of the simulated pass [52, 88]. TLEs are a parameterization of the satellite state at a set epoch. This state can be propagated with mean motion or the 2BP equations on a conversion of the

parameterization to a position and velocity vector. The classical 2BP relates the current state with a state propagated in time as

$$\ddot{\vec{r}} = \frac{-G(m_1 + m_2}{r^3}\vec{r} \approx \frac{-\mu}{r^3}\vec{r} \tag{3.2}$$

where the satellite object's acceleration, $\ddot{\vec{r}}$, is dictated by the force of gravity between the two objects with masses m_1 and m_2, the distance r between the two objects, and G the gravitational constant. Since the mass of the smaller satellite is often much less than the mass of the object being orbited, as with the case of a satellite around Earth, the equation simplify as $G(m_1 + m_2) \approx Gm_1 = \mu$ where μ is the Earth's gravitational parameter. This simplified model is improved with the inclusion of perturbations that take into account the imperfections and other forces not accounted for with the strictly gravitational model. The simplified general perturbations propagator (SGP4) informs the satellite position while including the major secular terms of drag [47]. TLEs should be as close as possible to the simulation time chosen because the error accumulation in this propagation method can be 1-3 km per day [99]. Calling SPG4 for each time step simulated and converting the output to (RA, DEC) produces the satellite track. Figure 3.3 depicts the bounded projection onto the focal plane. The full satellite track is displayed with only one time step for the stars and FOV.

3.2 Radiometry

Once I determine all relevant objects in each time step, these objects are simulated as point sources and propagated through an atmospheric simulation to determine the incident flux on the focal plane area in units of photons per second. I start with the definition of point sources, describe how I propagate light

through the atmosphere, and articulate the method to model the optical system in order to produce accurate spread of light on the focal plane. Finally, I also describe the method to attribute point sources on each simulated frame. That way, each generated event in the simulation is attributed to a source or noise, making assessment of the events produced and algorithm development from the model easier.

3.2.1 Source Definitions

In order to have an accurate estimate of the photon flux on the focal array, I first need to determine the ideal photons received by the sensor from each source, its irradiance. The Prophesee Gen3 sensor I model in this dissertation is sensitive to the visible spectrum through the near infrared (Table 3.1). The data collection procedure discussed in Section 3.4.1 does not apply additional filtering to the incident energy. Therefore, I define all sources over this bandwidth. Stars and satellites are handled separately as described in this Section due to the availability of magnitude data.

Starting with stars, to determine the appropriate radiance for the stars, I query the Gaia G broad passband catalogue for the visual magnitude, m_v. The broadband Gaia m_v is appropriate for a sensor sensitive over the visible spectrum. Therefore this catalog is an effective measurement to estimate the energy sent by each star that a standard EVS can discern. With the m_v of a star, I compute each star's flux

$$F = 3016 * 10^{-0.4 * m_v} \text{ [Jy]} \tag{3.3}$$

using the Vega zero point magnitude of 3.016e3 Janskys [44, 51]. Subsequently

I convert the resulting flux value to photons per meter squared per second per Angstrom and multiply by the bandwidth to determine the radiance of the point source.

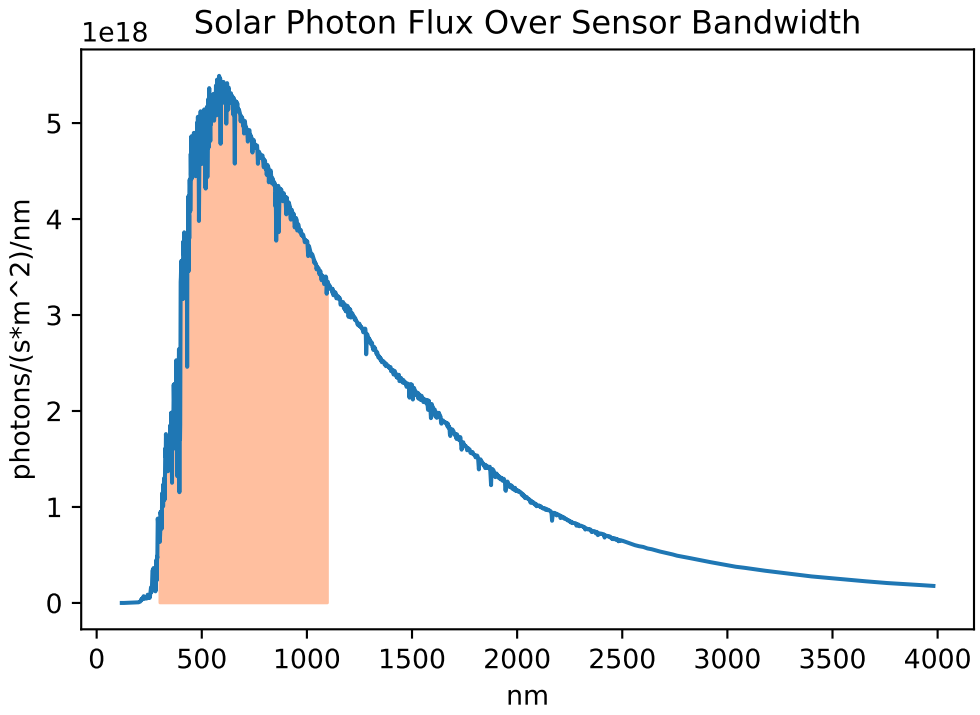


Figure 3.4: I integrate the solar photon flux over the Prophesee Gen3 bandwidth, indicated by the orange shaded region, to provide a better estimate of the photon flux detectable by the receiving EVS.

Simulating satellite radiance is more complex. Without more specific data about attitude pointing, materials and geometry, it is futile to reach a higher accuracy from a thorough ray tracing technique. Therefore, I simulate satellites with the goal of reaching the correct order of magnitude of photons per meter squared per second. To achieve this goal, I first estimate the projected area and materials with publicly available information. Then I calculate the reflected energy, Φ,

$$\Phi = \frac{L \sum A_i \rho_i}{2\pi}, \tag{3.4}$$

as a summation of the solar flux, L, multiplied by the area of each component, A_i and its reflectivity, ρ_i. I also assume the reflection is diffuse, due to lack of pointing information, making the energy sent per subtended angle the total reflected energy divided by 2π. I refine the solar photon flux over the bandwidth from a constant parameter by applying the EVS bandwidth to the solar flux as a function of wavelength as depicted in Figure 3.4. Integrating over the bandpass, the resultant flux is approximately 3.268e18 photons per meter squared per second. After calculating the radiance of the satellite, I determine the irradiance flux,

$$F = \frac{\Phi \cos\theta}{R^2}, \tag{3.5}$$

as a function of the distance of the satellite to the observing location, R, through the inverse squared law and off axis location of the point source through the angular difference, θ [28].

3.2.2 Light Propagation

These calculations of the power received by the sensor are idealized models. Observers in orbit without propagation through the atmosphere could use these numbers of flux received and multiply by the area of the optic aperture to estimate the total energy received. However, the data for verification in this dissertation is collected at a ground-based observation site and many users hoping to leverage EVS for inexpensive, autonomous SDA will take advantage of the less expensive ground-based observing options. The atmosphere distorts the wavefront, yielding imperfect spread of light on the focal plane. Therefore, to simulate these types of observations and accurately capture the atmospheric distortion, I must manipulate the uniform radiance from the point source ob-

jects before the numerical representation of the wavefront reaches the aperture of the optical system.

I model the distortion due to atmospheric propagation through the common phase screen, split-step propagation method with Zernike mode-constructed phase screens [89, 84, 80, 43]. Figure 3.5 lays out the general process of a single propagation. First, I define a centralized point source on the source plane, $U(r_1)$, in local dimensions. I draw a line between the off-axis location of the point source through the phase screens to determine what subsections of the screens will be sampled for the current point source. To propagate the source field to the observation plane, I divide the distance into distinct steps. Each of these steps alternates between free space diffraction regions of Δz and then refraction of the light through multiplication with a phase screen, τ. The final observation field, $U(r_n)$ typically covers some area larger than the observing aperture and is sampled for the final electrical field entering the optical aperture.

I start this overall process with construction of the phase screens. The number of phase screens I construct is dependent on the simulation parameters. In most cases I assume the simulation is sufficiently modeled with an assumption of frozen flow. Frozen flow is a combination of two assumptions: distinct layers within the atmosphere concentrate refractive index variations that are stable through time and these layers translate relative to an observing optic over time [75]. For most SDA operations the time scale of the satellite passing through the image frame is short relative to the time scale of the variation in atmosphere turbulence. As an example of what time scales are relevant in SDA data, the verification data sets I use in this dissertation are only around 20 seconds long for a staring sensor to collect sufficient information about a satellite pass. Due to the

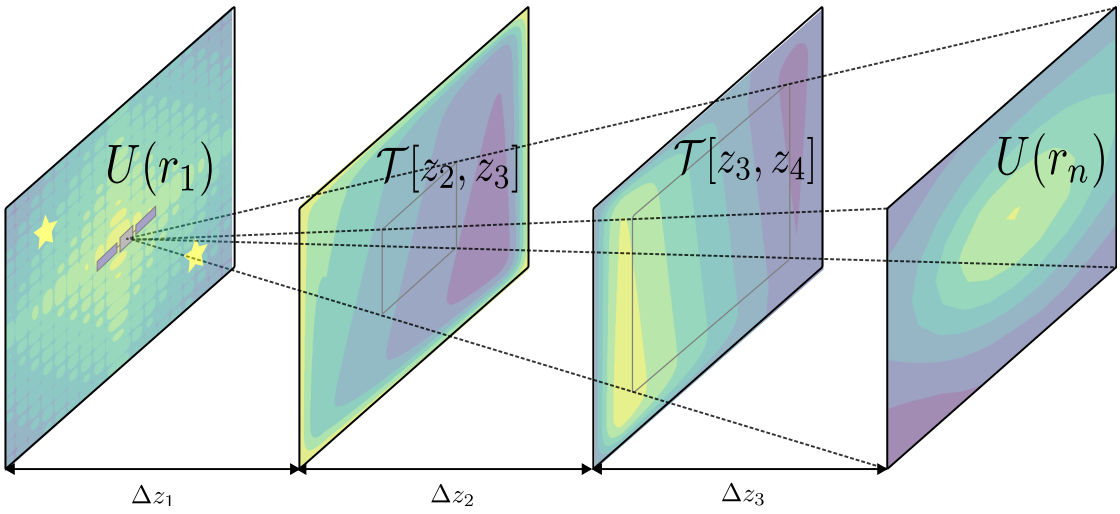


Figure 3.5: The split-step method of atmospheric propagation splits the transmission through the atmosphere into multiple steps. On the source plane a centralized point source is defined. To propagate that source the propagation distance I divide the distance into areas of free space diffraction and planes where I apply refraction through phase screens representing turbulence. The local phase screens are sampled from larger phase screen at off-axis locations drawing a line from the point source on the source plane to the center of the observation plane. Sampling from a larger phase screen ensures a continuous function applied to each source traversing different paths through the screens.

short time scale, individual layers of atmospheric phase modulation remain internally stagnant, meeting the first assumption of frozen flow. Treatment of the second assumption depends on the modality of the operations simulated. For example, if the observer slews to keep an object centered in the FOV, the frozen layer should shift appropriately to capture the translation of the observer and the frozen layer's translation with respect to the observing site over time. The translation of the frozen layer also applies for longer staring cases for more accurate distortion results. If the dynamics of the drifting turbulence is known then the phase screen is applicable for longer periods of time with the caveat that the phase screen is finite and must be extended for longer periods of translation. In the case of the data sets examined in this work, the short collection time works in my favor again. I apply a basic implementation of frozen flow.

I use a single set of phase screens and neglect translation because the observer and frozen layer should experience minimal relative translation within the 20 second simulations. I will implement the frozen layer translation in the future to increase simulation accuracy and allow for more sensing modalities.

Before I generate the phase screens, I first perform a dimensional analysis to pick grid spacings that prevent aliasing in the frequency domain as outlined in [89]. I determine the overall diameter of the phase screen by the camera's field of view at the distance of each screen. The screens are spaced evenly between the source and final plane. Once the grid spacing is determined, the size of the grid is scaled to reach the required diameter. This typically yields untenable grids for a typical high-end computer's system memory (64GiB). Therefore, I downsample each phase screen until it is of a reasonable size for the system's memory limitation. When the screen is sub-sampled for a particular point source's propagation, I interpolate the downsampled screen to create the desired grid size while maintaining a smooth function for the phase screen.

Now that I know how many screens to generate and the grid spacing and size of each phase screen, I create a relevant set of phase screens using Kolmogorov's statistical theory of turbulence. Following this theory, I construct each phase screen with Zernike polynomials. Zernike polynomials are an orthogonal basis defined on the unit circle. Zernike basis modes are a classic approach to capture the aberrations of optical systems with modes such as defocus and astigmatism. Randomly sampled coefficients of a set of Zernike modes can, therefore, capture atmospheric turbulence [84, 11]. The definition of the modes, as outlined in [68], organizes the orthogonal modes by radial degree, n, and azimuthal frequency, m. Those with higher frequencies have even and odd modes

defined with cosine and sine functions respectively as

$$\begin{aligned} Z_{even\,j} &= \sqrt{n+1}R_n^m(r)\sqrt{2}\cos(m\theta) \\ Z_{odd\,j} &= \sqrt{n+1}R_n^m(r)\sqrt{2}\sin(m\theta) \\ Z_j &= \sqrt{n+1}R_n^0(r), m = 0 \end{aligned} \tag{3.6}$$

where R_n^m is determined by

$$R_n^m(r) = \sum_{s=0}^{(n-m)/2} \frac{(-1)^s(n-s)!}{s![(n+m)/2-s]![(n-m)/2-s]!} r^{n-2s}. \tag{3.7}$$

These modes are functions of the polar coordinates, radius, r, and phase, θ, of the propagation plane and, therefore, must be computed for each cell for each phase screen generated unless the spacing is constant [84].

Each Zernike mode, Z_j describes a transformation to the phase component of the electric field. Combined they describe a full phase screen [68]. The combination with the Zernike modes comprises a summation over each mode with stochastically determined coefficients [84]. Therefore, to capture the atmospheric statistics with the Zernike modes and create a phase screen, $\Theta_{atm}(r,\theta)$, I evaluate the matrix summation

$$\Theta_{atm}(r,\theta) = \sum_j a_j Z_j(r,\theta). \tag{3.8}$$

The independent Karhunen-Loéve coefficients, a_j, are the weighting factors and scale the phase screen to appropriately represent the atmospheric turbulence. a_j is a multiplication between the decomposed covariance between each Zernike mode and a zero-mean unit-variance random vector [80]. I calculate the coefficients by computing the covariance of pairs of Zernike polynomials, Z_j and $Z_{j'}$, excluding the first mode, as

$$C_{j,j'} = \frac{K_{zz}\delta_z\Gamma\left[\frac{\left(n+n'-\frac{5}{3}\right)}{2}\right]\left(\frac{D}{r_o}\right)^{\frac{5}{3}}}{\Gamma\left[\frac{\left(-n+n'+\frac{17}{3}\right)}{2}\right]\Gamma\left[\frac{\left(n-n'+\frac{17}{3}\right)}{2}\right]\Gamma\left[\frac{\left(n+n'-\frac{23}{3}\right)}{2}\right]'} \tag{3.9}$$

where D is the aperture diameter of the observer, r_0 is Fried's parameter, and Γ represents the Gamma function. Fried's parameter ties the statistics of the Karhunen-Loéve coefficients to the severity of the atmospheric turbulence [31]. There are two more terms in the covariance matrix equation. First,

$$K_{zz'} = \frac{\Gamma\left(\frac{14}{3}\right)\left[\left(\frac{24}{5}\right)\Gamma\left(\frac{6}{5}\right)\right]^{\frac{5}{6}}\left[\Gamma\left(\frac{11}{6}\right)\right]^2}{2\pi^2} \times (-1)^{\frac{(n+n'-2m)}{2}} \sqrt{(n+1)(n'+1)}. \tag{3.10}$$

captures the frequency characteristics between the modes. Second, δ_z is the Kronecker symbol,

$$\delta_z = (m = m') \wedge (\overline{parity(j, j')} \vee (m = 0)), \tag{3.11}$$

which makes the covariance 0 for all terms that either do not sharing the same azimuthal frequency or are mismatched on their sine or cosine definitions. This logic ensures only the covariance of the interacting Zernike modes carries through to the final a_j weights.

After defining the covariance matrix and reordering the polynomials to ensure a block diagonal construction, the covariance matrix is now Hermitian and, therefore, there exists a unitary matrix, U, such that $U \cdot C \cdot U^T$ is diagonal. I now define the Karhunen-Loéve coefficients as a product of the unitary matrix, gained through a decomposition of the covariance matrix, with a column vector of Gaussian random variables, $\overrightarrow{n}$ with zero mean in

$$a_j = U^T \cdot \overrightarrow{n}\,. \tag{3.12}$$

For the decomposition of the covariance matrix, I apply the Cholesky decomposition method.

Five Zernike modes are depicted without their orthonormal complement along with a resulting phase screen in Figure 3.6. Higher-order phase screens

provide more complex variance in phase that captures beam intensity modification, but the predominant modes that capture distortion on the focal plane are tip and tilt, modes 2 and 3 respectively [10]. Since it is sufficient, I use a low-order model to generate phase screens for this simulation to conserve memory resources.

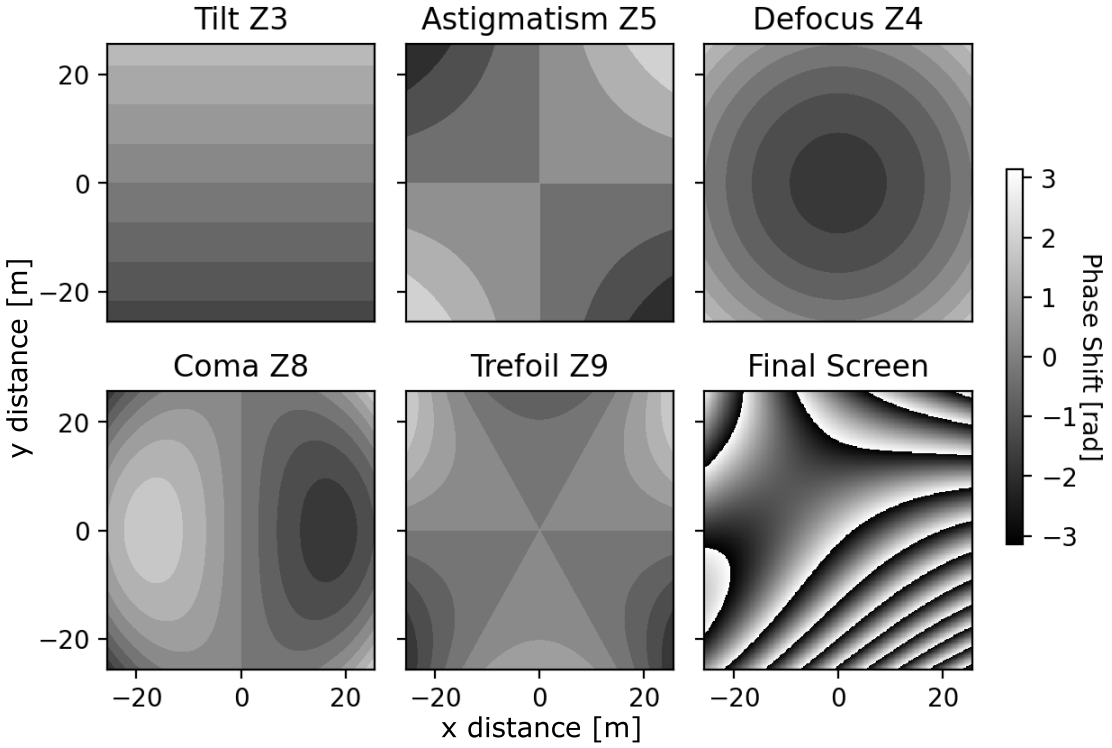


Figure 3.6: From the upper left to bottom right the Zernike phase distributions of increasing modes are shown. The Noll number associated with the mode is listed in each graph's title. Complementary modes with the opposite sign are not plotted. The resulting phase screen, on the lower right, is a combination of these modes and their complements scaled by the Karhunen-Loéve coefficients that capture the statistics of atmospheric turbulence.

Now that I have a set of phase screens to propagate through, I propagate the satellite and star sources of each simulated time step through the screens. I start by defining each source as a centralized sinc function as depicted in the leftmost

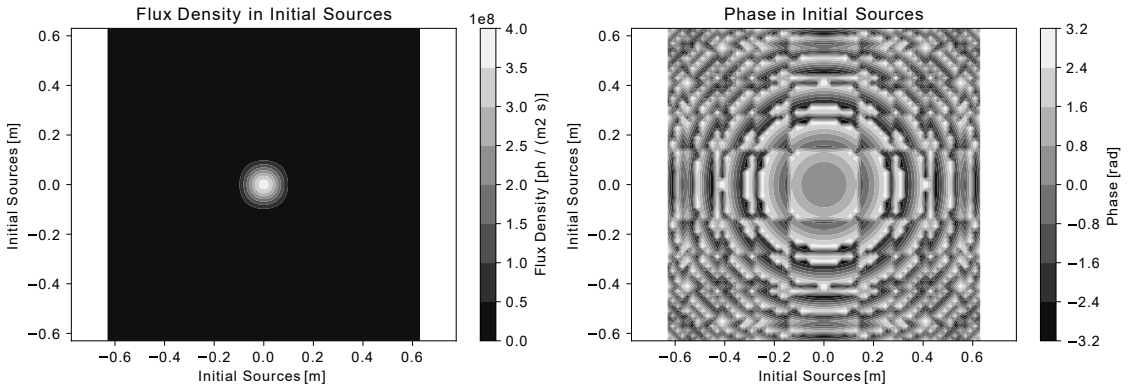


Figure 3.7: Initial plane point source photon flux and phase angles prior to propagation. The peak intensity is centrally located and falls off towards higher radii as expected from a sinc function definition.

pane of Figure 3.5. The centralized sinc function

$$U_{pt}(r) = Ae^{-i\frac{k}{2D_z}r_1{}^2}e^{-\left(\frac{Dr_1}{4\lambda D_z}\right)^2} \times \left(\frac{D}{\lambda D_z}\right)^2 \text{sinc}\left[\frac{Dx_1}{\lambda D_z}\right]\text{sinc}\left[\frac{Dy_1}{\lambda D_z}\right], \tag{3.13}$$

ensures the observation plane field is flat when no phase distortion is applied. In the centralized sinc function, A scales the amplitude of the function. The definition, $A = \Phi\lambda D_z$, sets the final field amplitude to be Φ, the power, sent to the full observation grid area. Polar coordinates of the local source plane capture the phase, where r is the radius of each grid space from the center of the grid. The sinc functions capture the drop off in the Cartesian coordinates x_1 and y_1. The other parameters in the sinc point source equation are constants which include the wavelength, λ, wave-number, k, propagation distance through the atmosphere, Δz, and the diameter of the observation plane, D. Since I define each point source locally on center, the resultant initial point source fields only vary by magnitude and have the appearance of the point source magnitude plotted in Figure 3.7. This uniformity is to be expected because the atmospheric distortion applied by the phase-screens produces the variation in the final fields.

After I define a point source, I simulate the propagation with alternating steps of free-space diffraction and refraction of light with the atmosphere

through the multiplication of Zernike phase screens. To propagate the energy on the satellite source plane to the optical frame, a multi-layer angular spectrum propagation is utilized. The multiple layers allow use of Fresnel diffraction and the inclusion of a multilayered phase-screen to model atmospheric turbulence. The split-step beam propagation method with n propagation steps has $n-1$ phase screens incorporated into the propagation by a Fourier product

$$\begin{aligned} U(r_n) =& Q\left[\frac{m_{n-1}-1}{m_{n-1}\Delta z_{n-1}}, r_n\right] \\ &\times \prod_{i=1}^{n-1}\left\{\mathcal{T}[z_i, z_{i+1}]\mathcal{F}^{-1}\left[f_i, \frac{r_{i+1}}{m_i}\right] Q_2\left[-\frac{\Delta z_i}{m_i}, f_i\right]\mathcal{F}[r_i, f_i]\frac{1}{m_i}\right\} \\ &\times \left\{Q\left[\frac{1-m_1}{\Delta z_1}\right]\mathcal{T}[z_1, z_2]U(r_1)\right\}. \end{aligned} \tag{3.14}$$

I find the observation plane field through this equation by multiplying the source field by a complex phase screen, $\mathcal{T}$, and a quadratic phase factor, Q. For each layer of propagation the Fourier transform is taken, I apply another quadratic phase factor to focus the spherical wave onto the current plane. Then I take the inverse Fourier transform and, finally, multiply the phase modification from the turbulence elementwise onto the output. As aforementioned, the phase screens for each point source are subsections of a larger Zernike phase screen that ensures the phase manipulation of the overall electric field is a smooth function. At the observation field, I apply a final quadratic factor to collimate the field which removes the spherical-wave phase [89, 11]. Figure 3.8 depicts a final resulting field. The aperture only covers a portion of the modeled field diameter, so a mask is overlaid the subsection of the field.

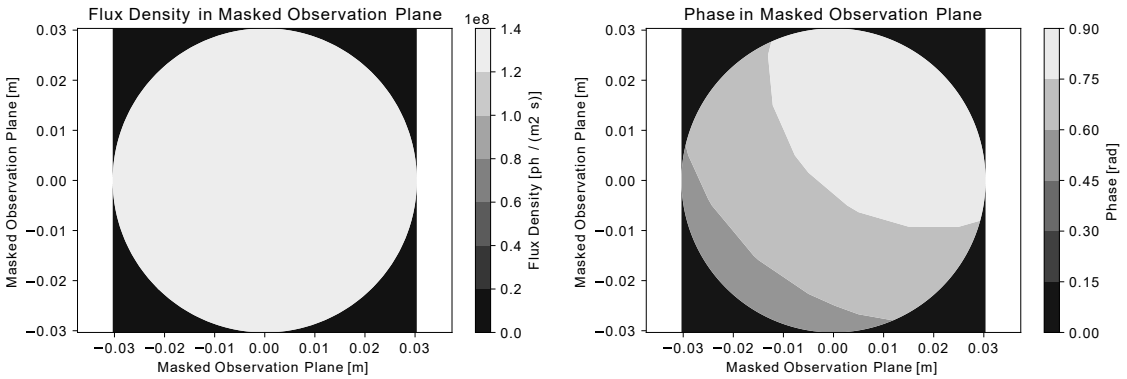


Figure 3.8: Final plane after propagation of one point source. The total flux entering the aperture is depicted by the marked aperture area. While the photon flux looks relatively uniform, the phase is not which indicates a successful manipulation of the phase.

3.2.3 Optical Train

I now have the field entering the aperture, but this relatively flat field is not what the focal plane images. The energy entering the optical system is focused onto the focal plane. To simulate the final image from the incoming field, I employ a semi-analytical Fourier transform of the masked final observational field [96]. This technique addresses the issue with small occulting masks, where the Shannon-Nyquist sampling of the focal plane $\frac{\lambda}{D}$ with D is the aperture diameter, will yield an insufficient sampling number in the frequency domain and, therefore, sample the frequency domain insufficiently [38]. Typically this leads to zero padding of 6-to-8 fold around a masked field before the optical propagation. Instead, starting with an analytical expression of the field in the focal plane,

$$U(r_f) = \mathcal{F}[U(r_n)](1 - \epsilon M(r_n)) \tag{3.15}$$

where the final field, $U(r_f)$, is the Fourier transform of the field entering the aperture, but truncated by the aperture mask, $M(r_n)$, with transmission of ϵ in the $M(r_n)$ regions. Therefore, I only need to capture the Fourier transform inside

of limited areas and circumnavigate the sampling problem by calculating a two limited area Fourier transform in the matrix form,

$$U(r_f) = \frac{m}{N_1 N_2} e^{-2i\pi \boldsymbol{X_2} \boldsymbol{X_1}^T} U(r_n) e^{-2i\pi \boldsymbol{Y_1} \boldsymbol{Y_2}^T}. \tag{3.16}$$

In this formulation, $\boldsymbol{X_1}$, $\boldsymbol{Y_1}$, $\boldsymbol{X_2}$, $\boldsymbol{Y_2}$, are the frequency sampling vectors, based on the grid sampling at the observing plane and the focal plane, respectively. $\boldsymbol{X_2}$ and $\boldsymbol{Y_2}$ are defined as the image size, m, divided by the number of points in the impulse response. $\frac{m}{N_1 N_2}$ is a normalization coefficient where N_1 and N_2 are the number of grid points at the observation plane and image plane. I define m in $\frac{\lambda}{D}$ resolution units. I dictate m by choosing the number of full size pixels I want inside the image, N_{pix}, produced by the semi-analytical Fourier Transform. Using the spatial resolution, the full diameter of the airy disk,

$$SR = \frac{2.44 t_{focus} \lambda}{D} \tag{3.17}$$

where t_{focus} is the effective focal length of the optical train, I derive

$$m = \frac{N_{pix} \mu_{pix}}{SR} \tag{3.18}$$

as a function of the spatial resolution, the number of full pixels to simulate, and the pixel pitch of each pixel, μ_{pix}. I also choose the number of sub-pixel samples, μ_{subpix}, which subsequently defines the grids X_2 and Y_2 as $\frac{m}{\mu_{subpix}}$. The final image is a distorted impulse response in resolution units with a direct relationship to the image plane spatial units because I choose how many full pixels to simulate. An example of the resulting response in Figure 3.9 demonstrates the scale of the resulting coordinates.

In order to depict the final image plane, I apply to more transformations to the impulse response. First, since the resolution of the image plane output from the semi-analytical transform is currently sub-pixel, I scale the response

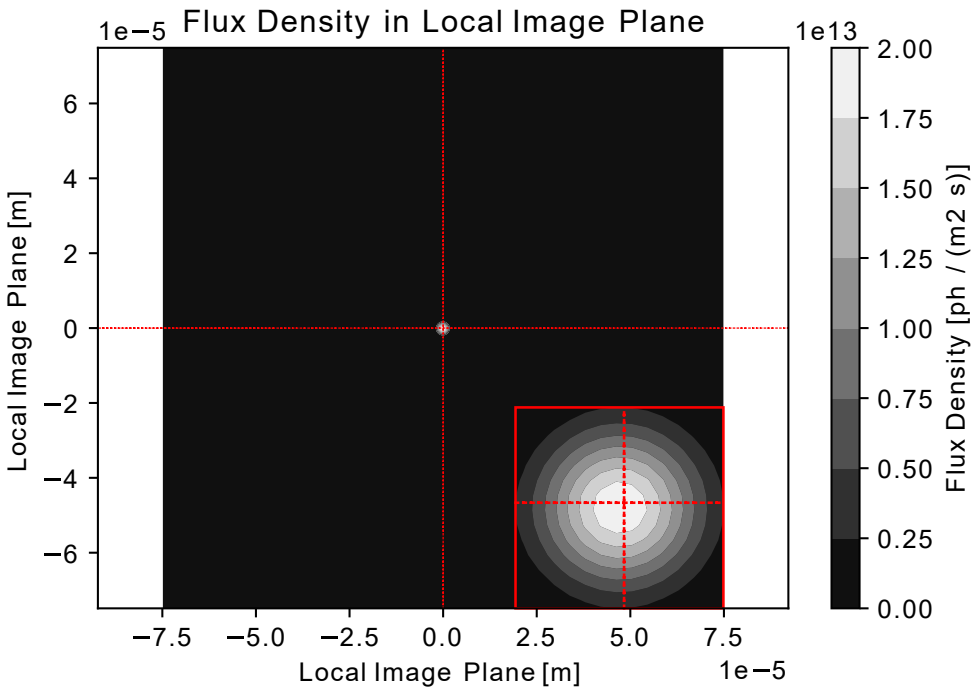


Figure 3.9: Resultant impulse response from the semi-analytical transform. An expansion of the image demonstrates the slight motion off axis for this simulation's Fried parameter.

by $\frac{1}{\mu_{subpix}}$. I translate the center of the resulting image plane to the focal plane coordinates (x, y) that correspond to the (RA, DEC) of the current point source. If a point source experiences tip or tilt during propagation, the resulting impulse response image on the image plane will be off-axis of the center pixel. Therefore, the scaled and shifted point source will also be off axis of the perfect (x, y) and accurately captures the atmospheric distortion. Once shifted to the correct location on the image plane, I calculate the overall field magnitude as a summation of the individual complex fields. Figure 3.10 depicts final image intensity which is the multiplication of the field by its conjugate.

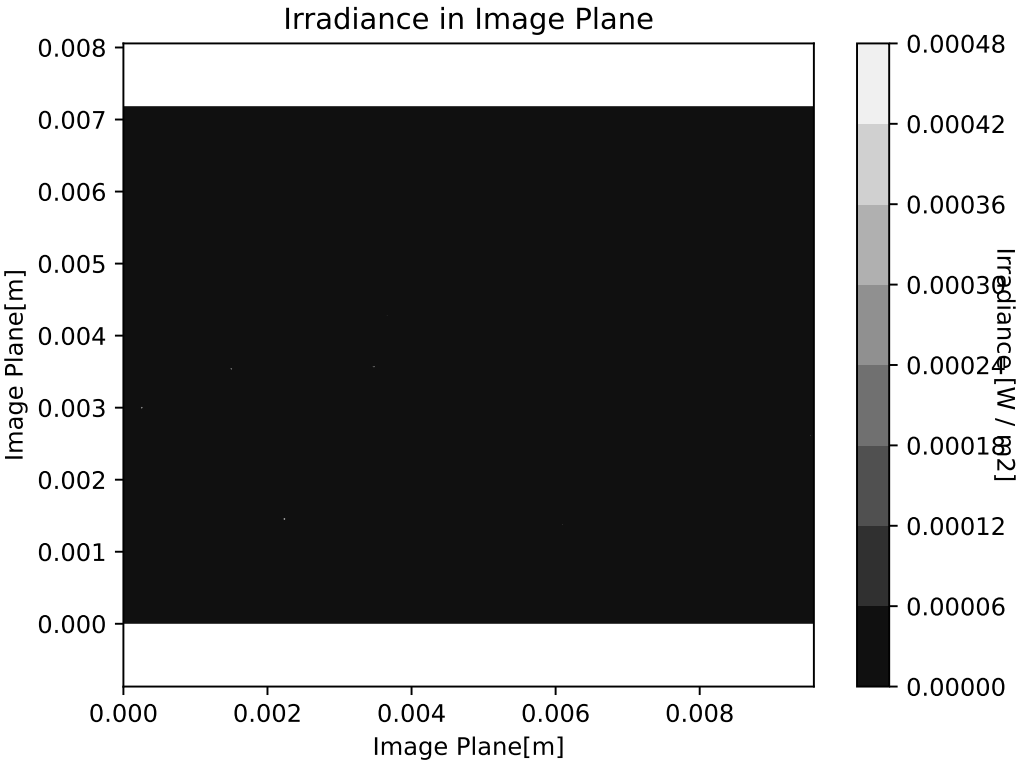


Figure 3.10: The final image field at the correct scale. Multiple sources are visible, but since the sources can be orders of magnitude different in photon flux, not all the sources can be seen in this depiction.

3.3 Circuitry

Up to this point, the simulation methodology is conventional. This Section covers the methodology to simulate EVS with specific considerations to leverage the photon flux level model I create with the first half of the simulation. The general processing flow I apply follows the layout of this Section and varies slightly from the methodology applied in the v2e flow shown in Figure 2.1. The general method, however, is inspired by the v2e processing steps with a goal to increase fidelity. The first step in both v2e and in my circuitry simulation is determination of the induced photocurrent. I calculate it directly from

the photon flux. This includes an estimation of the quantum efficiency of the simulated camera. Moving on from there, there are slight modifications to the conversion of logarithmic current, low pass filter, and comparator portions of the simulation to yield more accurate results for low-light sensing and improve the temporal fidelity of the event output. In order to capture the unique signatures of events, the arbitration process is highly modified from other simulation techniques. Finally, I address the application of noise. This is the only section slightly out of order. I apply noise after the calculation of induced photocurrent. Noise production is the last section because I develop a new way to model noise for EVS to try and capture the noise production given small simulation time steps overcoming the low pass filter frequency. In that section, I propose methods to capture both dark shot noise and appropriate noise levels on induced photocurrent.

3.3.1 Induced Photocurrent

The first step in the circuitry process is the conversion of the incident flux to current by the photodiode. From the very first step, I'm varying from the v2e process which estimates induced photocurrent in a video from the digital number output on each pixel. Since I have the radiometric information, I bypass this estimation. The output of the first half of the simulation output is in photon flux density in photons per second per meter squared, which is easily converted to photon flux per pixel by multiplying by the pixel area. In order to determine the generated current from the incident flux on the circuit's photodiode, I_{pd}, I use the quantum efficiency, QE, in conjunction with elementary charge, e, through

the directly proportional relationship

$$I_{pd} = QE * e * FLUX \tag{3.19}$$

to calculate the induced amperage on the circuit. This equation requires an approximation of the quantum efficiency. The Prophesee provided measurements of quantum efficiency are fairly coarse, with only 5 measurements across the visible spectrum, summarized in Table 3.1 [79]. Given the validation data is taken with the 3rd generation EVS, I conduct subsequent work referencing the values in the first row of the Table. I approximate the quantum efficiency for this sensor using these measurements, by integrating over the wavelengths reported and dividing by the bandpass.

Table 3.1: Prophesee Reported quantum efficiency values for their EVS available at the time of writing.

LED Wavelength	455nm	505nm	625nm	850nm	940nm
Gen31	.37	.39	.37	.15	.05
Gen40	.42	.52	.52	.23	.11
Gen41	.60	.78	.69	.28	.13

Given the coarseness of the measurements, I examine both an interpolation and curve fit between these points in order to integrate and estimate the overall quantum efficiency. As discussed previously in section 3.2.1, the optical train I simulate provides no additional filtering and responds over the visible to near infrared wavelengths which prompted use of the broadband Gaia catalog for the simulated point sources. Therefore, I add additional points at zero efficiency at 350nm and 980nm before fitting the quantum efficiency curves. Figure 3.11 shows the Gaia bandpass with the interpolated and curve fit QE. I multiply these two filters together to obtain a combined attenuation for the star sources. The Figure contains the interpolated estimation and the fourth order curve fit estimation. I also apply a second, third, and logarithmic fit with the results

summarized in Table 3.2. Ultimately, the fourth order fits to the reported points and does not overestimate the lower end of the scale, so I use that set of efficiencies in the final simulations. Note that the satellite sources are not defined in the same way as stars and are only multiplied by the quantum efficiency alone. Since I apply the quantum efficiency to all pixels in a frame with no discrimination of source, I use the average QE when applying Equation 3.19. I apply the additional scaling value to star sources during the radiometric portion of the simulation to ensure I apply the total attenuation to all star objects. While the average Gaia passband attenuation is 0.3936, the applied value in the radiometry simulation depends on the wavelengths of overlap and combined attenuation. Therefore, I calculate an appropriate attenuation to apply to stars during the radiometry portion of the simulation to get the final desired average attenuation. To do this I divide the final attenuation by the average quantum efficiency.

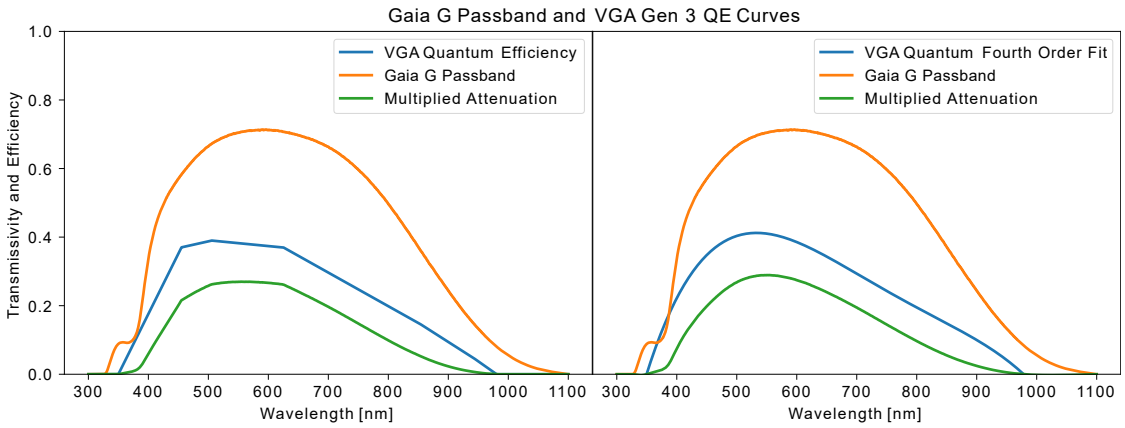


Figure 3.11: Quantum efficiency estimation of the 3rd generation Prophesee sensor requires either curve fitting and interpolating between the coarse reported values. The Gaia stars are also attenuated by the Gaia broad passband. An estimation of their overall attenuation is made in green.

Table 3.2: Estimating the quantum efficiency of the 3rd generation Prophesee EVS with multiple fits.

Fit	Average QE	Average Star Combined Attenuation	Gia Bandpass Attenuation	Star Attenuation in Radiometry
2nd Order Fit	0.2725	0.1296	0.3936	0.4757
3rd Order Fit	0.2165	0.1279	0.3936	0.5907
4th Order Fit	0.2508	0.1243	0.3936	0.4958
Logarithmic Fit	0.2757	0.1291	0.3936	0.4681
Interpolated	0.2422	0.1444	0.3936	0.5962

3.3.2 Logarithmic Conversion

The next step in the circuitry processing is the conversion to logarithmic current. The comparator circuit compares the logarithmic current values which makes the set threshold value akin to a standard percent change required to yield an event. Therefore, as the scene gets brighter, a larger change is required to produce the next ON event. To capture this phenomenon, I simply convert the induced current values from the previous step to their logarithmic equivalent after adding in the nominal dark current value. Surprisingly, this is not the methodology in the v2e simulation inspiring my processing pipeline. One of the perplexing things about v2e is their piecewise treatment of the induced photocurrent current a low levels of illumination [21]. Their conversion

$$I_{plog} = \begin{cases} I_p \frac{\ln(5)}{5} & I_p < \ln(5) \\ \ln(I_p) & I_p \geq \ln(5) \end{cases} \tag{3.20}$$

avoids the issue of the digital number estimation yielding a current value of 0, by converting into a linear conversion close to the origin. They start the linear region a few numbers out to create a smooth transition between the logarithm and the linear region. However, this is a particularly poor assumption for actual current values. If this conversion is linear, then the required change in photocurrent to yield an event becomes the threshold value. When working with induced currents on the order of femtoamperes in these low-light sensing conditions, a

real value change of 0.1 log scaled current will not happen. Instead, I avoid the issue of taking a logarithm of a 0 value by including the dark current. With a dark current value added to the induced photocurrent value, the overall current value will never be zero, the logarithm will always be valid, and the comparator process will not be undermined.

3.3.3 Low Pass Filter

The final step before the comparator is the application of a low-pass filter. Event-based pixels, being analog hardware, have finite analog bandwidth limiting the system's ability to respond to fast changes in current. This is particularly prominent in EVS hardware during low-light sensing conditions because the system's response time is proportional to the induced photocurrent [21]. Since low-light conditions are ubiquitous with SDA sensing, I apply a discrete low-pass filter with a scaled corner frequency into the model. The discrete version of Kirchoff's laws, the discrete difference equation, I implement

$$\begin{aligned} \epsilon &= \frac{\Delta t}{\frac{1}{2\pi f_{3db}} + \Delta t} \\ I_p &\longleftarrow (1-\epsilon) I_{p-1} + \epsilon I_{pd} \end{aligned} \tag{3.21}$$

is essentially a weighted average, through evaluation of ϵ, of the previous current on the pixel, I_{p-1}, and the new current induced by the photodiode, I_{pd}. As the corner frequency increases, ϵ approaches 1. If ϵ is 1 then the low-passed current of the pixel, I_p, is simply equal to the induced current. The cuttoff frequency, f_{3db}, changes with the strength of the current, so I cannot apply a constant low

pass frequency. Instead, I estimate the corner frequency,

$$f_{3dB} = \frac{(I_{dark} + I_{pd})}{I_{dark}} \times f_{3dBmin}$$
$$f_{3dB} \leq f_{3dBmax} \tag{3.22}$$

as a ratio between the combined dark and induced photocurrent and the dark current alone. The corner frequency has a maximum value, f_{3dBmax}, so I apply an inequality to ensure it is not surpassed. This is similar to the digital number computation in v2e which scales the corner frequency as a ratio of the digital number [21]. However, recent noise studies indicate a consistent relationship in the noise generation rate which shed light on the corner frequency relationship to induced photocurrent. These studies also help predict the minimum corner frequency for low-light sensing to be around 1-2.5 Hz [65]. EVS sensors are also capped on their frequency response and cannot follow infinitely fast changes in the photocurrent at high values of induced current. I assume that modern COTS EVS hardware has a maximum cutoff frequency at approximately 3 kHz [95].

3.3.4 Comparator

The fundamental construct of event-based circuitry is the current comparison to a previously memorized current value that creates a sensor that can handle high dynamic range and operate independently of all the other pixels in the array. An event triggering amounts to the ratio between the sensor current and the memorized current passing a threshold value. The threshold value, therefore, is a percentage change from the memorized current which allows for sensitivity over a wide dynamic range. Instead of evaluating the ratio, I follow v2e's lead

and I conduct the comparison in logarithmic current

$$\Delta I_p = \ln\left(\frac{I_{pd}}{I_{mem}}\right) = I_p - I_{memlog} \tag{3.23}$$

between the logarithmic current, I_p, and the logarithmic memorized current, I_{memlog}, because the logarithmic ratio of the current values is the difference between the those values [21]. When the logarithmic change between the two currents, ΔI_p, is greater than the ON threshold, θ_{ON}, or less than the OFF threshold, θ_{OFF}, an event triggers. v2e records an idealistic number of events within each frame's time step by assuming the number of thresholds passed in a time step will be the number of events recorded. v2e then evenly spaces the events through the time step. The v2e method does not take into account the circuit reset time, the refractory period, or the arbitration process which determines the time stamp of each event. Therefore, the timing of events output from the v2e process will likely not mimic real event frequencies making the synthetic data less useful for algorithm development.

Since I want to capture more accurate frequency information, I still conduct the logarithmic comparison, but estimate the time stamp of the first event on a pixel in the the current step. This process starts through interpolation between the current at the beginning of the time step, I_{p0}, and the current at the end of the time step, I_{p2},

$$\frac{\Delta I_p}{\Delta t} = \frac{I_{p2} - I_{p0}}{t_2 - t_0} \tag{3.24}$$

to determine a mean slope for each pixel over the time duration of t_0 to t_2. I can estimate any value of log current, I_{p1}, at time t_1 during the time step by

$$I_{p1} = I_{p0} + \frac{\Delta I_p}{\Delta t}(t_1 - t_0). \tag{3.25}$$

I also apply this linear interpolation technique to find the memorized current at t_1, $I_{memlog1}$. There is a drop in memorized current through a junction leak in EVS

which lowers the current from the start of the time step to the end. I discuss the memorized current leak more in Section 3.3.6, but it is included in this method since it directly influences the difference between the memorized current and the induced current over a time step.

Next, I take the difference between I_{p1} and $I_{memlog1}$. By setting that difference equal to the threshold value, I rearrange

$$t_1 = \begin{cases} t_{pos} = \frac{(\theta_{ON} - I_{p1} + I_{memlog1})\Delta t}{\Delta I_p - \Delta I_{memlog}} + t_0 & \Delta I_p > 0 \\ t_{neg} = \frac{(\theta_{OFF} + I_{p1} - I_{memlog1})\Delta t}{\Delta I_{memlog} - \Delta I_p} + t_0 & \Delta I_p < 0 \end{cases} \tag{3.26}$$

to solve for the time, t_1, an event will trigger on each pixel. This calculation is subtly different for the ON and OFF threshold, so the equations above are conditionally run depending on the trend in the current change. The final step in the comparator processes is keeping track of the next event trigger time of each pixel. I track an array of event times with the next anticipated event time for each pixel. If a pixel in this array has no event time, indicating no event waiting to record, and t_1 falls between t_0 and t_2, I add t_1 to the array.

While the threshold level is chosen by setting a bias level for the comparator circuit, the analog nature of the comparator induces variation between the threshold level for each pixel. The variation is dependent on the average bias level. Typical bias selections, requiring between 10% to 40% change to induce an event, lies between 2% to 2.5% of change with a mean of 2.1%. The variance increases as the percent change to induce an event increases [55]. To apply this variation in the log scale and account for the bias stochasticity, I apply a log normal distribution with the mean centered at the expected threshold change. While I leave selection of the variation up to the simulation user, all simulations in this thesis use a standard deviation of $\sqrt{2.1}$. I chose the mean for the lower

threshold levels because the anticipated threshold value matching the real data is suspected to be within the lower range as discussed in Section 4.1.1.

3.3.5 Arbitration

After an event occurs, the recording of the event is not instantaneous. The readout of the circuit, briefly mentioned in Sections 2.1, is a non-traditional readout to mimic the biological optical nerve and truly capture the asynchronous and independent operation of each neuron. Once an event triggers, the system arbitration begins. In an EVS system with a traditional address-event representation schema, the triggered pixel's row or column is first identified. Once selected, the other address, column or row respectively, is identified. The full address is delivered to the readout chip through time-division multiplexing, so that only one event is read out at a time. The final time series of events has a microsecond timestamp associated with each event. However, the final event readout can have multiple events on a single time step. The arbitration system and thus time-division multiplexing operates at a higher frequency than the microsecond readout. The published maximum readout rates of some EVS being in the GigaHertz range, despite the readout timestamps being limited to a MegaHertz [18]. Therefore, some EVS can read out approximately 1000 events in a given microsecond and will label them with the same timestamp. It should be noted that the 3rd Generation Prophesee sensor, providing the verification data in this analysis, has a maximum output of 50 MegaHertz or only 50 events per microsecond as verified by empirical analysis [9]. While it is not clear why the timestamps are limited to microsecond readout, there are presumably delays through the systematic arbitration process in the readout: time to read out other

pixels, time for the arbiters to iterate through pixels, and time to complete the time-division multiplexing itself. These delays, not being quantifiable or consistent in their biasing of events, reduce the resolution of the readout that make greater precision in the recording time not achievable. Therefore, recording at the microsecond level may be the most suitable.

Working in conjunction with the arbiters is the refractory circuit. When an event occurs, a transistor is triggered and applies current to a reset circuit for a set period of time, the refractory period. During this time the triggered pixel does not compare its current with the newly memorized value [21]. The refractory circuit signals the triggered event to the arbiters and prevents further comparison of events for a set refractory period. This period is purposefully longer than the readout time and can be changed by the user because it helps limit the frequency that a given pixel, neuron, can fire [58]. This helps prevent a monopoly of the readout bandwidth by one neuron or area of neurons. It is also a strategy to avoid pairs of noise events by reducing the time resolution [64]. For example, if the current changes enough to elicit an event during the refractory period, but returns to baseline before the period is over, the information about the current during that time frame is lost by under-sampling. This phenomenon of reducing time resolution also applies to true signals, however, and must be treated carefully or weaker signals may appear as noise. Since the refractory period is a user setting for COTS event-based sensors and clearly impacts both the frequency and resolution of the sensor, capturing the reset circuit's refractory period will assist with choosing an appropriate setting for SDA applications and produce realistic frequencies of events through its inclusion in the simulation.

The general flow of logic to mimic the arbitration process at a neuron level is as follows: an event is triggered, when the arbitration process reaches the pixel address the event is assigned a readout time, the pixel remains unavailable to additional events until both the event is read out and the refractory period is passed, and the memorized current is set to the instantaneous current after the pixel is available for another event. This flow of logic has not been captured before in synthetic event generation. V2e simplifies the process with larger time steps, identifying the maximum number of events by threshold change within a period and equally subdividing these events within the time step as discussed in Section 3.3.4 [21]. The simulation also does not apply a refractory period to reset the current and instead applies the same process on the next simulated time step as if the sensor has infinite bandwidth to create events. In this way, the event stream from v2e is optimistic on the event generation, producing events that can reconstruct the change in current over time.

One goal of this work is to generate a more realistic arbitration simulation, so that the timing data encoded in the event stream output, one advantage noted from the earliest of these sensors [58] and leveraged successfully in the data attribution portion of this dissertation, is generated realistically to provide a platform for possible algorithm development. Algorithm 1 captures the logic inside the current version of the arbitration code. In order to create a more realistic version of arbitration, I run the simulation inside the arbiter at the approximate frequency of the arbiter in order to appropriately assign realistic time stamps. Therefore, there are two time steps to consider inside of the Algorithm: the current simulation time step is the larger time step at the frequency I generate frames in the radiometric portion of the code and the current simulation time is the smaller time step iterating at the recording frequency. The first step in

Algorithm 1 generates a queue of events by flattening the event time array into a list of pixels with event triggered time stamps and an additional boolean to track if the event has already been recorded.

Algorithm 1 Arbiter Main

Generate Event Queue : Translate event time array into ordered list of events
Generate Refractory Period Time : Generate array of earliest reset refractory period times
Get Next Event : Determine the order to process events
while Current Simulation Time ≤ Current Simulation Time Step **and** An event is available to process **do**
  for Each Event to Process **do**
    Process Event : Add event to list of processed events
    Check if New Events Possible : Determine if any recorded pixels pass the refractory period
    if New Events Possible **then**
      Update Memorized Current : Set memorized current value I_{memlog} to I_p at reset time
      Check for New Events : Rerun comparator code on pixels where new events are possible
      if New Events **then**
        Add new events to the event queue
      end if
    end if
    Increase current simulation time by one recording frequency time step
  end for
  Get Next Event : Determine the order to process events
end while

The next step in Algorithm 1 generates an array with the earliest time a pixel can reset and start comparing current again. In Algorithm 2, I create the refractory period time array through a mask on pixels without timestamps and adding the refractory period time to the non-zero values. Users can adjust the refractory period with a bias on the camera. However, the refractory period experiences pixel mismatch, similar to the threshold values. Instead of assuming the hardware miraculously all reacts the same way to that setting bias, I initialize the refractory period with a Gaussian to capture the stochastic differences

between the hardware in each pixel.

Algorithm 2 Generate Refractory Period Time

```
if Event time array has non-zero values then
    Add refractory period time to non-zero values in array
end if
```

After determining the time each pixel might reset, I determine the order of events to process with Algorithm 3. Algorithm 3 contains the logic that applies the row and column determination of the next event going through the arbiter. There are three conditions that produce a list of events for the arbiter to record. First, if there is not a backlog of events to record, where the event triggering times are older than the current simulation time, then every event is ordered by the time when they triggered the comparator. In this case, the arbiter reads each event in the order they are received with the assumption of no latency for the arbiter to reach a particular row and column to read it out. If the first condition is not met, I only consider events triggered at or before the current simulation time. The second condition occurs if there is more than one event to read. In that case, I order the older events by their row and column. Then I wrap the list so that the first event in the list is the closest to the previous row and column arbitrated. The final case occurs when there is only one older event. This case is the simplest in construction, as it requires no additional ordering. It is the only event in the list to be recorded.

Once I have a list of events to record in the order that they should be recorded, I start the process of adding to the recorded events list. I loop through each event in the list awaiting processing being careful to ensure I stop the arbitration process if I reach the current simulation time step or if there are no more events to record. Recording an event requires two more steps prior to ap-

Algorithm 3 Get Next Event

```
if If any events are left to process in event queue then
  if If no events are currently waiting to process then
    Order list of events at and before the current simulation time step by the
    time they occur
  else
    Only consider events at and before the current simulation time
    if More than one event to process then
      Order shortened list by ordering by row and column
      Wrap shortened ordered list to start at row and column from the last
      recorded event
    else
      Only one event to process, no list organization required
    end if
  end if
end if
```

pending it to the recorded list as outlined in Algorithm 4. First, depending if the event triggered time is before or after the current simulation time, I either overwrite the event timestamp as the current simulation time for older events or I set the current simulation time to the event triggered time. Making this distinction enables for faster simulations when there are fewer events because the recording frequency is not being maximized. Instead of iterating through every time an event can be recorded, the simulation only iterates through the timestamps where events occur. The second step is to assist with future algorithm development. One of the challenging problems working with real event data is attributing events to their sources: stars, satellites, and noise. I develop an approach to solve this issue for real data in Section 3.4.3. In synthetic data where I generate the sources through the radiometry code, however, I link the events generated to the source influencing a pixel during a time step. This creates a pre-labeled event list ready to provide statistics for the Bayesian or machine learning methods I address in Section 6. After these two steps, I record the event

in consideration in an almost typical event-address representation

$$e = \{t, x, y, \rho, attribution\} \tag{3.27}$$

where x and y are the row and column indices of the pixel, ρ is the polarity of the event, and t is the simulation time the arbiter recorded. The only manipulation of the event-address I make is the addition of a column containing a string with the attribution for the event. I pull the attribution from a map of pixels for each time step. This map identifies the pixels influenced by each source from the radiometry portion of the simulation. If the pixel has a source providing photons during a time step, or within a threshold period of time, the event on that pixel attributes to that specific source. The attribution label is the source's name from the Hipparcos catalog for stars or from the title line of the TLE for satellites [25]. Any event that occurs without a source on its pixel during the simulation step or within the time threshold receives a noise attribution label.

Although the previous step finishes recording events, it is not the last step in Algorithm 1. I still must consider the refractory circuit process and handle the reset of the pixel to enable another event to register on a pixel. After each event generation, I check the array of pixels with time stamps which include the previously recorded events. For pixels that generated the event queue, if they have recorded their event and their refractory period has passed, I check them for a new event as I outline in Algorithm 5. Next, I reset the memorized current of the pixels to the current value using linear interpolation between the initial current and final current at the simulation time the pixels pass through the logic gate. Then I check that pixel for any additional events in the remaining time of the simulation time step using the methodology outlined in Section 3.3.4.

Algorithm 4 Process Event

Determine Time of Recording
if Event time stamp is before the current simulation time **then**
  Assign current simulation time stamp rounded to microseconds as the recording time
else
  Assign the time the event triggered as the recording time
  Set the current simulation time to the time the event triggered
end if
Attribute Event to Source
if Pixel is attributed to a source in the current simulation time step **then**
  Attribute event to the source from radiometric code
else
  if Prior recorded event on pixel within time window **then**
    Attribute to previous event source
  else
    Attribute to noise
  end if
end if
Add Event to Recorded List : Include timestamp, x and y locations, polarity, and attribution
$e = \{t, x, y, \rho, attribution\}$

Algorithm 5 Check if New Event Possible

if Refractory Time ≤ Current Simulation Time **and** Event Recprded **then**
  Pixels satisfying these conditions are available for another event
  They must be checked for another event before the end of the current simulation time step
end if

3.3.6 Noise Modeling

Not all events in an event stream are from the perfect tracking of instantaneous current reaching the contrast threshold and yielding an event. As with other sensors there are events generated unrelated to a real signal and imperfections in real signals. The typical analog circuity of EVS is complex and yields multiple possible sources of noise: shot noise through the photodiode, high frequency noise through circuitry components such as the source follower, leaking

current from the memorized current level, and parasitic photocurrent [39, 69]. Typically, treatment of noise in other EVS simulations and models is focused on the frequency of events generated. In particular, simulations generating a simulated event output inject noise artificially on top of the other events generated using user defined frequencies of noise event rates [21]. This is a way to induce the right magnitude of noise, but it provides no insight into the mechanisms driving the production of noise and does not guarantee the generated noise resembles the noise on a real sensor that I'm interested in rejecting algorithmically. Since I have insight into the electron-level information with this simulation, I explore the generation of noise within the circuitry based on fundamental noise sources to gain insight into the noise mechanisms. Fortunately, the understanding of EVS noise in low-light sensing environments is an active area of research [65, 64, 40, 42]. The EVS SDA community's research into this area generally focuses on quantifying EVS performance in low-light environments because the COTS EVS are designed for typical daylight dynamic ranges for the general user. Using this research as a backbone, I propose two new methods to model EVS noise to capture shot noise and high frequency noise from internal components.

Starting at the front-end of the circuitry model, the photodiode is no different than any other silicon photodiode and, therefore, has a dark current. In a typical photodiode, there is a small leakage of current over the depletion layer between the n-type and p-type layers which is called the dark current. This leakage is time varying causing fluctuation in the dark current. As more photons strike the outside layer, more electrons and holes populate the n-type and p-type layers respectively leading to a greater current flow that outpaces the small leakage of electrons, so the variation in the electron leakage is less noticeable. However,

in dark parts of the scene, the leakage is spread over less bandwidth and produces events. Since dark portions of the scene are a common hallmark of space imaging, it is important to properly capture the events generated by dark shot noise. The electron flow across the depletion layer is not deterministic. The number of electrons must be integers and there is some randomness to when they flow between the layers. The number of electrons following during any simulated length of time is best parameterized as a Poisson process.

I am not the first to develop a method to include dark shot noise in synthetic event generation. The v2e dark shot noise applies a Poisson methodology

$$\begin{aligned} r &= ((F-1) \times (I_{pd}/I_{max}) + 1) \times R_n \\ p &= r \times \delta t \\ u < p &: event_{OFF} \\ u > (1-p) &: event_{ON}, \end{aligned} \tag{3.28}$$

which scales a predefined rate noise, R_n, to a local rate value, r, by a linear function with the slope F of the normalized log photocurrent. The scaling ensures only pixels with low intensities of light experience events at the predefined dark shot noise rate. This method does not operate on the electron flux through the circuit and instead assumes a maximum log photocurrent, I_{max}, of 5.54 corresponding to the maximum digital number, 255, of the silicon array converted to log scale. The local rate, r, then multiplies the time step, δt, to find the probability of an event, p. The final step compares a uniformly distributed sample between 0 and 1, u, to the Poisson probability to determine if an event occurred. This model amounts to a random injection of events scaled by the intensity of the induced photocurrent and I can apply it my simulation. However, I have the fidelity of photons per second arriving on each pixel within the simulation, so I look to leverage that information with a novel method inspired by non-event

photon-level simulations. In particular, the dark current driving the temporal variation is always present on the circuit. I maintain the temporal variance of the dark current through the whole simulation. Then when I add the induced photocurrent it will eventually outweigh the dark current variance. This phenomenon naturally reduces the events induced by dark shot noise where there are real signals.

In non-event simulations with photon-level fidelity, the photon flux information informs the Poisson process. Generally to apply this process, after summation of the dark current and induced current from the incoming photon flux, I convert current levels in amperes to the number of expected electrons per second, λ_e, by

$$\lambda_e = 2qI$$

where q is the elementary charge and I_{pd} is the induced photocurrent. Then using the electron rate value, I sample from the Poisson distribution

$$P(X = k) = \frac{\lambda_e^k e^{-\lambda_e}}{k!} \tag{3.29}$$

to have an electron rate at a given time step. As the electron rate increases, the Poisson distribution more closely reassembles a Gaussian distribution. In most simulation environments this technique, getting a different electron per second flux for any simulated time step and integrated over the exposure time is sufficient to capture the dark shot noise of a sensor.

The problems which hinder traditional dark shot noise modeling for event type simulations where the time encoding of the events matter are tied to the sensor's temporal fidelity and its analog low pass filter. First, I consider the temporal fidelity. With the ability to encode at a microsecond level, the simulation can run at the microsecond level. This is cumbersome computationally,

so the choice of sampling time may be more aptly chosen to capture the rate of discernible spatial and temporal changes in the FOV. Low-light sensing of satellites and stars for example, may be capped at a spatial change limit of 250 pixels per second for the DAVIS346 EVS Sensor[65]. I choose to sample at a Shannon-Nyquist frequency of 500 samples per second, 2 milliseconds. This sampling frequency should adequately capture the spatial changes in the scene, with the understanding that fidelity is diminished in the temporal dimension through the process I implement in Section 3.3.4. I linearly interpolate the change in log current to yield a triggered event time between time steps. However, since the interpolation assumes a linear relationship between time and log current, the event may not be perfectly timed within the larger simulated time step. Therefore, simulating at a smaller time step yields higher fidelity in the final timestamp output and is a trade-off with computational efficiency.

The advantage of smaller time step modeling is not only at odds with computational time, however. The low pass filter implemented by the analog circuitry of the event based circuit also struggles with the random sampling at higher simulation frequencies in low-light sensing scenarios. A low pass filter inherently smooths the incoming signal and the speed at which the system can follow a signal is dependent on the corner frequency. As discussed in Section 3.3.3, I implement an empirically-derived model of the corner frequency. This model linearly scales the corner frequency based on the ratio between the induced photocurrent and the dark current. It maximizes the corner frequency at 3 kHz. At this maximum corner frequency, most EVSs should be able to follow noise sampled below 3 kHz with a 3 dB drop-off in transmission at 3 kHz. 3 kHz corresponds to a period of approximately 333 microseconds. Therefore, if I simulate event generation at 2 milliseconds to capture the spatial changes

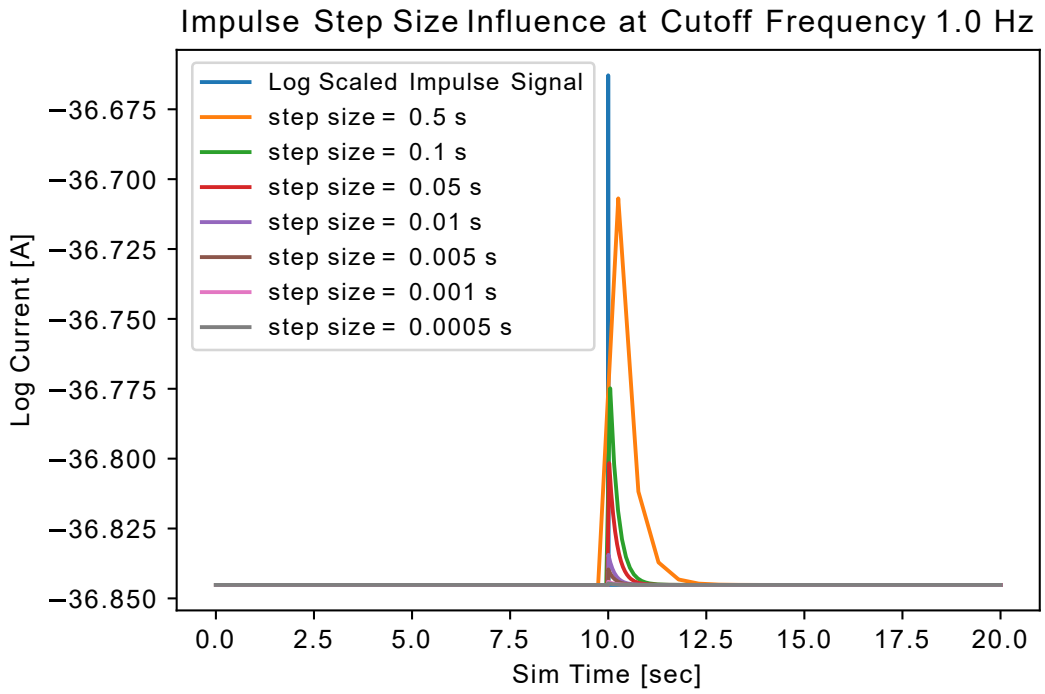


Figure 3.12: Simulation of an impulse response through an event generation simulation at the estimated lower-end corner frequency for EVS of 1 Hz and varying simulation step sizes. As the step sizes decrease in size, the change in photocurrent through the low pass filter decreases which reduces the chance of an event from an impulse.

in the scene, the maximum corner frequency is much faster than the simulated time step. Following Equation 3.21, the ϵ is 0.97 given these simulation values. Consequently, in regions operating at the maximum frequency, the current following the low pass filter is weighted as 97% towards the newly sampled value and 3% towards the previous current value. However, dark shot noise occurs at the opposite side of the corner frequency range. Since dark shot noise occurs at low levels of induced photocurrent, the corner frequency is between 1-2.5 Hz. The ϵ, with the 2 millisecond simulation step and a 1 Hz corner frequency, is only 1.2%.

Figure 3.12 captures the impact of the simulation step size on randomly sam-

pled noise in regions of low corner frequency values. As the simulation step size decreases, the simulated impulse, a 0.2 deviation in log current from the nominal value, yields smaller amplitudes in the final simulated current. As a result, the simulated current is less likely to reach the threshold difference to yield an event, just through selection of a smaller simulation step size. This trade-off of sampling noise joins the higher computational times to be at odds with higher temporal sampling that produces higher accuracy event trigger times and captures the spatial motion through the focal plane.

Figure 3.13 demonstrates this phenomenon specifically for traditional dark shot noise modeling. If I sample a Poisson distribution defined by the electrons per second of a 1 fA dark current at every 2 millisecond time step and assume that the sensor is operating at a nominal 1 Hz corner frequency, the maximum current deviation from the nominal dark current decreases by approximately 70% after application of the low-pass filter. This simulation suggests that EVS operating off of instantaneous current comparisons with theoretically infinitesimally small time divisions should not experience dark shot noise, but EVS sensors are not impervious to this noise. Sensor calibration measurements simply taken without stimulus with the lens cap on capture dark shot noise on EVS [86]. Since dark shot noise should exist and the simulation time step ideally should not be any larger to preserve event trigger time accuracy and capture of spatial motion through the sensor, I propose an alternative method to sample the dark current at any simulation time step.

Functionally, I need a value of electrons per second that is approximately around the dark current at each simulated time step and is resilient to the choice of simulation time step in its production of events. Theoretically the readout of

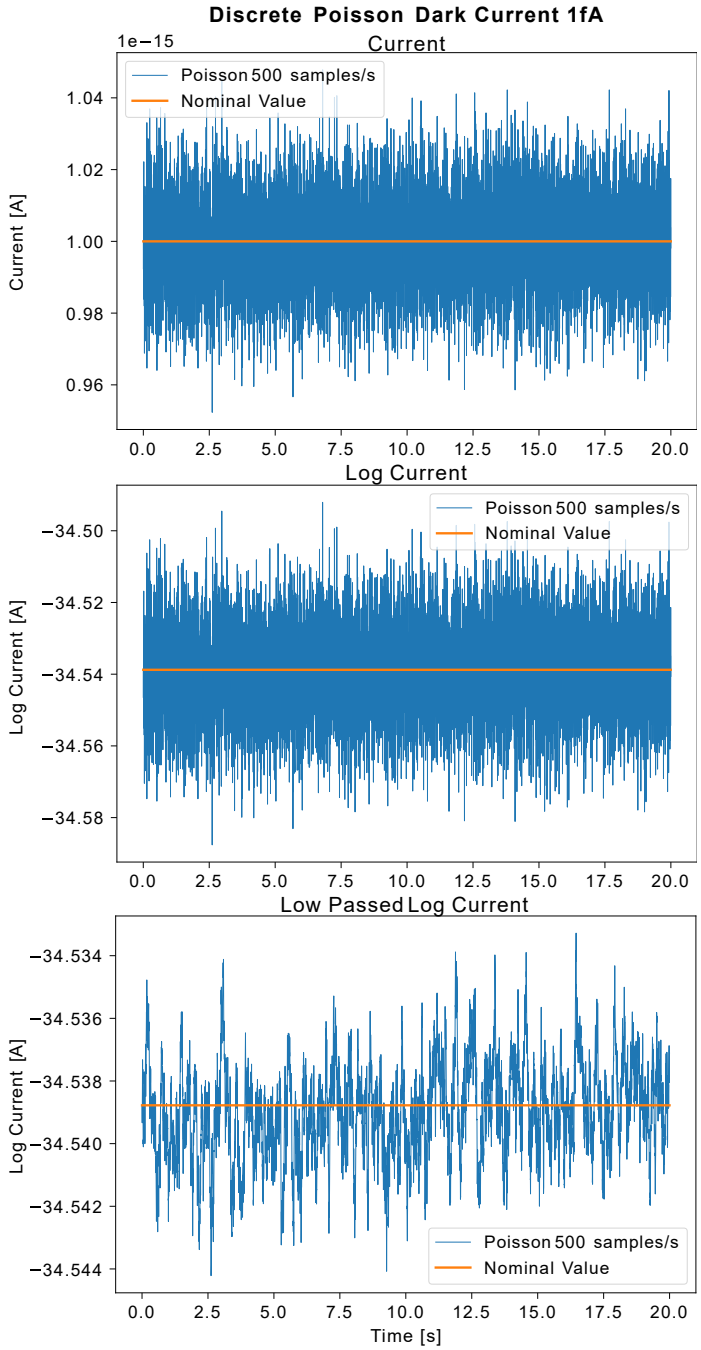


Figure 3.13: Sampling a Poisson distribution from a 1 fA dark current at each simulated time step of 2 milliseconds over a 20 second simulation produces the following current values. While the resultant log current only yields just beyond 0.02 log amps beyond the nominal value, the low pass filter, operating at 1 Hz, makes the change an order of magnitude less. This phenomenon significantly reduces the chance of events simulated due to a traditional dark shot noise model.

a dark current measurement is the accumulation of the total electrons crossing between the layers over the previous second. If I break a second down into smaller time steps, I have a number of electrons in each of those smaller time steps crossing between the n-type and p-type layers that contribute to the overall electrons per second current. With this logic, I create an estimate of the total electrons per second at any simulation step as a summation of the electrons leaked through the photodiode over each smaller time step. I sample a Poisson distribution defined by the rate of electrons per simulation time step for enough time steps to equate to a total second and then sum over the distributions. The result is an estimation of the electrons per second at the simulation time that should approximately equal the dark current value and, importantly, that carries the history of the previous leakage of electrons across the photodiode. Since the value at any simulation step is related to the previous steps, the change of the total electrons passing between the layers between time steps is effectively pre-smoothed, leaving neighboring current values without drastic changes between them. When passed through the low pass filter, the previous filter value is still heavily weighted, but the current values from the Poisson draws have enough consistency to influence output current over time. I call this method the rolling Poisson method.

To implement the rolling Poisson method I initialize a 3-dimensional array of random Poisson draws, with two dimensions defined by the number of sensor pixels in the x and y directions and the third defined by the number of simulation steps in a total second. Note that the simulation time step must divide one second evenly, so that the number of simulation steps in a total second is an integer. To reduce memory requirements, I replace a slice of the array, depicted in Figure 3.14, at each time step. After replacing the *i*th slice on the *i*th step in

an overall second of the simulation, I sum the resulting array across the time axis to get the dark current at that simulation time. At the next simulation step, I replace the $i + 1$ slice. When I reach the last slice of the array, I loop back to replace the first slice.

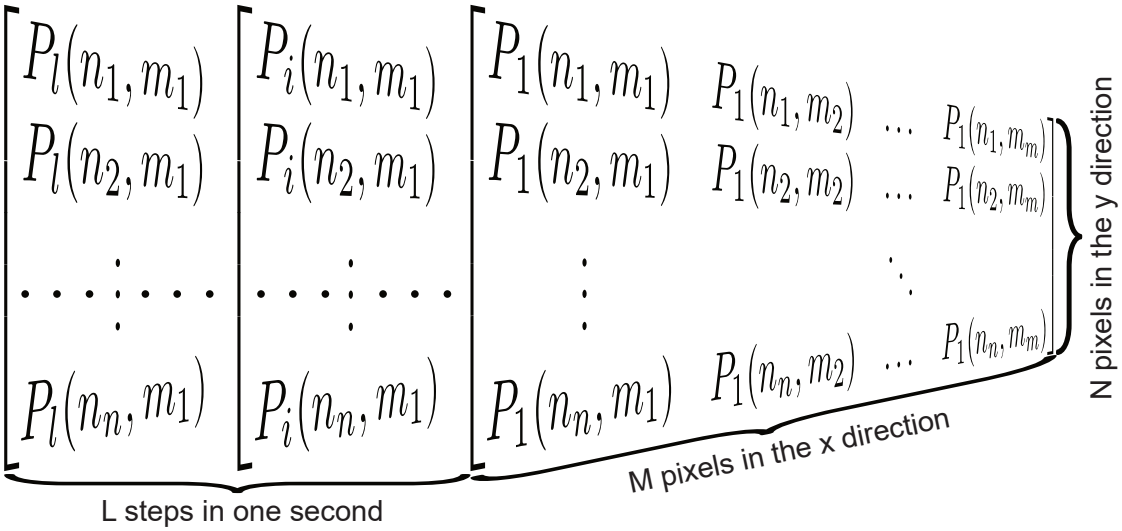


Figure 3.14: The rolling Poisson method requires pre-allocating an array that matches the sensor dimensions and has a depth of the total number of simulation steps within a second. To reduce memory requirements, the ith slice of the array is replaced at each time step.

The fundamental principal behind my rolling Poisson method is maintaining consistency within the varying dark current value allowing for event generation despite the unfavorable temporal sampling of the simulation when compared to the low-pass filter cutoff frequency. I call this consistency inertia of the dark current. Another way to achieve the inertia is through less frequent sampling sampling of the Poisson distribution or, equivalently, a Gaussian distribution because the electron rate per second is on the order of thousands for a 1 fA of current. By sampling the distribution less frequently, the resulting consistent current value provides the same type inertia achieved by the rolling Poisson method to ensure enough deviation to generate events. I compare the rolling Poisson method and the reduced frequency Gaussian methods in Section

4.1.1 which highlights the difficulty of choosing a sampling frequency when the corner frequency is changing and inconsistent across the focal plane.

I make one final consideration regarding the variability of the baseline dark current of the sensor, which in turn affects the dark shot noise generation. Dark current through photodiodes is temperature-sensitive and EVSs are no exception to this phenomenon as proven with empirical measurements on the dark current of earlier versions the DAVIS sensors [69]. That work shows the relationship between the logarithmic dark current, $I_{darklog}$, and temperature, T, to be

$$I_{darklog} = \ln(I_{dark}) = \text{constant} - \frac{E_a A_{pix}}{kT^4}. \tag{3.30}$$

They determine through experimentation the constant activation energy, E_a, which dictates the linear relationship between the quadratic absolute temperature in Kelvin and the logarithmic dark current, while k is Boltzmann's constant and A_{pix} is the area of the pixel. The temperature dependency of the photodiode still holds true, though the definition of the activation energy has not yet been empirically examined and published for newer sensors.

Instead of relying on Equation 3.30 and the definition of the activation energy to determine the dark current for any simulation, I relate the noise rate to the outdoor temperature during the collections through examination of the noise in the data sets described in Section 3.4.1. I can only extrapolate the noise within these data sets after processing. I outline my processing method in Section 3.4.3. To match temperature data with the provided data sets, I take hourly temperature information from the Albuquerque airport, within 500m of the observation site, and interpolate between data points to estimate the ambient temperature at the time of each data set's collection. The relationship between the

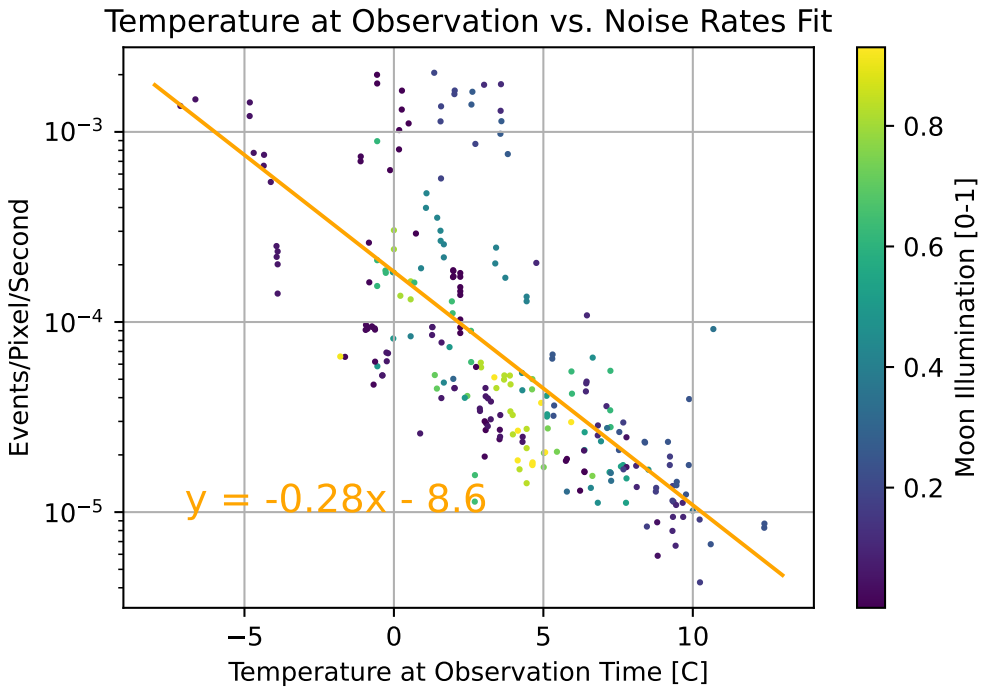


Figure 3.15: Temperature at time of measurement plotted against the noise rate, excluding hot pixels, with a fit linear relationship between the temperature and logarithmic noise rate. The moon illumination, captured in the color scaling of the points, demonstrates the lack of correlation between the moon phase and background noise event rate generated.

temperature and resulting logarithmic noise rate captured in Figure 3.15 is linear. This figure also features the moon illumination on each night to examine the affect of overall background brightness on event generation. No discernible pattern is visible regarding the moon illumination, but there is, as expected, a clear correlation between the temperature and the noise rate. Therefore, given a temperature of observation T, I estimate the noise event rate, r, with an exponential relationship

$$r = e^{P(\mu_m,\sigma_m)T+P(\mu_b,\sigma_b)} \tag{3.31}$$

by applying a least squares regression between the data sets' noise event rates and temperature. By employing an assumption of residual normality, I capture

the spread in the solution space through the standard deviation in the slope and intercept parameters. The mean of the slope, m, and intercept, b, are -0.28 and -8.6 as depicted in Figure 3.15 and their standard deviations are 0.017 and 0.089 respectively. With this model, I start with a temperature in degrees Celsius of a collection I want to simulate, conduct a normal random draw to computed the slope and intercept, then apply the linear relationship to approximate the desired background event rate driven by dark shot noise. Next, I connect the desired noise event rate to an appropriate dark current to apply in the simulation.

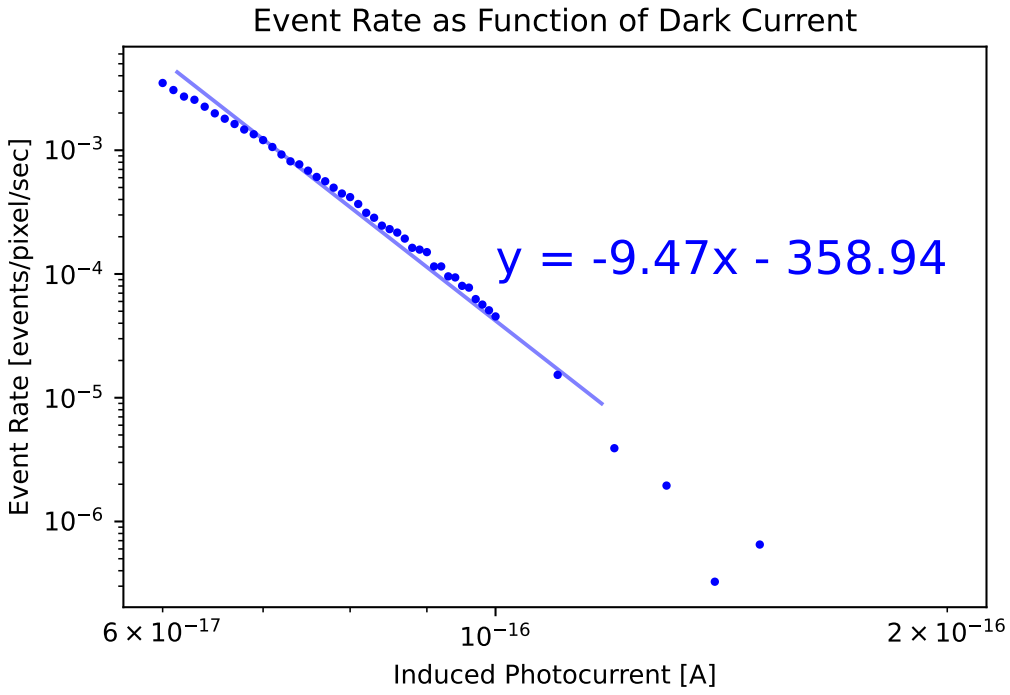


Figure 3.16: Simulating the event generation with the rolling Poisson method exposes a slightly quadratic relationship that can be approximated with a linear function to relate the noise rate to a particular simulated dark current.

I make this connection through experimentation of the rolling Poisson process with varying dark currents. The variation of the dark current parameters exposes a relationship between the logarithmic dark current through the pho-

todiode and the resultant logarithmic event rate. I run these simulations with no additional induced photocurrent and I only apply the rolling Poisson noise methodology. Each noise event generation simulation follows a full size 3rd generation Prophesee focal array, 480 by 640 pixels, over a 20 second period with the aforementioned 2 millisecond time step and threshold of 0.2. I calculate the noise event rate, E_r,

$$E_r = \frac{e_{tot}}{x_{pix} y_{pix} t_{tot}} \tag{3.32}$$

by dividing the total events in a simulation, e_{tot}, by the total pixels calculated through the multiplication of the number of pixels in each direction, x_{pix} and y_{pix}, and the total time I simulate, t_{tot}. Note, this full array event rate is equivalent to the average individual pixel event rate calculated as the total events on one pixel divided by total time simulated and in this way 307,200 samples contribute to the resulting event rate. This metric is common in the EVS community to evaluate noise [66, 39, 65]. Therefore, I continue to use this metric throughout Section 4.1 to evaluate the noise event generation of my simulation. In this case, the simulation yields the event rates in Figure 3.16. The Figure depicts a linear fit of the resultant event rates. While a 4th order quadratic arguably fits these points better, I do not need that level of fidelity when connecting this estimation to the linear fit of the noise rate as a function of temperature. Through examination of the range of noise event rates in Figure 3.15, I know the resulting estimated event rate for any particular temperature can have an order of magnitude spread. Therefore, a linear estimation of the relationship between the desired noise event rate, likely yields a dark photocurrent that produces an event rate within the desired order of magnitude. After defining these two linear relationships, I skip the intermediate step of the desired event rate, combining the two linear relationships to solve for a simulation dark current value

from the temperature.

$$I_{dark} = e^{\frac{P(\mu_m,\sigma_m)T + P(\mu_b,\sigma_b) - b_{dark}}{m_{dark}}} \tag{3.33}$$

calculates the dark photocurrent, I_{dark}, in amperes from the aforementioned distributions derived from the temperature to noise relationship and the intercept, b_{dark}, and slope, m_{dark}, from the noise to dark current relationship.

Dark shot noise, despite being a critically important portion of the SDA-focused simulation since large portions of the focal plane have minimal photon interaction, is not the only noise source to consider. These additional noise components include leaks within the circuitry and high frequency noise from other analog components such as the source follower. The v2e simulation treats all these other noise sources under the leak rate umbrella. The commonality between leak noise generation sources is their effect on the memorized current. Through leak sources, the memorized current drops over time and eventually the logarithmic delta between the memorized current and the stable induced current reaches the θ_{ON} threshold value and produces an ON event without any change in the photocurrent required. As such, leak rate events are only ON events. The primary leak producing this phenomenon within an EVS circuit is the current flow between the differencing amplifier into the floating input node of the same amplifier. Additionally, parasitic photocurrent, caused by stray photons either penetrating the protective metal or scattered into the n-well also produces this affect.

V2e models leak noise events by assigning a rate at which the leak events occur, R_{leak}. Through multiplication with the simulation time step, Δt and the threshold required to yield an on event, θ_{ON}, they define

$$\delta_{leak} = \Delta t R_{leak} \theta_{ON} \tag{3.34}$$

as the change to the memorized current at each time step, δ_{leak}. They apply the change through a subtraction of δ_{leak} from the logarithmic memorized current at each time step which yields a consistent decrease. Nominally, the v2e paper [69] suggests a leak rate of 0.1 Hz is appropriate for most applications and covers both events generated by the leak through the reset switch and the parasitic photocurrent.

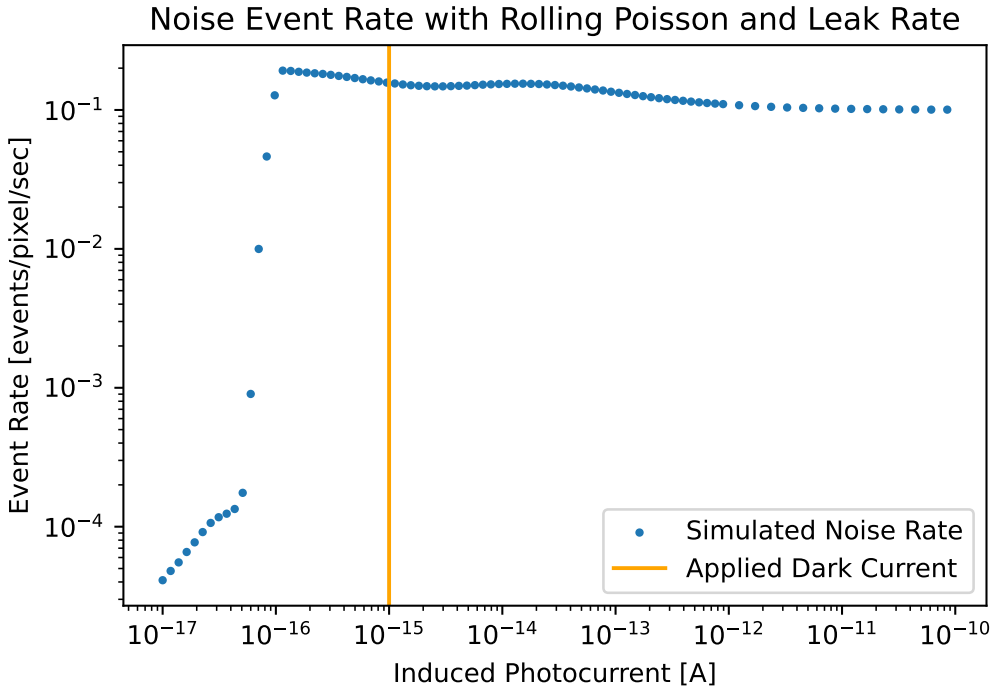


Figure 3.17: Applying the v2e leak rate model to a rolling Poisson shot noise method generates a discontinuous rate of events. For levels of induced photocurrent above the applied dark current, the noise event rate is close to the applied 0.1Hz the v2e authors suggest as a good estimate for noise event generation outside of shot noise. For one decade below the dark current, the events generated exceed the suggested value as the induced photocurrent is on the same magnitude at the dark current and the additional change in the dark current through the rolling Poisson method creates additional events. As the dark current becomes the predominant contributor to the overall photocurrent, a sharp decline in the event rate occurs because the primary method of noise event generation switches to the shot noise method.

I apply the aforementioned nominal leak rate at 0.1 Hz to the rolling Poisson

method with a set dark current of 1fA over a range of induced current values. I start the simulation by sampling a uniform distribution for each pixel to randomly set the memorized current between the induced photocurrent value and one ON threshold below the induced current in the logarithmic scale. The random but even spread of a uniform distribution prevents all the pixels triggering simultaneously which can lead to a large number of events awaiting arbitration at the end of the simulation if the total simulation time is a multiple of 10 seconds. As I run the simulation for 20 seconds, the normal initialization is necessary. I then evaluate the noise event rate with Equation 3.32. There are three distinct regions in the results. First, the nominal region, where the induced photocurrent is greater than the applied dark current. In this region the simulation approximately produces the nominal leak rate requested. The second region is between one decade below and two decades above the simulated dark current value. This region exhibits a noise event rate above the 0.1 Hz leak rate frequency. For induced photocurrents closer and, particularly, below the dark current value, the influence of dark shot noise produced by the rolling Poisson method grows. Variance above the nominal dark current triggers leak rate events earlier and variance below the nominal resets the memorized current to a lower value than the nominal induced photocurrent plus dark current. These two effects combine to elevate the resultant noise event rate. The final region, one decade below the dark current and lower, the induced photocurrent is no longer relevant and the noise events are primarily driven by the dark shot noise induced by the rolling Poisson method.

While I can explain the discontinuous noise event rate generated by the v2e leak rate method and rolling Poisson method combined, Figure 3.18 demonstrates how erroneous the leak rate method is at producing the variation in noise

events of a real EVS over a range of induced photocurrent. The empirical data in this Figure depicts events generated by a 4th generation Prophesee EVS. The Air Force Research Laboratories providing the data, subject the EVS to the constant photon flux output from an integrating sphere. They collect events at each lux level for 60 seconds and then evaluate the overall event rate, the OFF event rate, and the ON event rate. Unlike the flat region the v2e leak rate method creates, the empirical data shows a peak noise rate at a photocurrent equivalent to the incident 4-5 lux. At both lower and higher levels of photocurrent the noise event rate drops off. It is also important to note, the OFF event rate is universally greater than the ON event rate. The v2e leak rate method only generates ON events on its own. While the mechanism of the memorized photocurrent decreasing overtime is real, the mixture of ON to OFF events in the empirical data indicates the leak noise is not a primary source of noise of events in regions of higher induced photocurrent. Considering the polarity makeup and the magnitude of resultant noise events from the v2e leak rate method is not indicative of real event data, I do not continue to use the method moving forward. As a reminder, my goal is to generate noise events with realistic polarities and frequencies in my synthetic data so I can apply them in future algorithmic development. Therefore, I propose a new method to capture the characteristic frequencies and polarities.

My method takes a second look at a source of noise unaddressed by v2e. Components, such as the source follower, introduce higher frequency noise into the EVS current. While there are possibly many contributing elements of noise within the circuitry, I apply the central limit theorem to assume I can encompass all of them with a Gaussian distribution. The Gaussian distribution about the central current value in non-log space theoretically also fits the real data

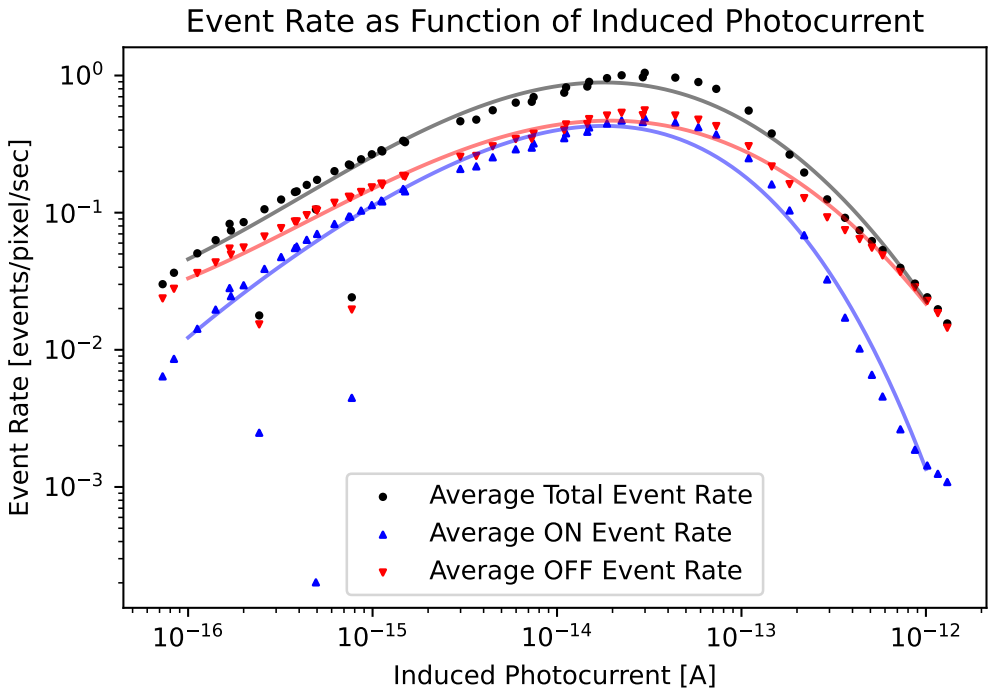


Figure 3.18: The Air Force Research Laboratories experimentally collected noise event rates while varying levels of induced photocurrent through the use of an integrating sphere. The sphere, providing a uniform photon flux in both space and time allows for isolation of the noise generation at varying levels of stimulation. The resulting curve demonstrates a region of peak noise activity and decreased activity at higher and lower levels of induced current. Additionally, the noise on real signals is weighted towards OFF events.

noise profile of more OFF than ON events. Because it takes less change to produce an OFF event than an ON event, a normal distribution crosses the OFF event threshold line more frequently than the ON event threshold assuming the thresholds for both polarities are equivalent. I also assume that this Gaussian noise is higher frequency because it peaks at a region of higher induced photocurrent as shown in Figure 3.18 and the peak may be a result of noise making it past the low-pass filter at higher induced current values. Consequently, I anticipate capturing the drop-off towards lower induced current values in the Figure from the Gaussian noise being modeled as high frequency. I also expect the

drop-off in the higher induced photocurrent regions is the result of the overall noise amplitude growing at a different rate than the photocurrent and becoming less consequential for high levels of current.

I use data provided by the Air Force Research Laboratories to define the standard deviation of the Gaussian [66]. I propose equating the area under the tails of the Gaussian probability density function, the complementary error function, to the probability of an event occurring in any simulated time step as defined by the total event rate per simulated time step. From each background event rate data set, I evaluate the overall event rate as usual with Equation 3.32 and multiply by my simulation time step of 2 milliseconds to produce a probability of an event on a pixel during a unique time step. I minimize the difference

$$\arg\min\left((P_{event} - erfc(z))^2\right) = \arg\min\left(\left(P_{event} - \left(1 - \frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}\,dt\right)\right)^2\right) \tag{3.35}$$

between the complementary error function, erfc(z), and the probability of an event, P_{event}, squared to find the z score of the error function generating the desired area under the probability density function [36, 105]. Rearranging the z score formula to solve for the standard deviation, σ

$$\sigma = \frac{|x - \mu|}{z} \tag{3.36}$$

as a function of the difference between the nominal mean induced photocurrent, μ, and the photocurrent that yields an OFF threshold change, x, divided by the z score. Working with the Air Force Research Lab's event data and assuming a threshold change of 0.1 for the OFF events, I plot the Gaussian standard deviation as a function of induced photocurrent in Figure 3.19 on the log scale. The standard deviation to produce 0.1 threshold change is nearly 1.5 decades lower than the induced photocurrent for all values. While appearing linear, a

quadratic fit captures the slight drop in the ratio between the induced photocurrent and the applied standard deviation at higher levels of photocurrent more effectively. For this specific camera and an assumed threshold value of 0.1, the quadratic component, -0.01, is small, as compared to the linear components, but provides the slight change in the standard deviation required to have a drop-off of noise events within a few decades of induced photocurrent further discussed in Section 4.1.2. With this continuous fit, I compute the appropriate standard deviation to sample around each pixel's induced photocurrent in the simulation at each time step. Since I assume this noise to be high frequency, I apply no super sampling or rolling methods to maintain the inertia of individual draws.

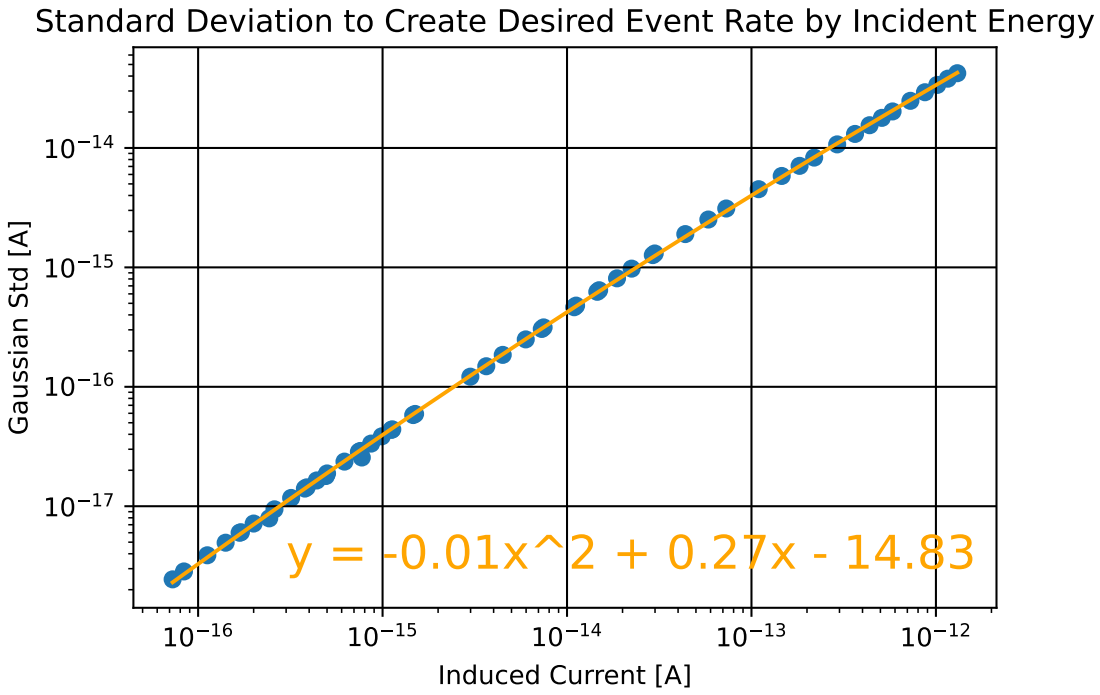


Figure 3.19: The standard deviation assuming the noise event rate is the area under the Gaussian probability density function tails defines a nearly a linear relationship between the induced photocurrent and the standard deviation required to generate the expected noise events. I apply a quadratic fit to ensure the slight reduction in slope for higher values of induced photocurrent.

I must address one final noise source beyond the shot noise and high fre-

quency noise in my simulation, the parasitic photocurrent. V2e addresses this noise within its leak rate noise generation. While I have ruled out the overall leak rate process being the primary contributor to the noise rate on real signals, I have no conclusive evidence to remove a similar leak model completely for the parasitic photocurrent. The parasitic photocurrent leak rate is not a blanket 0.1 Hz. Instead, it is dependent on the number of photons interacting with a pixel. As the number of photons increases, a greater number of them make it into the n-well. Therefore, the rate varies according to the incident photon flux. Using the induced photocurrent, I_{pd}, I compute the parasitic leak rate, R_{para}

$$R_{para} = I_{pd}\frac{f_{para}}{R_{\lambda}} = I_{pd}\frac{f_{para}\eta * \mu_{pix}^2 * h * c}{QE\lambda * e} \tag{3.37}$$

in Hz by scaling the photocurrent with a empirically derived rate parameter which scales hertz as a function of lux, f_{para} [69]. I convert the hertz per lux rate value to hertz per watts with luminous efficacy, η, and the area of the pixel, μ_{pix}^2. I assume the luminous efficacy to be ideal as 680 lum/W for these calculations. I estimate the received wattage on the pixel to do the final calculation to hertz by dividing by the pixel responsivity, R_{λ}, which is a function of the pixel's quantum efficiency, QE, Plank's constant, h, the speed of light, c, and the elementary charge e. Once I determine the event rate, I still apply this rate in the same way as the universal leak rate of v2e. Ultimately, the influence of this leak rate is not scrutinized in this dissertation. The primary reason is the minimal influence of the parasitic leak rate over the photocurrents I analyze. Rearranging Equation 3.37 to solve for the induced photocurrent and assuming a 2.7e-3 hertz per lux efficiency reported in [69], it takes an induced photocurrent at 1.5e-10 amperes to generate the 0.1 Hz leak rate assumed by the v2e authors. Therefore, the influence is trivial in the photocurrents I present in this analysis.

To summarize, I propose two new methods of noise event generation to bet-

ter capture the appropriate frequency and polarity of events in both dark regions of the scene and pixels with induced photocurrent. First, I suggest the rolling Poisson method. This method samples the dark current leak of electrons per simulation step size and keeps a history of one second of samples to create a dark current value with the inertia to pass through the low-pass filter. The second method I offer is a tuned standard deviation on a Gaussian sampled at the same rate as the simulation. The tuning of the standard deviation aims to equate the event rate at each simulation step as the probability of the complementary error function. Each of these methods aims to produce more authentic noise event signatures in order to provide synthetic data with realistic noise characteristics for algorithm development.

3.4 Event-Based Space Domain Awareness Data

I need real data to validate the model described in Sections 3.1, 3.2, and 3.3. Sections 3.4.1 and 3.4.2 describe the data provided to me, courtesy of Dr. David Monet of the Air Force Research Laboratories. Here I describe his collection and processing methods. Section 3.4.3 describes the post-processing method I create to label the dataset for validation and algorithm development purposes. Then I compare my new processes to the traditional techniques applied by Dr. Monet in Section 3.4.4.

3.4.1 Collection Methods

There are multiple COTS EVS currently available, as described in Section 2.2. The model validation analysis in this dissertation is constrained to available truth data from a Prophesee Gen3 camera with an 85 mm, f/1.4, lens with no additional filters applied, taken in Albuquerque, New Mexico from January to February 2021[1]. In 2021, the Prophesee Gen3 was the latest generation of EVS available and, therefore, the logical choice to collect a practical dataset to evaluate current performance for this use case. Dr. Monet's primary collection purposes and conclusions are covered in Section 2.3. While my model validation is constrained to this one sensor, my validation methods are designed to be independent of the specific sensor and can be applied to additional sensors in the future.

For this data set, the Prophesee Gen3 camera was pointed to a stationary azimuth and elevation via an electronically controlled mount. This procedure was selected because both satellite and star streaks are present in the data when the sensor's aim-point is rotating with the Earth. The nominal collection time was 30 seconds. After each collection, Dr. Monet programmed the mount to slew the pointing of the EVS to another azimuth and elevation. Then the collection process was repeated. Since the camera reports changes in current levels on a logarithmic scale, the EVS is very sensitive to small changes when the scene is dark. The background radiation rate of space is not uniform and propagation through the atmosphere creates further distortions. Motion of the observer exacerbates the non-uniform field hitting the sensor by inducing a faster progression in space through the varying incident flux. Therefore, the slew between

[1]Provided courtesy of Dr. David Monet of the Air Force Research Laboratory Space Vehicles Directorate.

telescope mount pointings yields a higher number of events than during staring operations. Data during slewing is not examined in this dissertation. Even after reaching a set azimuth and elevation, approximately the first 8 seconds of each dataset produces higher noise levels until the pixels reach a nominal intensity for the new staring angle. This portion of the data is also not examined in this analysis. Further insight can be gleaned from the nosier datasets on noise generation and should be understood for practical applications of moving observers. The importance of this analysis will be discussed in Section 8.3

Dr. Monet's observation schedule was not informed with an analysis on the probability of a satellite observation because star observations provided a better metric for the visual magnitude analysis he was performing [63]. Therefore, his experimental plan did not require such planning to meet his primary objectives. In order to identify datasets that positively caught a satellite, Dr. Monet co-boresighted a traditional SLR Camera with the Prophesee Gen3 EVS. During EVS collections, Dr. Monet set the camera to take a corresponding traditional images. This creates an image with streaks corresponding to stars and satellites over the total exposure time. He then identified datasets with satellites using the traditional processing techniques described in Section 3.4.2.

3.4.2 Labeling Events

Taking his collected EVS data, Dr. Monet devised a method to apply traditional processing techniques to label events in the time series as belonging to a satellite, star, or noise. This process follows the flow chart in Figure 3.20.

Dr. Monet first integrated the data over 3 second time slices. With positive

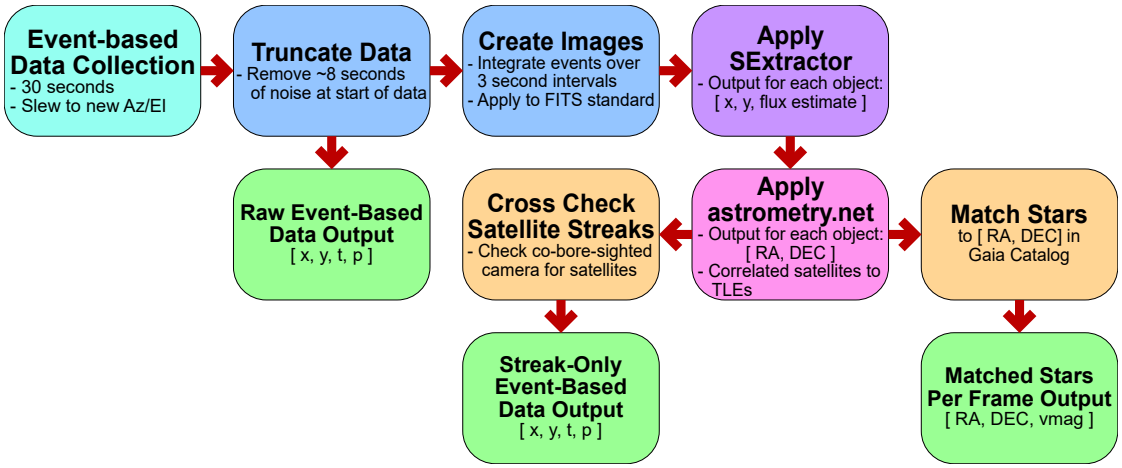


Figure 3.20: Traditional astronomical processing method applied to event-based data stream.

events and negative events given a label of 1 and 0 respectively, the integration amounts to a summation of positive events within a set time window. After the summation, the slices are compiled into a flexible image transport system (FITS) image. While the FITS format has expanded since its 1981 debut where the application was onto magnetic tape, the intent of the format remains. The FITS format provides a standard for the transmission of n-dimensional regularly spaced data arrays and allows for the attachment of an unlimited number of auxiliary parameters that are associated with the image [103]. This enables development of processing methods to be applied to data conforming to this format and is, therefore, widely used in the astronomical community that developed it originally. It is important to note that event-based data streams do not conform to the uniform spacing in the temporal dimension requirement. Therefore, in its raw form the event streams do not quality for the FITS format. A sacrifice to the information in the temporal spacing must be made to access the processing algorithms traditionally available to the astronomical sensing community.

Now that the data fit the regularly spaced standard, Dr. Monet fed the FITS

images into the program SExtractor to group together the streaks as objects [4]. SExtractor passes through the FITS information twice. The first pass creates a model of the sky background and the second applies the series of image transformations in Figure 3.21. This processing is applied to classify pixels into belonging to grouped objects or the background, and then output a catalog of rudimentary classified objects [46].

The first step in the SExtractor process is to estimate the sky background. This process determines what the image readout would be without any sources in the FOV. This is done to easily eliminate pixels that fall below a certain threshold of brightness without placing hard requirements on the type of sensor, optics, and parameters under which the image was generated. During the first pass the image is separated into a rectangular grid where a local background is estimated. SExtractor amasses these estimations into a histogram of brightness levels and calculates the median and standard deviation. The histogram is then clipped around the median at $\pm 3\sigma$. This process is repeated until the change in the standard deviation falls below a given threshold. Given dark sky imaging has high contrast levels and relatively small number of pixels impacted by the higher intensities of sky objects, the median is typically within the dark sky background. Breaking this process into the local grids allows a median filter to reduce the influence of the brightest objects dominating a local background. After filtering, a bicubic-spline interpolation, averages the median sky background levels for an overall intensity cutoff that identifies pixels as part of the background [4].

After the first pass to identify the sky background level, the second pass through a FITS image applies a series of image processing techniques to group

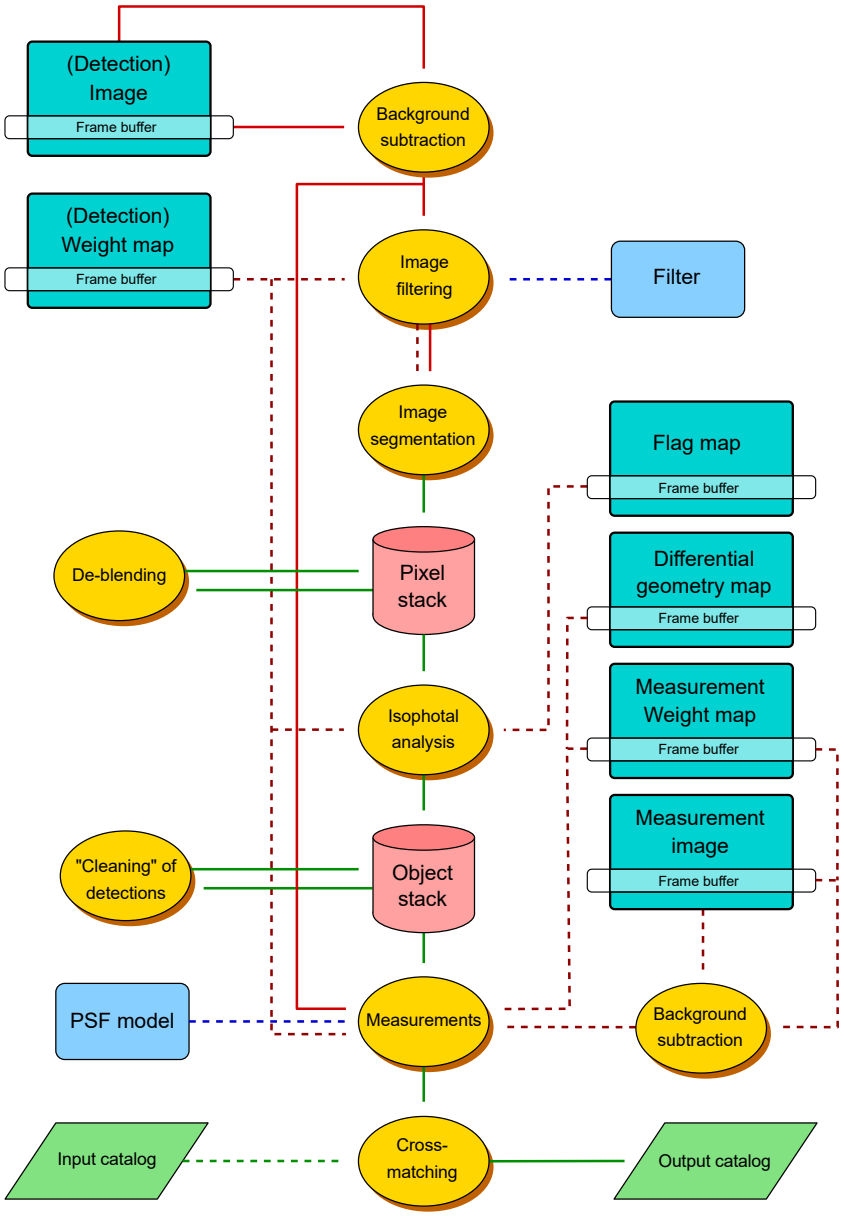


Figure 3.21: Image processing methods applied by SExtractor. The dashed line indicate optional processing techniques that are not applicable to every situation. The diagram is from SExtractor documentation[4].

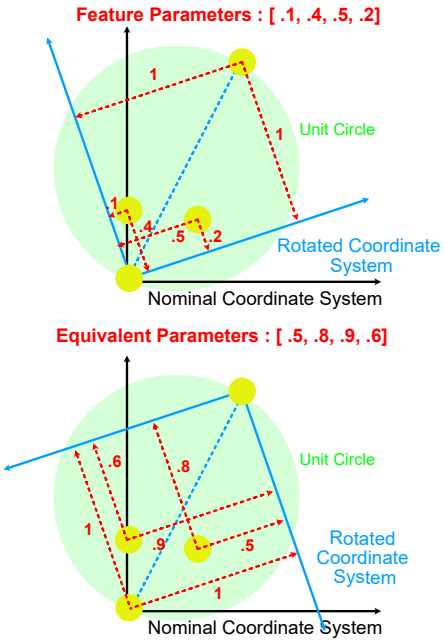
pixels into the background and different objects output as a catalog. At a fundamental level, if there are two peaks of intensity divided by an area of background, the objects are broken into two objects. Given two sky objects, a sensor may not have the resolution to create uniform troughs between objects and, therefore, this process is not foolproof. Figure 3.21's flow chart aims to reduce

this phenomenon. First, the background intensity level is subtracted from the entire image to effectively remove the background pixels from consideration to be objects. Then the image is convolved with a specified profile to sharpen the edges of the remaining objects. At this point the image is segmented into object and background labels. Afterwards, the "de-blending" process separates the object pixels into different objects. The "isophotoal" analysis then measures the shapes and positions of each object where the position is chosen as the geometric center of the object. The subsequent cleaning process reconsiders the pixels on the edge of each object considering the contributions from the neighboring objects. Finally, a version of photometry is conducted in the "measurements" block and fed into a network to classify the object as a star, galaxy, or satellite. This classification is added to the output catalog and cross matched to any provided input [46].

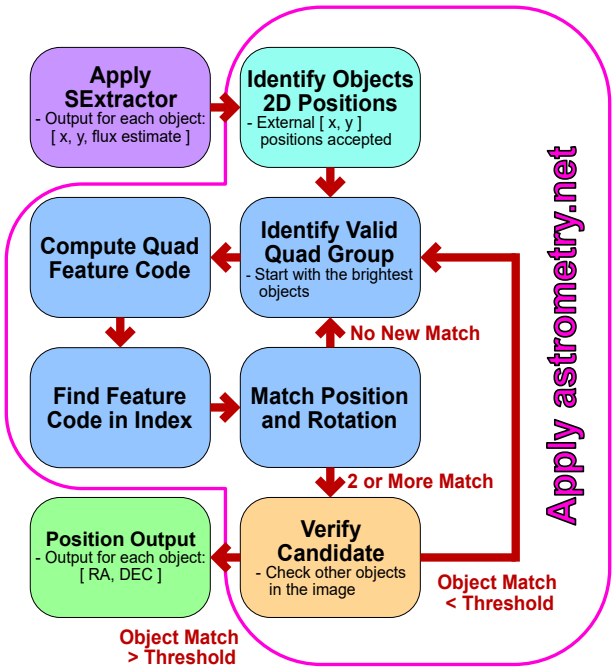
The result of the SExtractor process is a catalog with the column and row center, estimated flux, and preliminary classification for each object for the 3 second frames Dr. Monet fed into the system. Given that events cannot be directly tied to intensity due to temporal delays as discussed in Section 2.3, the flux output of SExtractor is relatively meaningless beyond a measure of relative brightness of each object within the FOV. The relative brightness, however, is a useful input to the next step in Dr. Monet's processing, applying astrometry.net to achieve a plate solve [87]. Plate solving in astronomy is the process of matching the center of an image to a right ascension and declination based on matching the pattern of stars in the image to a catalog of stars. After finding the right ascension and declination of the center of the image, the other pixels are assigned to right ascension and declination values in a similar manner to the gnomonic projection discussed in Section 3.1.

There are multiple plate solving algorithms publicly available. One of the strengths of astrometry.net is that it can be applied as a blind solver, with no prior reduction of the sky catalog of stars to a limited region. An extensive search without a reduction in catalog would be time consuming and any given image may be missing or have additional features, like satellites, in the image. To work around these issues, astrometry.net groups patterns of features, relative orientations of stars, into an index. If a particular pattern of features is within an image, it reduces the locations in the entire star catalog that the imaging system can be pointed towards. The list of possible locations is reduced by intersections between the distinct features found in the inverted index. Using a list of features and the locations they occur front loads the computational time when the index is created.

The astrometry.net creators also took special consideration when developing the features list to make the features invariant to scale and orientation. This is done by defining a local feature coordinate system as in Figure 3.22a, defined by the two most widely separated points. One point is set as the origin and the coordinate system is rotated until the x and y defining the second point location are equivalent. These distance values are set to 1 defining a unit circle. The astrometry.net creators found quadruple point features effective, so the other two point locations in x and y define a 4 parameter definition for the feature. If the most widely separated points are swapped, the feature parameters are mirrored making these two sets of parameter sets equivalent. It is important to note that not all quadruple pairs are features in astrometry.net's index, only pairs where the additional two points lie within the unit circle defined by the coordinate system are utilized to limit the scope of the index and the feature search for any new image.



(a) Extracting feature parameters with the astronmetry.net procedure takes 4 neighboring sources and parameterizes their relative location within a unit circle. There are two equivalent parameterizations for any 4 sets of stars.



(b) To apply the astronmetry.net processing on a new reconstructed image from events, SExtractor first identifies the (x, y) locations of groups of events. This input feeds the astronmetry.net parameterization and matching process. The output of the process is a (RA, DEC) for each (x, y) of the input.

Figure 3.22: Astrometry.net processing

The astrometry.net process applied to a new image is outlined in Figure 3.22b. The two dimensional positions of objects are identified in the image. Starting with the brightest objects first, quadruple point features are calculated. The features are given a tolerance and matches within the tolerance are found within the precomputed index. The output from each feature is a list of candidate positions and rotations. As soon as two quadruple features agree on the position and rotation, that candidate image is verified against all objects in the image with a given tolerance for objects to be distractors or dropouts from the original catalog.

This process works best with point source images where the star field is

either tracked or motion is removed from the image in post processing. This highlights the purpose of applying SExtractor to the FITS image prior to using astrometry.net. The fixed azimuth and elevation collection method described in Section 3.4.1 creates oblong streaks for stars as the mount rotates with the Earth. Instead of providing those shapes through a traditional image to astrometry.net, Dr. Monet takes advantage of an alternative input format of a list of x and y object locations in order of relative magnitude. Dr. Monet's final step mapped astrometry.net's output right ascension and declination of the objects to objects in the Gaia catalog. Astometry.net also contains the capability to use the public TLE catalog given an accurate collection time. While not always successfully correlated to a satellite, Dr. Monet provided a list of events from each data set corresponding to the steak he identified in the co-bore-sighted SLR camera. He did not produce a list of corresponding events for stars. Instead, he generated a list for each 3 second slice of stars identified and their corresponding Gaia catalog right ascension, declination, and visual magnitude. I will discuss the implications of this data attribution method and the resulting difficulty to compare to a non-traditional method in Sections 3.4.3 and 3.4.4.

3.4.3 Clustering

The data attribution achieved in Section 3.4.2 relies on the flattening of events into an image with a 3 second exposure time, so that traditional astrometric processing techniques can be applied. The downside to this technique is that the temporal fidelity is lost. It's possible a pixel has two sources, true signals or noise, contributing to its events in a time window. Since disregarding the temporal fidelity of event-based data eliminates one of its strongest advantages,

I set out to discover other ways to group and classify events effectively both in a batch and online context. This section covers batch clustering which is motivated by the event data provided by Dr. Monet. Since, Dr. Monet only provided specific events he found associated with non-star streaks, I did not have data labels for stars and noise events within the data set. Additionally, I do not have clarification from Dr. Monet in how the identified events from the satellite streaks were mapped. I assume that pixels are assigned to the satellite object for each time slice, and all events on that pixel are mapped to the satellite. With my method outlined in this Section, I produce a label for each event and cluster related events fully utilizing the temporal aspect of the data.

The data attribution task is not trivial. An example of one of the data sets is plotted in Figure 3.23 with all 3 dimensions to demonstrate its complexity. There are thousands of events dispersed throughout the 640 pixel by 480 pixel by approximately 22 second volume. On events are colored blue and off events are red. Within the volume, clusters are visible, appearing as discontinuous lines. Since each data set has many thousands of events, it is unreasonable to manually attribute each event to a type of detected signal by hand. However, the linear patterns that are visible in Figure 3.23 indicate that the labeling task can be simplified by a couple orders of magnitude when the events are grouped into 3 dimensional clusters.

The 3 dimensional clustering approach is originally inspired by examination of a 2-dimensional projection of the data as seen in Figure 3.24. This Figure contains the last event on each pixel displayed in Figure 3.23. With this 2D projection, the human eye can more clearly pick out the patterns of streaks associated with clusters of related events. In a way, the projection resembles the

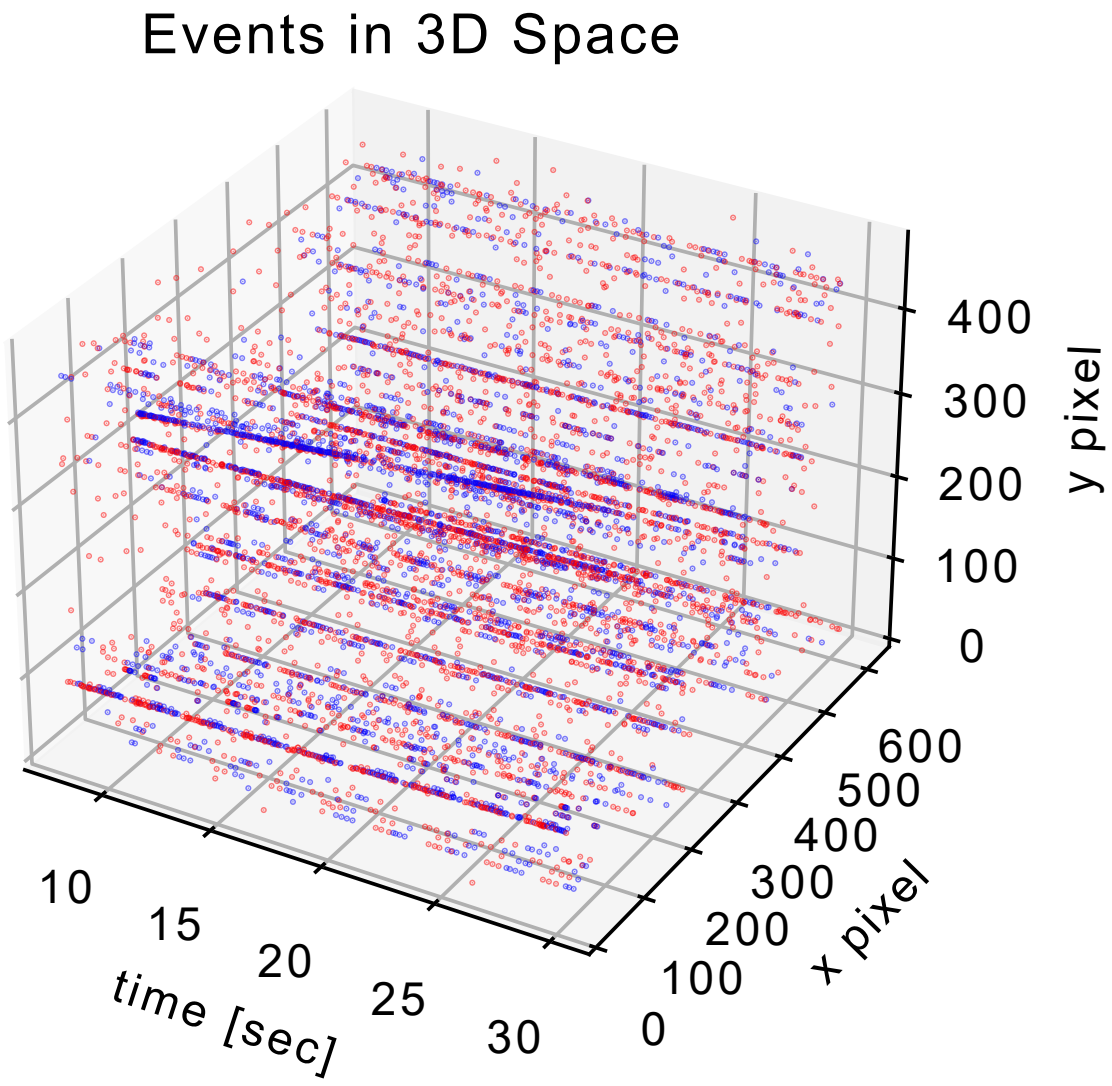


Figure 3.23: Example event-based data set plotted in 3 dimensions. In the 3 dimensional volume of events, the grouping patterns resemble lines. Positive ON events are blue and negative OFF events are red. Despite the lines being discernible, it is difficult to see all the lines in this isometric view.

flattening required for traditional image processing to be applied to the data set and it is more familiar for the average user that handles typical imagery. In this image the smaller streaks moving in a generally uniform direction are stars. The movement is due to the fixed nature of the observer with respect to Earth. As the Earth rotates, the stars are relatively stationary in the barycentric coordinate system over a short period of time such as the 20 seconds in this data set. There-

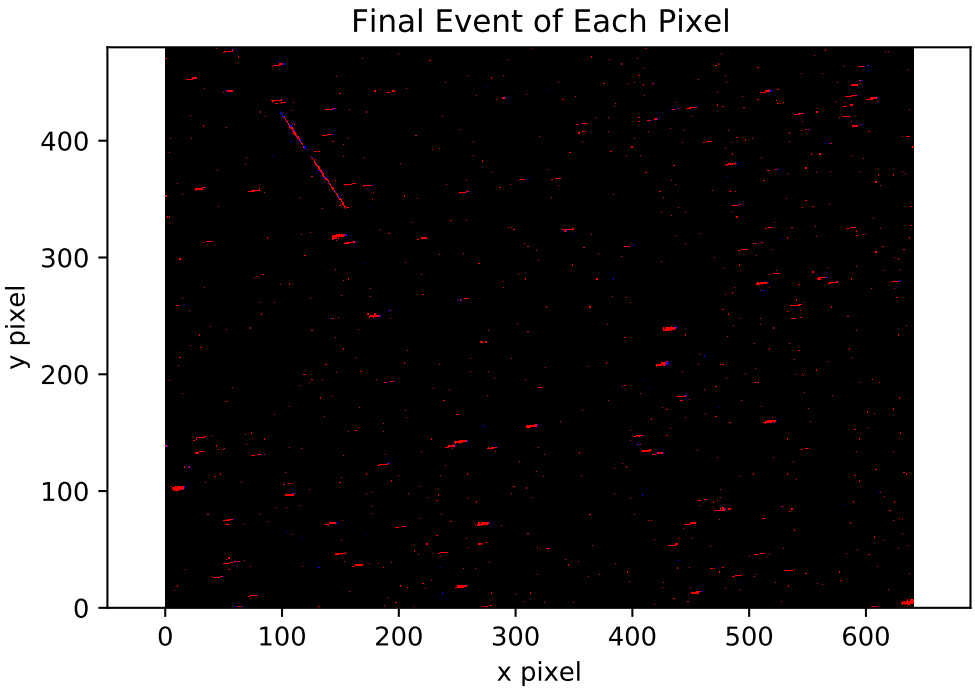


Figure 3.24: Example data set reassembled into a typical frame. Positive ON events are blue and negative OFF events are red. A satellite or star signal typically provides more photometric current than the background. Therefore, the first events are typically positive and the final events are typically negative. This graphic shows the last event on a pixel and therefore, is mainly red. The satellite track in the upper left-hand corner goes in a different direction from the stars.

fore, they display relatively straight line motion due to Earth's rotation. Since this motion is induced by the observer for essentially fixed points, this motion appears uniform for all stars. Variance is attributable to the star magnitude and the spread of light of each star not necessarily passing through the center of each pixel. The streak in the upper left hand quadrant of Figure 3.24 is the satellite streak. Its unique length and direction identify it as the outlier in this data set.

Given that the data set is 3 dimensional not 2 dimensional, I exploit a 3 dimensional clustering algorithm to cluster the events. Specifically, I use the density-based spatial clustering of application with noise (DBSCAN) algorithm.

This clustering algorithm, developed in 1996, addresses some of the shortfalls of other spatially focused clustering methods. In particular, the researchers desired an algorithm that could handle large databases, on the order of thousands or greater objects. Not only did they push for faster processing, but reducing the need for prior information about the large datasets. Therefore, the user requires minimal knowledge of the dataset to choose input parameters and will not need to sift through thousands of objects before implementation of the algorithm. The researcher's other goal was to create an algorithm that can cluster objects with an arbitrary shape in the spaces they occupy. This feature is the primary benefit of DBSCAN in the use case of sorting events over a traditional k-means clusters algorithm. K-means clustering optimizes the location of a chosen number of gravity centers. Then each object is assigned to the closest gravity center as long as it is within a limiting radius. The result are circles, or spheres in 3 dimensional space, of clustered objects [26]. The oblong nature of the groups of events in Figure 3.23 is the primary reason for choosing DBSCAN. Imagine a sphere chosen to capture the temporal change within each cluster. Even a scaled version of the time axis, the extreme of this being the completely flattened array in Figure 3.24, is likely to capture neighboring events belonging to a different source or noise within the sphere because the separation between cluster groups is less than the radius required to capture the events furthest from the center of the cluster.

The DBSCAN algorithm provides more flexibility to the shape of the clusters that it generates by operating on a threshold of density as opposed to a limit of influence within the clustering space. It estimates the density of a region by the number of neighbors to an object. Each object has its own circular area or spherical volume to determine its neighbors. The DBSCAN algorithm calculates the number of objects, N_{object}, in the data set of objects, D, within a radius, r, of

each object

$$N_{obj} = \{\text{object} \in D < r\}. \tag{3.38}$$

If the number of events is greater than a minimum value, $N_{obj} > N_{min}$, then it is assigned as a core point. Objects within its radius neighborhood are assigned to the core point's cluster. Next, the neighboring points are tested to be core points. To reduce computations for a full cluster, intermediate points are not tested to be a core point. If a tested neighboring point is a core point by meeting the minimum neighbor requirement, their neighbor objects are added to the cluster. Those that do no qualify as core points are labeled as border points and their neighbors are not added or tested to be a core point of the border point's cluster [26, 90]. Figure 3.25 demonstrates the difference between core, border, and noise points with a minimum number of objects within the radius threshold set at 3. The core points that meet the given threshold are marked by circles. The border points, marked by triangles, do not meet the threshold. Finally, the square noise point does not quality as a border or core point because it is outside the radius of every core point. The result is a cluster that is locally bound by a radius, but the full cluster of dense points can take on any shape.

I make one modification to the data before I apply the DBSCAN algorithm. While not strictly necessary, scaling the time axis to be better conditioned with the spatial axes assists in successful clustering output. DBSCAN still functions with a limiting radius. Events happening both close together in time and space set the maximum limit for the radius. The easiest way to choose this DBSCAN input parameter is through examination of the spatial distance between clusters in the 2 dimensional projection of Figure 3.26. With Dave's data sets, the mode cluster radius I apply with DBSCAN is 2 pixel lengths with 80% percent of data sets using this setting. The DBSCAN cluster settings are summarized in Table

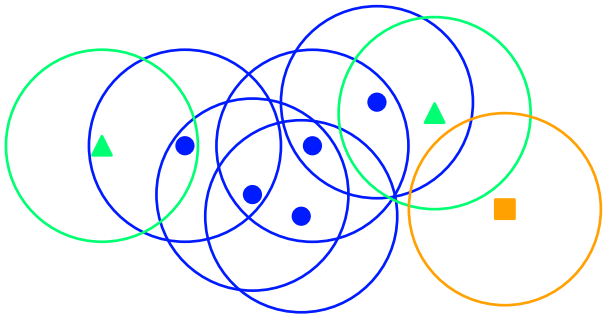


Figure 3.25: A 2D Illustration of density-based clustering. The core points, defined by having at least the threshold number of neighboring points within a radius, are illustrated as circles. Border points, marked as triangles, are neighbors of core points, but do not reach the limiting number of neighboring points within the radius. The square noise points are not neighbors of a core point and do not reach the threshold themselves.

3.3. These settings are attributable to the common modality in which these data sets are taken and are not necessarily optimal settings given different angular fields of view. After I choose a radial parameter, the time axis is scaled by

$$m = \frac{r_{max}}{t_{max}}. \tag{3.39}$$

The scale, m, makes the maximum time change selected by the user, t_{max}, from a previous core point to be within their chosen radius. Dividing the timestamp vector by that value will make the difference between time stamps equal to the t_{max} value equivalent to 1. To then scale it to the chosen radius, the vector is multiplied by the maximum radius, r_{max} for the DBSCAN algorithm. I conservatively apply an initial estimate of 8 seconds for t_{max} to this data to ensure capturing of the off event trails following an on event. Due to the low pass filter induced by the hardware, the response of the compared current to a change in induced current is not instantaneous. The movement to a new current value is initially rapid and then that rate of change slows as the value converges. This

behavior is observed with both increases and decreases in current. However, the logarithmic scale in which the current is compared yields more threshold changes in the slow return to the nominal value during a decrease in current. There are two factors driving this phenomenon. First, the change in actual current to yield a change in the logarithmic scale decreases exponentially as the current decreases. As a result, the possible threshold changes are more densely populated around the nominal value. Since the number of events is limited by the sensor's refractory period, the minimum time between events, the slow drop through the dense possible changes yields more events assuming the same setting for the on and off thresholds. The conservative 8 second setting is the applied cluster maximum time setting for all data sets as summarized in 3.3. This is attributed to my assessment of the 2 dimensional projection of the clusters,like that of Figure 3.26, being sufficient without need to adjust the 8 second nominal maximum time setting. However, the 2D projection does not expose the possible temporal issues. In fact, after observing individual clusters post processing, the trails of events on most pixels did not approach this t_{max} value. Being more aggressive with this value, a smaller t_{max} such as 1 or 2 seconds, will help reject tailing noise.

The final DBSCAN parameter is the number of neighboring objects that are within the radius before an object is considered a core point. This parameter is important because it helps separate event clusters that just happen to be close in space and time. The most common setting in Table 3.3 was 2 pixels when only considering the DBSCAN method. This is lower than the default method and might have been driven by the aggressive scaling of the temporal axis applied to every data set. It is important to be able to adjust this parameter for each data set, for not only temporal reasons, but because different lighting conditions

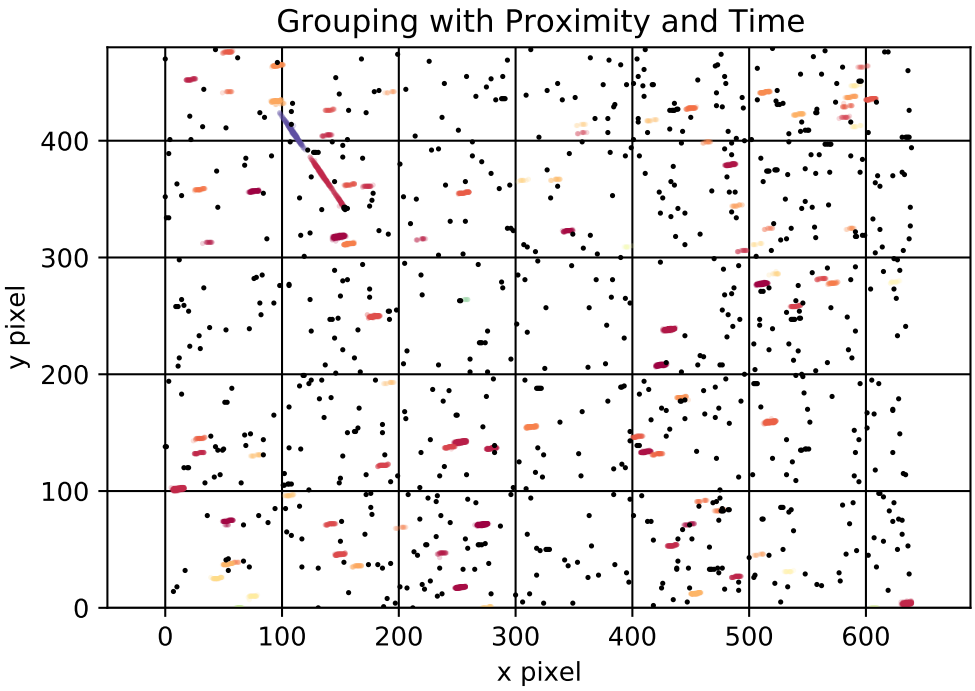


Figure 3.26: The final cluster output from the clustering algorithm mapped to different colors for each cluster. The black dots are un-clustered events assigned to noise. This 2 dimensional projection of the clusters provides a means to assess the clustering performance without needing to observe the thousands of events in 3 dimensional space. Note in this cluster output the star track is segmented into 2 separate clusters.

produce different noise densities and spread of events from real signals.

In addition to the rejection of possible clusters due to minimum number of events within a core point's radius, I enforce a minimum number of pixels and minimum number of events within a cluster outside of the formal DBSCAN algorithm. These two checks reduce clusters of noise events that were close enough in time and space to be grouped with DBSCAN from being assigned as a former cluster. The minimum number of pixels limit is dependent on the staring modality of collections causing the previously described motion of the observer. With this motion, the star and satellite consistently stimulate more

Table 3.3: Summary of the common cluster settings utilized to cluster the empirical event data.

	Maximum Radius	Minimum Points	Maximum Time	Minimum Pixels	Minimum Events	Cluster Method
Mode	4 pixels	3 points	8 seconds	3 pixels	3 events	Time Cluster Method
% at Mode	44.4%	60%	100%	73.4%	43.7%	96.0%
Default Values	4 pixels	3 points	8 seconds	4 pixels	8 events	3D DBSCAN
DBSCAN Mode	2 pixels	3 points	8 seconds	5 pixels	5 events	N/A
% at DBSCAN Mode	80%	60%	100%	30%	40%	N/A
Time Cluster Mode	4 pixels	N/A	8 seconds	3 pixels	3 events	N/A
% at Time Cluster Mode	45.9%	N/A	100%	75.6%	45.0%	N/A

than a minimum number of pixels. I assume clusters that are only on one pixel to be a hot pixel. All other pixel limited clusters are added to the general noise cluster. Since the minimum pixels parameter is dependent on imaging method and field of view, the reported settings in Table 3.3 should only be utilized after inspection of the data to be clustered.

The second external to the DBSCAN check is the minimum number of events. While this parameter may seem redundant to the minimum number of points, recall that the minimum number of points is only used in the determination of which points are core points. The minimum number of events is a check on a global number of events in a cluster. This theoretically should help reduce qualifying clusters with one or only a couple core points, that span enough pixels to pass the minimum pixel check, but may be too sparse to verify as a true cluster. In practice, I did not leverage this parameter frequently, as summarized in Table 3.3. The most common event minimum being equivalent to the minimum number of pixels will not weed out further potential clusters of noise. However, this remains a viable method to reduce the most uncertain clusters and will remain within the algorithm as an option.

Algorithm 6 DBSCAN Event Cluster Algorithm

```
Calculate the scaling factor and apply it to the timestamp vector
Run DBSCAN : clusters = DBSCAN(events, points_max, radius_min)
Apply cluster minimums
for Clusters do
  if pixels_cluster < pixels_min then
    if pixels_cluster < 1 then
      Add pixel to hot pixel list
    else
      Add cluster events to general noise list
    end if
  end if
  if events_cluster < events_min then
    Add cluster events to general noise list
  end if
end for
```

The DBSCAN algorithm works well when the data have adequate low-density areas in all three dimensions and no-low density areas within clusters. The algorithm parameters should be selected to leverage the low-density regions and not bridge the gaps between clusters. However, it only takes one dimension with insufficient separation to combine two unrelated signals or, conversely, too much separation to separate related signals. For example, consider an event at the end of one cluster near another cluster in space but later in time. If the nearby, but later, cluster provides enough events within the spherical radius to meet the minimum event requirement, the non-core point will be assigned as a core point to merge the two unrelated clusters of points. The opposite, the separation of two related clusters, is common in these data sets due to inconsistent satellite signals. If a satellite is tumbling, for example, the photon flux from the satellite's reflection is oscillatory and will result in clusters of events with significant separation in space and time. It is easiest to see this phenomenon when the satellite motion is starkly different from the stars in the field of view, as in Figure 3.26.

It is difficult to identify two improperly merged clusters from the 2 dimensional projection making it the more challenging of these two problems to solve. One way to work around this issue is stricter DBSCAN parameters that will produce more clusters overall. This solution will rely on a solution to the second issue to combine separated but clusters from the same source. Instead, I develop a second clustering method that leverages the sequential nature of the time series data to reduce the aforementioned error. This method, the proximity clustering method, also serves as a baseline for online clustering addressed in 6.1.1. Instead of evaluating the density of events within a spherical radius in a batch processing method where any events before or after an event in time can make a point a core point, the events are clustered only considering the events that come before them. Each event is matched to the closest event within a maximum prior time and within a radius in the spatial dimension. The region considered is, therefore, a cylinder as opposed to the spherical volume of the DBSCAN algorithm as depicted in Figure 3.27. I choose a cylindrical volume to aid capture of the trailing OFF events on a pixel after a bright source creates at least one ON event as there can be a delay in the potential OFF event generated as the event source's photons move off the pixel. This can cause simultaneous events from the same source but offset in space despite the source's influence moving to another pixel. The resulting ribbon of events is better served with the cylinder assumption to capture that temporal offset.

This algorithm's sequential method is outlined in Algorithm 7. Through this algorithm, I check each event for being within the cylindrical volume of its closest neighbor in time and space. If the event is within a volume, I add it to the previously established cluster. If it is not within range, the event establishes a new cluster. Once every event belongs to a grouping, groupings are discarded in

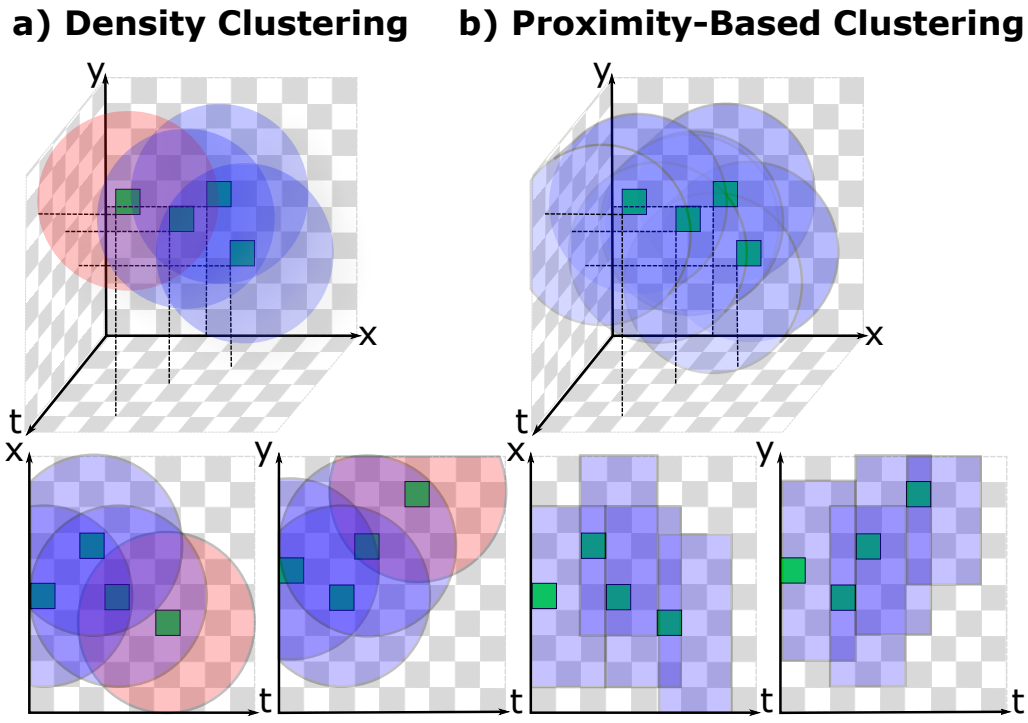


Figure 3.27: Visual comparison of the DBSCAN and the proximity-based clustering in 3-dimensions on the same 4 events with 2-dimensional projections of the x and y dimensions versus time. a) DBSCAN clustering relies on reaching a threshold minimum events within a radius of each point. The final event in the time dimension does not have two neighbors and, while a member of this cluster, it will be a border point. It will not connect other events to this cluster. b) Proximity-based clustering considers each event sequentially and assigns the new event to the closest past event inside the cylindrical envelope. The final event in this cluster can still group a subsequent event.

the same manner as those generated by the DBSCAN Algorithm for not having enough pixels or events within the cluster.

To compare the two clustering methods' performance, I run each data set through both clustering methods with the same cluster settings to assess both the duration of the methods and number of clusters produced as the total event number grows. Considering the cluster reduction method is the same between the two, these algorithms share the minimum pixels and minimum events pa-

Algorithm 7 Sequential Time Cluster Algorithm

```
for Events in Data Set do
  if First Event then
    Start new cluster
  else
    Check prior events
    if Any Events in Spatial Temporal Cylinder then
      Add to closest veent
    else
      Start new cluster
    end if
  end if
end for
for Clusters do
  if pixels_cluster < pixels_min then
    if pixels_cluster < 1 then
      Add pixel to hot pixel list
    else
      Add cluster events to general noise list
    end if
  end if
  if events_cluster < events_min then
    Add cluster events to general noise list
  end if
end for
```

rameters. I nominally set these values to 3 minimum pixels and 3 minimum events. These settings are the most common for these parameters in the final clustered data as summarized in Table 3.3. Both methods also apply a maximum time parameter, but in different ways. The DBSCAN algorithm scales the time axis by the maximum time parameter while the proximity based clustering uses the parameter to set the height of the cylindrical volume of interest. I set the maximum time to 4 seconds instead of the universally applied 8 seconds for that parameter in Table 3.3. I select this parameter purposefully to assess the impact of not viewing the data in 3 dimensions during the manual clustering which may have led to finer tuning of that parameter. Like the maximum time param-

eter, the maximum radius parameter is also shared between the two algorithms. The DBSCAN algorithm applies it spherically while the proximity-based clustering applies it only radially in the spatial dimensions. The most common setting of 4 pixels from Table 3.3 is chosen for this analysis. The only setting that is unique to DBSCAN, the minimum points for determination of core points, is set the most common setting in Table 3.3 as 3 neighbouring points. These results are discussed in Section 4.2.

The final result of either clustering method is a dictionary of events assigned to a cluster number. Figure 3.26 depicts this assignment by assigning a unique color to each cluster. By visual inspection, the clustering appears fairly successful. The noise events (shown in black) appear mostly solitary over the 20 second collection duration. A few patterns are discernible with the human eye, the most common of which is the uniform motion of the stars through the spatial plane. The common slope and duration of these highlighted lines builds confidence that the clustering method properly identified stars within the field of view and rejected other trends which only exist in the 2 dimensional projection.

Despite the demonstrated success, both algorithms have spatial limitations. In some cases, both methods require restrictive bounds to keep neighboring clusters from merging together which also results in multiple clusters associated with the same object. A great example of this phenomenon is in Figure 3.26. In this Figure, the satellite track is between (98,423) and (156,342), but is separated into two clusters. There are a couple of reasons for the gap to exist in the track of events. First, the observing optic has a layer of atmosphere distorting the incoming energy. This distortion can prevent enough photons from reaching a single pixel to enable it to register an event from dim stars and satellites. The physical

spreading and movement of the light aberrations are discussed in greater detail in the propagation portion of the simulation in Section 3.2. Another reason for gaps in time and space, likely the reason for the gap in Figure 3.26, is the geometry of a satellite does not yield constant incoming flux of photons. The sharp and inconsistent surface properties and potential rotation make it likely to have fluctuations in the incident energy. Light curve examination is another field of SDA study that exploits the changes in flux to identify satellites [50, 20]. While I do not delve into the possibility of satellite characterization through the light curve analysis method, there is potential for those concepts to extend the binary classification between satellites and stars discussed in Chapter 6.2. Regardless of the reason, if a gap of events exists in time and space and the clustering algorithm employed is sensitive to that gap width, then what should be a single cluster is split into multiple clusters.

To solve this issue and combine clusters without manual assignment, I exploit the linear nature of the grouped data as seen in Figure 3.24. Due to the short collection period and small field of view, the true satellite and star detections in the analyzed data set appear linear on the focal plane projection. I leverage the linearity of the data to automatically combine separate but co-linear clusters by way of a 2 dimensional Hough transform which finds best fit lines in a data set. Typically a best fit line is defined using the standard equation, $y = mx + b$, where the the slope, m, is undefined for a vertical line. To avoid any singularities, the Hough transform defines lines in the (x, y) plane with an intercepting line from the origin to the line being defined that intercepts it at a right angle as depicted in Figure 3.28. The intercepting line uses an angle, θ, and radius, r, to define itself. These parameters translate the original $y = mx+b$ equation to the form $y = \tan(\theta + \frac{\pi}{2})x + r\sin(\theta) - r\cos(\theta)\tan(\theta + \frac{\pi}{2})$. Therefore, the Hough

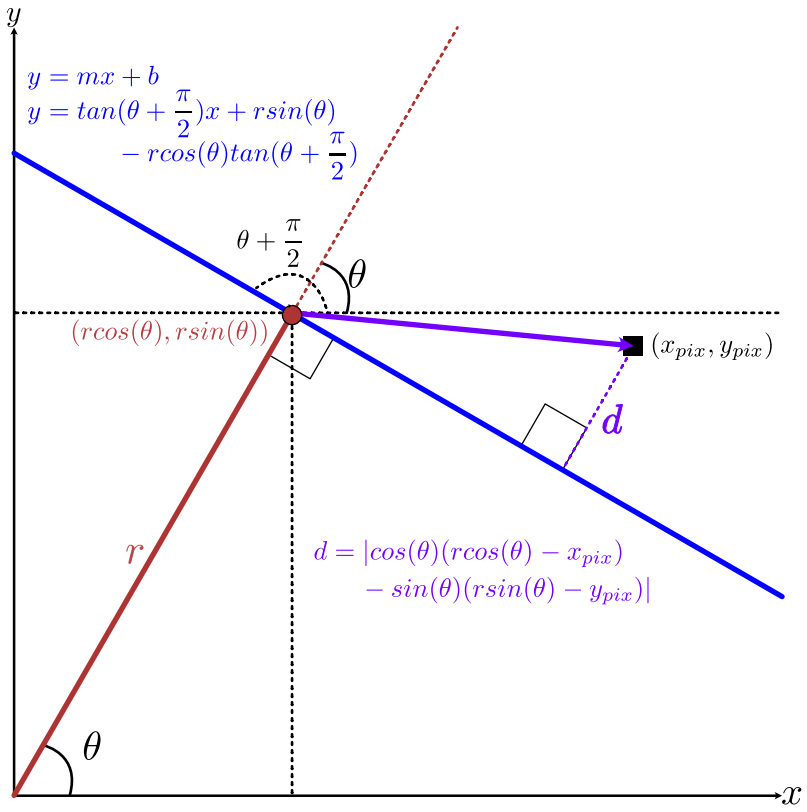


Figure 3.28: The Hough space is defined by an intersecting line drawn from the origin to the line being defined. It intersects the line being defined at a right angle. The equation defining the line can be written in the Hough parameterization. Any pixel's shortest distance to the line will also be a right angle from the line. Therefore, that distance is a projection of the vector between the point of intersection of the original two lines to the pixel back onto the original intersection line. Using the dot product definition of the projection, the distance calculation reduces to equation d.

space is parameterized by a range of angles and radii that each describe a different line. The points in the data, anywhere in an image not set to a value of 0, essentially "vote" on the line candidates by increasing the Hough space value for any line that contains it. Peaks form for lines that contain a higher amount of the data in the image. Applying a threshold to the peak values selects a limited number of the strongest line definitions for the data set [24, 91].

I am not the first to implement the Hough transform technique on event-

based data. The transform is applied to extract the linear trends and angles of event-based data in a related effort. Bagchi and Chin [2] prove linear trends of stars in event-based data are extractable through a progressive Hough transform. The progressive Hough transform uses a random subset of the data to speed up computation and still provides high confidence in the resulting line [91]. Bagchi and Chin find that the lines and angles from the progressive Hough transform can identify star tracks and determine relative rotation of the observing body as the angles of those tracks change over time [2]. I do not apply the same techniques as Bagchi and Chin for a couple of reasons. First, in my data sets, the observer is not rotating, the field of view is relatively small, and the time of the collection is relatively short, so the data set tracks are essentially linear. I am not trying to capture online rotation as done in Bagchi and Chin. Since I am not any time restrictions for an online algorithm, I avoid randomly selecting a subset of data to progressively evaluate the Hough transform. Instead, I apply the Hough transform to the entirety of each data set flattened into a 2 dimensional array. Due to the large batch of flattened data being processed disregards the temporal dimension, the Hough transform lines likely pass through noise and other clusters on the same linear trend, but separated by time and space. Therefore, I apply careful additional assessment to the Hough transform output when assigning events to the final line clusters to avoid undoing the noise rejection and clustering achieved via the initially applied clustering methods.

As outlined in Algorithm 8, I apply the Hough Transform to a flattened array similar to that of the final event displayed in Figure 3.24. If a pixel has an event at any point in the dataset, the array is given a value of 1. I run this array through SciKit-Image Hough transform [91]. This is a two step process. First, I generate a Hough space using a given range of angular values for the

Algorithm 8 Hough Transform Cluster Management

Create Binary Array
for Pixels in Array **do**
  if Event on Pixel **then**
    Pixel in array = 1
  else
    Pixel in array = 0
  end if
end for
Run Hough Transform
for Pixels in Array with Events **do**
  Identify distance, d, to the closest line
  if $d < d_{max}$ **then**
    Assign pixel to closest line
  else
    Label pixel as noise
  end if
end for
for Events in Data Set **do**
  if Event is in Prior Cluster Hot Pixel Group **then**
    Update line assignment to the hot pixel label
  else if Event is in a Prior Cluster **then**
    Preserve the prior cluster grouping
    Count up most common line label in the cluster, $label_{count}$
    if $label_{count} = label_{total}$ **and** $label_{mode} \neq label_{noise}$ **and** $pixels_{cluster} = pixels_{Hough}$ **then**
      Line label is unique to this cluster and does not need to be modified
    end if
  else if Event is not in Prior Cluster **then**
    $label_{count} = 1$
  end if
  if $label_{count} \geq label_{total} * pixel_{\%}$ **then**
    Acceptable agreement with most common line label
    Find next cluster in line label
    Calculate spatial distance to nearest event, d_{next}, and time to event, t_{next}
    if $d_{next} \leq d_{merge}$ **and** $t_{next} \leq t_{merge}$ **then**
      Merge two neighbouring clusters
    else
      Run Hough Transform on cluster
    end if
  else
    Unacceptable agreement with most common line label
    Run Hough Transform on cluster
  end if
end for

Table 3.4: Summary of the common Hough settings utilized to finalize clusters of the empirical event data.

	Hough Threshold	Minimum Distance	Minimum Angle	Maximum Distance	Pixel %	Cluster Distance	Maximum Time
Mode	0.9	5 pixels	10 degrees	2 pixels	0.5	8 pixels	8 seconds
% at Mode	82.1%	60.3%	48.8%	95.6%	99.2%	57.9%	100%

aforementioned polar transformation. In this case, I apply no restriction to the definition of possible slope definitions, with the possible θ values ranging from $\frac{\pi}{2}$ to $-\frac{\pi}{2}$. After defining the Hough space, I limit the considered lines by defining a lower bound threshold between 0 and 1 and multiplying by the maximum Hough space value to define the lower bound in Hough space. The threshold value listed in Table 3.4, indicates that I use the top 10% of lines from the Hough space on 82% of the data sets. The considered lines are also limited by the minimum distance and minimum angle between output lines in the Hough space to ensure a variety of line solutions are offered. I vary these settings more greatly depending on the data set. The respective 5 pixels and 10 degrees in Table 3.4 are the default settings, so I adjust 39.7% of the minimum distances and 48.8% of the minimum angles to yield a favorable final grouping result.

Next, given the angles and closest radial distances to the origin of the lines identified by the Hough transform, I assign each pixel with events in the array, $(x_{pix}, y_{pix)}$, to the closest line by calculating the shortest distance to each line. The vector describing the closest distance from a point to a line is perpendicular to the line. Since the vector from the Hough transform is also perpendicular to the line, I calculate the distance as the dot product of the vector between the Hough tranform vector point of intersection with the line $(r\cos(\theta), r\sin(\theta))$ to the pixel in question projected onto the Hough transform vector. The dot product definition

of this distance

$$d = |\cos(\theta)(r\cos(\theta) - x_{pix}) - \sin(\theta)(r\sin(\theta) - y_{pix})| \tag{3.40}$$

simplifies due to the perpendicular nature of each vector. If the minimum distance to one of the lines is below the maximum distance, I associate the pixel with the minimum distance line in a dictionary. I apply a nominal maximum distance of 2 pixels on 95.6% of the data sets which I rarely change because the next couple of steps maintaining the cluster information from the DBSCAN or proximity-based clustering. I do not need every event in a cluster to associate with a line or the same line.

To incorporate the prior cluster information, I assume that the data clustered together in either the DBSCAN or proximity based clustering should remain together. For each event in the data set, I first check whether the event is assigned as a hot pixel from the clustering algorithms. If so, I leave that label intact. If the event is part of a prior cluster, I count up the line labels for each pixel in the associated cluster. If the count of the most common label, $label_{count}$, is equivalent to the total number of labels, $label_{total}$, all the pixels in that cluster are within the acceptable distance and closest to the same line. If the total pixel count in the cluster, $pixels_{cluster}$, is equivalent to the total pixel count in the Hough line, $pixels_{Hough}$, then it is the only cluster associated with that line. As long as the most common label, $label_{mode}$, is not the label for noise events, $label_{noise}$, and meets the two prior conditions, then no additional modifications are made to the line label and events within it.

If one of those conditions isn't met, then I treat the number of the most common label as a voting metric. If the count of the label mode is greater than a chosen percentage, $pixel_{\%}$, of the total labels, then I assume enough pixels are

associated with the Hough line for the line to be an acceptable estimate of the cluster's slope in the spatial dimension. For 99.2% of the data sets, I only require 50% of the pixels to be assigned to the Hough line as summarized in Table 3.4. Being more relaxed with this metric allows for faster run times because if the cluster does not reach this threshold I take an additional Hough transform to define a more appropriate line. The relaxed approach has less refined final θ and r parameters which may be less useful for future applications of the line information. Currently this is not an issue, as I do not apply the batch Hough line information in any algorithm.

Given an event and its cluster make it through the voting metric, I check if the next event along the Hough line is within the chosen maximum spatial distance and absolute time gap thresholds, d_{merge} and t_{merge} respectively. If the next event along the line is within the proximity window, the event or cluster of events is merged with the cluster currently under evaluation. Otherwise, if the cluster is not close enough to another cluster, another cluster is on the line, and the event is not clustered as noise initially, I run the Hough transform on only the current cluster of data to define its own line. This leaves the other clusters on the line with the possibility of being combined. This step is the one that truly leverages the Hough transform to only consider linearly associated events to merge clusters. Through my leveraging of the Hough transform, I overcome the issue of larger gaps between events from the same source than gaps between different sources in the data. In general, the cluster distance in Table 3.4 is a larger value than the maximum radius in Table 3.3 used in the clustering algorithms, with 57.9% being at 8 pixels. Again, I do not manipulate the maximum absolute time gap in Table 3.4. Clearly, 8 seconds is sufficient to bridge the gaps I see within the 2 dimensional projection of the data. The linear

restrictions of the Hough line reduce the need to decrease the 8 second value.

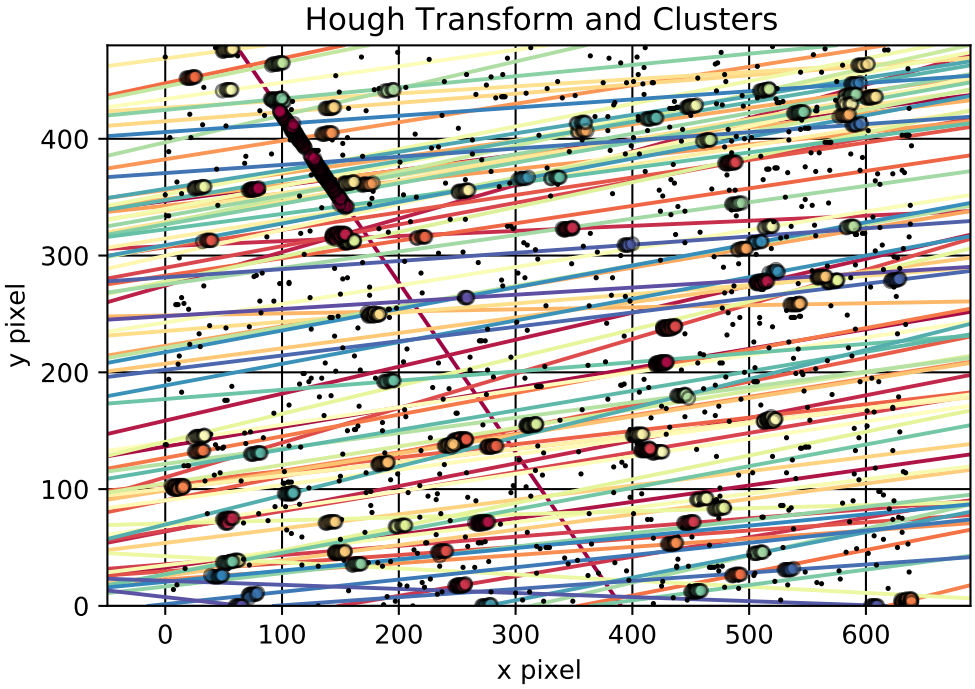


Figure 3.29: The cluster output is adjusted with a Hough Transform. Co-linear, but separate clusters of the satellite track in the upper left-hand corner are combined without merging with the two clusters. Lines are drawn for each identified solution out of the Hough transform. Considering the similar trajectory of the star sources, the drawn lines simplifies identification of satellite clusters or incorrectly clustered noise.

After the co-linear combinations are complete, I assign each cluster to the classes of satellite, star, noise, or hot pixel. I manually complete this selection with line labels plotted over the 2 dimensional projection. The final result is a combination of clusters as seen in Figure 3.29 for the two satellite grouping of events originally identified as likely being from the same source in Figure 3.26 through visual inspection.

3.4.4 Grouping Comparison

I developed the clustering methods discussed in section 3.4.3 with the explicit intent to leverage the time information encoded with the data. This section covers the methodology I apply in order to compare the classic approach of identifying objects in Section 3.4.2 to the methods developed in Section 3.4.3.

Dr. Monet's data is an invaluable resource that I apply for model verification, analyze to identify group characteristics and inspire algorithms, and employ in algorithm testing. However, the data set does have some limitations. In particular, I cannot reverse engineer, with 100 % certainty, Dr. Monet's processing to determine how he identified satellite events. I also do not have attribution for each event's generation outside of the satellite-specific events. Therefore, it becomes a challenge to compare his object identification to my own. I must apply additional processing to the data in order to do a comparison.

Taking a step back to examine the data provided through the processing outlined in Figure 3.20, besides having the raw list of events for each data set, I have a list of the stars in each time slice with one centered pixel assigned to each star source. I also have a more detailed list of events for the satellite. At the simplest level to prepare the data for comparison, I can apply all events within the assigned star pixel within each image slice to a star label and the detailed satellite list to a satellite label. In a 3 second window, however, I expect most stars will stimulate more than one pixel. This is a function of the staring collection method inducing motion with the rotation of the Earth, the arc second resolution of each pixel, and the signal magnitude and intersection of pixels during the time period. Applying only events of pixels assigned to the center of the group in each slice will reduce the number of events attributed to that

source and, consequently, will yield more noise events.

I want to produce the most accurate data labeling possible given the traditional processing method and limited information about the groups produced by the SExtractor processing. Therefore, instead of only attributing the one pixel per star per slice assigned through the processing, I attribute multiple pixels by averaging the rate of change in the 2-dimensional plane between slices. Using the initial x and y pixel location of an identified star and the slope, I set a reference location of each star in the first time slice. This location can be inside or outside the frame, it is just important that the estimated locations are approximately at the same point in time. With this definition, I expect the reference to be approximately halfway through the duration of the first frame. To protect the remaining temporal fidelity after the compression into 3 second slices, I process each time slice separately using the average change rate. I first extract events bounded by each 3 second slice and filter for unique x and y pixel locations. Then I define a line for each star bounded within the timestamps to determine the pixels the star should pass through during a window. I associate all events on those pixels within that window with a star and keep a running total of star events. With these metrics, I compare the number of events associated with star and satellites in Section 4.2.

CHAPTER 4

SIMULATION ANALYSIS

4.1 Noise Rate Verification

As I describe in Section 3.3.6, I propose two new methods to simulate noise generated through two different mechanisms of the circuitry. By leveraging the electron flux information I gain from a photon-level front-end model, I attempt to create a dark shot noise and high frequency noise models that have greater realism as opposed to artificial noise injection. The reason for attempting higher fidelity noise modeling is two fold: it provides insight into the underlying mechanics and sensor parameters that are difficult to measure and, if a more accurate noise event signatures in the output event stream are possible, I have a powerful tool to improve algorithm development such as that discussed in Section 6. I address the two new methods' performance, the rolling Poisson method and the high frequency tuned Gaussian, in Sections 4.1.1 and 4.1 respectively.

4.1.1 Dark Shot Noise Analysis

Dark shot noise should be the dominant producer of noise in low-flux environments such as the dark regions between sources in SDA. Therefore, producing consistent and realistic dark shot noise is a priority for the simulation. While random injection of events using a Poisson distribution yields the correct rate of events, a dark shot noise rate must be provided for each simulation. Providing a rate is not a trivial task. First, the influence of dark shot noise is tied to the intensity of light on a pixel with the highest rates on pixels with only the

dark current flowing. Second, the dark shot noise is temperature dependent as shown in Figure 3.15. In the SDA data sets I analyze in this dissertation, which all use the same 3rd Generation Prophesee sensor with the same sensor bias parameters, the dark shot noise varies over two orders of magnitude. Therefore, I cannot choose a static setting for the dark current or a rate parameter and expect it to produce a relatively realistic event stream. I, instead, relate the temperature at the time of the observation to the dark current and apply a rolling Poisson method of sampling that current throughout the simulation as I describe in Section 3.3.6. Figure 4.1 demonstrates the difference between standard Poisson sampling at each time step and maintaining the inertia of the samples of 1 second of simulation time. This simulation is only on one pixel with a 2 millisecond time step, 1 Hz low-pass corner frequency, and a total simulated time of 19.5 seconds. In this example, the dark current at each simulation step varies above and below the nominal dark current of 1 fA. Therefore, the summation of the Poisson samples yields something close to the proper expected value. In this one pixel example, the maximum variation off the nominal log dark current due to a rolling Poisson method is 49.2% less than the maximum variation from the normal Poisson sampling. While this may seem concerning because the goal is to induce occasions when a threshold change occurs on the log scale, the inverse is true on the low-passed maximum variation due to the enforced inertia. After the low-pass filter, the standard Poisson sampling method maximum is 81.2% less than the rolling Poisson method. It is also important to understand the resulting density of current values away from the nominal dark current. Assuming normality, one standard deviation from the nominal log current value is approximately 16% less than with the traditional sampling method. Again, this statistic inverts after the low-pass filter with an 89.6% reduction in the standard

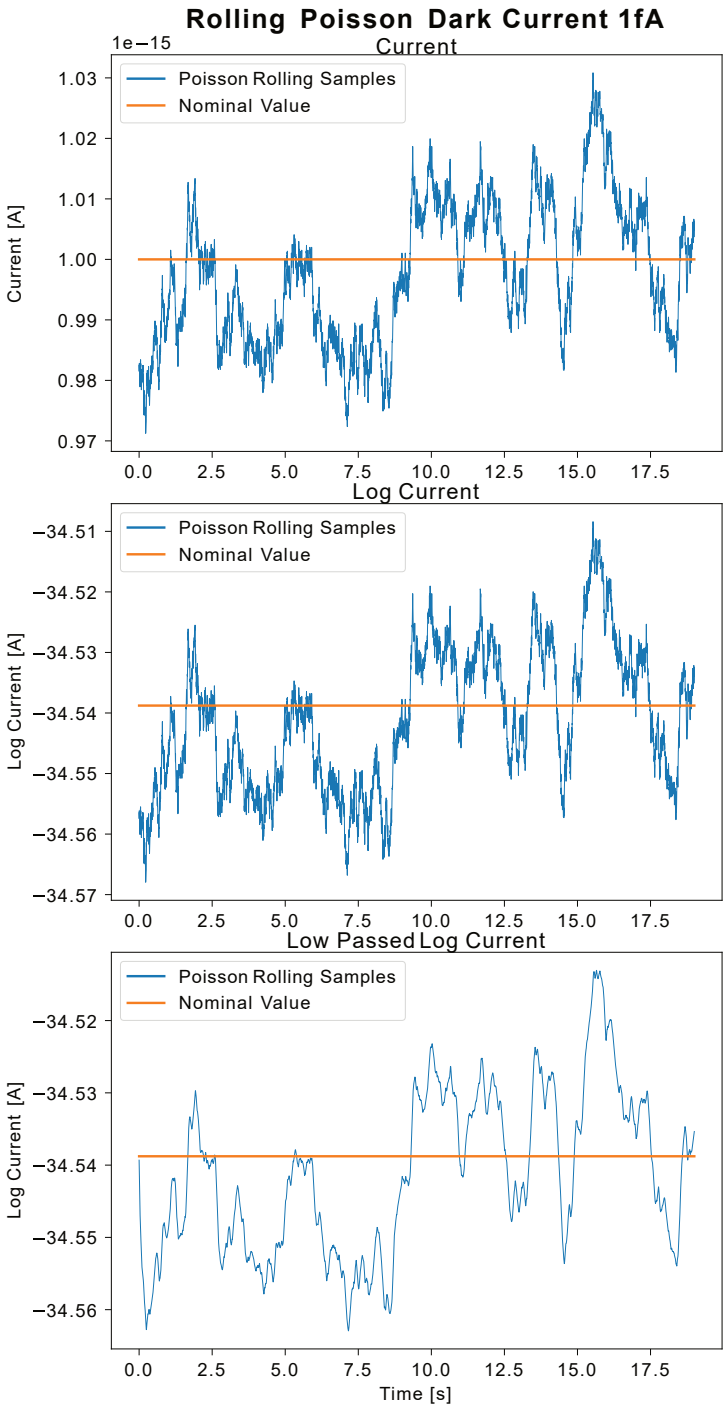


Figure 4.1: The rolling Poisson method in practice ensures the deviation of the current from the nominal current value has enough inertia to allow the deviations through the low pass filter. The loss in the current standard deviation is only 10% as opposed to the 93% loss for the simulation run with standard Poisson sampling.

deviation between the rolling Poisson and traditional methods. This follows the order of magnitude lost in the current variation that Figure 3.13 highlights. In fact, comparing the change in the standard deviation before and after the low-pass filter, the traditional Poisson method has a 92.1% loss and the rolling Poisson method as only a 10.4% loss. These statistics confirm the efficacy of the inertia theory behind the rolling Poisson method to create deviations far enough from the dark current to yield events. I can effectively produce the same inertia method with lower rate sampling of either a Poisson or Gaussian distribution. At the larger time steps with higher numbers of electron flow rate, these distributions are effectively equivalent. Figure 4.2 demonstrates the effectiveness of sampling at a lower rate. In this 20 second simulation, I maintain the dark current, corner frequency, and simulation time step length. I sample a Gaussian at an interval twice the corner frequency with the full electron per second rate value, not one subdivided for the time step, as in the rolling Poisson method. Because the Poisson distribution is sampled from the full electrons per second rate, the maximum deviation from the nominal log current is only 6.6% less for the low-rate Gaussian method when compared with the traditional Poisson sampling method. Since it also has some inertia, the low-rate Gaussian only experiences 19.4% decrease in its standard deviation during the low-pass filtering. Between the more comparable initial maximum deviation and the medium performance on maintaining value through the filter, the low-rate Gaussian out performs the rolling Poisson method on its maximum variation value after the low-pass filter. The traditional method has a 92.3% decrease in the maximum variation value when compared to the low-rate Gaussian.

Despite the perceived advantages from a maximum variation standpoint, the low-rate Gaussian has some implementation issues. First, the amount of de-

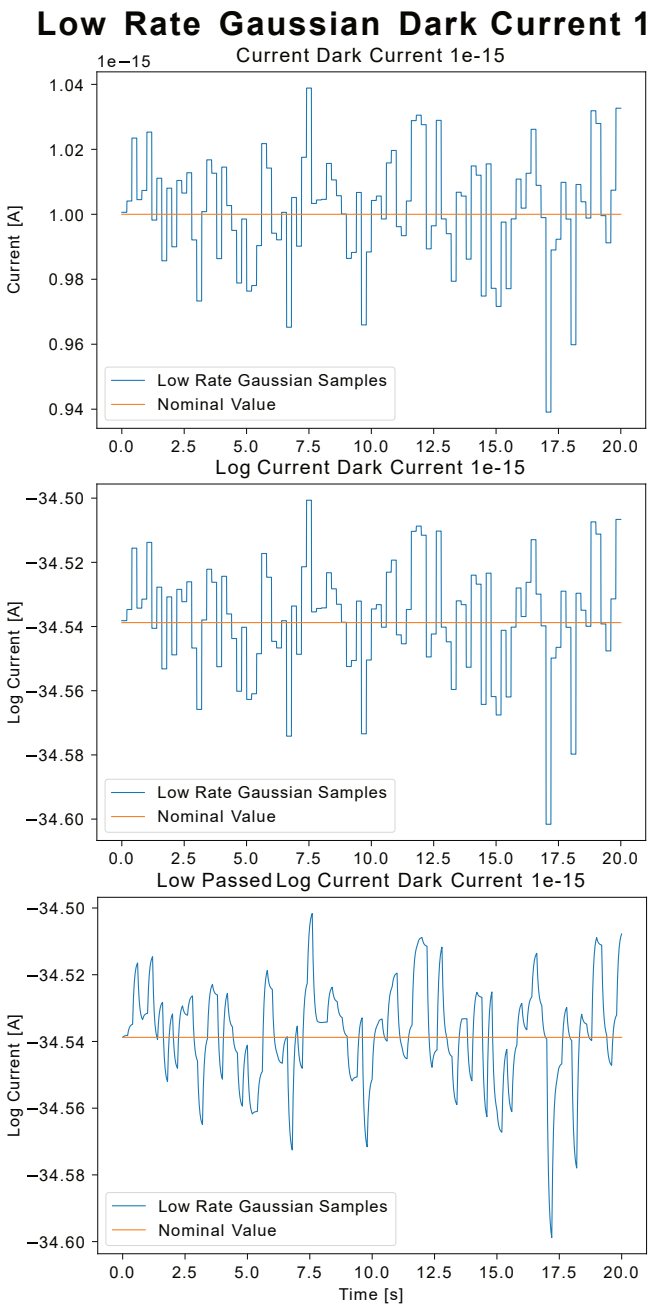


Figure 4.2: Reducing the random sampling frequency of a Gaussian or Poisson distribution also creates the inertia required to pass the deviation in the dark current through the low-pass filter. Due to the use of the total electron per second rate and the same initial seed, the maximum deviation from the dark current is closer to the traditional Poisson distribution. While the inertia maintains the maximum variation advantage through the low-pass filter, the unique slopes in the final current graph highlight the challenge in choosing the right sampling and the need to adjust that sampling at different induced current levels may make this method less tenable.

viation achieved is highly dependent on the rate of sampling. One extreme is sampling at every time step as seen with the traditional method and in that case the low-pass filter effectively filters out the deviations. At the other extreme the filter perfectly follows the noise samples when the sampling rate is at half the frequency of the corner frequency. Perfect following is not necessarily desirable. As in Figure 4.2 where the noise sampling is twice the corner frequency value, the low-pass filtered current displays the classical shark fin shape, initial steep changes after choosing a new set point that levels out as that point is reached. This behavior may lead to event trigger times being disproportionally close to new sample times creating unwanted frequencies in the dark shot noise generation. The correct noise sampling frequency is not initially evident to produce realistic event time stamps. In addition to the potential unrealistic trigger times, the corner frequency changing as a function of induced current on each pixel must also be considered. It is necessary to update the noise sampling frequency as a function of the corner frequency to maintain a particular performance in the low-passed dark current variation. It is a challenge to implement different sampling rates across the array with a fixed step size simulation.

While the previous simulations capture the effectiveness of inertia in the dark current methods, the number of current samples is too small for the methods with inertia to effectively evaluate if the resulting distributions resemble Gaussian distributions. Increasing the number of time steps sampled to 100,000, Figure 4.3 depicting the traditional Poisson sampling method serves as the baseline distribution expectation besides the scale on the low-passed distribution. Given the sampling rate is in the 1000s of electrons per second, the Poisson sampling creates a Gaussian distribution of current values. The co-location of the mean and median and the Fisher excess kurtosis and skew metrics being close

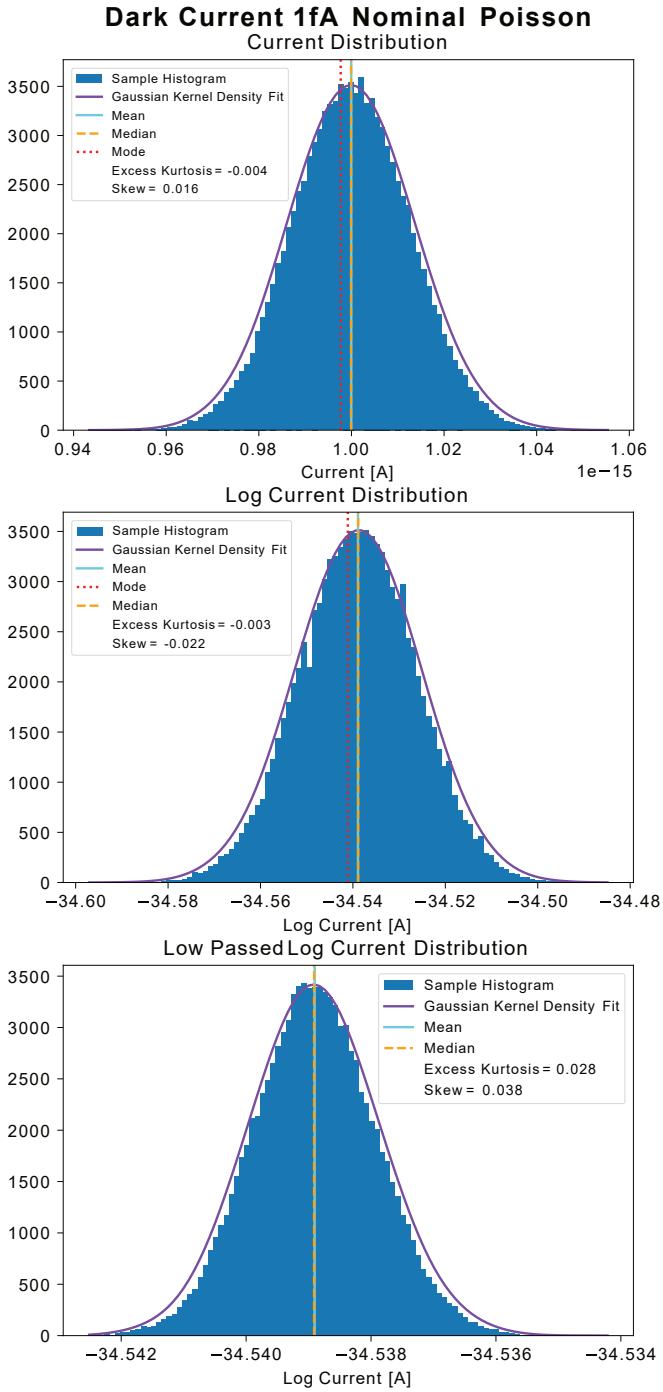


Figure 4.3: Traditional Poisson current sampling over 100,000 time steps and a rate parameter in the 1000s of electrons per second creates an effectively Gaussian distribution; The mean and median are co-located at the actual dark current value of 1 fA and the Fischer kurtosis and skew metrics are nearly 0. The kurtosis increases slightly after the low-pass filter, becoming slightly more leptokurtic. Due to the low-pass filter the magnitude of the standard deviation decreases 92.2% effectively matching the simulation with fewer samples.

to 0 confirm the Gaussian distribution. The larger number of samples minimally affects the loss statistics for the traditional Gaussian. Both the standard deviation and the maximum variation reduce by 92% through the application of the low pass filter as they did with the shorter simulation. The effective concentration at values closer to the mean making it more leptokurtic.

As Figure 4.4 indicates, with enough samples, the rolling Poisson method distribution also has the hallmarks of a Gaussian distribution. The mean and the median are co-located and the Fischer kurtosis and skew are on the same order of magnitude as the traditional Poisson method. For further assurance, I check the normality of the low passed rolling Poisson distribution using the Kolmogorov-Smirnov test for goodness of fit [22, 70]. In this test, I check for agreement with the null hypothesis, the distribution is normal. Using a 95% confidence interval, the resulting p-value of 0.7 is greater than 0.05 and, therefore, the rolling Poisson distribution is consistent with the null hypothesis. Now that I've verified the normality of the rolling Poisson distribution, I compare the standard deviation of the current with the traditional Poisson sampling method's standard deviation. With more samples, the difference reduces in the standard deviations reduces from 16% to only 5.4%. While the differences in the current standard deviations tighten with more samples, the distribution also highlights the rolling Poisson's co-located mean and median are slightly displaced from the nominal 1 fA at a value of approximately 9.98E-16A, a 0.004% difference from the nominal. This displacement is a result of the subdivided Poisson rate summation. At the 2 millisecond simulation time, the 1000s of electrons per second becomes single digit rates of electrons per 2 milliseconds, so the distribution acts less Gaussian and more Poisson. The larger sample size of this second test not only allows for distribution testing but provides more in-

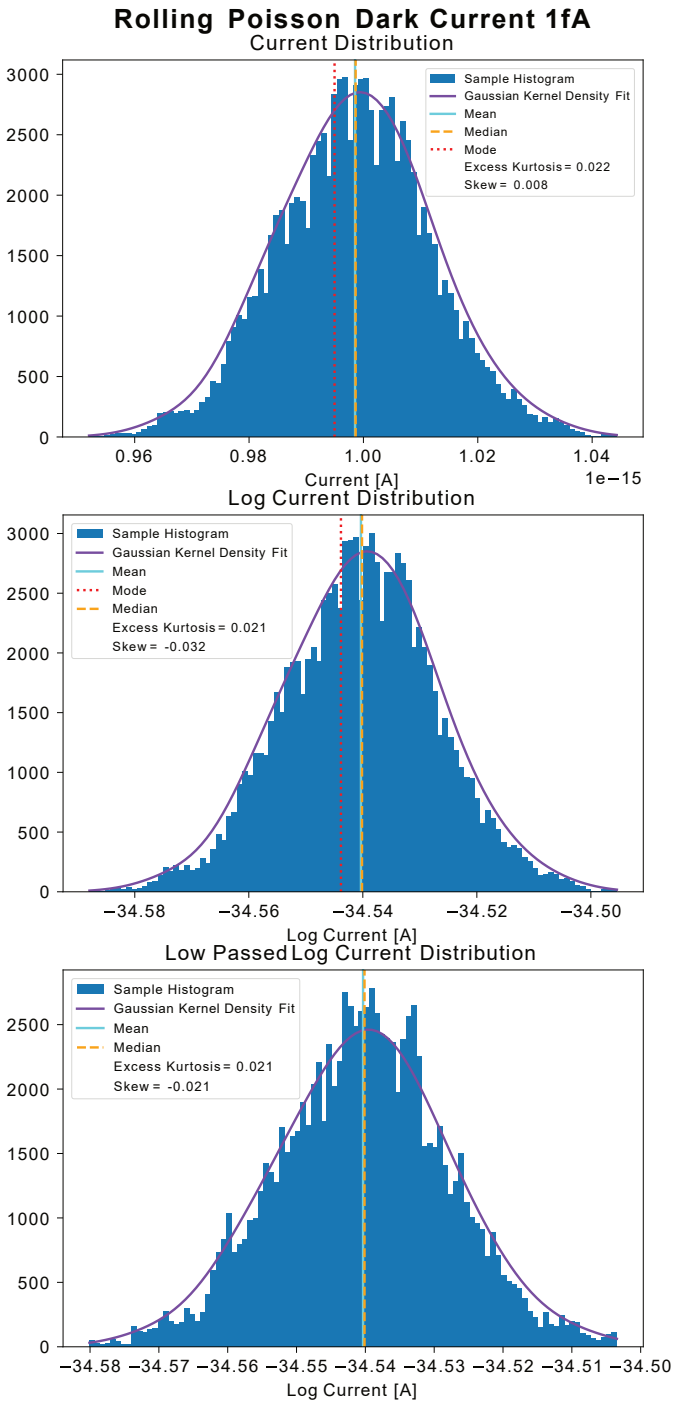


Figure 4.4: A more cohesive distribution of current values is visible after sampling 100,000 time steps with the rolling Poisson distribution. The resulting mean is 9.98E-16, 0.004% away from the nominal dark current of 1 fA. Despite this slight offset, the distribution is promisingly Gaussian; the mean and median are co-located and the kurtosis and skew metrics remain consistent through the low-pass filter at values close to 0. Like the traditional Gaussian sampling the final distribution is slightly leptokurtic.

sight on the expected affect of the low-pass filter on the dark current values. In particular, the reduction in the standard deviation, capturing the overall affect of the low-pass filter, reduces to 7.5% from 10.4% with the larger sample size. In total, the rolling Poisson maintains the expected normality while enabling the magnitude of the variance to pass through the low pass filter verifying the method provides the desired affect on the simulation.

Even though the low-rate Gaussian time series demonstrates some glaring downsides from the potential to induce unintended frequencies in the induced dark shot noise events and provides additional complexity for implementation, Figure 4.2 provides further evidence that the method is less suitable than my proposed rolling Poisson method for modeling the dark shot noise. Like the rolling Poisson method, the low-rate Gaussian's resultant distribution is clearer with more samples and sets expectations on how the distribution will change. Like the rolling Poisson distribution, with more samples the low-rate Gaussian's standard deviation loss through the low-pass filter drops from 19.5% to 15.9%. However, the low-rate Gaussian demonstrates the furthest deviation from a normal distribution with its kurtosis and skew values being double those of the traditional and rolling Poisson methods. The distribution, both before and after the low-pass filter, has negative kurtosis, indicating a platykurtic distribution as opposed to the leptokurtic distributions of the other two methods. The broader, platykurtic, distribution is a direct result of the low-rate sampling and values held over a set sampling time. The fixed time resampling intervals allows values further from the mean, sampled less often, to have the same number of current measurements as those closer to the mean; in this case, I apply 250 samples for each Gaussian draw. The binning in the current and log current histograms depicts added weight through this sampling method to samples further from

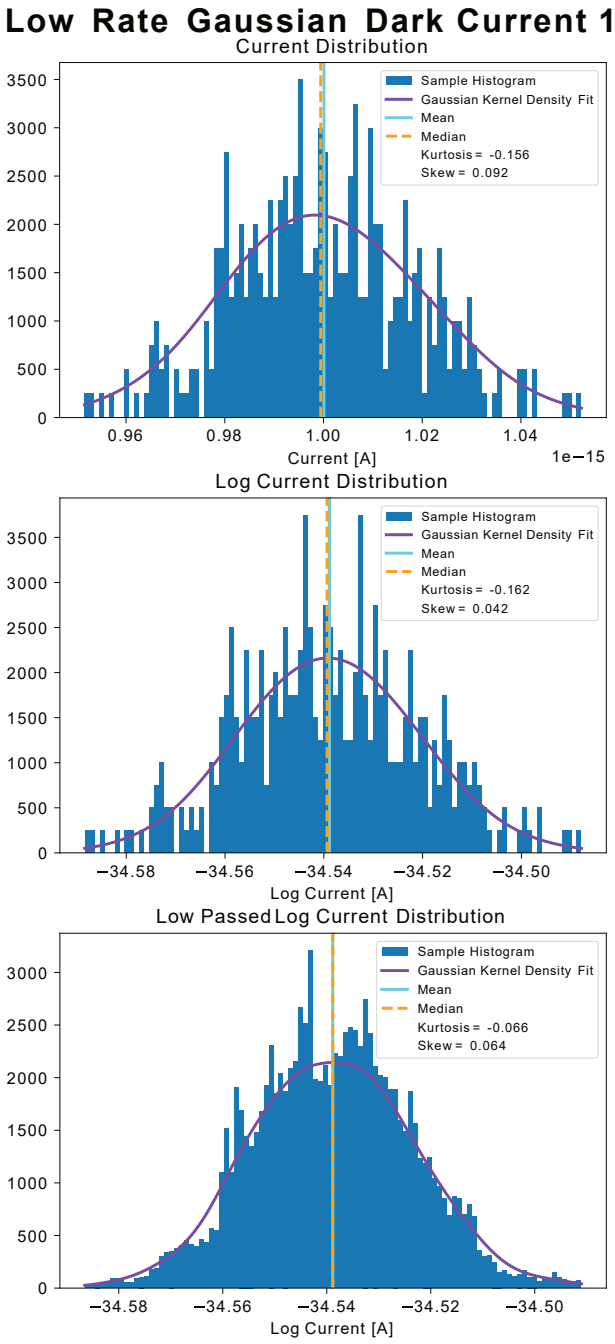


Figure 4.5: The low-rate Gaussian at 1000000 samples is the least cohesive distribution when displaying the samples as a histogram, particularly for the non-low passed current. Due to the constant settings for 1/2 second intervals, many of the values before the low pass filter belong to the same bin. Since each draw, no matter the deviation, holds its value for this period of time, weight in the resulting distribution is distributed further from the mean. The negative kurtosis also indicates the broadening of the distribution as it is slightly negative indicating that it is platyukurtic.

the nominal dark current mean. The weighting also throws the median slightly to the left of the mean, matching the positively measured skewness. It's reasonable to assume that more samples will improve the kurtosis and skew characteristics as more samples are taken closer to and balanced above and below the mean. Interestingly, the low-pass filter smooths between the larger bins creating a smoother histogram that is less broad since less current timestamps are attributed to maximum values of the variations. It follows that the less the shark fin shape in Figure 4.2 appears, the less current values will remain at the furthest out points and the more the distribution will trend towards the leptokurtic Poisson distributions. This could be done by setting the Gaussian sampling rate to the suspected rise time, but that would be dependent on the cutoff frequency and, therefore, difficult to implement as previously described.

While I did not have the resources to take empirical measurements of the EVS dark current to assess the behavior over time and the distribution of current values it creates, the traditional Poisson sampling method, commonly used to model circuit level noise, serves as a baseline to evaluate the distributions created by the current inertia maintaining rolling Poisson and low-rate Gaussian methods. The rolling Poisson method retains the normality of the traditional Poisson sampling method while maintaining the magnitude of variation required to induce events. The low-rate Gaussian method proves to be another way to maintain the variation in the dark current through the low-pass filter. However, the method's statistics prove highly dependent on the sampling rate of the Gaussian distribution. Consequently, the ideal sampling is dependent on the moving corner frequency which is difficult to implement with a constant step size. Because of these downsides and the relative similarities of the rolling Poisson distribution to the traditional method, the rolling Poisson method is the

preferred method to model dark shot noise in this simulation going forward.

Dark Current Variation

At a pixel level, the rolling Poisson method provides the desired effect of maintaining the dark current variation through the low-pass filter with a Poisson sampling method. One additional benefit of this method is that there are only two parameters that influence the dark shot noise event generation rate: the dark current and the threshold values. No tuning of a sampling rate is needed.

Taking a closer look at the dark current parameter, the dark current value determines the electron rate that defines the sampled Poisson distribution. At lower levels of dark current, the number of time steps without any electrons leaking between the photodiode layers increases. This increase drives the rolling Poisson summation further from the nominal value. Note, the comparison to generate an event is a ratio of the actual current and the memorized current, presumably around the dark current value. Therefore, I expect as the dark current value decreases, the amount of current variation required to generate an event decreases. I purely vary the dark current values for a set threshold of 0.1 on a 480 by 640 array during a 20 second simulation in Figure 4.6. The compounding factors of less variation occurring and more variation required to generate an event produces a quadratic drop off in the noise event rate as the dark current increases.

As I discuss in Section 3.3.6, the dark current through the photodiode varies with temperature and the spread in the temperature to noise event rate relationship only requires a linear mapping between the noise event rate and the

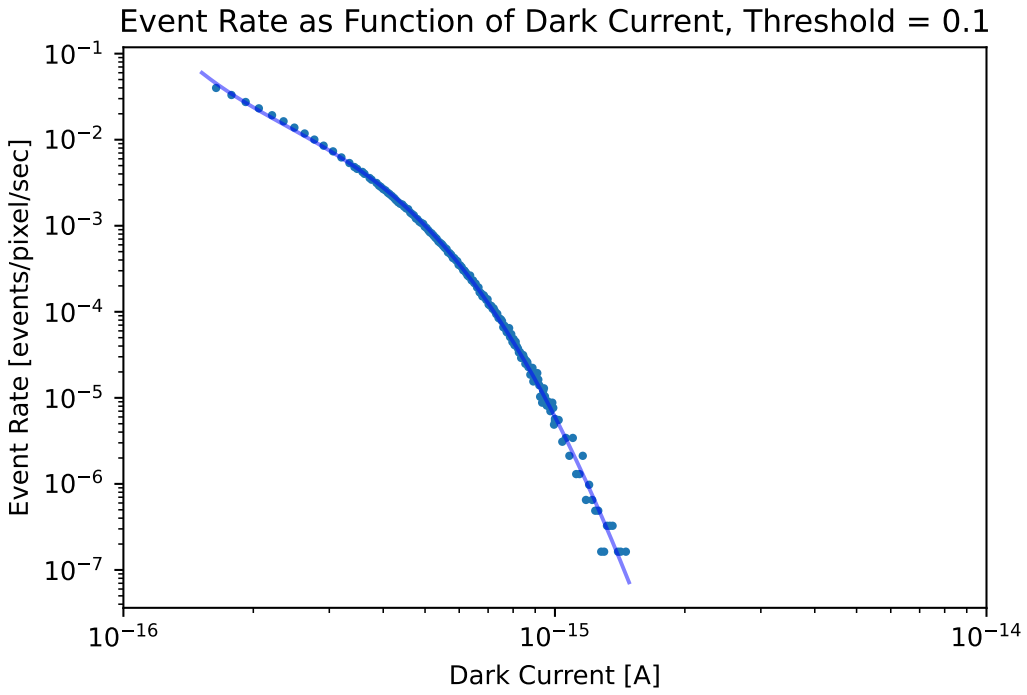


Figure 4.6: As the dark current increases the event rate attributable to the dark shot noise decreases. The response of the event rate to tuning this parameter is a quadratic drop-off in the log space of both the dark current and the noise event rate. Within smaller regions of performance, to match the temperature of a measurement, a linear fit is sufficient.

dark current. Its important to note the scale of dark current rates in the real data range from 10E-3 to 10E-5, the middle range of the values in Figure 4.6. The generated event rates with respect to the dark current best fit to a quadratic function. However, the dark shot noise event rate varies more significantly with temperature. I accept the variability of the noise rate from the temperature relationship and match the noise rate to the corresponding dark current with a linear relationship. I only linearize the relationship between the dark shot noise and dark current for noise rates from 10E-3 to 10E-5. The linear mapping is a poor assumption outside of these couple decades of the noise event rate values.

There is a slight shift towards larger dark currents producing this range of

values between Figures 3.16 and 4.6. The primary factor driving that shift is a change in the threshold value. Therefore, it is important to understand the camera bias settings before mapping the observation temperature to the dark current for the event simulation. In addition, this simulation supports the idea that the dark shot noise generation is cut through the increase of the event threshold.

Threshold Variation

Taking a closer look at the threshold to event relationship, I simulate a 1 fA dark current over the same 480 by 640 pixel array and 20 seconds of simulation time as the other simulations while only varying the thresholds in Figure 4.7. The threshold, being applied to the logarithmic change in photocurrent, determines where in the tail of the rolling Poisson method's Gaussian distribution of the logarithmic current a threshold change is met. It follows as the threshold decreases the number of samples in the tails of the distribution resembles the complementary error function evaluated at increasingly smaller values of z. In the limit where $z = 0$, every pixel generates an event at its maximum frequency, typically limited by the chosen refractory period or the sensor readout when the refractory period is short. This relationship supports higher thresholds being a significant tool to limit the dark shot noise event generation. The trade-off is decreased sensitivity to weaker point sources, a common issue with space EVS applications [65, 63]. The drop off in the dark shot noise event rate is a quadratic function with respect to the threshold value. Potentially an optimum bias setting exists between the dark shot noise event suppression and weak source event generation. This optimum will work best with improved knowledge of the dark current and control of the external parameters such as

temperature affecting the dark current to ensure consistent performance.

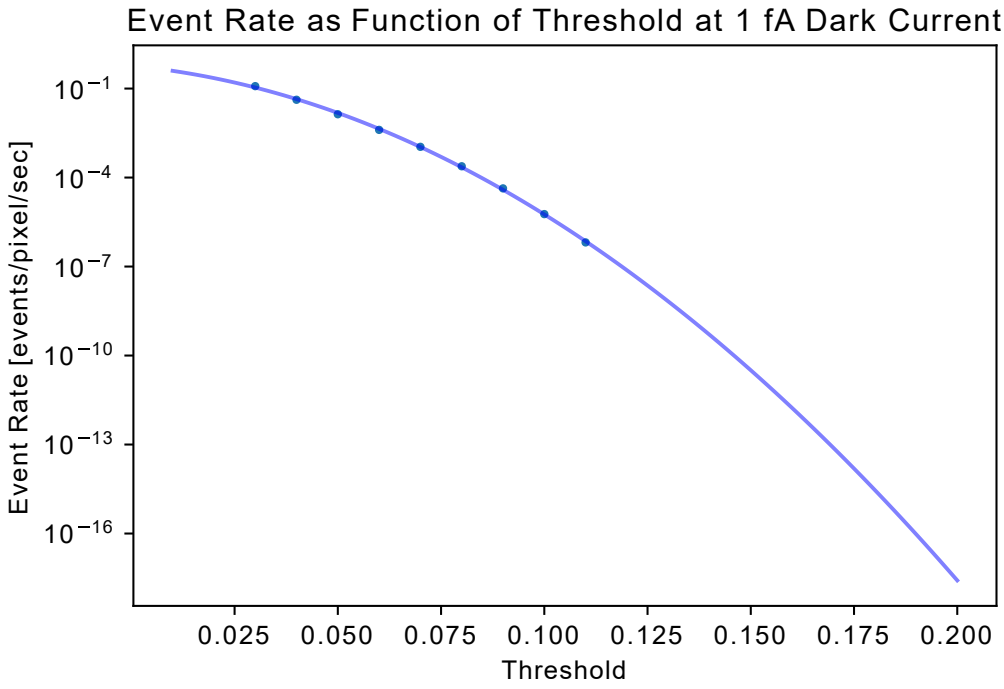


Figure 4.7: The dark shot noise event rate generation is closely tied to the threshold bias selection and amounts to increased area in the tails of the complementary error function. Choosing a higher threshold limits the dark shot noise generated, but will also limits the sensitivity of the sensor to weak signal sources.

4.1.2 Signal Noise Analysis

Regions of the focal plane where the dark current and, therefore, dark shot noise does not dominate the current, are subject to leaks and high frequency noise. As stated in Section 3.3.6, modeling noise on these regions with only v2e's leak rate approach does not generate the polarity signatures and frequency of noise event rates of EVS data, seen in Figure 4.8. Instead, I propose another new method with a tuned Gaussian, sampled on the dark current and induced photocurrent combined. This method equates the noise event rate for any time step, events

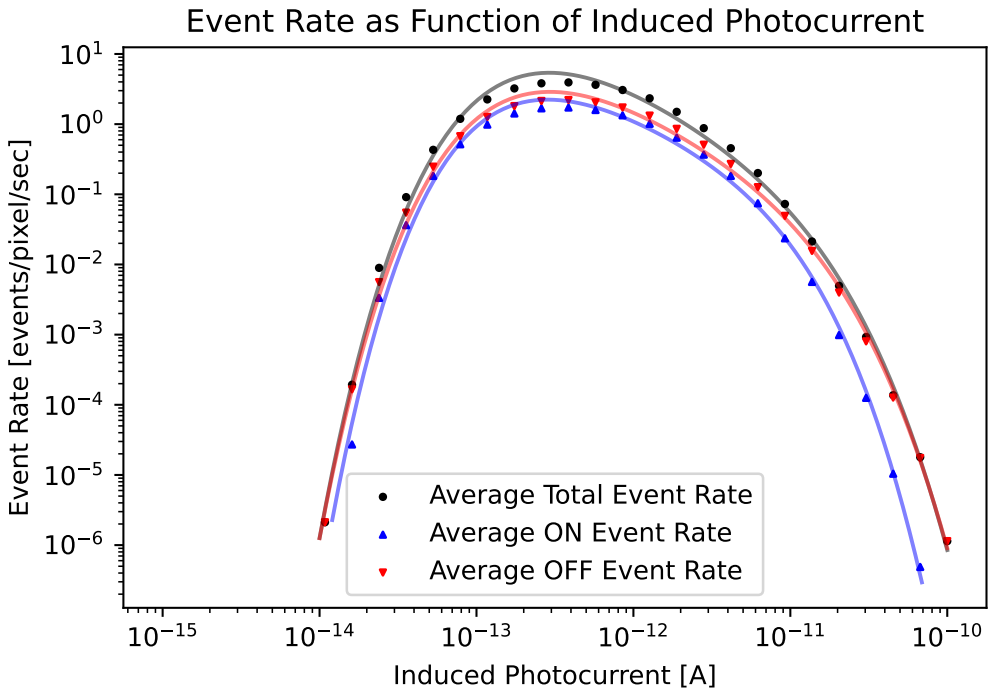


Figure 4.8: The quadratic fit of the required Gaussian standard deviations to implement as high frequency noise produces many of the attributes in the empirical data. First, the event rate peaks at a mid-range photocurrent value and drops-off towards higher and lower induced photocurrents. The OFF event rate is the primary source of events throughout the simulation with a more equal distribution of ON and OFF events in the regions with the highest event rates. The one unanticipated effect is the steep drop-off of events one decade above the dark current. This steep decline may be indicative of an incorrectly selected dark current for this simulation or a need to further refine the cutoff frequency relationship to dark current and induced photocurrent.

per pixel per subdivided time step, to the probability of an event occurring in that time step on a pixel.

I apply the high frequency Gaussian fit method to a 480 by 640 array for 20 seconds of simulation time assuming a threshold of 0.1, a dark current of 1 fA, and a corner frequency of 2.5 Hz. I evaluate the noise event rates for the total event and ON and OFF events individually. Figure 4.8 plots these rates as a function of the induced photocurrent for each simulation. Unequivocally,

the Gaussian fit method produces a more realistic noise event rate than the leak rate method in Figure 3.17. First, the noise generated produces both ON and OFF events instead of typically ON events. While the simulation always produces more OFF events, like the empirical data, the ratio of ON and OFF events benefits from the quadratic fit of the standard deviation. The largest standard deviation with respect to the induced photocurrent occurs at the peak in the curvature of the quadratic fit. The larger relative standard deviation provides more opportunities for ON events as a greater portion of the distribution in the tails passes the ON threshold. Consequently, as the fit moves away from that peak, the ON events decay faster than the OFF events. In those regions the OFF events are the primary contributor to the noise. Inspection of the empirical data also indicates this phenomenon.

Besides the magnitudes of the overall noise event rates generated, which differ from the empirical data due to the size of the array modeled, the decay of the noise event rate at lower induced photocurrent levels does not follow the empirical data's trend of decay over a couple decades of photocurrent values. I attribute the sharp decline in events to the change in the frequency cutoff in that region. The corner frequency following the proportional relationship in Equation 3.21 grows approximately by an order of magnitude per decade above the nominal dark current and the original corner frequency until reaching 3 kHz. With these simulation parameters, the simulation reaches the maximum corner frequency at approximately 1E-12 A. The issue in the simulation could simply be that the maximum corner frequency should be reached at a lower induced current value, which would extend the influence of the high frequency Gaussian fit into lower induced current values. Working with the originally proposed proportional relationship, either a greater low end corner frequency or a

dark current less than 1 fA will move the initial maximum corner frequency to the left. In this simulation, the dark shot noise event rate requires a threshold around 0.1 to produce the dark shot noise event rates in the empirical data. The default threshold bias value is around 0.2. Assuming a nominal bias setting, the dark current values in Figure 3.16, may be more realistic than the 1 fA estimate. Alternatively, the proportional relationship defining the slope of the corner frequency may be more complex than originally anticipated. While, I can currently work around this issue through manipulation of the dark current value, experimentation through empirical measurements of the dark current and simulations with alternative corner frequency functions will clarify the underlying cause of the rapid decay from the high frequency noise.

Since the output of the simulation is a full event data stream, I go a step further than magnitude of event generation to evaluate the performance of the high frequency Gaussian noise. With the data stream of events, I subdivide the time series of event data into distinct time subsets and evaluate the event rate within the subdivisions to ascertain the consistency and frequency of the noise event occurrence. Working with the empirical data first, Figure 4.9 depicts the distribution of event rates normalized as a density function over multiple time step intervals. The original intention of this analysis is to identify a relationship between the time sampling and the distribution sampled to generate noise consistent with the frequency of event occurrence. However, the Figure shows that the noise events generated by a set induced photocurrent are remarkably consistent. Unsurprisingly, the mean value of the event rate is constant at a value of 0.9 events per pixel per second for all subdivisions of time because ultimately the same number of events in the data set contribute to the overall event rate. For each time subdivision, the resulting distribution has a positive skew with

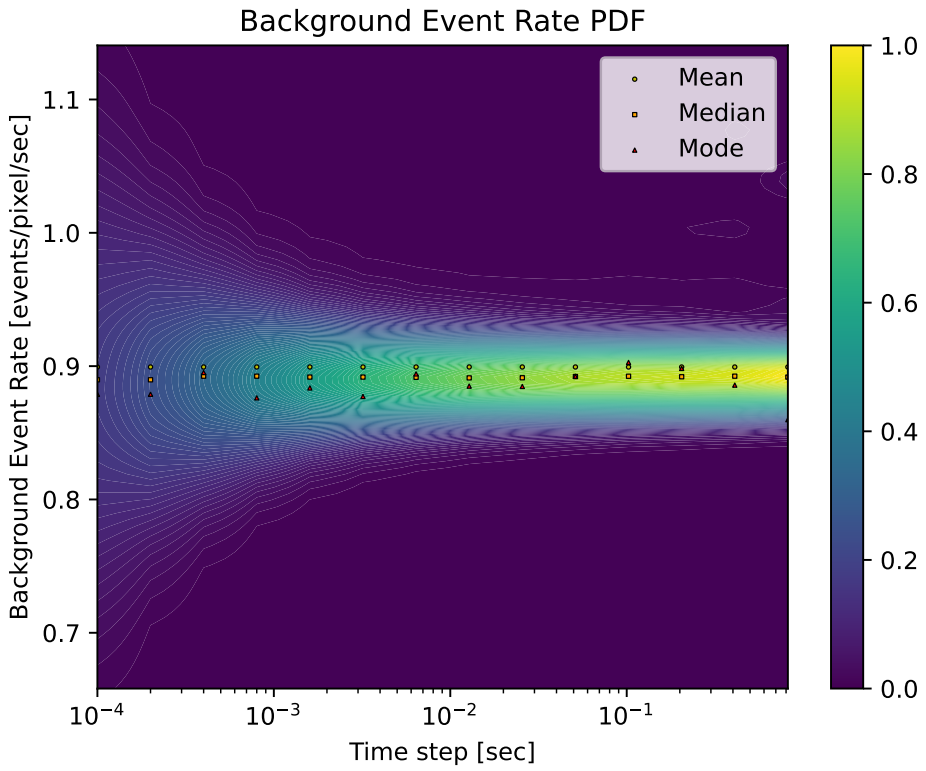


Figure 4.9: Subdivision of the noise event time series empirically collected at a set current value and fit to a probability density function demonstrates the consistent spread of the noise event generation when an EVS sensor is subjected to consistent incident energy. The distribution has a positive skew for all time subdivisions and the distribution grows further platykurtic as the time steps decrease. The distribution's smoothness at the smallest time divisions indicates consistent event generation without additional underlying frequencies.

the median at a slightly lower rate than the mean. The mode deviates more, especially at the largest step sizes, because there are only a few time step samples and they may all be unique despite being close in value. As the step size decreases the density of the distribution widens gaining more negative kurtosis, becoming platykurtic. The broadening of the distribution indicates a larger variation in the smaller time increments. The smoothness of the distribution at the smaller time increments indicates a fairly consistent rate of event generation.

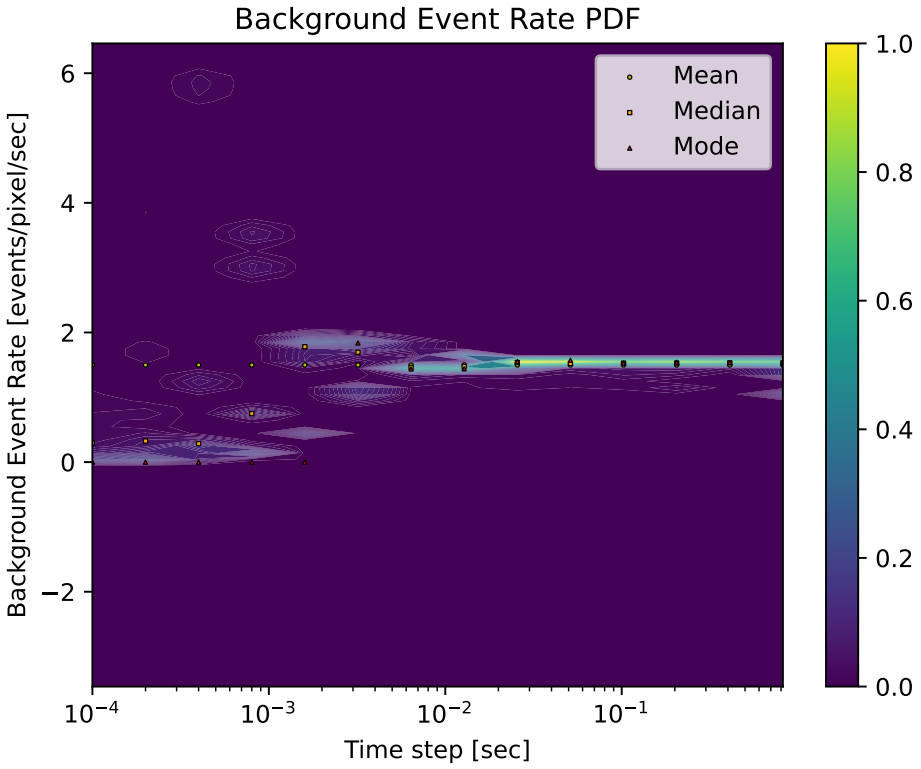


Figure 4.10: Subdivision of the simulated noise event time series generated at a set incident current value and fit to a probability density function demonstrates a less consistent spread of events than the empirical data. Up to the time steps on the same order of magnitude of the simulation time steps, 2 milliseconds, the distribution has a positive skew for all time subdivisions and begins its broadening of the noise event rate. However, at smaller subdivisions, the event rates generated do not share the smoothness seen in the empirical data. The binning is indicative of the event trigger times being weighted towards one side of the time step and refinement of the time stamp method is necessary to remove the artificially imposed frequency.

I conduct the same time subdivision analysis on the simulation generated event stream. Figure 4.10 highlights that at the macroscale the high frequency tuned Gaussian method produces a positively skewed distribution of events just like the empirical data. The rate is not directly comparable because I simulate the 3rd generation Prophesee EVS which has a different pixel count which scales the noise event rate. Additionally, the sensor itself is expected to have different

noise performance. Therefore, I do not select the two induced photocurrents in Figures 4.9 and 4.10 to match but to highlight the noise characteristics. As I decrease the time increment, the positively skewed distribution breaks down into multiple clusters of event rates instead of becoming a broad platykurtic distribution. This occurs at time increments on the same order of magnitude at the simulation step size, 2 milliseconds. This behavior indicates the generation of events within a simulation step are being clustered together and, therefore, the linear assumption determining the event trigger times of events is applying its own frequency and is erroneous. I recommend modification of the event trigger time determination method to be a linear change in the non-log current scale as opposed to being evaluated in the log scale as it is now. This modification is required before generation of synthetic data with reliable frequency information is possible through my simulation methods. It applies to all changes in current, not just noise, the noise rate analysis just highlights the impact of this oversight.

One final aspect I examine in the synthetically generated noise data returns to the macroscale generation of overall event rates. I wish to find the time of a simulation required to get a reliable noise event rate value. The empirically collected data sets are each 60 seconds long and my simulated noise data sets are 20 seconds long, matching the approximate length of the SDA data sets. I evaluate the total event rate after each new event in each data set, which provides an estimate of the total event rate over time. The noise event rate as a function of the simulation time, as in Figure 4.11, typically has an initial spike from the first few events happening early in the time history, a period of underestimation, and then a rise to the nominal noise event rate as the simulation length increases. For the data set in this Figure, the estimate is within 90% of the final event rate at approximately 2.5 seconds and 1.5 million events.

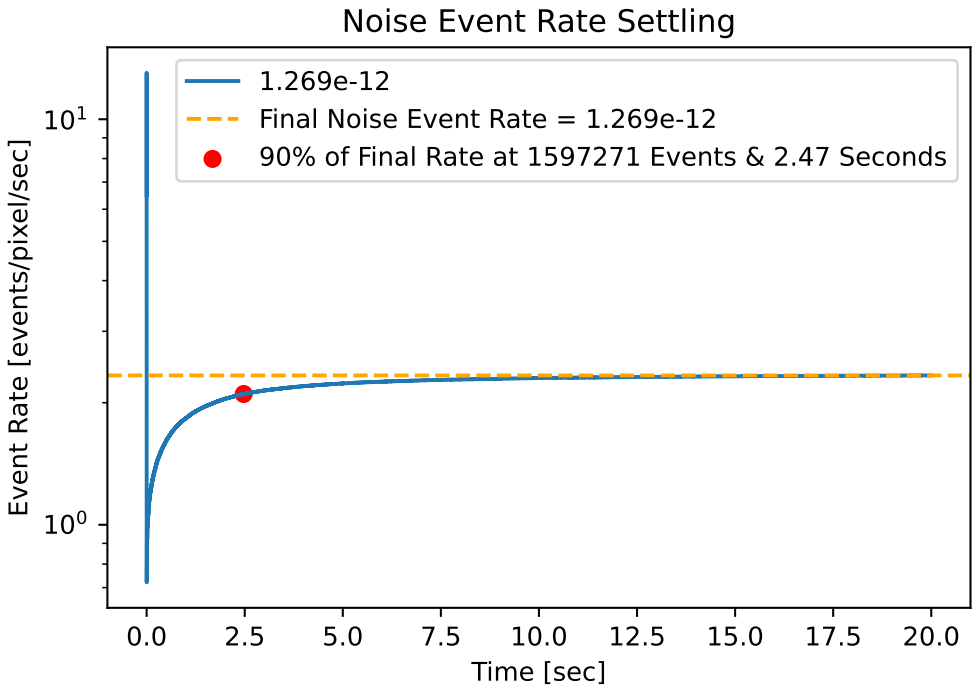


Figure 4.11: The rise time to the final noise event rate is extracted from the time series data by evaluating the total event rate after each new event. The red dot indicates the time the noise event rate is within 90% of the nominal value.

Figure 4.12 depicts the relationship between the number of events to reach the settling threshold and the total number of events in a data set. For data sets with higher event rates and, therefore, more total events output from the 20 seconds simulated, the number of events to reach the 90% of the final estimate plateaus at 1.5 million events. Therefore, there are diminishing returns to generating longer data sets when the anticipated noise event rate is high. For lower event rate data sets, the relationship between the total number of events and the 90% convergence is more linear with approximately 88% of the events needed to settle. The deviation for data sets at and below 100 events indicates longer simulation times are required to accurately capture the noise event rates. This is further reiterated by Figure 4.13 where the total time to settle is plotted against the total number of events in the data sets. The data sets in the 100s of

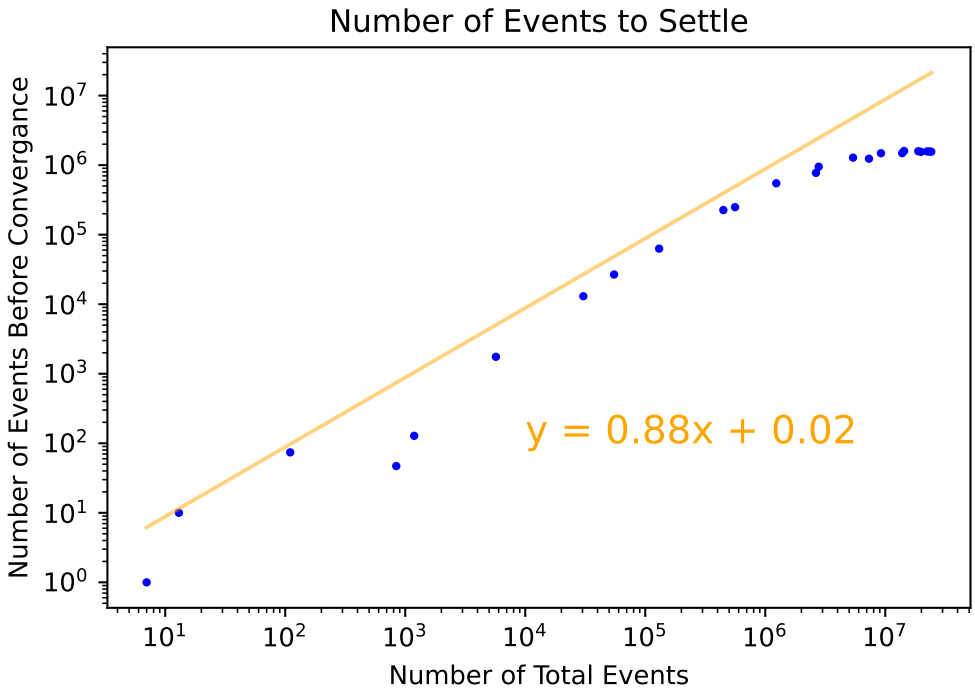


Figure 4.12: The number of events to reach 90% of the final reported event rate as a function of the total number of events demonstrates the advantage of a large number of event samples. The largest data sets plateau for convergence around 1.5 million events. Since these data sets have the highest event rates, they converge to their respective rates in under 2.5 minutes.

events do not fit the exponentially decreasing curve and probably do not have enough events within the 20 second simulation to prove convergence of their noise event rate. Therefore, in future noise analyses, I recommend tailoring the length of simulations by the number of samples generated instead of a fixed simulation time.

In summary, the high-frequency Gaussian with a tuned standard deviation proves promising as a method to properly mimic the event generation of EVS sensors in simulation. At a macro level, the method captures many characteristic in the empricial data: the peak noise event rates and drop-off at higher current levels, the use of a Gaussian distribution creates a realistic stronger prefer-

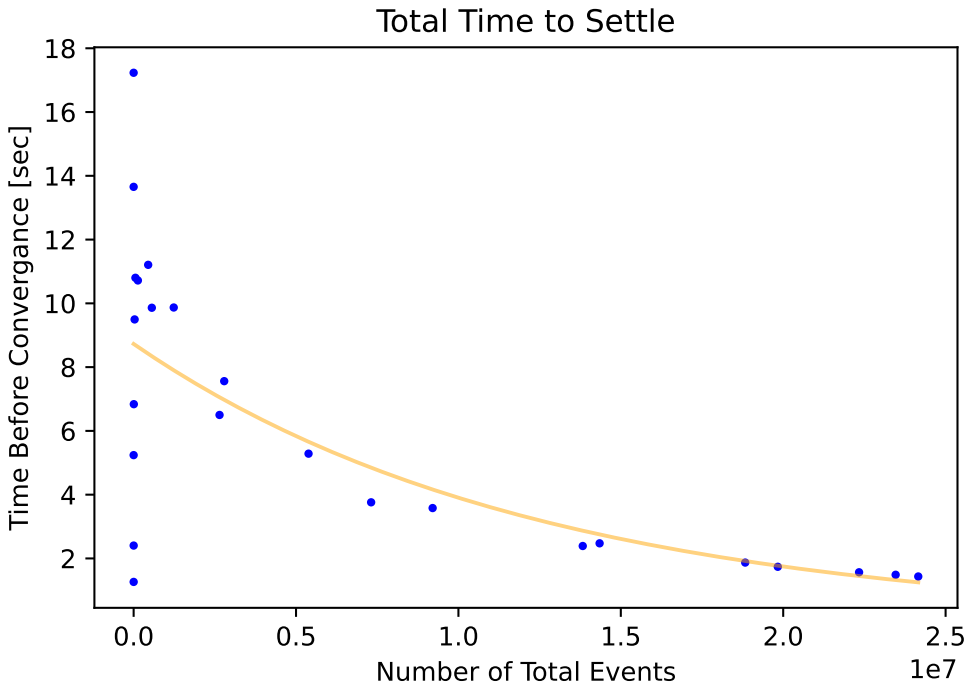


Figure 4.13: The simulation time to reach 90% of the final reported event rate as a function of the total number of events demonstrates the need to change the simulation length to have a minimum number of event samples. While most data sets show an exponential decrease in convergence time as the total number of events grow, the low number of event data sets time of convergence is arbitrarily set by the first event time stamp.

ence to OFF events than ON events, and the distribution of events demonstrates the positive skewness a larger time increments. The analysis also highlights a need to revisit the relationship between the frequency cutoff and the determination of event trigger times between simulation time steps.

4.2 Cluster Analysis

Clustering is an important component of being able to analyze event data, both for analysis of empirical data and assessment of synthetically generated events.

Clustering also provides statistical information on the attributes of event groups which I use in both Bayesian attribution and to train machine learning models in Section 6. In this section, I evaluate clustering methods with the intent to recommend batch clustering techniques for future event data processing.

4.2.1 Proximity-Based and DBSCAN Evaluation

I develop two batch clustering methods in Section 3.4.3, one reliant on the existing DBSCAN density-based clustering algorithm and the other applying a proximity-based clustering method. While in practice, I apply the proximity clustering 96% of the time, this metric is largely biased on my own choices. I only select one method to evaluate the group at a time and once the 2 dimensional projection looks sufficient, I accept the grouping output. This process does not lend itself well to online comparison of the techniques to evaluate which one should be recommended for future processing.

To assess the capabilities of each algorithm, I set fixed parameters for the algorithms and run each data set through each clustering method 100 times. The universal settings chosen are the mean values reported in Table 3.3. Since both algorithms are deterministic, the number of clusters does not change between each run taken to collect the run time information. Figure 4.14 depicts the number of clusters generated with each data set and these settings. 95.6% of data sets fit to a line with a slope of 1.14 and intercept of 7.11 indicating, with these general parameters, the DBSCAN algorithm produces more clusters than the proximity clustering algorithm. What is particularly interesting within the data in Figure 4.14 is the discrepancy between the clusters produced and the

total number of events in each data set as indicated by the color scale. This is likely due to different lighting conditions between observations. These data sets cover an entire month, so lighting conditions from the Moon's phase and weather affecting the light scattering within the atmosphere are expected to be inconsistent. This changes the ratio between the background and the signals coming from the satellites and stars yielding less or more events, depending on the background. Therefore, a data set with a large number of events does not necessarily correlate to a large number of clusters to identify.

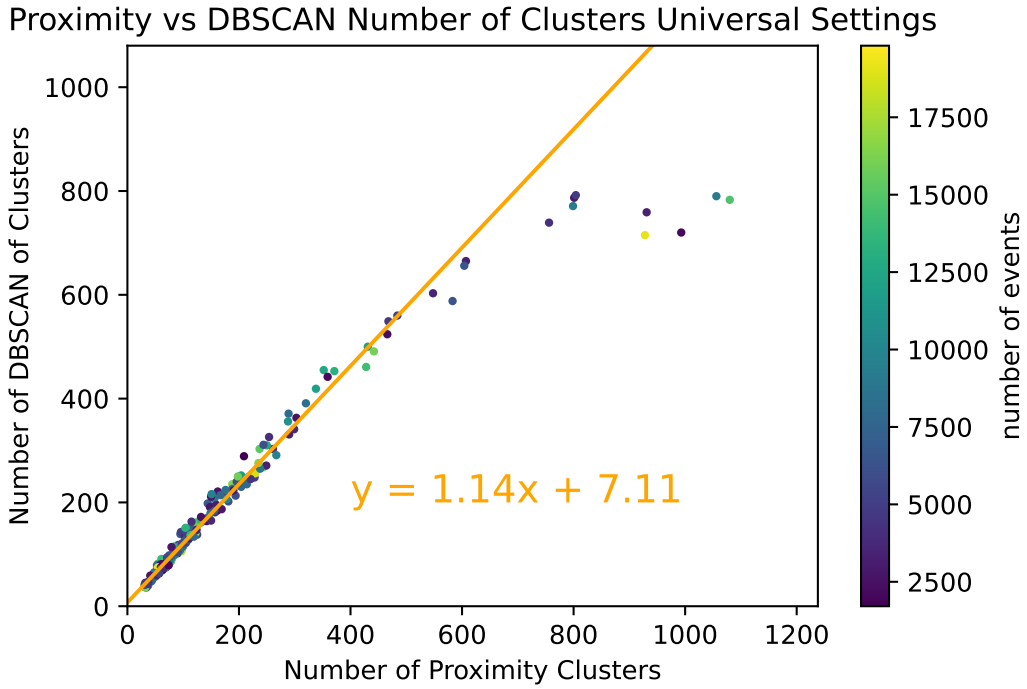


Figure 4.14: With fixed parameters, the DBSCAN algorithm produces more parameters for each dataset. The number of events is not greatly tied to the number of clusters with some of the largest and smallest data sets in terms of number of events yielding the largest number of clusters with both methods.

When I examine the number of clusters generated with the ideal settings chosen manually for each data set, applying the settings chosen for only one of the methods to both methods, the relationship between the modified DBSCAN

and proximity-based clustering is still not an exact match. In fact, the proximity-based clustering generates slightly more clusters than the DBSCAN clustering as indicated by the fit slope in Figure 4.15 now being below 1. However, both slopes are within 0.05 of 1, so the number of clusters produced is not significantly impacted by the choice of the clustering algorithm. Figure 4.15 does highlight the importance of parameter selection for each data set. The 4.4% of data sets diverging from the linear least squares fit of Figure 4.14 reduces to one major outlier in Figure 4.15. Therefore, when I choose parameters that are well tuned for a given data set, then the production of clusters is nearly identical between the two methods.

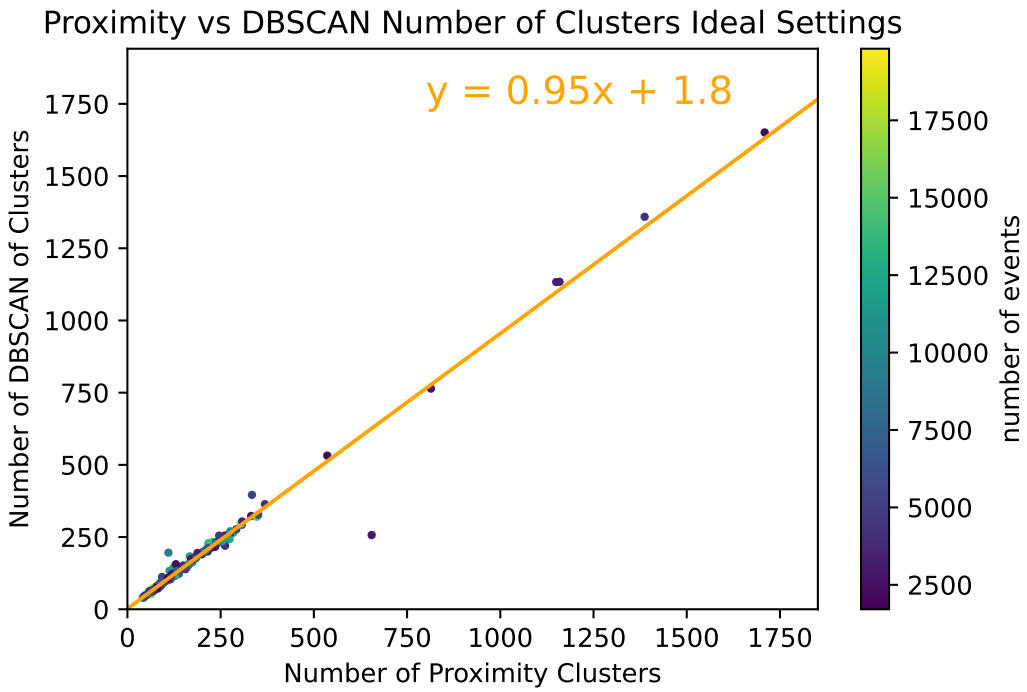


Figure 4.15: With parameters specified for each data set and applied to each clustering algorithm, the DBSCAN algorithm yields fewer clusters for each data set. The number of major outliers reduces to 1 from the linear fit indicating the importance of tuning parameters for each data set.

Since production of clusters is nearly uniform between the methods, I as-

sess the cluster algorithms with their temporal cost using the uniform parameters. Since the computational power of the computer determines the overall run time, the absolute time in Figure 4.16 is less important than the ratio between the trend lines. These lines, fit to the mean processing time as a function of the number of clusters in the dataset, demonstrate the proximity clustering computational time increases 10 times faster over the DBSCAN algorithm. The results indicate an overall preference for the DBSCAN algorithm. It nearly matches the clustering performance of the proximity clustering while being 10 times faster. While the proximity clustering does fall short in its run time and the run time diverges from the prediction with higher cluster numbers, the standard deviation between runs is smaller than for the DBSCAN algorithm, so the length of an iteration of the proximity cluster on a data set is very consistent.

Through examination of the total processing time and verification of the similar clustering achieved with desirable parameters, I recommend future batch-based clustering is completed with the DBSCAN algorithm. It is approximately 10 times faster than the proximity-based clustering for a given number of clusters while still producing approximately the same number of clusters. While the proximity-based clustering does not compete with the DBSCAN on speed in batch processing, DBSCAN theory is dependent on a batch of data being provided at one time. Therefore, it is complex to implement in an online fashion. Implementation of a batch clustering algorithm online is not impossible. In fact, I consider another algorithm, RANSAC, in Section 6.1.2 and discuss the complexities of implementation in more detail. In that same section, I treat the proximity-based clustering as the baseline grouping method to develop a tracking algorithm. The assumption that only proceeding events contribute to grouping is well suited for that online implementation.

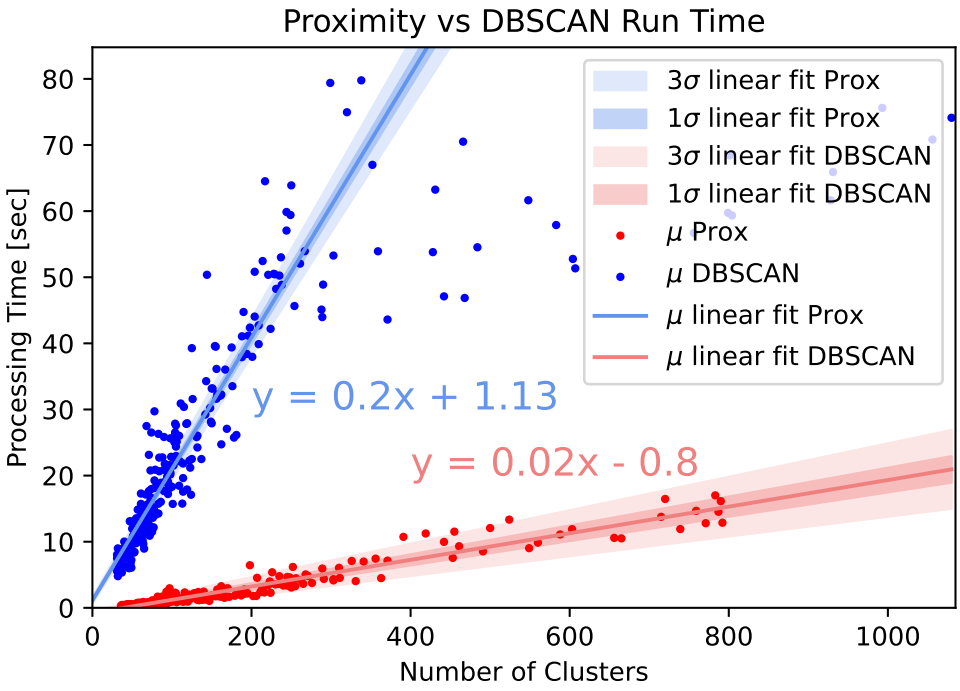


Figure 4.16: The means of the processing time for the proximity-based and DBSCAN clustering with respect to the number of clusters produced show linear growth with the number of clusters in the data set. Each data set is run through each processing method 100 times. The proximity clustering takes longer to process, but is more consistent in the processing time.

4.2.2 Cluster Grouping Analysis

Regardless of the clustering method I choose to separate related events while addressing the time component, I compare the results to the clusters derived with the process utilizing more classic dark sky processing techniques discussed in Section 3.4.1. The classic techniques require flattening the array into a traditional image, losing the temporal fidelity that may separate groups of events. In the case of the data mentioned in Section 3.4.1, the artificial event frames each compress 3 seconds of imagery before applying the post processing. The output of that processing critically only identifies one pixel for each source, the centroid of each group in the integrated image. I apply additional processing to connect

between the centroids and better estimate all the events associated with each source, assuming all events in a traversed pixel within an image belong to the centroid cluster.

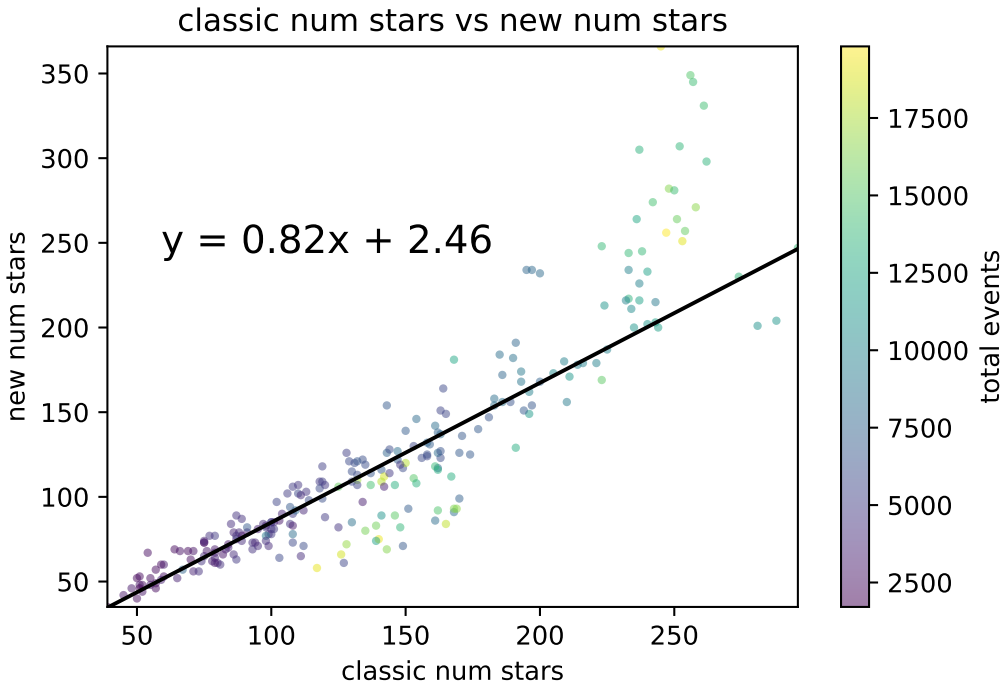


Figure 4.17: The classic processing technique to identify clusters compared to the proposed methods without compression to the temporal dimension. The total number of star clusters are compared because they are the most common type of cluster. With only rare occurrences of data sets identifying more than one satellite, the primary difference between these two techniques is the number of clusters produced.

While I must apply the additional data manipulation to evaluate the events associated with each cluster, the total number of clusters identified is simply the number of stars and satellites identified within each data set. I directly compare the number of stars identified in Figure 4.17. I only compare the number of star clusters because they are the most common type of cluster. As expected, the number of clusters generally increases as the total event count grows for both methods. The least-squares fit with a cauchy loss function, reduces the

influence of outlier events, to produce a fit closer the median ratio of the number of clusters identified. In general, the numbers of stars identified through the 3-dimensional clustering algorithm is around 82% of the number of stars identified by the classic method. There are two possible explanations for this decrease: either the classic algorithm is integrating noise events and identifying stars that are not really there or the 3-dimensional clustering algorithm is more conservative and is missing some star groups.

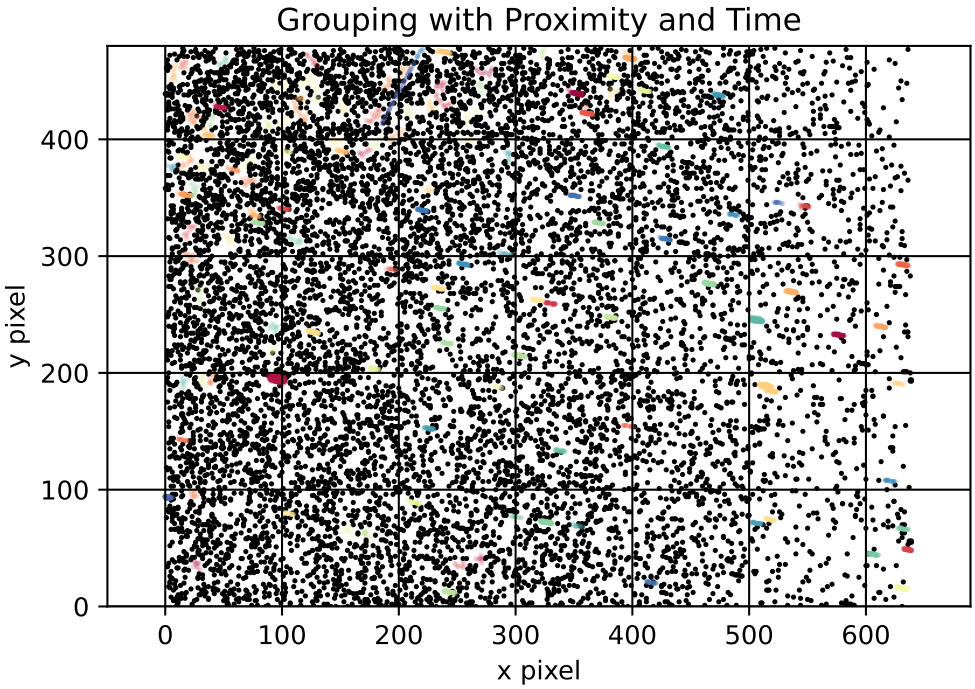


Figure 4.18: The 3-dimensional clustering output for high event rate data set with 16475 events in the approximately 20 second collection period that yields more star clusters with the classical method than the 3-dimensional clustering method. This is likely due to the high noise rate within this data set that yields clusters in the upper left of the plot that do not follow the trend of star clusters observable on the right hand side of plot.

One key insight into what is happening is in the location of the higher event groups that diverge from the fit line. Equating the total event metric to event rate since the data sets are approximately the same length, data sets with higher

event rates diverge from the predicted line. Higher event rates yield two separate groups of outliers from the linear fit. There is a group below the fit line where the 3-dimensional clustering produces fewer clusters than the classical method. This group likely belongs to the first explanation for higher star matches from the classic processing method; A higher noise rate produces non-existent clusters when integrated and subsequently matches to real star locations during the astrometry.net processing. Figure 4.18 is a clustering example of one of the data sets in that region with the output of the proximity-based clustering method. Every black dot corresponds to a noise event. Prior to applying the Hough transform, the events in the upper left corner of the plot are grouped together which means they are relatively close enough in time they may integrate together during the classical processing method and match to a star location. The drop down from the trend line may indicate a critical point in the event rate where this issue is exacerbated for the length of integrating time chosen. Possibly changing the length of integration based on the event rate would reduce this issue. However, it may be more difficult to find weak real signals with shorter integration times.

The other group of outliers, in the upper right hand corner of Figure 4.17, also have high event rates, but produce more star clusters with the 3-dimensional clustering methods than the number produced with the classical methods. These data sets are distinct from the first outlier group through the organization of their events as shown in Figure 4.19. This data set has nearly the same number of events as the previous one, with 15,483 and 16,475 events respectively. However, the high number of events are mostly associated with real signals, which yields less erroneous matching of clusters to stars with the classic processing method. Essentially, the observing conditions of this data set

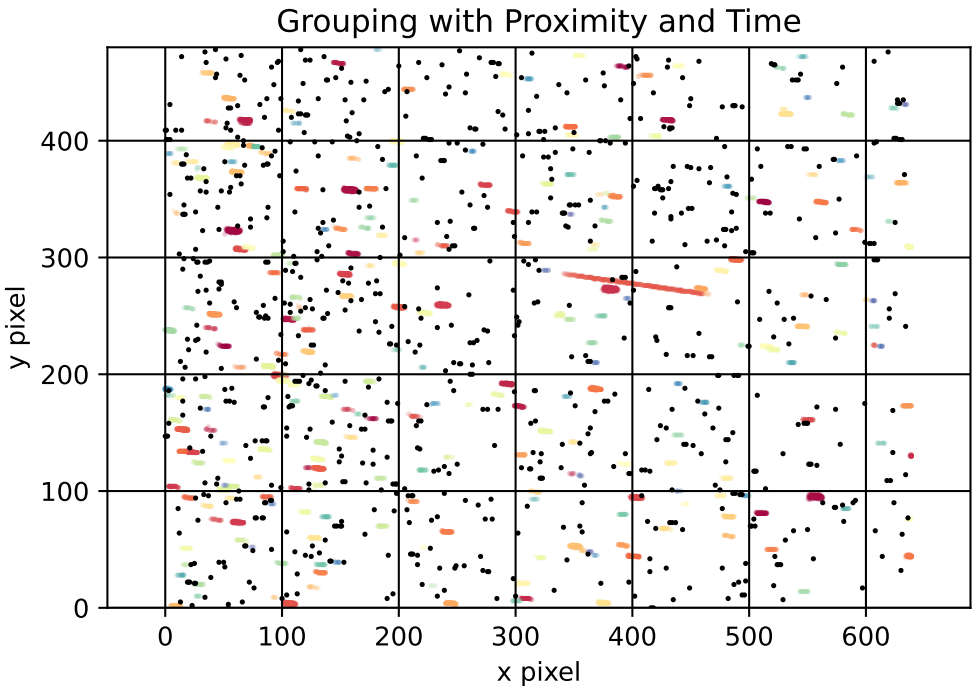


Figure 4.19: The 3-dimensional clustering output for high event rate data set with 15483 events in the approximately 20 second collection period that yields fewer star clusters with the classical method than the 3-dimensional clustering method. The events in this data set are more organized under the real signals indicating a preferable signal to noise ratio.

are more favorable, producing a higher signal to noise ratio. It is not immediately obvious why the classical processing method would match to fewer stars in these cases, but it is possible the 3-dimensional clustering methods are generating too many clusters that are not matched to real stars since that is not a check within the current processing method.

Moving beyond the generation of star clusters, I examine the total events grouped to each class of identified object starting with the star clusters in Figure 4.20. Unsurprisingly, this relationship is relatively similar to the relationship in Figure 4.17. Where fewer stars clusters are generated with the 3-dimensional clustering, fewer total events are associated with those clusters. Overall, how-

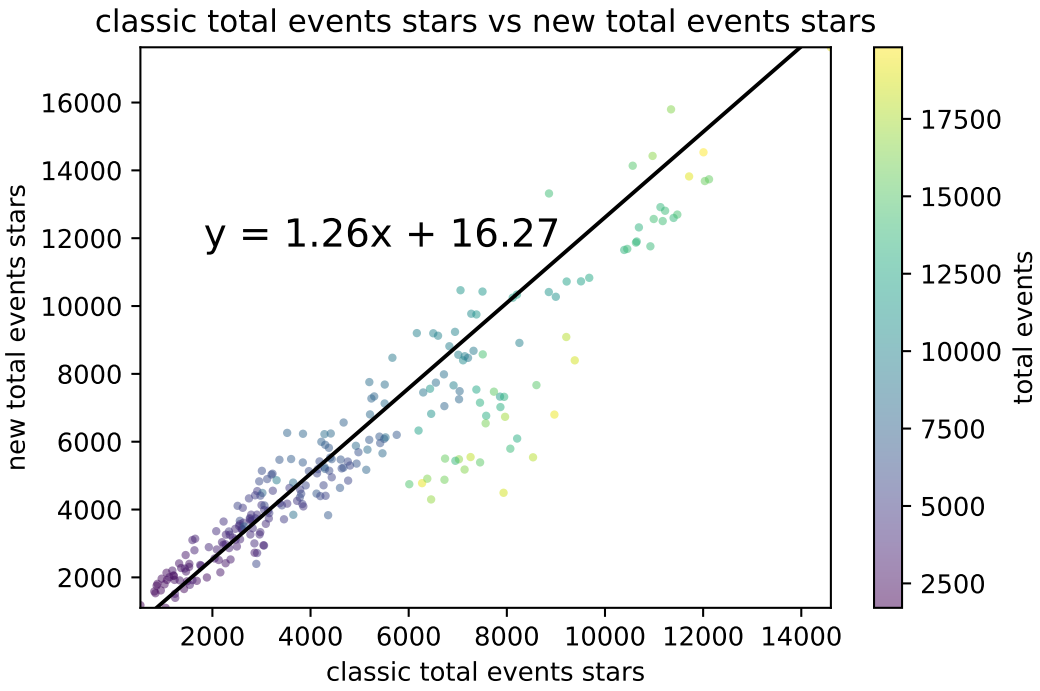


Figure 4.20: Comparing the total events attributed to stars with the classical and 3-dimensional processing methods demonstrates the under attribution of events to stars following the method suggested in Section 3.4.4. The only exception is in data sets with high levels of noise where the 3-dimensional processing methods more easily reject noise data.

ever, the 3-dimensional clustering attributes more events to the star clusters generated than the classical processing method. This is partially due to the rudimentary way of determining which events in each slice should be assigned to each star cluster. Recall, I assign events to a group based on the pixels traversed in one time step by the star with the assumption that the pixel location of the star reported within the time step is in the middle its associated grouping in both space and time. Active pixels in one time slice, particularly towards the end of each time slice, may have events in the next slice that are not added to the associated cluster. Overall, this method underestimates the events and the underestimation grows as the signal strength increases as indicated by the slope over 1 on the least squares fit. Stronger signals have more ON events which

yields a longer tail of OFF events as the circuit returns to a nominal level for the background. This process extends the time of the trailing events into the other slices exacerbating the issue.

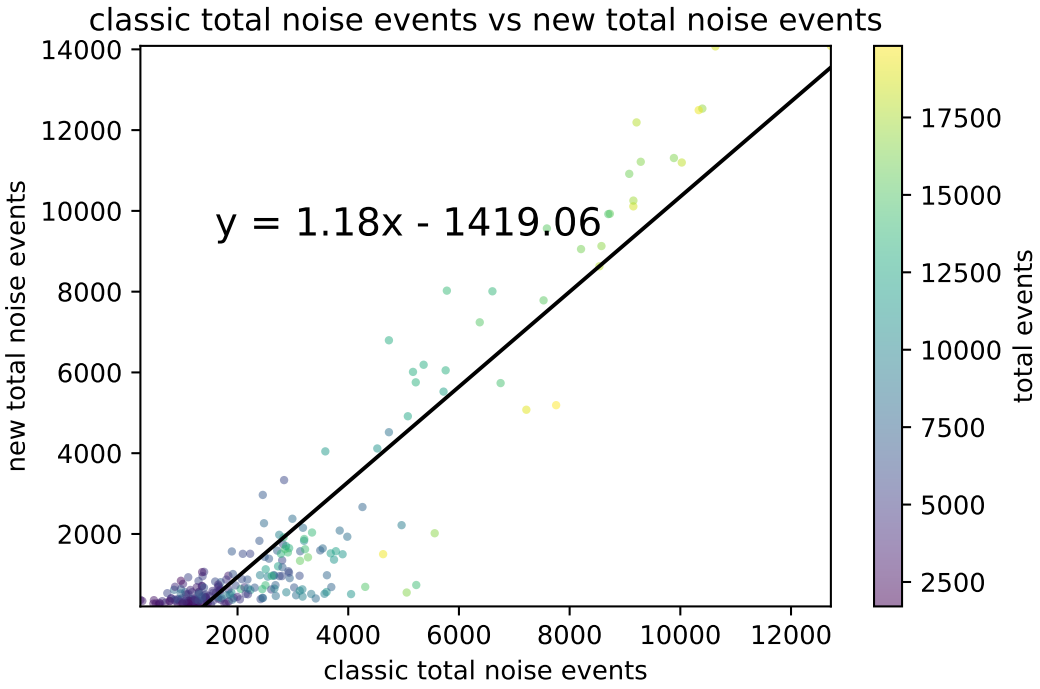


Figure 4.21: The total events attributed to noise with the classical processing method is compared to the 3-dimensional processing method. At lower levels of noise the 3-dimensional method rejects fewer events from true groupings, likely an artifact of the attribution method being applied to the classical processing output. Conversely, for data sets with higher noise levels, the 3-dimensional processing method rejects more than the classical method.

Pivoting to examine the events attributed to noise, the same outlier group seen in Figures 4.17 and 4.20 appears again. Figure 4.21 confirms the lower total events attributed to stars is due to the 3-dimensional clustering method assigning more events to noise. When clustering high noise rates data sets with the classical processing method, the previously mentioned over attribution of star clusters and the density of the events within the time slices yields a larger portion of events being assigned to star clusters instead of being rejected as

noise.

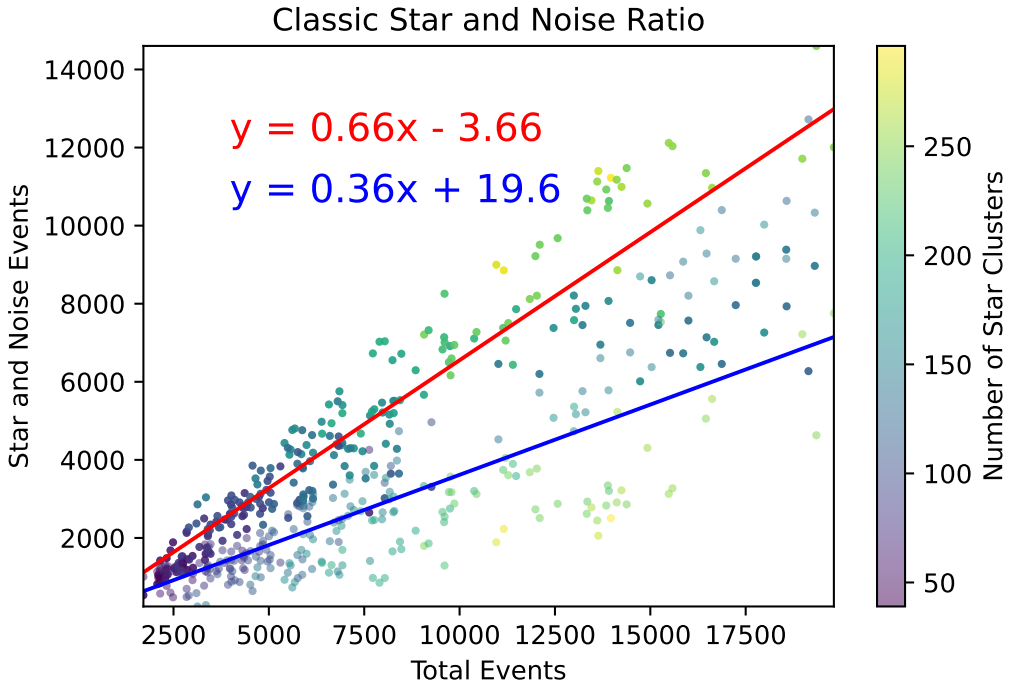


Figure 4.22: A relationship exists between the star and noise events attributed through the classical processing method. A least squares fit to the the star and noise counts is drawn in red and blue respectively. Approximately two-thirds of the events are attributed to the star clusters and one-third are attributed to noise for most low noise data sets.

It's clear there is a trade-off between events attributed to stars or rejected as noise and this is partially tied to the number of clusters assigned to stars and the overall noise rate with the classical processing method. Figure 4.22 depicts only the classic processing method output of associated star and noise events as a function of the total number of events. The number of star clusters from the classical processing output is displayed as the color of each plotted point. The star and noise event points are distinguishable as the noise events are plotted as with 50% transparency. I fit a line to each spread of resulting points, with the red line associated with the star event count and the blue line associated with the noise event count. The slopes indicate that through the classical

method, approximately two-thirds of events are associated with the star clusters and one-third is associated with noise. Divergence from the fit lines occurs at the higher levels of events in the range of 120 to 170 star clusters, which follows the previous analysis of data sets with higher levels of noise.

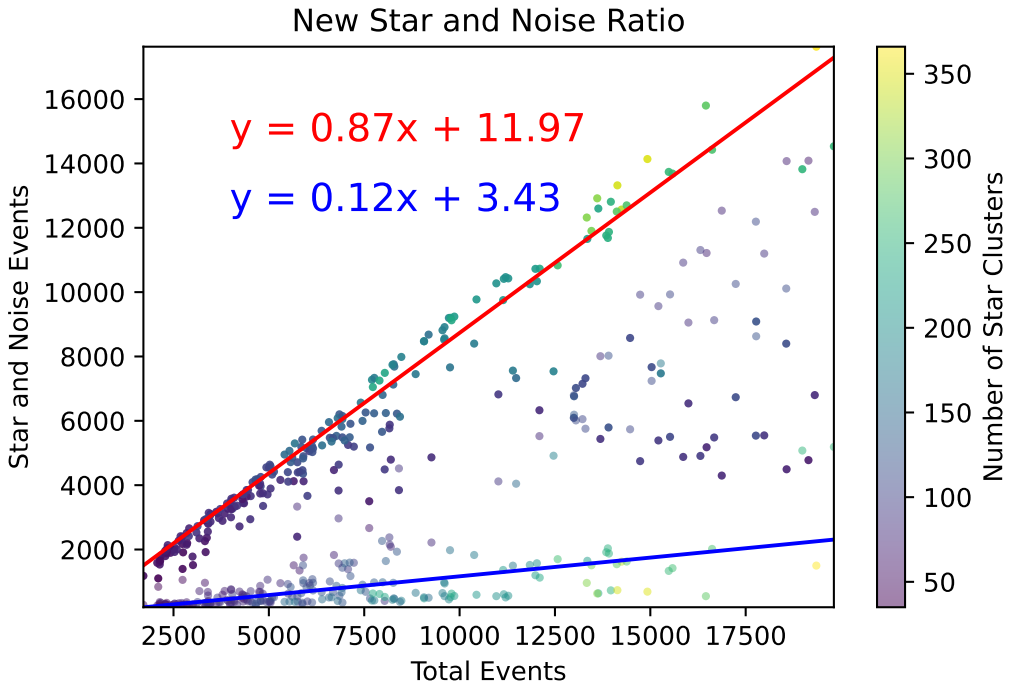


Figure 4.23: A relationship exists between the star and noise events attributed through the 3-dimensional clustering methods. A least squares fit to the the star and noise counts is drawn in red and blue respectively. Approximately 87% of the events are attributed to the star clusters and 12% are attributed to noise for most low noise data sets.

Applying the same analysis to the output of the 3-dimensional clustering method in Figure 4.23, the differentiation between the noise and star lines are more pronounced. In general, with the 3-dimensional clustering methods 87% of events are attributed to star clusters as indicated by the slope of the red line. On the other hand, the blue line indicates only 12% of the events are attributed to noise. Presumably, this relationship follows the lower loss of trailing events to noise attribution with the 3-dimensional clustering. Figure 4.23 also supports

my interpretation that the 3-dimensional clustering rejects more noise in the data sets with high noise rates. Outliers from the fit lines are mostly in regions of the plot with higher total event count and lower numbers of attributable clusters. The higher event counts switch between darker star associated events and more transparent noise associated events around 11000 events for only data sets with lower numbers of star clusters indicated by their purple coloring.

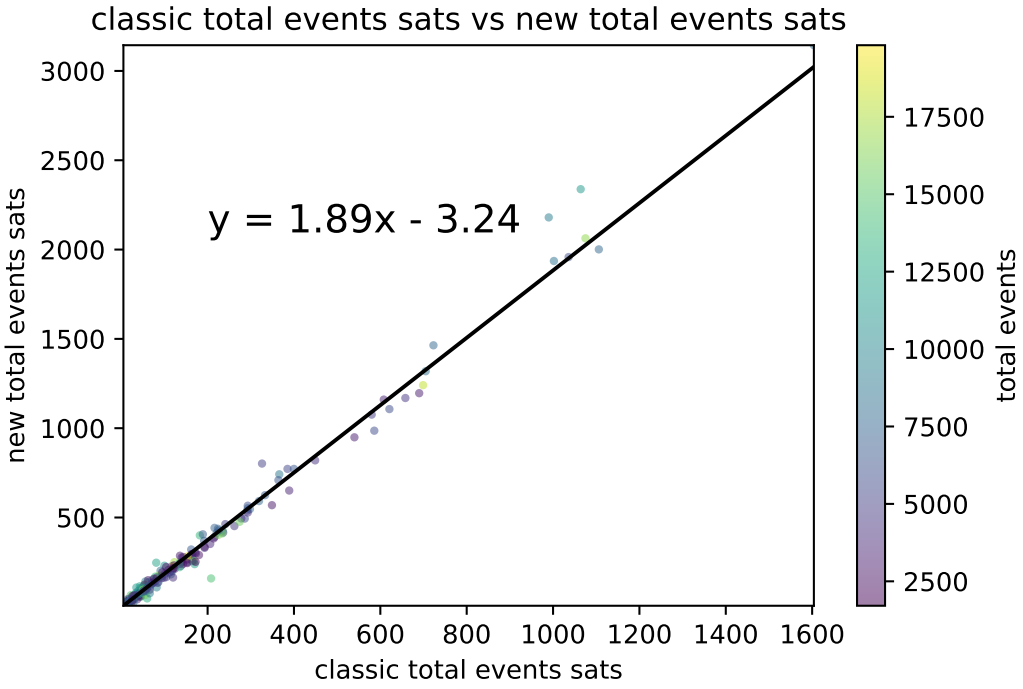


Figure 4.24: Comparing the total events attributed to satellites with the classical and 3-dimensional processing methods demonstrates a surprising lack of OFF events attributed to satellites with the classical processing method. This issue nearly doubles the events attributed to satellites with the 3-dimensional clustering.

From the previous two Figures, it is apparent that most of the events for either processing method belong to star clusters or noise. The final attribution option is a satellite cluster. With each data set containing either 1 or 2 satellites, I expect satellite events to be in the minority of the data. I apply no additional processing to the classical processing method to extract satellite events as they

are already provided. Figure 4.24 compares the events attributed to satellites with either processing method. This relationship contains the least outliers of the three total event attribution relationships and is fit to a line with a slope of 1.89. Therefore, the 3-dimensional clustering method is capturing nearly twice as many events as the classical processing method. The reason for this discrepancy is clear. In the satellite attributed classical processing method data, only 1 data set has OFF events attributed to a satellite. It is not clear, however, what in the post processing chain of the classical processing method yields this inconsistency. However, with the production of ON events and an assumed equal setting for ON and OFF thresholds, I expect OFF event production. Figure 4.25 depicts one satellite group from the 3-dimensional clustering algorithm. The ON and OFF events are depicted in blue and red respectively. The tail of OFF events closely follows the earlier ON events in the time dimension and has a consistent length building confidence in their proper association.

In summary, the 3-dimensional clustering methods prove advantageous over the classical processing method for identifying stars and satellite clusters in event-based data sets. The classical processing method has a clear drawback in the extraction of satellite events. It is not clear why the OFF events are neglected. The classical method also underperforms in data sets that require heavy noise event rejection because clusters of noise events can match to a star in SExtractor. My simple method to attribute star events to the identified stars also yields less distinction between star and noise events. The 3-dimensional clustering methods using either the DBSCAN or proximity-based clustering in conjunction with a Hough transform identifies the OFF events attributed to satellites missing in the classical processing output. They also adapt to the higher noise event data sets. While the 3-dimensional clustering may conservatively overlook some real

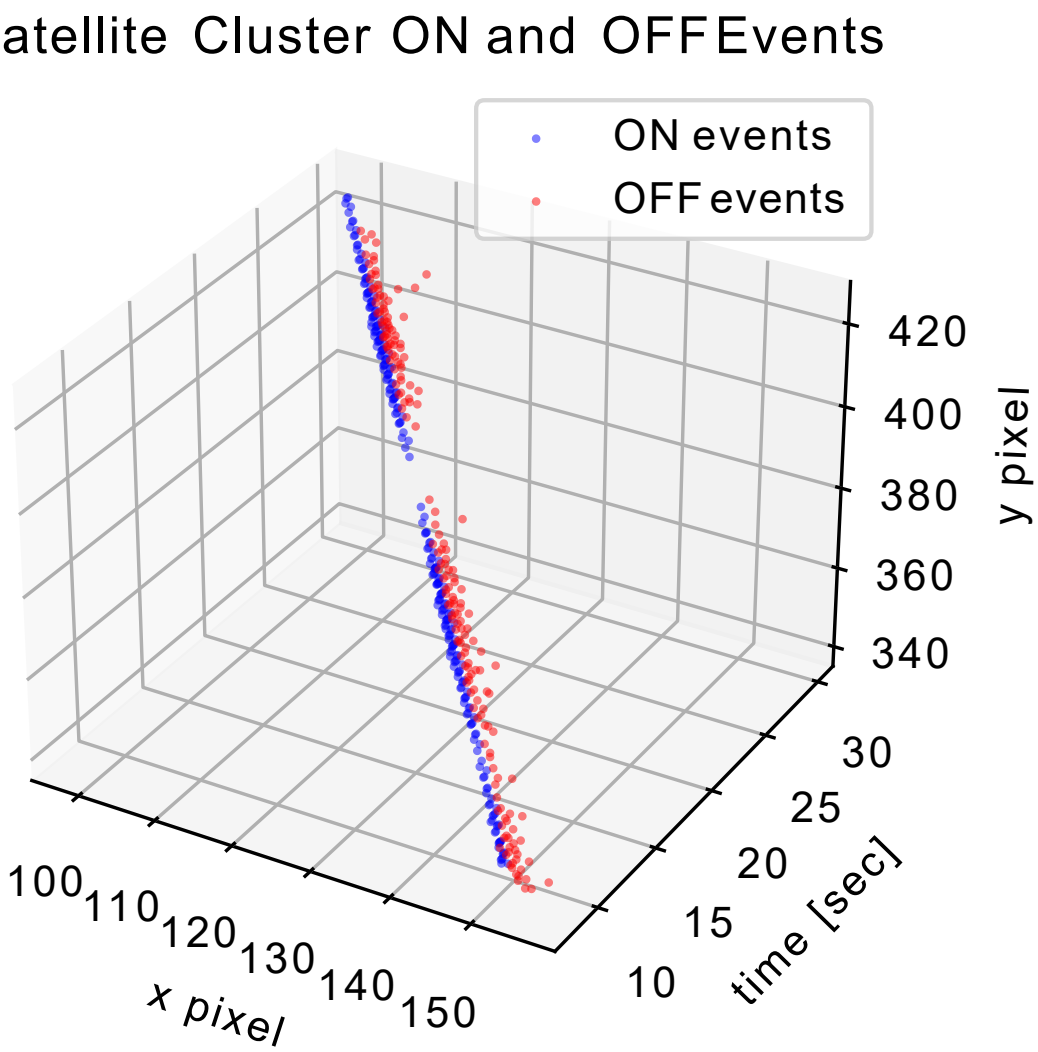


Figure 4.25: An example of a satellite cluster's ON and closely following OFF events. For this particular signal level, a second or two of OFF events trail behind the original ON events from a satellite stimulating these pixels.

star clusters, its output indicates reasonable identification of clusters above the signal to noise ratio.

CHAPTER 5

SIMULATION CONCLUSIONS

5.1 Noise Generation Performance

There are two main sources of noise I account for in the SDA simulation: dark shot noise and high frequency noise. Each requires a new methodology to fully leverage the electron per second flux information provided from the radiometric portion of the model. Ideally, from leveraging this information, the noise generated mimics the statistics of EVS noise event generation.

For the dark shot noise, I identify a trade-off between high temporal sampling required to capture spatial motion through the array and passing randomly sampled noise through the low-pass filter. In order to maintain the temporal sampling I desire in the overall simulation and still yield dark shot noise events, I develop a rolling Poisson summation method to track the effective dark current at each simulation time step. With the history of the previous time steps, up to one second of draws, the dark current retains some inertia from the previous time steps. Since it does not rapidly change, the dark current variation maintains a greater amplitude from the nominal dark current than a traditional Poisson sampling at each time step. With enough time steps, I prove the resulting simulation has the same qualities in the resulting distribution as the traditional Poisson method. One great advantage of my new method is the minimal parameter space to implement it properly. I only need an estimate of the dark current and the change thresholds of the EVS in order to apply this method.

This simplicity is in stark contrast to a alternative method of lower frequency

sampling that enables the pass of information through the low-pass filter. It's difficult to balance a low sampling rate method; One must let enough information through the filter but not impose an artificial curve to the current which could yield a unnatural frequency into the address-event representation data stream. The low-rate Gaussian I implement highlights these deficiencies with its final distribution being more platykurtic than the high-rate sampled Poisson methods.

In total, the rolling Poisson method succeeds in its goal to pass a varying dark current value to the comparator portion of the EVS circuitry. On the macro-scale, my rolling Poisson method event rates are on the same order of magnitude as those I extract from the empirical data I label as noise through my clustering processes. With its simplicity and effectiveness, the rolling Poisson method now is the method of choice within the EVS simulation.

After dark shot noise, I address the deficiency found in modeling high rate noise only through the leak rate process. The previously implemented leak rate process only produces ON events and the rate of events is uniform regardless of induced photocurrent. Inspired by background noise curves for EVS sensors, I apply a tuned Gaussian method setting the noise event rate per simulation step as the probability under the tails of a Gaussian curve. After conversion to a standard deviation to produce that rate, I apply the quadratic trend to determine the standard deviation of the Gaussian taken at each time step. The result has remarkable similarities to the original background noise curve. There is a peak noise event rate in the middle of the induced photocurrent I simulate and a drop-off on both higher and lower photocurrent. The Gaussian, tuned to the lower end threshold to define the required standard deviation, produces more

OFF events than ON events which is another characteristic in the empirical data.

Despite these successes, the results highlight two changes that must be made to the simulation in order to capture the right frequency of noise events at lower photocurrent, levels relevant to SDA information, and achieve the goal of synthetic event time series with accurate frequency information. First, the high-frequency Gaussian method is particularly susceptible to the drop off of the cut-off frequency from approximately 3000 to 1 hertz. With the current model, the cutoff frequency effectively scales by an order of magnitude for each decade of photocurrent. It is not evident with the current information if the cause of modeled noise event rate drop off is simply an overestimation of the dark current frequency or if the equation defining the frequency cutoff is erroneous. Dark current measurements and study of the frequency cutoff may be required to answer this open question. The other simulation issue I identify during the event rate per simulation step analysis is an unintended imposed frequency in the event readout. I theorize the clustering of events into common rate bins is tied to the event trigger times being assigned improperly. The current method applies linear interpolation in the log scale. I suggest the modification of this method to impose linear change between photocurrents in the non-log scale and a subsequent timing analysis to verify the correction method.

In summary, my proposed methods of noise event generation both utilize the electron-level information about the circuit and create noise event rates on the order of magnitude with correct polarity statistics and type of distributions. The models themselves identify shortcomings in other portions of the model, the cutoff frequency and event trigger time determination. It is important to update these components and verify the frequency of the events through the

noise generation methods before progressing to generating synthetic data sets to compare against real EVS SDA collections.

5.2 Clustering Performance

I develop a new event data processing pipeline to cluster events into attributable groups. The combination of either DBSCAN or proximity-based clustering with a Hough transform allows me to identify groups of events belonging to a signal and attribute them to particular sources for future use. While I rely more on the proximity-based clustering to process the data set utilized in this dissertation, the performance of applying DBSCAN is on par with the number of clusters produced, particularly with tuned settings for each data set. Since DBSCAN will run 10 times faster than the proximity-based clustering on batch data, I ultimately recommend the DBSCAN and Hough transform combination for future event-based data processing. This analysis however is not an exhaustive list of clustering and line drawing algorithms. In particular, the RANSAC algorithm, which I compare against the baseline online version of the proximity-based clustering method in the Section 6.1, combines the line drawing and clustering for online event attribution. It may also prove useful in batch data processing, but I leave that comparison for future work.

To assess the validity of my processing pipeline, I compare its output to that of the classical processing method which integrates slices of events into images and applies traditional SDA image processing methods to identify stars and satellites. I must reconstruct the events associated with stars from the classical processing method which highlights the downsides of integrating events into a

time slice. The loss of temporal fidelity results noise events grouped together and identified as possible stars. This is particularly apparent in high noise data sets. Additionally, my reconstruction suffers from trailing events of a cluster in the next time slice not being attributed to the previous slice's cluster. It would be difficult to improve that performance because the tail length of associated events within a cluster varies between data sets and source strength. Beyond the difficultly of reconstructing which events belong to each identified source, the classical processing method provided satellite event data suffers from no OFF events being included. The satellite ON events likely lead to associated OFF events as properly identified by my 3 dimensional clustering pipeline. Overall, while my data attribution pipeline may not be perfect, the use of the temporal dimension in the data is a positive step towards properly identifying all ON and OFF events of real signals and improving noise rejection. The process serves its purpose of providing labeled truth data for both simulation verification purposes and the statistics behind the Bayseian and machine learning classifiers I discuss in Section 6.2. With regards to the simulation verification conducted in this dissertation, the labeled event data sets provide samples of dark shot noise event rates which verify the temporal sensitivity of the photodiode. Going forward, I expect to utilize the data output in verification of generating realistic signals and checking event characteristics.

5.3 Future Work

5.3.1 True Signal Analysis

The next step in the simulation verification process is to compare the simulation output against EVS SDA data sets. Through the clustering process, I have characteristics for each data set which I derive in Section 6.2.1. These attributes enable data association between satellite, star, and noise events. In Section 7.2.6, I show the attributes selected have promise for the data association problem for SDA. I can also extract the same characteristics in the synthetically generated data. Once corrections are complete to the simulation for the issues raised by the noise analysis in Section 4.1, I can generate synthetic data for each of the empirical datasets. I will start this process though generation of events for a single scenario. Since, there is imperfect knowledge on some of the simulation input parameters, such as the threshold bias settings, these parameters will be tuned on the first data set. I will use a gradient decent technique to modify the setting parameters and conduct iterative runs of the circuitry portion of the simulation until the data attributes converge. As the parameters prove effective in data classification, I want to ensure these characteristics are properly captured in the synthetic data sets. Additional simulation iterations may be necessary if inconsistencies are found in these parameters.

5.3.2 Simulation Robustness

After I have shown the simulation captures noise sources based on incident energy levels and the simulation can match the event statistics of a data set, the

next question to answer is whether the simulation can robustly capture the sensor's performance in other environmental conditions, other cameras of the same model, and the wider spectrum of the new generation of cameras. This effort will require new sources of validation data. Possible test collections could include: collections in multiple lighting environments with external measurement of the ambient background light, collections with multiple sensors versions on the same observation, and collections with the newer generations of cameras. Through these new data sets and matching synthetic data sets, I hope to verify the universal applicability of my rigorous simulation method or identify further fixes to improve the simulation performance and strive closer to the goal of reliable event signatures from a simulation.

5.3.3 Space-Based Simulations

The primary method of verification of the simulation developed in this Chapter requires real data collections. All the collections so far analyzed are ground-based. This fact confined the scope of the work in this dissertation to ground-based applications. While these sensors provide a relatively cheap way to create a proliferated constellation of ground-based sensors for SDA, their real promise is in space-based applications for both satellite and star tracking. Therefore, the goal is to modify the current simulation to replicate a space-based observation platform.

Fortunately, there is a current EVS abord the ISS, the Falcon Neuro experiment [61]. Up until recently, this experiment has been observing the Earth in the nadir direction. The experiment is now re-oriented and has a view in the

zenith direction. The new data is an exciting opportunity to update and verify the model for space-based simulations. In some ways, a space-based simulation will be simpler. For one, no propagation through an atmospheric model will be necessary. On the other hand, there are new noise and observing considerations. For one, as outlined in the EVS radiation study, radiation is another primary source of noise on the sensor in a space-based environment to consider with 22 times in the overall background noise rate in their testing [86]. Therefore, I plan to implement radiation noise sources to match these possibilities. As for observing conditions, I intend to implement additional dynamics and logic for viewing conditions that were unnecessary for the ground-based viewing simulations. After these additions are made and the new data is clustered, I will go through a similar verification process for the space-based data as I implement on the original SDA data sets.

CHAPTER 6

EVENT-BASED TRACKING METHODOLOGY

Maholwald recognizes that "Information is encoded in the time between events" and cites it as motivation for developing the first EVS [58]. Traditional imaging tracking algorithms function on integrated images that minimize the temporal resolution to the rate at which the images are taken. EVS are designed to reduce the information collected to only changes in the frame in order to capture the temporal information that evolves between frames. This advantage motivates this dissertation by providing opportunities to capture the high resolution temporal information of fast moving LEO satellites and leverage the information encoded in the events to conduct SDA identification and tracking tasks. In fact, the simulation development of Chapter 3 aims to create realistic synthetic data sets to further this effort. While the simulation is not yet ready to provide data sets with event characteristics, I begin the development process with the SDA collections described in Section 3.4.1. Through these means, I develop a general methodology of an EVS tracker that does not need to integrate events into traditional images.

I draw inspiration for an EVS tracking algorithm from the multiple hypothesis trackers (MHT) first published in 1979 [83]. The foundational idea of MHTs is to evaluate multiple hypotheses in tandem until enough data are available to infer which hypotheses are true positives and which are true negatives [53]. Considering its age, the MHT was designed for frame based imagery, but the rigorous processing concept is relevant to the handling of new information that includes uncertainty about the nature of information being provided. A fundamental MHT, as depicted in Figure 6.1, is a repetitive process and has a similar

structure to a Kalman filter. The loop starts with either a priori information about the location of targets that are being tracked or the first piece of information, classically an image with discernible centroids. If the MHT is tracking possible objects (the hypotheses) these are updated in each time step. The MHT then uses the new information in the image to update the target clusters it is tracking and generate new hypotheses. For example, if the new information is within a set radius of a previously tracked centroid then it can either be the previous tracked object updated in time, a completely new object to track, or noise that should be rejected. During the process of updating clusters and generating hypotheses, clusters and hypotheses can be combined or eliminated to remove low likelihood targets. The final step of the MHT process confirms targets that have a probability of unity and removes remaining hypotheses that included information that is now confirmed to belong to a different target [83]. There are several ways to organize MHT hypotheses in a multi-target and multi-measurement process. Measurement-oriented hypotheses list every possible target for each measurement. Target-oriented hypotheses list every possible measurement for each target [83]. Alternatively, a Bayesian method employs Gaussian mixtures to build probability density functions of the joint distribution of the targets being tracked [5].

The MHT framework is the foundation for the processing methods for EVS data developed in this chapter. I view each EVS event as a new piece of information that can contribute to the understanding of the scene being imaged. On its own, the event in 3 dimensional space does not have much meaning. Unfortunately, it is improbable to tune the EVS to respond with only positive polarity events to satellites, negative polarity events to stars, and completely eliminate noise from being generated. Therefore, each event on its own has some proba-

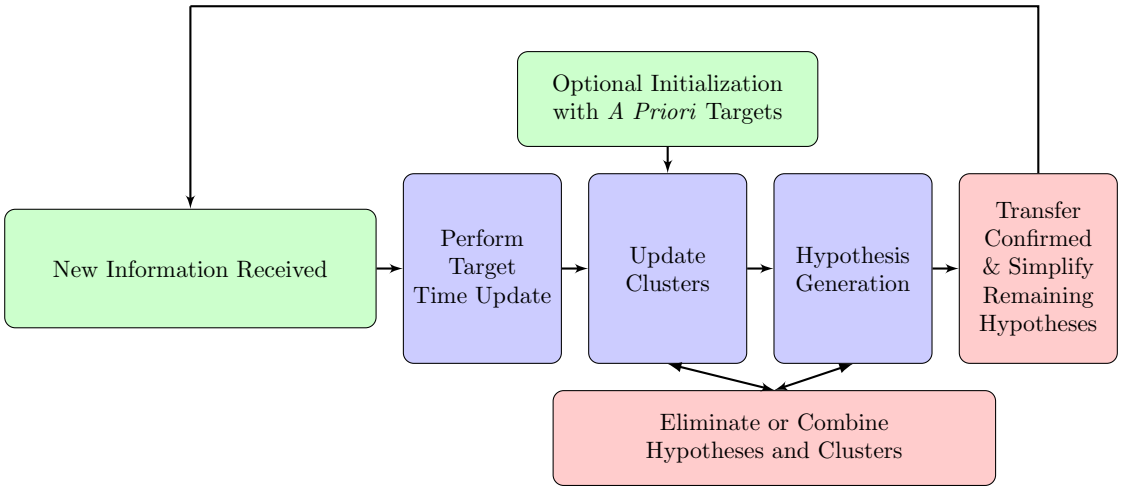


Figure 6.1: The foundation of the MHT inspires the processing of event data in this dissertation. The general flow is to use new information to generate the possible hypotheses of associated information. A traditional MHT has four steps when a new dataset is available: form new clusters, form new sets of hypotheses, reduce the number of hypotheses by elimination or combination, and segregate confirmed tracks. Through this method, low likelihood hypotheses are eliminated while high likelihood hypotheses become confirmed targets.

bility of being from any of these sources. Despite this uncertainty, the event has a 3 dimensional location in time and space which is enough context to associate it with surrounding events. Through the act of grouping events, the information about the possible object compounds over time. This grouping of all possible measurements is a target-oriented hypotheses approach. Hypotheses of varying combinations of events are possible, but not explored in this dissertation. Instead, I explore two possibilities of leveraging the clustered event information. First, I investigate pixel level rejection with the assumption the pixel profile provides a unique enough signature to distinguish between sources and, therefore, isolate signatures of interest. With this method, I develop an estimated track from pixels passing a hypothesis test threshold as shown in Figure 6.2. The second method I leverage moves away from pixel specific profiles and, instead, considers classification of clustered event information at a larger whole group

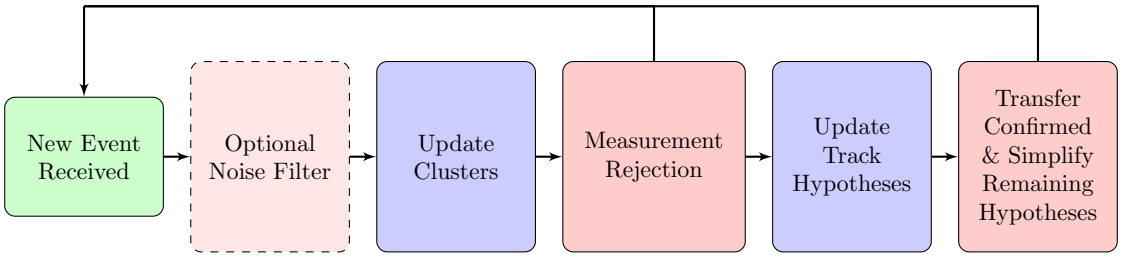


Figure 6.2: The first iteration of the EVS tracker focuses on individual measurement rejection. This process focuses on the pixel profile of events and only updates a track hypothesis with a pixel's information once the probability of being a satellite passes a hypothesis test threshold.

level to reject full groups of data before being considered for tracking.

Both of these methods are reliant on the grouping of event information beforehand. Therefore, Section 6.1 covers the two primary ways I develop to cluster event information in the spirit of an online algorithm. The proximity-based grouping derives from my existing algorithm for batch clustering of events covered in Section 3.4.3. The alternative method leverages a custom version of the Random Sample Consensus (RANSAC) algorithm to accelerate the event processing. These two methods are compared in Section 7.1.

After grouping, the data classification and leveraging for algorithmic implementation is possible. These methods are broken into two larger groups. First, I apply Bayesian methods in Section 6.2.3 to both of the aforementioned leveraging of event information, data rejection at the pixel level and overall cluster assessment. Then, I implement multiple machine learning methods in place of the Bayesian methods to evaluate the overall cluster. I outline the techniques and processing required to implement the machine learning methods I explore in Section 6.2.4.

6.1 Grouping Methods

The first step in any of the tracking implementations is grouping events. Events on their own do not provide enough information to determine anything about the originating source which stimulated the EVS. In groups, however, their individual data encode a profile into the grouped information such as space and time between events in the group and the group's arrangement of event polarities. Online grouping of events is not a trivial task. Section 3.4.3 covers multiple ways to cluster events on batch information. With the goal of leveraging the event data as they arrive via MHT, I develop online grouping methods. Section 6.1.1 adapts the proximity-based clustering method for an online implementation. Section 6.1.2, on the other hand, is a completely new methodology aimed at increasing the rate of the grouping process while also capturing the slope information of the grouped events.

6.1.1 Proximity-Based Grouping

The first method of online grouping I explore is a proximity-based method inspired by the method I develop for the batch-based clustering to label events in Section 3.4.3. In essence, Algorithm 9, searches a cylindrical volume proceeding the event being grouped and assigns the event to the closest group just like the batch clustering method. I apply a couple additional modifications, however. In more complex algorithms, the measurements would be assigned to all groups within the cylindrical volume and only removed once the measurement has probability of unity association with one of the tracks. The approach in Algorithm 9 reduces the need to aggressively prune hypotheses, but also degrades

the algorithm's robustness by not representing all possible data associations.

The first addition to the baseline proximity-based clustering method is a noise filter inspired by the high frequency of events in true signals. This is an optional noise filter. If implemented, the filter keeps an array of the most recent event on a pixel. Events only pass through the filter when two neighboring events in only the temporal dimension are within a time threshold of each other. The first event on a pixel to pass the filter also retroactively passes the previous event on that pixel to ensure its inclusion in the grouping process. This rudimentary filter operates under the assumption true signals have more than one event on a pixel in close succession. While that is not always true for weak sources, I examine the trade-offs for this non-selective filter in 7.1.1.

After the noise filter, I check the event Euclidean distance threshold to the most recent track pixels. I only check the Euclidean distance assuming the most recent pixels on confirmed tracks should have pixels with recent updates. I add the data to the closest hypothesis within the threshold. Before I include the event information at the track level, I test each pixel's probability of being a part of the track. I discuss this implementation of data rejection in Section 6.2. If I do not find a hypothesis within the distance threshold and the event's polarity is negative, I check for previous positive events on the pixel within a time threshold. If the negative event's time is within the threshold, I add the event to the hypothesis of the previous positive event. Finally, if I have not associated the data with hypotheses, I create a new hypothesis associated with the data. After processing the event(s), I then check if any hypotheses have not received a new event within a certain period, and if not, remove them as outdated.

Algorithm 9 Online Proximity Grouping

Require: $dist_{max}$, $time_{neg.max}$, $time_{keep}$
  while $t \leq t_{max}$ **do**
    $pixel = (x[t], y[t])$
    if $t == 0$ **then**
      H_1 {New Hypothesis}
    else
      for $pixel_{previous}$ **do**
        $dist = pixel - pixel_{previous}$
      end for
      if $min(dist) \leq dist_{max}$ **then**
        $H_{min(dist)}$ {Add to Hypothesis}
      else if $event.polarity = 0$ & $lastPositiveEventOnPixel.time - event.time \leq time_{neg.max}$ **then**
        $H_{lastPositiveEventOnPixel}$ {Add to Hypothesis}
      else
        H_{new} {New Hypothesis}
      end if
    end if
    for H in $AllHypotheses$ **do**
      if $H.lastEvent.time < (t - time_{keep})$ **then**
        Delete H {Remove Hypothesis due to lack of updates}
      end if
    end for
  end while

6.1.2 RANSAC Grouping

One of the downsides of the proximity-based grouping method is the time it takes to run. For each event, the algorithm checks the distance to the last pixel in each hypothesis to determine where the event should be assigned. Over time, as the number of hypotheses grow, this process becomes more cumbersome. This connection to the number of clusters is similar to the time metrics of the batch proximity-based clustering I discuss in Section 4.2.1. Ideally, the tracking algorithm should take the same or less time to process the events in a data set as the total time of the data collection. Decreasing the time to group events makes the overall tracking algorithm more resilient to increased event densities where there may be large numbers of hypotheses. With the goal of decreasing

the grouping time in mind, I explore the use of a Random Sample Consensus (RANSAC) algorithm. Algorithm 10 outlines my custom implementation of the RANSAC algorithm.

RANSAC iteratively fits a mathematical model to data containing outliers by randomly sub-sampling the data, fitting the model to the sub-sample. Then it computes the number of inliers to the model fit based on a threshold and, ultimately, selects the model with the most inliers[57]. As previously discussed, the SDA data sets I use in this dissertation have a small FOV relative to the sky. Therefore, the tracks of real signals closely approximate a line. For this reason, I use a linear mathematical model for RANSAC to fit and the definition of the line is set through the selection of two events in the sample. While the RANSAC algorithm can be applied to an entire data set in 3 dimensions, that implementation is not online and, thus, will find erroneous lines in the data. Therefore, to perform well and online, RANSAC requires customization in selecting the sub-sample of data and the inlier threshold. To decrease the number of events considered for a line's definition, I constrain the events I apply to the RANSAC algorithm to a cuboid of events prior to the event that triggers the RANSAC algorithm. I explore the bounds' impact on the grouping performance in Section 7.1.3.

In addition to the restriction of events I apply to the RANSAC method at one time, I discover there is typically a slight gap between ON and OFF events from a real source that first raises the current and then returns to the nominal background. Essentially, negative events from a source are offset in time from the positive event leading edge. So if I consider both positive and negative events, the fit of the RANSAC line may be poor in the temporal dimension. This mat-

ters because one way my implementation of RANSAC decreases computational time is by providing a line future events can be matched to as inliers. If the estimated slope is a poor approximation, it is harder to fit future events as inliers. Therefore, I customize RANSAC to consider only positive events for an initial fit, and use a time-offset line with the same slope to fit negative events. While the initial fit selects positive events from within a cuboid sub-sample of the dataset, the quantity of inliers for each RANSAC generated line are considered across the entire dataset of previous positive events. This combination reduces the number of unique combinations for line selection, but ensures the final line selection also best fits all the previous data. Practically, some time cutoff of previous events will need to be implemented for continuous online operations. However, for the approximately 20-second data sets I utilize in this dissertation, I implement no cutoff. Finally, after the initial implementation I note that many events within the sub-sampling cuboid correspond to the same source as I discuss in Section 7.1.3. Since RANSAC is not a deterministic algorithm, the points fit are randomly selected. It is, therefore, not guaranteed to pick a line equivalent to the "best fit" line as defined by linear regression. For that reason, I make the first iteration of the RANSAC algorithm a linear regression fit of all of the points in the cuboid, with subsequent iterations executing the random N-point selection, to ensure that fit is always one of the lines being considered.

Deciding what the time-offset should be to capture the negative events is non-trivial; The offset is dependent on camera settings determining responsivity and the logarithmic strength of the source signal as roughly indicated by the number of ON events for each pixel before a decrease in energy occurs. As a signal gets stronger, it requires a larger logarithmic change before triggering the

Algorithm 10 RANSAC Implementation

Require: $Number_{PointstoSelect}$, $Number_{iterations}$, $Threshold_{inlier}$, $Cuboid$
  while Iteration Number ≤ $Number_{iterations}$ **do**
    if Iteration Number = 1 **then**
      Select all positive events in $Cuboid$ as sub-sample to fit
    else
      Select $Number_{PointstoSelect}$ positive events from $Cuboid$
    end if
    Compute line via least squares fit to selected events
    Compute number of positive events in all dataset with distance ≤ $Threshold_{inlier}$ to line
    $Residual_{positive}$ = Mean of inlier event distances to line
    if Number of inliers > Best previous number positive inliers
    or
    Number of inliers = Best number positive inliers **and** $Residual_{positive}$ < Previous best mean positive residual **then**
      Compute Negative Event Time offset
      Offset line model by negative time offset
      Determine which negative events in all dataset are inliers to the offset line
      $Residual_{negative}$ = Mean of inlier event distances to line
      if Number of inliers > Best previous number negative inliers
      or
      Number of inliers = Best number negative inliers **and** $Residual_{negative}$ < Previous best mean negative residual **then**
        Save line models as best fit found
        Save number of positive and negative inliers and average residuals
      end if
    end if
    Iteration Number = Iteration Number + 1
  end while
  return Best Line Models (positive and negative), Inliers

first OFF event. The refractory period and the OFF threshold setting contribute to the needed amount of change. I experimentally determine a linear relationship between the number of ON events and the time-offset for the OFF events line, so that offset could be adjusted. The difficulty in implementation is that not every pixel will have 100% of a point source cross through its pixel area, so not every pixel has the same number of ON events in a group. As a con-

sequence, the offset is not consistent within a group. To estimate a reasonable offset to capture most OFF events, I average the number of ON events within the group. Algorithm 11 outlines the current implementation. To get an offset for the negative time line, I average the number of positive events on each pixel and use that number in a linear model. I discuss the linear model's coefficient and intercept determination in Section 7.1.3.

Algorithm 11 Compute Negative Event Time offset

Require: $Sequence_{events}$, $slope$, $intercept$
$Pixels_{unique}$ = number of unique pixels in $Sequence_{events}$
$Avg_{eventsperpixel}$ = length($Sequence_{events}$) / $Pixels_{unique}$
$Timeoffset = slope * Avg_{eventsperpixel} + intercept$
return 0 - $Timeoffset$

I do not apply the full custom RANSAC algorithm on each event to reduce computational time. The output of RANSAC is a linear definition that I next leverage to reduce the number of times the full RANSAC algorithm runs. To implement the overall online RANSAC-based grouping algorithm, Algorithm 12, I trigger the following sequence with each event: group cleanup, event attribution, group combination, and preliminary classification.

The first step is group cleanup. The intention of group cleanup is to prune existing groups that have low event numbers or have not had new events added to the group within a certain time span. In this process, the algorithm looks for groups that have too few events to fit a line and removes them. I release the events in these groups to be considered in future grouping. As for groups that have not been updated recently, these can be groups of noise that have not picked up recent events or they can be legitimate signals that are now out of frame and will not be collecting more events. If a group times out, the group is archived so as not to be considered for future event attribution and the events

Algorithm 12 RANSAC-Based Grouper

Require: *event*, $threshold_{inlier}$
  Remove groups with not enough events
  Archive groups that have not been updated recently
  Attributed = False
  if *event* distance to nearest group ≤ $threshold_{inlier}$ **then**
    Associate event with group
    Attributed = True
  else if *event* is negative and *event.time* ≤ max time of negative event association **then**
    Associate event with group
    Attributed = True
  else
    Attributed = **Attempt Refit Of Nearest Group**
  end if
  if Not *Attributed* **then**
    Cuboid = **Compute bounding cuboid** around *event*
    $model_{post}$, $model_{neg}$, *inliers* = **RANSAC implementation** with *Cuboid*
  end if
  if *event.time* > time since last star group determination and line combination **then**
    Combine Nearby Parallel Groups
    Compute Star Groups
  end if
  for Updated Group in All Updated Groups **do**
    if Updated Group is in Star Groups **then**
      Classification of Updated Group = Star
    else
      Classification = Classifier predict class of Updated Group
    end if
  end for

within the group are not considered for future groups.

In the next phase, I attribute the new event to a group. Like the proximity-based grouping algorithm, I first try to attribute the event to an existing group instead of creating a brand new one. I identify the nearest group and attribute the event to that group if its 3-dimensional distance is within the inlier threshold of the group's line with the same polarity as the event. Associating an event

with a group registers the event as belonging to the group in a group attribution map. If the event is positive, it re-computes the least-squares line fit of the groups positive events, and fits a new time-offset negative line as Algorithm 13 outlines.

Algorithm 13 Associate Event with Group

Require: *event*, *group*
  Group Map(*event*) = *group*
  if *event.polarity* is positive **then**
    Group Slope = Compute line via least squares fit to positive events in group
    Group Negative Slope = Re-compute negative event line via time offset from positive event line
  end if

If the event is not close enough to an existing group line, I try to attribute the events in two different ways. These ways are dependent on the polarity of the event. First, negative events tend to have a longer tail of associated events in time before being exhausted due to the use of the logarithmic scale. This creates a longer offset from the approximate leading OFF event fit line to the final OFF events in a group. Therefore, if the event is negative and fails to be attributed to the nearest group, I check for previous positive events on the event's pixel. If there is a positive event is within a chosen interval, I attribute the negative event to the positive event's group.

I handle positive events differently when they do not match to an existing line. If the new event has positive polarity and is unattributed, the nearest group attempts to re-fit to a line that would make the new event as an inlier, as I document in Algorithm 14. I only run RANSAC against the group and the new event that is trying to associate. If RANSAC returns a fit retaining greater than a quarter of the original events with lower residuals while also fitting the new event, a new group is created by including the inlying points and any outliers are dis-

associated from the old group. Before attempting to refit the nearest group to an event, I check that the event is a reasonable match to the closest group. This check defines a cone from the most recent event in the group. Thus, as time increases a greater distance in (x,y) passes the check. If the new event is within the distance threshold at the offset in time, then I run the refit of the nearest group. The time-scaling distance threshold assists with catching events on inconsistent source signals, such as satellites with some periodicity to their reflected light, weak sources impacted by atmospheric transmission, and sources with their spread of light not going directly over the center of a pixel.

Algorithm 14 Attempt Refit of Nearest Group

Require: *event*, *group*, *eventDistanceFromGroup*
  Threshold = Calculate modified threshold based on time of last group update
  if *eventDistanceFromGroup* ≤ Threshold **then**
    $residuals_{old}$ = Compute residuals of group
    $model_{new}$, *inliers*, $residuals_{new}$ = **RANSAC Implementation** group & new event
    if *inliers* ≥ 1/4 * # old inliers **and** $residuals_{new} \leq residuals_{old}$ **then**
      Create new group with new inliers
      Disassociate all events from old group
      return Attributed = True
    end if
  end if
  return Attributed = False

Now if the new event passes all the previous gates attempting to associate it with existing groups of events, I call the original RANSAC from Algorithm 10. These previous checks prevent the RANSAC algorithm from running unnecessarily if a group already exists where the event could reasonably belong. Algorithm 15 selects the events within the cuboid bounds before the current event. It returns a filtered list of events that are within a distance in x or y dimension of a center point, the event, and within a certain amount of time prior to the event. Note that the RANSAC implementation attempts to fit any line

to all events within the cuboid, but it is not required to fit the new event as an inlier. As aforementioned, the line selected has the greatest number of inliers in the entire list of previous non-archived events. All events may switch from a previous group to the new line. I note any groups that change as updated during the Attribution phase for potential re-classification.

Algorithm 15 Compute Bounding Cuboid

Require: $radius$, $timeDepth$, $center$, $Sequence_{events}$
$output = Sequence_{events}$ where ($event.x \geq center.x$ - $radius$ **and** $event.x \leq center.x$ + $radius$ **and** $event.y \geq center.y$ - $radius$ **and** $event.y \leq center.y$ + $radius$ **and** $event.time \geq center.time$ - $timeDepth$)
return $output$

After the attribution phase, I apply one more round of group maintenance prior to classification. Instead of pruning groups as in the first step of Algorithm 12, this process combines groups. I apply the methods in Algorithm 16 and Algorithm 17 periodically, triggered by a fixed time to both combine lines and re-evaluate star groups. The combination algorithm considers groupings that are within half of the inlier distance from each other and within a set angular slope measurement across all axes (x, y, time). If groups meet both these conditions, the algorithm combines them into a single group. If two groups form that are close in their linear traversal of the 3-dimensional space, they actually could be redundant where a slight spatial deviation in time generates two separate groups. Algorithm 16 helps manage the generation of duplicates by periodically combining groups meeting these conditions.

The second algorithm is the first step in the classification process. To reduce the load of the classifier, I rely on the fact the stars are essentially fixed points in the barycentric coordinate system over short periods of time. The movement of the sensor, therefore, results in the star motion across the focal plane. De-

Algorithm 16 Combine Nearby Parallel Groups

Require: $threshold_{inlier}$, $angle_{maxcombine}$
  for *group* in *allGroups* **do**
    for *othergroup* in *allGroups* **do**
      if *group* ≠ *othergroup* **then**
        if 3D distance between *group* and *othergroup* ≤ $threshold_{inlier}$ /2 **then**
          if $\sum(\|angle_{group} - angle_{othergroup}\|) \leq angle_{maxcombine}$ **then**
            attribute *othergroup* events to *group*
            update *group* line models and negative time offset
          end if
        end if
      end if
    end for
  end for

spite some distortion towards the edges of the field of view, the resulting lines are fairly consistent in 3-dimensional space. Earlier event-based tracking algorithms leverage this aspect in the data, but do not go so far as classification [1]. In my implementation, I periodically identify likely stars by the median slope in the data and only apply extra scrutiny to slopes deviating from this pattern. Algorithm 17 identifies the groups close to the median slope and through periodic updates determines which groups should pass to the classifier's higher level of scrutiny. The grouper compares the slopes of all of the groups. It computes star groups as being within the Nth percentile difference from the median group slope. Then it passes groups in the upper 1 - Nth percentile difference from the median slope on to the classifier. Overall this method decreases the number of groups requiring evaluation by the classifier. I cover this method's impact on performance identifying star groups in Section 7.1.3.

Algorithm 17 Compute Star Groups

Require: *percentile*
  Compute Median motion vector of all groups
  Compute difference from each group's motion vector to the median
  Designate groups with difference ≤ *percentile* of median as star groups

After all the proceeding steps to group the events, updated groups that are deemed not a Star group are evaluated by a classifier. I describe these classification methods in Section 6.2.

6.1.3 Measuring Grouping Performance

Before applying a classifier, I compare the two grouping methods against the batch clustered data processed with methods I outline in Section 3.4.3. Against the truth batch clustered data, I consider multiple factors to evaluate grouping performance at both an individual event and final group levels.

At an event level, there are 4 factors I consider to describe the grouping performance. First, events are either in a group or not in a group as associated during the course of the online grouping methods. Depending on the event, being in a group or not in a group is the proper condition. Next, if an event is in a group, the event is either grouped properly with associated events or improperly with another real signal and/or noise events. If an event is grouped with noise events, the event may still belong to a group that corresponds to a real signal. Therefore, it is important to determine whether a group consists of a majority of noise events, whether or not it also contains events corresponding to a real signal. Finally, if a noise event is part of a group, it can belong to a real signal group or a group of noise.

At a group level, I consider two factors that describe the grouping performance. First, in the final groups, a one-to-one match with the truth clustered data indicates if the method generates no duplicate groups or additional noise groups. In addition, if any groups have a majority of noise events, then the

grouping method produces one less group with respect to the total number of the truth groups.

Given these considerations, I develop 10 metrics for grouping performance: 7 specific to individual events and 3 to full groups. I use these definitions in the grouping performance discussion found in Section 7.1. The 7 specific to individual events, organizes each event into a category that captures the aforementioned possible behaviors for events. Summation of the category totals results in the total number of events. The truth batch clustered data contains a group label for each event. While the labels will not be one-for-one, I leverage the information about the events association with other events to evaluate the grouping performance at the event level. I apply this comparison at the end of the data set for the online methods. Out of the 7 metrics capturing the event level performance, the first 3 categories are for noise events: noise events grouped with a majority real source event group, noise events grouped with a majority noise event group, and noise events not grouped. The first adds up noise events included in a grouping that consists of a majority of events corresponding to a real source. The second finds noise events included in a grouping that consists of a majority of events that are noise. The final totals the noise events correctly determined not to be associated with a group. The second 4 categories are for real source events: real events included in a correctly identified real event group, real events included in an incorrectly identified real event group, real source events included in a majority noise event group, and real source events not grouped. The first and second categories capture the events grouped with real sources. Unlike the noise category that checks the same, I delineate between events belonging to the same majority truth data label and those that belong to a different label. I do not need to check for the same condition with the noise

events because they are all in the category of not belonging to the majority truth label. The other two categories mirror the noise event categories with the third totaling real source events that belong to a majority noise event group and the forth adding all incorrectly non-grouped real source events.

The 3 full group metrics are at a higher level evaluating the number of erroneous groups the online grouping creates. These metrics include the duplicate group count, the duplicate noise group count, and the majority of noise groups. The first metric totals the number of incidents where multiple formed groups map to a single label in the truth data which I call duplicates. This metric includes majority noise groups. The second metric is a subset of the first, totaling the duplicate groups that map the noise group label and helps identify the number of real group duplicates from the overall duplicate metric. The final evaluation metric totals all the groups that have a majority of noise events which evaluates the tendency for the online method to group noise events.

I take these measures of event grouping and form a metric to evaluate the percentage of events matching the truth label of an event from a real source or noise. I call this group optimality, $group_{opt}$.

$$group_{opt} = \frac{r + n}{event_{tot}} \tag{6.1}$$

is the amount of events that are correctly grouped into the larger categories of real sources or noise divided by the event total, $event_{tot}$. I determine the total correctly grouped events through the aforementioned measures where r is the real event grouped with correct real event group category and n is the number of noise events that are not grouped.

In some circumstances, groups consisting mostly of noise present a classifier the chance to reject the events. In fact, when grouped they are not tied to the

physics that appear to give all stars similar slopes and, therefore, are more likely to go through formal classification. While it is ideal to reject noise events at the grouping stage, it is interesting to examine the noise rejection performance in certain portions of the analysis in Section 7.1. Given fixed grouping parameters, the incidence of grouped noise goes up, so noise rejection by the classifier indicates robust performance of the overall tracking algorithm. The second group optimality metric

$$group_{opt2} = \frac{r + n + m}{event_{tot}} \tag{6.2}$$

adds the number of majority noise grouped events, m to the total events in the numerator of the first optimality metric. Despite being grouped, the labels of these events are considered as correct after the classification portion of the tracker. Therefore, this metric captures overall online algorithm performance assigning events between real sources and noise.

6.2 Group Classification

The next step in the online tracking process, after grouping each event, is classifying the groups. Classification takes the grouping of events further by leveraging the group's information to infer what in the scene should pass classification to be applied in eventual track estimation processes. I design the classification to work in an online process that adds intricacies to the implementation. Not only do I explore the methods of classification, but I implement classifiers that operate on pixel and group level information. This Section covers the intricacies of each type of classification process implemented. Understanding the methods I apply prepares for conclusions about classification methods applied to the tracking algorithm to be drawn from the performance analysis.

I start this process with a discussion of the working principle of data rejection I apply through the use of a classifier in Section 6.2. Then I cover the attributes within the data sets in Section 6.2.1. The data attributes are the inspiration for leveraging classification within the tracking process. After discussing the data attributes I find, Section 6.2.2 describes the partition of the data set for training and validation before application of any methods to ensure that they are comparable. Subsequently, Sections 6.2.3 and 6.2.4 introduce the two overarching classification methods, Bayesian statistics and machine learning respectively. Finally, I cover the metrics I use to evaluate the classifier performance in Section 6.2.5.

Data Rejection Principle

One of the main steps of my proposed tracking algorithm in Figure 6.2 is the use of the grouped event information for informed data rejection. I originally derive this premise from the polarity profile of events on each pixel. The combination of ON and OFF events from pixels experiencing the same point source, assuming the source is subpixel in size, should have similar trends throughout a track. However, as I discuss in Section 6.2.1, there are more qualities in the timing of events, not just the polarity profile, which can identify sources of interest. I apply the data rejection principle in two ways in this dissertation. First, comes from the original premise, I evaluate individual pixels of a group to determine if their event information should contribute to a track estimation process. The second approach moves away from thresholding pixels and instead evaluates the group as a whole.

In the pixel data rejection method, I test each pixel's probability that the pixel

is from a satellite through a joint conditional probability test. Online, this test occurs after the addition of an event to a pixel within the same group. Once a pixel receives new information, the pixel probability is re-evaluated. If the probability surpasses a chosen threshold, the pixel moves to a list of pixels associated with the group that constitutes the satellite track. Once two pixels in one group reach the hypothesis threshold and are added to the track, I create a linear estimate in the 3-dimensional space for the next pixel's leading edge of ON events and compare it to the subsequent observations in Section 7.2.1.

Because of the variation in signal strength on pixels and other attributes proving to be more important than the pixel profile in the classification of satellite sources, I turn to overall group classification to improve performance of the data rejection method. The machine learning models I discuss in Section 6.2.4 take this approach. While not implemented, the track generation from the classifier method will be different than individual pixel thresholding. To recreate the linear estimate, instead of evaluating the slope given a new pixel and its leading edge information, I will use the whole group's worth of information. When the overall group passes the threshold, all of the current events contribute to the slope estimate. With the RANSAC implementation of grouping, I can use the RANSAC line information. For the proximity-based grouping, the leading edge of events on each pixel provides the needed information. As time progresses each new event may adjust the RANSAC estimate or contribute to the leading edge of a group currently classified as a satellite.

Moving forward with either pixel or group based data rejection, the track estimation will move beyond a linear estimate. Whether star or satellite tracking, the (x,y) pixel coordinate system is not useful on its own to provide any

information about where the observer's EVS is pointing or what the satellites' positions and velocities are over time. After data rejection, the remaining source information will pass to a Kalman filter. I leave this portion of the tracking algorithm to future work as I discuss in Section 8.3.5. However, I mention the future of the tracking algorithm here to emphasize the importance of the data rejection step. Accurately reducing the event-based data through grouping simplifies the implementation of a star or satellite tracking algorithm. It limits the possible matches to a finite number of groups.

6.2.1 Data Attributes

I explore the attributes of my batch clustered data, processed with the method outlined in Section 3.4.3, to inform the proposed methods of informed data rejection before applying any classification method. From this data, I search for characteristic attributes that can be leveraged for algorithm development because they distinguish between sources. While much of the focus in this dissertation is to differentiate satellites from other signals for the purposes of SDA, as a consequence the methods I develop also differentiate star sources. If the sensor is onboard a satellite or other moving object, identifying stars for an event-based star-tracker algorithm is of particular interest. Therefore, all the attributes I outline in this Section are also useful for star tracking purposes.

Figure 6.3 is an example data set in a reassembled frame over an approximately 22 second collection. The Figure expresses polarity as blue for positive events and red for negative. From visual inspection, the streak in the upper right hand corner traverses the projected plane in a different direction than those in

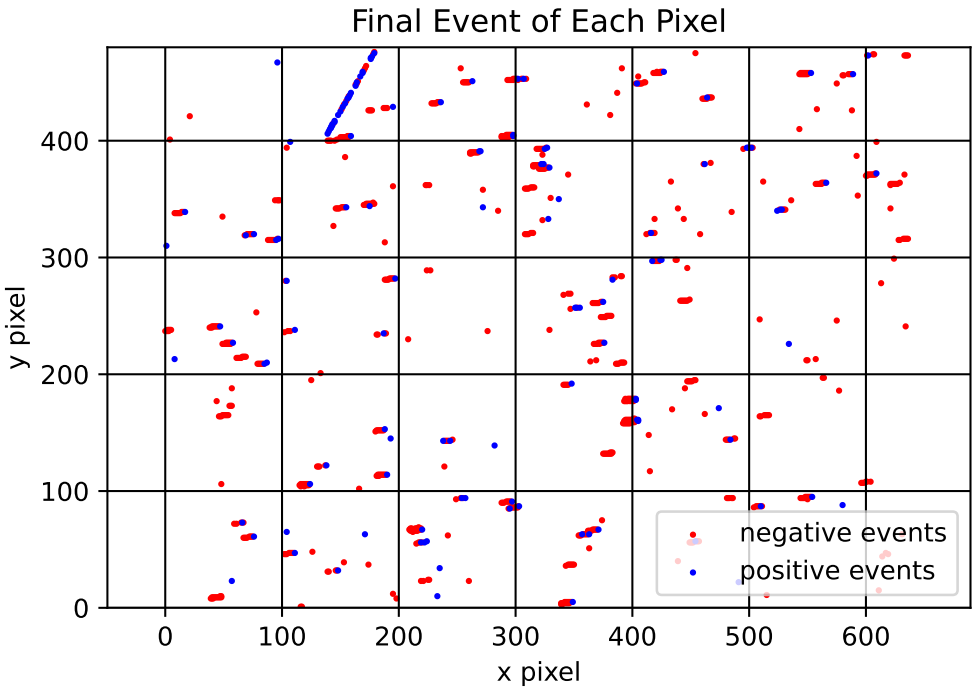


Figure 6.3: Through examination of a 2 dimensional projection of the final events on a pixel through one of the processed data sets, I can identify attributes in the data that have potential to assist with classification such as consistent star directions and the common final satellite event being positive.

the middle of the field of view. This is the satellite, whereas the other lines are stars. I have leveraged this spatial characteristic to develop the batch cluster and online grouping methods. Figure 6.3, however, provides some additional insight into what attributes distinguish a star from a satellite. The individual pixel event profiles of the stars almost uniformly end in a negative event, while some of the final satellite pixels end in blue indicating a positive event. Therefore, the ordered event profile helps discriminate between the two types of detected RSOs. The figure also shows the density of the star events leaves little or no gaps between the pixels whereas the satellite track has gaps. The satellite may have a different flux and motion with respect to the sensor than the stars. Therefore, the rate of travel and distance between events in a track also

differentiate the satellite from the star pixels. While it cannot be perceived in the two-dimensional format displayed in this figure, the duration a pixel receives information from a satellite can also be a discriminating characteristic.

The other events are presumed to be from hot pixels and noise. The attributes I propose are sufficient to discriminate these types of events. Hot pixels produce temporally long strings of events that surpass the length of all other types of detected events. Noise events, on the other hand, have a lower frequency than those from true signals. Neither of these types of detected signals traverse to multiple pixels sequentially, so they are easily identified when spatially isolated.

I explore these characteristics more by examining pixels within the groups in the batch clustered data. Taking each pixel within a group, I add the total ON events, OFF events, and the events overall that attribute to the group. Then I evaluate the average the time between events and derive the total time as the difference between the first and last event on the pixel for the group. Finally, I calculate the distance and time to the next pixel in the group based on the leading edge of events in the group. Evaluating these characteristics for each pixel within a group in the data sets, I have a list of pixel level attributes to compare. I evaluate the correlation between these characteristics as reported in Table 6.1.

Unsurprisingly, there is a high level of correlation between the total events. Stronger sources produce more ON events, so it takes more OFF events for the signal to decay back to the nominal level and, therefore, there are more events total. This also, typically, produces a longer total pixel time. As such, having all four attributes as components in a classifier evaluating the pixel level pro-

vides minimal additional information for the inference. The other attributes, the frequency of events occurring on the pixel, the distance to the next pixel in the group, and the rate at which the next pixel joins the group or the speed of the object, are less correlated to the pixel's event totals. These attributes, consequently, have a higher chance of providing additional valuable information for inferencing the event source on the pixel. The weak correlation between the average time between events and other attributes shows that Mahowald's expectation of information being encoded between the events is both true and verifies its utility for classification. In terms of Bayesian inferencing, the lack of correlation also supports an assumption of independence between variables for the joint conditional probability evaluation.

Table 6.1: Pixel attributes' correlation demonstrates the redundancy of some of the attributes which decreases the additional input they provide to the classifying process. Blue cells indicate higher correlation and red cells indicate less correlation.

	ON Events	OFF Events	Total Events	Avg Time B/W	Next Pix Dist	Pix Total Time	Next Pix Rate
Total ON Events	1.000	0.682	0.933	0.034	0.039	0.605	-0.001
Total OFF Events	0.682	1.000	0.900	0.072	0.030	0.569	-0.001
Total Events	0.933	0.900	1.000	0.056	0.038	0.641	-0.001
Avg Time B/W Events [sec]	0.034	0.072	0.056	1.000	0.014	0.655	-0.001
Next Pixel Distance	0.039	0.030	0.038	0.014	1.000	0.034	0.179
Pixel Total Time [sec]	0.605	0.569	0.641	0.655	0.034	1.000	-0.001
Next Pixel Rate [pixel per sec]	-0.001	-0.001	-0.001	-0.001	0.179	-0.001	1.000

Taking this same analysis to the group level, I change the attributes evaluated to handle multiple pixels as Table 6.2 summarizes. This time I add up all the ON events, OFF event, and total events across an entire group in the batch data sets. Because I have the pixel information, I average the ON, OFF, and total events from the pixel level to see if they correlate to the group level totals. In addition to the total and average pixel totals, I also take the ratio of ON and OFF events at the pixel level and average that ratio for all pixels in each group. The last four metrics average the time between events, total pixel time, next pixel distance, and next pixel rate from the pixel level attributes to see how they

correlate at a group level.

Interestingly, the total ON events at the group level is less correlated to the total OFF events and the total events overall than the pixel level averages. One possibility causing this effect is the variance of pixel profiles across the group. The average profile may lean towards the pixels with the strongest signals and, therefore, better reflect the total ON events in the group. The weaker correlation between the average events and the OFF and total events suggests the individual pixel profile may still have some utility at the group level. Keep in mind, pixel profiles are lists of ON and OFF events and are difficult to combine into a group level profile. This is one of the primary reasons I first explore pixel level data rejection. I try one possible way to capture the pixel level profile by evaluating the ratio of ON to OFF events on a pixel. This profile substitute cannot distinguish between signatures that are purely ON events followed by OFF events from something more complex. This attribute shows promise by being mildly correlated to the average event totals and less correlated to the group event totals.

The final four group attributes retain their status from the pixel level attributes. The average total pixel time adds minimal information beyond another attribute dependent on the pixel polarity profile. The average time between events, average next pixel distance, and average rate of new pixels still remain weakly correlated to the other attributes and should be considered favorable for classification purposes. Overall, the group level characteristics' correlations suggest there are more options of attributes to inference with than at the pixel level alone. Additionally, some level of connection to the pixel profiles is possible. In light of these characteristics, I consider group level classification with

the machine learning models.

Table 6.2: Group attributes' correlation demonstrates the redundancy in the same attributes as a pixel classifier alone, but provides more options with less correlation. Refer to Table 6.1 for color definitions.

	ON	OFF	Total	Avg ON	Avg OFF	Avg Total	Avg ON/OFF	Avg Time B/W	Avg NP Dist	Avg Pix Total Time	Avg NP Rate
Positive Events	1.000	0.407	0.526	0.579	0.571	0.602	0.208	0.080	-0.025	0.542	0.000
Negative Events	0.407	1.000	0.991	0.036	0.071	0.054	-0.052	-0.028	-0.026	0.041	0.000
Total Events	0.526	0.991	1.000	0.119	0.150	0.138	-0.018	-0.015	-0.028	0.118	0.000
Avg Positive Events	0.579	0.036	0.119	1.000	0.825	0.967	0.454	0.199	-0.057	0.914	-0.002
Avg Negative Events	0.571	0.071	0.150	0.825	1.000	0.941	0.123	0.112	-0.044	0.771	-0.002
Avg Total Events	0.602	0.054	0.138	0.967	0.941	1.000	0.327	0.169	-0.054	0.892	-0.002
Avg Positive Negative Ratio	0.208	-0.052	-0.018	0.454	0.123	0.327	1.000	0.166	-0.019	0.359	-0.002
Avg Time B/W Events [sec]	0.080	-0.028	-0.015	0.199	0.112	0.169	0.166	1.000	-0.030	0.457	0.002
Avg Next Pixel Distance	-0.025	-0.026	-0.028	-0.057	-0.044	-0.054	-0.019	-0.030	1.000	-0.057	0.279
Avg Pixel Total Time [sec]	0.542	0.041	0.118	0.914	0.771	0.892	0.359	0.457	-0.057	1.000	-0.002
Avg Next Pixel Rate [pixel per sec]	0.000	0.000	0.000	-0.002	-0.002	-0.002	-0.002	0.002	0.279	-0.002	1.000

Now that I have a list of the possible classifying attributes, I can prepare the data for use in both Bayesian and machine learning techniques. For both methods, I require each event to be labeled according to their class of detection. I break the classes into 4 categories: hot pixel events on pixels that erroneously output many events not associated with true signals, noise events that do not correspond to other groups, star events for groups with star labels, and satellite events for groups with satellite labels. For the Bayesian method, I create discrete conditional probability tables relating the probability of a type of detection given each attribute: the event profile, average time between events on a pixel, total time a pixel is active, average distance to the closest event, and rate that new pixels are added to the track. Currently, I do not reduce the attributes I consider. I describe the creation of these tables in Section 6.2.3 and the tables are available in Appendix A. In contrast, the machine learning methods require data preparation dependent on method and augmentation to balance the discrepancy between the number of satellite events and the noise and star totals. Section 6.2.4 covers these preparatory steps.

6.2.2 Training, Validation, and Testing Data

Once I sort the data into detection classifications, for purposes of being able to evaluate the efficacy of my approach and being able to compare performance of multiple algorithmic approaches, I separate the real datasets into training and validation sub-sets, with a 70:30 split. I document the data sets in the training and validation sets in Appendix B. Eventually, I intend to expand the overall data set and provide testing data with simulated scenarios as I describe in Section 8.3.4.

6.2.3 Bayesian

The Bayesian inference method that I outline in this Section functions on the principle of data rejection at the pixel level and specifically aims to identify satellites from the other groups. First, I discuss the implementation of Bayesian inferencing. Then I describe how I tune the threshold of my hypothesis test. Finally, I cover the online hypothesis test implementation and subsequent track generation.

Probability Tables

I utilize the 70% split the processed data described in Section 6.2.2 as the prior knowledge required to develop a hypothesis test. The Bayesian use of this training data is different than for the machine learning methods because I must define explicit attributes in the data to use for inferencing. From the training data with the attributes I describe in Section 6.2.1, I extract discrete conditional prob-

ability tables to apply in the evaluation of the joint conditional probability.

Each data attribute provides information to infer the possibility the data come from a satellite. In order to leverage multiple data attributes, I employ Bayes theorem, which defines the conditional probability of A given B as

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{6.3}$$

The probability of B can be evaluated as

$$P(B) = P(B|A)P(A) + P(B|\sim A)P(\sim A) \tag{6.4}$$

where $\sim A$ is the possibility other than A. This can be further expanded for multiple pieces of information using

$$P(A|B \wedge C) = \frac{P(B|A)P(C|A)P(A)}{P(B|A)P(C|A)P(A) + P(B|\sim A)P(C|\sim A)P(\sim A)} \tag{6.5}$$

with the assumption that the information in B and C are independent [102]. In my specific application, I calculate the probability that the detection is a satellite, the alternative hypothesis, with the assumption that the chosen data characteristics are independent. The discussion in Section 6.2.1 supports this assumption.

In order to implement the joint conditional probability, I must first generate conditional probability tables from the clustered data classifications. I assign each event a class: hot noise, noise, star, and satellite. I use these class labels for both the machine learning and Bayesian methods. In the case of the Bayesian methods, I discretely organize pixel level counts of each class by the values of the attributes for the pixel level Bayesian classifier. Two of the attributes I describe in Section 6.2.1 are discrete in their definitions. First, the polarity profiles are lists of 1s and 0s capturing the ON and OFF events of a pixel. There are a finite number of combinations for each length of profile. The other discrete attribute is the next pixel distances. These distances are discrete in their definition

following the distance formula

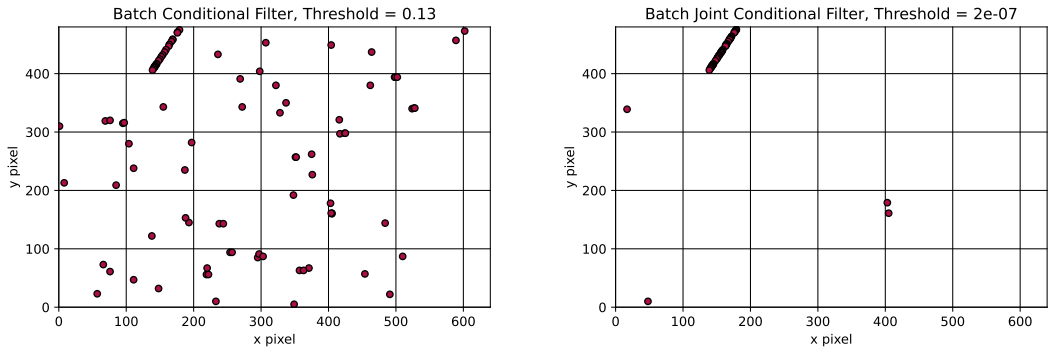
$$d = \sqrt{x^2 + y^2} \tag{6.6}$$

where the distance, d, is a combination of integer pixel distances in the spatial dimensions of x and y. The continuous attributes, all associated with time, are average time between events, total pixel time, and new pixel rate. I separate these attributes into discrete bins to enable the generation of their class probability tables. Repeated values are unlikely with continuous variables, so binning the values allows for accumulation of the probabilities in discrete ranges. After binning for both discrete and continuous attributes, I sum the counts of an attribute for each class. Then I divide the counts of each discrete binned attribute in each class by the total of that class. The resulting table represents the probability of each attribute's profile or binned values for a given class, which is the conditional probability $P(B|A)$. For example, Table A.2 in Appendix A reports the conditional probability of the four classes of event sources detected given the pixel profile derived from the SDA data set I leverage in this dissertation. It should be noted, these values are dependent on the sensor and the sensor settings and should not be applied directly.

Batch Hypothesis Filter

After the development of discrete probability tables, I choose an appropriate decision threshold to formally define the satellite inferring hypothesis test for the initial implementation of the pixel-thresholding tracking algorithm. I inform the choice of a threshold by employing the joint conditional probability filter on the clustered data sets in a batch format. Employed on the total data set, the filter calculates the probabilities with the prior grouped information, which is

with the greatest amount of pixel level information available in each data set. This method provides a quick way to estimate final performance of the measurement rejection. Lower hypothesis test thresholds that isolate the satellite track in the batch method increase the chance that the online probability calculation reaches the targeted threshold before a full 30 seconds of data are available. The lower threshold is a trade-off as it may also let true negatives, star tracks, noise, or hot pixels, register as satellites in the early data sets. I select the threshold value from inspection of the batch joint conditional hypothesis filter plots that balance the rejection of true negative measurements and retention of the true positive measurements. The threshold is unique to these data sets because it is sensitive to the group attributes that are a function of the sensor and bias settings. Therefore, the absolute magnitude of the probability threshold chosen is inconsequential.



(a) Threshold = 0.13

(b) Threshold = 2e-7

Figure 6.4: a) Conditional probability based only on the event profile. This is the maximum threshold that allows data past the hypothesis test. b) Joint conditional probability considering other data factors reduces false positives in the data association.

Figure 6.4a depicts an implementation of the batch data filter that only considers the conditional probability due to the event profile. The probability of the true positives and true negatives overlap. The threshold depicted is the maxi-

mum for any data to pass the hypothesis test because pixels separate from the satellite track have the same polarity profiles as some of the satellite data. Therefore, it is not possible to completely remove star detections with the conditional probability hypothesis test that considers only the event profile on a pixel.

Consideration of multiple data attributes reduces the probability overlap, hence the improvement from Figures 6.4a to 6.4b. However, each binned attribute is not perfectly unique to satellites. Some attributes that are high probability for satellites may also have a substantial probability for other classes. Therefore given the right combination of attributes, some overlap in class probability still exists. The multiplication of multiple conditional probabilities makes the joint conditional probability threshold much lower than if considering only the pixel polarity profile. As a result, the joint conditional probability in Figure 6.4b does not have perfect performance in terms of null hypothesis measurement rejection.

I only use the threshold chosen from inspection on the first iteration of the tracking algorithm. I modify selection of the threshold to a choice informed by a receiver operating characteristic (ROC) curve, plotting true positive rate (TPR) against the false positive rate (FPR) in subsequent versions of the algorithm. I generate the ROC curves by running the algorithms with different threshold settings and subsequently calculating the final TPR and FPR values. I also apply the ROC curve approach to the machine learning classifier output. Section 7.2.4 contains these curves and the thresholds I select for these data sets. It is important to note, selection of a threshold value from an ROC curve is still subjective to the user's desired output. Generally, the maximum TPR to FPR ratio indicates the best classifying performance. However, selection of that ra-

tio can reject more true positives in favor of rejecting false positives. For these algorithms, I select the thresholds from the ROC curves that maximize the total count of true positives, thus maximizing the satellite information passing through the classifier's hypothesis test.

Bayesian Implementation

In the online algorithm, after grouping, I apply the Bayesian classifier to the pixels within a group that gains a new event. Since I only apply each new event to a single group, there is only one group that goes through the classification process. Algorithm 18 is my original implementation of the joint conditional probability based hypothesis test developed on the batch information. I evaluate each pixel in the group during the update step because the spatial attributes of the group are modified with the new information. I keep a running count of pixels satisfying the hypothesis test. A track update triggers when the count increases in value. The pixels above the threshold are considered part of the possible track and are added to a track list.

Essentially, the hypothesis test applies measurement rejection to the incoming data until the confidence in the information is high enough to form or add to a track. From the pixels in the track list, I estimate the next pixel by regressing a line and using the average Manhattan distance in the x direction to estimate the next y value. I prune the output tracks after the probability update. Tracks without new information within a chosen time frame or if all of the track pixels fall below the probability threshold after subsequent updates, I remove it from the possible track list. Pruning at the track level acts as a final filter to remove unlikely tracks where a few pixels of non-satellite groups share attributes with

satellite pixels and make it through the classification step.

Algorithm 18 Probability Update

Require: $prob_{minimum}$
  for $pixel_{H_{current}}$ **do**
    $prob_{current}$ {Run Joint-Cond Prob}
    if $prob_{current} \geq prob_{minimum}$ **then**
      $count = count + 1$ {Threshold Count}
    end if
  end for
  if $count \geq count_{previous}$ **then**
    for $pixel_{new}$ **do**
      $track_{update} = pixel_{new}$ {Update Track}
      $error_{estimate} = pixel_{new} - est_{previous}$ {Estimate Error}
      $y = m * x + b$ {Fit Data to Line}
      $est_{new} = m * x_{diff} + b$ {Estimate Next}
      $count_{previous} = count$ {Update Count}
    end for
  end if

I modify the implementation of the Bayesian classifier at the group level to more closely match the output of the machine learning classifiers. Instead of thresholding at the pixel level to build an overall track, I keep the pixel level inferencing and allow pixels to vote at the overall group's classification. In future work I intend to consider group level statistics which provides more inferencing information as I discuss in Section 6.2.1. I use the group level information to train the machine learning methods, so the performance gap may close with Bayesian statistics at that level.

6.2.4 Machine Learning

My first tracking algorithm, using a Bayesian approach, evaluates conditional probabilities to reject data from track generation. That method proves that the information contained in event polarity and timing distinguishes between

sources and is, therefore, useful for intelligently filtering for satellite groups. The performance of that initial implementation covered in Section 7.2.1 has room for improvement in the sensitivity and specificity metrics. Therefore, I look for ways to improve the inferencing portion of the tracking algorithm through machine learning and consideration of the data at the group level instead of data at the pixel level. To implement machine learning, I prepare the training data to capture the right attributes. I also augment the data to account for the relative prevalence of star and noise data compared to satellite data. Once prepared, I train a selection of OTS models to cover a wide range of possible options and assess initial performance. Finally, initial performance in sensitivity and specificity prompts a retraining of the models with data grouped with the RANSAC method. I describe the treatment of the data to get RANSAC groups.

Data Preparation

Similar to the probability tables for the Bayesian classification approach, I must manipulate the batch labeled data for the machine learning algorithms prior to its use. In the case of the machine learning methods, I do not identify specific attributes for the machine learning models to use. Instead, I must prepare the data to remove the influence of unimportant or method breaking attributes and divide the event groups into fixed sizes to limit the number of models I train.

I have three main steps to precondition the event groups addressing the influence of the representation of these events on classification. First, grouped events take up space and time in the block of data, which has a reference frame origin at one corner of the array and an initial time at the start of the record-

ing. The actual location of the grouped events within the overall block of data should not affect the classification of the group. Therefore, I remove the influence of particular x, y, and t values from the classification of groups by retaining only the relative information of events within a group. To do this, I first center the events of each group around the average x and y pixel location. Next, I subtract the first event's time from all of the event times and scale the new time by 1E-5. Scaling by 1E-5 allows the event time features to be at approximately the same order of magnitude as the x, y, and polarity features. The last step of preconditioning modifies the events with negative polarity. Standard output from an EVS labels negative events with a data value of 0 in the polarity field of the address event representation. Machine learning methods may multiply by this 0 value effectively nulling out its influence. Therefore, to provide more useful information to the classifier, I set all negative event polarity values to -1.

Time (us)	**X**	**Y**	**Polarity**
8007708	54	149	0
8013398	55	150	1
8028147	56	151	1

Table 6.3: Example grouping before preconditioning for classifier.

Table 6.3 shows a 3-event grouping before preconditioning, and Table 6.4 shows the same grouping after preconditioning. In the first Table, the time values are integers in microseconds, approximately 8 seconds into a data set. After preconditioning, the first time value is 0 and the relative time difference from the first event remains 0.1 seconds. Likewise the second table highlights the relative change from the middle event and is centered at (0,0) instead of (55,150). Finally, the polarity of the negative event is -1 in the preconditioned group of events.

In the machine learning models I apply, the models evaluate a fixed amount

Time (1 = 0.1s)	X	Y	Polarity
0	-1	-1	-1
0.0569	0	0	1
0.20439	1	1	1

Table 6.4: Example grouping preconditioned for 3-event classifier.

of information provided to the classifier. Therefore, I train separate classifier models to classify group sizes of 5, 10, 20, 40, and 80 events. I do not classify groups with fewer than 5 events with the assumption that too little information is available to accurately infer the class of smaller groups. For groups that contain a number of events not equal to the chosen group sizes, I apply a sliding window over the groups in their time order to provide all event possibilities at a chosen group size. This diversifies the data combinations on which the machine learning models will train. For groups of events smaller than a chosen model size, I try zero-padding to fill out the events in the group up to the model size. Unfortunately, the zero-padding negatively affects the model performance and, thus, I avoid further training with this method.

Table 6.5: Example event grouping window 1 on preconditioned events for 2-event classifier.

Event	Time (1 = 0.1s)	X	Y	Polarity
1	0	-0.5	-0.5	-1
2	0.0569	0.5	0.5	1

Tables 6.5 and 6.6 show an example of this sliding window applied to the group in Table 6.4 as it would be presented to a notional 2-event group classifier. In the first Table, the first combination of events provided for training a 2-event group classifier are the first two events sequentially. The second and third event, in the second Table, build the second combination of events for the same classifier to train.

Table 6.6: Example event grouping window 2 on preconditioned events for 2-event classifier.

Event	Time (1 = 0.1s)	X	Y	Polarity
2	0	-0.5	-0.5	1
3	0.14749	0.5	0.5	1

My data preparation methods have implications on the implementation of the classifier models. Groups evaluated by the classifiers must go through the same preconditioning so as to provide only relative event information within the group and ensure part of the model is not ignored because of a 0 value for a negative polarity event. Additionally, the classifier only evaluates the most recent set of events up to the next model size. For example, given a group with 25 events, I apply a 20 event rolling window across the group and supply each to the 20 event model, averaging the resulting classification results. If that group grows to 40 events and I reapply the classifier, I now use a 40 event model. Finally, I train all models on a flattened version of the event-address representation to define the input features. An event group of 20 events, for example, contains 80 pieces of information: the preconditioned x and y location, time stamp, and polarity for each event.

Table 6.7: Example event grouping for 3-event x 9x9 CNN input.

Time Index	X	Y	Value
0	3	3	-1
1	4	4	1
2	5	5	1

While all the aforementioned data preparation works for most of the classifiers I apply, I have slightly different preparation requirements for a convolutional neural network (CNN). First, I only prepare a 20 event model for the CNN. Just like the other models I apply a sliding event window on event groups greater than 20 events to make multiple group combinations from 1

larger group. I prepare the 20 events 3-dimensionally by collapsing the time dimension into indexed event numbers. Table 6.7 shows the same example event group in Table 6.4 after preparation for a 3-event by 9-pixel by 9-pixel CNN. It is important to note that unlike input for the other models, the only value seen by the CNN is the polarity, where the time index, x, and y dictate the location of the polarity value. All the indexes must be positive integers, so I place the group centers on (4,4) in the 9-pixel by 9-pixel dimension and the event indexing starts at 0. If the events exceed the 9-pixel by 9-pixel dimension, I set the indexes above 8 or below 0 to 8 and 0 respectively.

A key limitation of the CNN is the time resolution at the microsecond scale; if I model each microsecond as a separate index in the input data and fill in the intervening microseconds with zeros, a 2-second event group considering 9x9 pixels requires 162 MB of memory. When dealing with millions of groupings for training and validation, this easily exceeds current consumer hardware GPU RAM sizes which range between 8 and 24 GB. For example, a data set with 4 million groupings at 162 MB of memory each is a total of 648 TB. With these memory requirements, the application of a CNN demonstrates how not leveraging the data sparsity of EVS through time series focused techniques quickly diminishes one of its key advantages as a technology. To work around these memory limitations, I currently create a sparse to dense input for the CNN, remove all the non-event zero polarity steps and only keep event indexes within the 20 event group, thus limiting the training batch sizes. Despite these changes, the CNN model still has significantly higher memory requirements than the other models I implement. Between the removal of spatial data outside the 9-pixel by 9-pixel index and the reduction of timestamp to indexed event numbers, there is a loss of information in the current CNN implementation. Groups go-

ing through the classifier are treated the same way. With the loss of information there is a resulting loss of inferencing power.

Data Augmentation

One of the downsides of the empirical SDA data set I use to develop the classifiers in this dissertation is the heavy weighting of events towards noise and star data. All told, noise events are 50.1% and stars are 46.0% of all events in the data set, whereas satellite events are only 2.6% of the events. At the group level, each data set has 1 or 2 satellites compared to hundreds of stars in each. While it is accurate that satellite observations are more rare than stars in most SDA data sets, training the models with unmodified data creates classifiers that are particularly weighted towards star identification over satellite identification. I fix the imbalance in the data through augmentation with satellite group variations. This process trains the classifiers more robustly.

To capture the disparity of class occurrence in the training data groups, I apply a sliding window to the labeled groups in the data sets assigned for training at the different classifier model sizes. I count up the number of groups generated for each class to determine how many examples of each class the machine learning methods use for training. Table 6.8 shows the quantity of groups of each class at the various event group sizes. It is important to note, I conduct this analysis on truth-labeled RANSAC grouper output. Section 7.2.3 covers the rationale to the switch to RANSAC grouping. The following methodology only has one difference between the two truth-labeled data sets on the generatation of training noise groups. The RANSAC grouper innately generates noise event groups while groups of noise do not exist in the original truth data set. To han-

dle a lack of noise groups, I treat all the noise events in the original truth data set as one large group and assume neighboring noise events create groups of the chosen model size.

Despite the rolling group creation, the group totals still follow the large disparity in the raw event percentages with one exception. Noise, while the most common event in the raw events, produces fewer groups than the star class groups in Table 6.8. I do not address the noise disparity with augmentation methods because it is on the same order of magnitude, on average, as the star groups and the methods are less applicable to the noise groups. I do, however, augment the data for the satellite and hot pixel classes.

Table 6.8: Group class representation before augmentation.

Events/Group	**Hot Pixel**	**Noise**	**Star**	**Satellite**
5	57620	382609	1297800	66016
10	47884	381349	1163575	64725
20	35414	378829	959178	62299
40	21418	373799	692666	57940
80	9543	364050	398912	50788
Mean	34375.8	376127.2	902426.2	60353.6

I augment the data to increase satellite groups by generating new satellite examples from the originals. I work from the original samples to ensure the signatures through time are reasonable estimates of real satellite sources. I conduct the augmentation by rotating the (x, y) coordinates of events around $x = 0$, $y = 0$. I apply 3 rotations to each group at an angle of $\pi/2$ radians per rotation to capture 4 orientations covering the unit circle for each group. Increments of π resemble each other, but the third dimension of time makes the group signatures unique because the earlier and later events switch quadrants. Since each original group has a unique slope through the (x, y) plane, the 4 orientations I apply to each group provide a large selection of angles for classifier training.

Through this method, I make the classifier more robust to different angles of satellite track projection onto the focal plane because only a finite number of angles exist in the original data set. Now with the augmented data there is less of a connection to angle through the focal plane and the satellite class. It is important to note that rotation augmentation does not work for hot pixels, which are also under represented in the rolling group creation, as the events should all originate from the same (x, y) coordinate.

I also modify the temporal signature. I apply a uniform distribution with a maximum deviation of 5 ms and an additional check to ensure the event order does not change. This method ensures a slight variation in the event time signatures. For each of the 4 orientations that maintain the original satellite group timestamps, I produce 3 additional variations in the temporal dimension. At the end of the satellite data augmentation, I have a total of 16 groups generated from 1 original satellite group. Since the hot pixel groups are only on one pixel, I augment the hot pixel groups by applying the time variation to the events only. I maintain the same maximum time variation distance of 5 ms from each original event time. I produce 20 time variations from the original to have a total of 21 hot pixel groups from each hot pixel group of the chosen model size. Table 6.9 lists the number of groups I provide to the classifiers after augmentation. With this implementation, the hot pixels and satellites are on the same order of magnitude as the star groups.

I apply this same style of augmentation to the CNN groups, but apply modifications to hot pixels, stars, and satellites. It is important to note that I only construct a 20 event model for the CNN. To augment the data, I first apply 40 time augmentations to the hot pixels to get 41 samples from each 20 event hot

Table 6.9: Group class representation after augmentation (non-CNN).

Events/Group	Hot Pixel	Noise	Star	Satellite
5	1210020	382609	1297800	1056256
10	1005564	381349	1163575	1035600
20	743694	378829	959178	996784
40	449778	373799	692666	927040
80	200403	364050	398912	812608
Mean	721891.8	376127.2	902426.2	965657.6

pixel group. Then I rotate the (x,y) values about the center of the 9-pixel by 9-pixel spatial dimension and for both satellites and stars. I rotate stars an angle of π and apply no time modifications to the star data. In contrast, I choose 8 satellite orientations for the CNN, equally spaced at $\pi/4$ increments. I do apply the time augmentation on the rotated satellite groups, 3 each, for a total of 32 groups for each 20 event satellite group. Table 6.10 shows that through this method I level out the number of hot pixel, satellite, and star groups.

Table 6.10: CNN group class representation after augmentation.

Events/Group	Hot Pixel	Noise	Star	Satellite
20	1451974	378829	1918356	1993568

Models Implemented

Now that I have appropriate preconditioned and augmented data to train machine learning algorithms, I apply a variety of machine learning models in order to estimate the class of the event groupings. I implement a wide breadth of the models because it is not immediately obvious what style of classifier will perform the best on the event-based data. Additionally, models require different levels of computational power. Through their exploration, I foster an understanding of the resources needed to apply machine learning classification on EVS SDA data sets and potential for space-based applications of EVS with

these methods. The classification models I apply include K-Nearest Neighbours (KNN), Gaussian Naïve Bayes, Random Forest with and without Extra Trees and Gradient Boosting, Dense Neural Network, and Convolutional Neural Networks (CNN).

The first model I implement is the K-Nearest Neighbors classifier. This classifier computes the feature-space distance of a sample from previous samples with known class, and averages the class of the nearest k neighbors to predict the class of the sample. The implementation of this model essentially places the prepared and augmented groups to be in an N-dimensional feature space. Then a new group fits somewhere within that space and the class of k neighbors contributes to the classifier output class. While the neighbors can be weighted, I keep equal weights for 5 neighbors in my classifier. I choose the KNN as a mathematically simple model that tends to perform well even with small training sets [29, 19]. While, KKNs can be computationally heavy on the back-end with larger data sets, it serves as a baseline to understand the efficacy of machine learning with this data and interpret the efficacy of the other models.

Next, I try Gaussian Naïve Bayes. This machine learning model uses Bayes' theorem with the assumption that features are conditionally independent and, therefore, each feature has the capacity to provide information towards the inference of a class. I apply this model due to its similarity to the Bayesian classifier. There are some differences, however. First, I am not picking the features applied in the Gaussian Naïve Bayes method as I am with the Bayesian classifier. In this case, the model features are the flattened and preconditioned address-event representation. Next, the Gaussian Naïve Bayes model assumes that the

feature likelihood is Gaussian[92]. Due to this fact

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \tag{6.7}$$

describes the conditional probability of a feature x_i given the class y where the mean and standard deviation of the feature are μ and σ. On the other hand, the Bayesian classifier conditional probability lookup tables can capture other distributions. While there is less nuance in the feature selection and possible simplification of the conditional distributions, Gaussian Naïve Bayes involves the same math introduced in Section 6.2.3 when computing a group of features' predicted class and has potential to run quickly even on limited hardware resources. Comparing the results of this classifier to the Bayesian implementation also provides feedback on my manual feature selection.

The third model I apply is a Random Forest model. A Random Forest consists of many Decision Tree Classifiers, each of which I train independently. Decision trees operate by inspecting a number of input features and selecting a threshold for one of those features to split the dataset. Subsequent decision nodes receive the split dataset and then split it again [7]. Chosen criterion define when to form an end node; usually when the subset of data at the node represent a single class [8]. To train the Random Forest and grow the trees, I do not apply any random sampling with replacement, bootstrapping, of the data set. Therefore, the decision trees all receive the entire data set broken into the aforementioned groups of a set size. The process creates a number of trees, hence the "forest" portion of the name. Online these trees receive a group to classify and each tree calculates the probability of each class. The model averages the class values through the forest of decision trees to output a final class label. I choose to implement the Random Forest model as a moderately compu-

tationally intensive model. The tuning choices of tree-depths limits and number of trees creates an effective classifier with limited computational requirements.

I also try two modifications to the Random Forest model. First, I implement Random Forest with Extra Trees. Extra Trees is a variation on a Random Forest with heavy randomization of the decision trees [37]. During tree generation, training of each tree draws thresholds at random for each candidate feature and selects the best of these thresholds. The best threshold is one that reduces the most entropy by dividing classes onto the next node of the tree. This version of Random Forest maintains the medium complexity and reduces variance of the class outcomes between decision trees. While lower variance can create greater confidence in class, the trees can have greater bias than the regular Random Forest model. The second Random Forest model modification I try is Gradient Boosting. Gradient Boosting uses multiple stages of regression trees to fit a log loss error function output between the previous tree's output and true data labels [32]. Each tree tries to capture the loss of the preceding ensemble of trees, up to a specified number of trees.

After Random Forest, I implement a Dense Feed-Forward Neural Network. These networks consist of fully connected layers of nodes. Input features are the input to nodes in the first layers, each of which conducts linear regression with a series of weights for each input, and a constant node bias. The output of the node feeds through an activation function such as a sigmoid so as to allow for non-linearity in relationships from inputs to outputs. The output of the node's activation function connects as an input for the next layer. I constrain the final layer to be 4 nodes, corresponding to the 4 classes, which output a normalized exponential, known as Soft Max [49]. The normalized exponential

values represent the classifier confidence in each output class.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 120)               9720

 dropout (Dropout)           (None, 120)               0

 dense_1 (Dense)             (None, 120)               14520

 dropout_1 (Dropout)         (None, 120)               0

 dense_2 (Dense)             (None, 4)                 484

=================================================================
Total params: 24,724
Trainable params: 24,724
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.5: TensorFlow Dense Neural Network Composition, 20 Event Model Summary

Figures 6.5 and 6.6 show the Dense network I implement in TensorFlow 2.10. It consists of an input layer, a dense node layer with 120 nodes with a scaled exponential linear unit (SELU) activation function, a dropout layer dropping 50% of interconnections when training, a second 120 node SELU dense layer, another Dropout layer (50%), and a final 4 node dense layer with the Soft Max activation function. I choose SELU after a quick survey of available activation functions not included in this dissertation. I expect the slight performance improvement is a result of SELU's small negative component for values less than 0, which "damps the variance for negative net inputs and increases the variance for positive net inputs", inducing a self-normalizing property [54].

I train the Dense network with a batch size of 1024, using the RMSprop [45] optimizer at an initial learning rate of 1e-3, with the loss function set to categorical cross-entropy. I implement a scheduler to reduce the learning rate on plateau. The scheduler inspects the accuracy of the model on the validation set and decreases the learning rate by an order of magnitude, on the condition that 3 epochs in a row do not increase the validation accuracy over the previous best

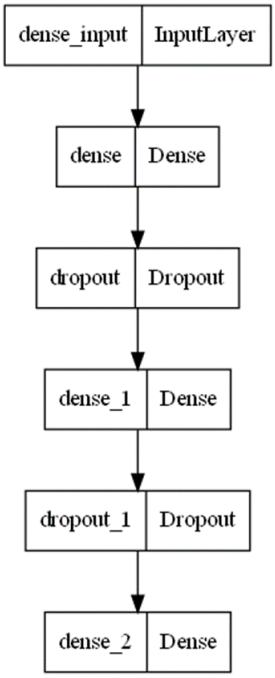


Figure 6.6: TensorFlow Dense Neural Network Flow Graph

observed validation accuracy.

The final model I train is a CNN. CNNs, unlike Dense Networks, are not fully-connected; being only a subset of the next layer's nodes considers each layer's output. Nodes interconnect using weighted filters that convolve with windows of features of the previous layer [34]. Using a CNN replicates relative positioning of events, instead of using x and y location and time values as input features to a network. I try the CNN method because the relative positioning at the microsecond level could better represent the true 3-dimensional relationship of the events than presenting them as features.

Despite these grandiose plans, I encountered a problem when attempting to create a CNN for evaluating a grouping: the size of the requisite input feature space, and thereby the size of the model are extremely large, with the vast majority of features being 0 for no event at a given microsecond. For example, take

a group that spans 7 seconds at the microsecond precision of the reported event times, the input array size, $size_{array}$, is

$$size_{array}(n, m, t) = t \times n \times m \tag{6.8}$$

where t is the time in microseconds and n and m are the pixel dimensions of the region. If I preserve the array size from the original dataset, where $n = 640$ and $m = 480$, the total number of input features for the 7 second grouping is approximately 2.15 Trillion features, or 2.15 TB in memory assuming 8-bit input values.

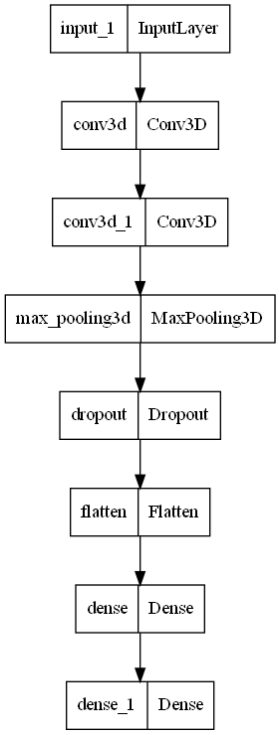


Figure 6.7: TensorFlow convolutional neural network composition, 20 event model summary.

An inspection of the centered datasets indicates that approximately 92% of the X and Y locations for a 20-event group window are within +/- 4 pixels of 0. Further analysis for a 9x9 X,Y window size indicates a total of $\tilde{9}$1.6% of events remain entirely within the window. As I discuss in the preconditioning section,

I change the input array size to 9x9 pixels. This restriction reduces the input features for a 7 second sample of a group to 567 million features (567 MB). Unfortunately, this still proves relatively unusable when dealing with multiple groupings, especially when training. Therefore, I further reduce the feature space by changing the size of the time dimension to the number of events. In this way, a 20 event group results in an input feature quantity of 20 events and a 9-pixel by 9-pixel grid only requires 1620 Bytes. This concession to have tenable model sizes for current consumer grade hardware reduces the information available to the classifier, as the time differences between events are lost. Conveniently, switching defining the CNN groups by the number of events as opposed to a specific length of time also aligns with the input feature space size of the other models.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 20, 9, 9, 1)]     0

conv3d (Conv3D)              (None, 19, 8, 8, 16)      144

conv3d_1 (Conv3D)            (None, 18, 7, 7, 16)      2064

max_pooling3d (MaxPooling3D  (None, 9, 3, 3, 16)       0
)

dropout (Dropout)            (None, 9, 3, 3, 16)       0

flatten (Flatten)            (None, 1296)              0

dense (Dense)                (None, 128)               166016

dense_1 (Dense)              (None, 4)                 516

=================================================================
Total params: 168,740
Trainable params: 168,740
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.8: TensorFlow convolutional neural network flow graph.

Figures 6.7 and 6.8 show the CNN implemented for the 20-event groupings. There are three main layers that I implement inside the CNN. The first two layers are a 3-dimensional convolution which convolves the input features along all three of its dimensions with a 2 by 2 by 2 grid. Therefore, the volume input into the next layer reduces by 1 in each dimension. The next layer is a max pool-

ing layer which further reduces the volumetric components of the CNN array by retaining the maximum values in volumes of 2 by 2 by 2 without overlap in the feature map. Finally, the CNN flattens the volume of feature to feed into a dense layer with 4 output nodes, similar to the structure of the last layer of the Dense network.

6.2.5 Measuring Classifier Performance

Given all the different combinations of grouping and classifiers I cover in the previous Sections, I choose a standard set of statistics to evaluate the performance of the first two steps in the tracking framework. The goal of the grouper and classifier combination is to provide filtered desirable information, satellite events in my use case, to the next step in the tracking process. I equate performance to the amount of information properly assigned to my class of choice and the amount of information properly rejected. These two metrics, sensitivity and specificity, highlight the performance of the filtering process.

Starting with sensitivity, through which I assess the ability of the classifier to find the correct alternative hypothesis class events through the filtering process of the tracker. To evaluate these statistics for the satellite class, I record a true positive, TP, for each event assigned to the satellite class online and identified as a satellite detection in the clustered data. I also record the number of satellite events in the truth data falsely labeled as the null hypothesis in the online filtering process. These are the false negatives, FN. I calculate the sensitivity, TPR, as

$$TPR = \frac{TP}{TP + FN} \tag{6.9}$$

where the denominator equates to all satellite events in the truth data. The sensitivity is, therefore, the probability of an event properly passing through the grouper and classifier combination.

My second metric of choice, specificity, tests for the opposite conditions of sensitivity. It is a measure of how well the filter rejects the null hypothesis classes through the online process. I calculate the specificity, TNR, as

$$TNR = \frac{TN}{TN + FP} \tag{6.10}$$

where the total assignments to the null hypothesis in the truth data is a summation of the total true negatives, TN, and total false positives, FP. In the case of a test isolating satellite data, the null hypothesis corresponds to every other class in the post filtered data: star, noise, or hot pixel. Events receive the true negative designation if they have any null hypothesis class from both the online filtering and truth data. False positives are the events with the null hypothesis class in the truth data, but an alternative hypothesis class label in the filter output.

CHAPTER 7

EVENT-BASED TRACKING ANALYSIS

The grouping and classification components of the tracking algorithm I develop in Chapter 6 have multiple tuning parameters to optimize their performance. Before I evaluate relative method performance, I consider the sensitivity of these methods to their input parameters and explain parameters chosen for method comparison. I organize the discussion of grouping and classification into two overarching Sections. Section 7.1 covers the grouping methods and Section 7.2 addresses the classifying methods. It is important to note, the classification method performance depends on the grouping method, so some analysis within Section 7.2 discusses grouping method impact on classification.

7.1 Grouping Methods Analysis

Proximity-based and RANSAC-based grouping are the two methods I develop in Section 6.1 to associate events with their neighbors. Each individual event, when working with point sources, is difficult to distinguish between noise and a real source. To be able to inference, I need to group events so that the information between events and combined polarity signatures both contribute. Each of these methods have their own set of parameters to chose. I describe the grouping parameters for the proximity-based grouping method in Section 7.1.1 and the RANSAC-based grouping method in Section 7.1.3. Using parameters chosen in those Sections, I cover the performance compared to the batch-clustered events labeled using the metrics I outline in Section 6.1.3. Sections 7.1.2 and 7.1.4 report on the grouping results of the proximity-based and RANSAC-based methods respectively. Finally, in Section 7.1.5, I compare the two method's per-

formances relative to each other.

7.1.1 Proximity-based Grouper Parameter Determination

The Proximity-based Grouper has the following parameters to choose by the user that define its deterministic performance: a 2-dimensional distance to filter the group hypotheses end points, whether the noise filter is on or off, whether the pruning is on or off, and time bound in which negative events are attributed to the most recent positive event on their pixel. If I initialize the noise filter or pruning, each has a time threshold. In Section 6.1.1 I discuss the proximity-based grouper construction and these parameter requirements. To summarize, the 2-dimensional distance is the threshold filter of the most recent pixels added to hypothesis groups. Events join the closest hypothesis by euclidean distance within this region and start their own hypothesis if there is not a previous hypothesis within this region. If it is a negative event, I also check if a positive event is within the negative attribution time on the event's pixel before starting a new hypothesis. When the noise filter is on, the algorithm treats events as noise unless multiple positive events occur on the same pixel within a time bound. If the most recent two events on a pixel are within the time bound, both events pass the noise filter. Finally, I prune non-satellite hypothesis groups when a defined period of time occurs without an update. After pruning, I do not consider the hypothesis for future event association.

I run a sensitivity analysis on the proximity-based grouper parameters with a fixed time of 3 seconds for the noise filter and pruning if enabled. Because the pruning depends on the group classification, I use the Random Forest classifier

model, Dense Network, and the Bayesian classifier to evaluate the performance with pruning. Using all 3 verifies whether the proximity parameter settings are consistent across classifying methods. For these classifiers, I use the thresholds chosen by the ROC analysis in Section 7.2.4. With each run I evaluate the optimalities I define in Section 6.1.3. I report these metrics for a subset of parameter combinations in Table 7.1. In this Table, I also report the TPR and TNR. These metrics, related to the classification, inform conclusions on the pruning process.

Table 7.1: Proximity-Based grouper parameter variation optimality. High relative TPR, TNR, Optimality, and Real Duplicates are highlighted in blue, with poor performance highlighted in red for each column.

2D Distance	Classifier	Class Thresh	Noise Filter	Pruning	Neg Attr Time	Sat TPR	Sat TNR	Optimality 1	Optimality 2	Real Duplicates
2	Forest	0.99	FALSE	TRUE	4	0.6938	0.99982	0.8664	0.8855	2227
3	Forest	0.99	FALSE	TRUE	4	0.7923	0.99951	0.8989	0.9185	1344
4	Bayesian	0.32	FALSE	TRUE	4	0.4763	0.95927	0.8745	0.9229	1022
4	Forest	0.99	FALSE	TRUE	4	0.7888	0.99908	0.9083	0.9283	968
5	Bayesian	0.32	FALSE	TRUE	4	0.5752	0.93802	0.8662	0.9265	830
5	Forest	0.99	FALSE	TRUE	4	0.8297	0.99694	0.9071	0.9291	766
5	Forest	0.99	FALSE	FALSE	4	0.8068	0.99687	0.7343	0.9887	1587
5	Forest	0.99	TRUE	FALSE	4	0.8152	0.99683	0.7379	0.9861	1533
5	Forest	0.99	TRUE	TRUE	4	0.8107	0.99691	0.9078	0.9282	647
6	Bayesian	0.32	FALSE	TRUE	4	0.3517	0.95394	0.8577	0.9260	766
6	Dense	0.8	FALSE	TRUE	4	0.9764	0.96167	0.8731	0.9295	736
6	Dense	0.96	FALSE	TRUE	4	0.9516	0.99875	0.9096	0.9298	697
6	Dense	0.96	FALSE	TRUE	10	0.9516	0.99871	0.9095	0.9298	697
6	Forest	0.99	FALSE	TRUE	4	0.8824	0.99444	0.9044	0.9291	701
6	Forest	0.76	FALSE	TRUE	4	0.9817	0.99238	0.9042	0.9303	719
7	Forest	0.99	FALSE	TRUE	4	0.8558	0.98992	0.8969	0.9261	678

Table 7.1 displays the optimality results both with and without the noise filter and pruning enabled through a range of 2 to 7 pixels 2-dimensional distances. The pruning parameter stands out as one of the most influential parameters in the proximity-based grouping process. When disabled at a 2-dimensional distance of 5 pixels, the optimality 1 metric drops from some of the best performance values, where over 90% of the events are properly assigned in or not in a group, to the lowest values, where only approximately 73% are properly grouped. Interestingly, for these same cases, the optimality 2 metric reaches relatively high values around 98%. However, Table 7.2 shows that this high op-

timality is due to a very large number of majority noise groups existing, with many noise events belonging to those groups. In this case, the noise rejection would be left to the classifier. Therefore, choosing a 2-dimensional distance that maximizes optimality 1 to reduce use of the classifier to reject noise, I recommend higher values of 2-dimensional distance. As an additional benefit, higher distance values also reduce duplicate groups. There is an upper end to larger distance performance gains. Eventually, the larger 2-dimensional distances capture more noise events into the groups, reducing optimality 1 for the 2-dimensional distance of 7 pixels.

Table 7.2: No pruning (False) group metrics with and without noise filter.

Noise Filter	**Noise event in majority noise group**	**Duplicate groups**	**Duplicate Noise Groups**	**Majority Noise Groups**
FALSE	148926	66599	65012	65087
TRUE	145329	66464	64931	65006

With the highest performing combination in Table 7.1, a Dense Neural Network at a 2-dimensional distance of 6 pixels without the noise filter and with pruning, I also try a greater negative event attribution time to examine the grouping sensitivity to that parameter. The results are underwhelming because grouped events do not have trails of off events with that much offset. Optimality 1 decreases by 0.01% by increasing the offset from 4 to 10 seconds. This decrease is likely from the inclusion of subsequent noise events into a group. However, optimality 2 stays the same because these additional noise events are added to majority noise groups. In general, since the grouping method is not sensitive to this parameter, I use an offset of 10 seconds for all subsequent runs of proximity-based grouping method to match the offset chosen for the RANSAC method as I cover in Section 7.1.3.

7.1.2 Proximity-based Grouper Results

Selection of the proximity-based grouping parameters is a balance between achieving optimal segregation between events through the grouping method and allowing some of those groups to be erroneous with the hope that the classification methods will reject them. Table 7.3 highlights my selection to use in all subsequent proximity-based grouping applications. With an optimality 1 of 90.95% and an optimality 2 of 92.98%, a majority of events are being properly grouped. Of the 9.05% that are not, 22.4% are grouped into majority noise groups and have a higher chance of being rejected by the classifier.

Table 7.3: Proximity-Based grouper final parameter selection.

Parameter	**Setting**
Inlier Threshold	6
Noise Filter	FALSE
Pruning	TRUE
Negative Event Attribution Time	10s
Classifier Model	Dense Neural Net
Classifier Threshold	0.96

The breakdown of all the individual event metrics in Table 7.4 further exposes the performance of the chosen settings across all data sets because it details the breakdown within the groups. For real events, the number of events in a group with other real events with the same label from the batch data is two orders of magnitude larger than those in groups where the majority of events have a different label. The number of real events joining a majority noise group is even less with under 100 instances. These metrics expose how much information may be lost before the classification process. In general, the chosen grouping parameters ensures a large portion of real events end up in groups. However, the real events associated with the wrong group do not help in de-

termining whether the satellite group's underlying source is really a satellite. Therefore, minimizing the portion of real grouped events in these categories indicates an already refined set of groups will go to the classifier.

Table 7.4: Proximity-Based grouper metrics.

Metric	**Count**
Real event with correct group	394193
Real event with wrong group	3336
Real event in majority noise group	66
Real event labeled as noise	35336
Noise event in real group	2356
Noise event not grouped	138292
Noise event in majority noise group	11886
Duplicate groups	5172
Duplicate groups that are noise	4475
Majority Noise Groups	4550

The noise metrics also uncovers more information about the grouping performance. While most of the noise remains ungrouped, 83.5% of the grouped noise is within a majority noise group and may be rejected by the classifier. This is the ideal condition for grouped noise. The remaining 16.5% of grouped noise are within real groups. These noise events add information to real groups which may reduce classifier performance. Because of the potential to feed poor information into the classifier, it is important to look beyond the original optimality metrics when choosing a set of parameters for any clustering method. While the performance of the parameters in Table 7.3 are acceptable for this data set, the performance is likely unique to these particular data and should be reevaluated before application.

7.1.3 RANSAC Grouper Parameter Determination

When running the RANSAC Grouper there are many values to select that govern its functionality. Many of these parameters depend on the characteristics of the data sets which in turn depend on sensor settings and physical factors during collection such as slew rate or environmental conditions. Like the proximity-based grouping parameters, these parameters will need reevaluation for each use case. There are a couple parameters which may be applicable to other scenarios with minimal modification. The parameters, minimum number of events for RANSAC Line Fit, group combination distance, and star group percentile, may require little or no modification for use in other scenarios. On the other hand, the current version of the RANSAC grouper requires tuning of all of the following: the size of the bounding cuboid, the inlier threshold, the relative scale of the temporal dimension, the negative event fit line time offset, the maximum time of negative event attribution, and the RANSAC iteration count, for different given operating and sensing conditions. In this Section, I discuss my RANSAC parameter selections that I choose to optimize grouping performance and also apply to all classifier runs to determine classifier performance.

Choosing the Bounding Cuboid

The first parameter I examine is the bounding cuboid. The cuboid in x, y, and temporal dimensions determines which events the RANSAC method draws its random points from to form linear models to fit the larger sample of previous events. A "good" cuboid exhibits a few properties. First, the cuboid dimensions I select should have a high likelihood that at least 2 events are in the cuboid to enable fitting a linear model. On the other hand, the cuboid should contain only

a few events, such that the random draw of events without replacement has a small number of unique combinations. Only have a few combination limits the execution time and decreases randomness in the final lines. Ideally, the majority of the few events belong to the same source. If the events belonging to the same source are spread out on different pixels, I increase the likelihood of estimating a slope for the linear model that closely follows the approximated linear motion of the source. If I satisfy these conditions, I maximize the likelihood the fit line applies to events outside the cuboid and future events.

Given the above conditions for a "good" cuboid, I inspect the training subset of labeled batch truth data sets at each positive event belonging to real signal groups, satellite and star groups, at a variety of cuboid sizes, and the record the following information: the total number of events in the cuboid, percentage of those events belonging to the same source as the event being inspected, and the number of unique pixels in the cuboid that have events from that group. I only inspect positive events, as the RANSAC grouper runs only against positive events. I try cuboid sizes with equal x and y dimensions ranging between 1, where I only consider the pixel of the event, through 31. The values I explore are always odd with the assumption the event is in the center pixel with an equal number of pixels on either side of it. I range the temporal dimension between 0.5 and 18 seconds.

Starting with the number of events in the cuboids, Figure 7.1 shows an increasing relationship between the number of events in a cuboid and the cuboid's volume. This Figure displays the average number of events other than the inspected event given a cuboid size. I disregard cuboids with only one event for this analysis. The positive correlation to volume is expected, as a larger cuboid

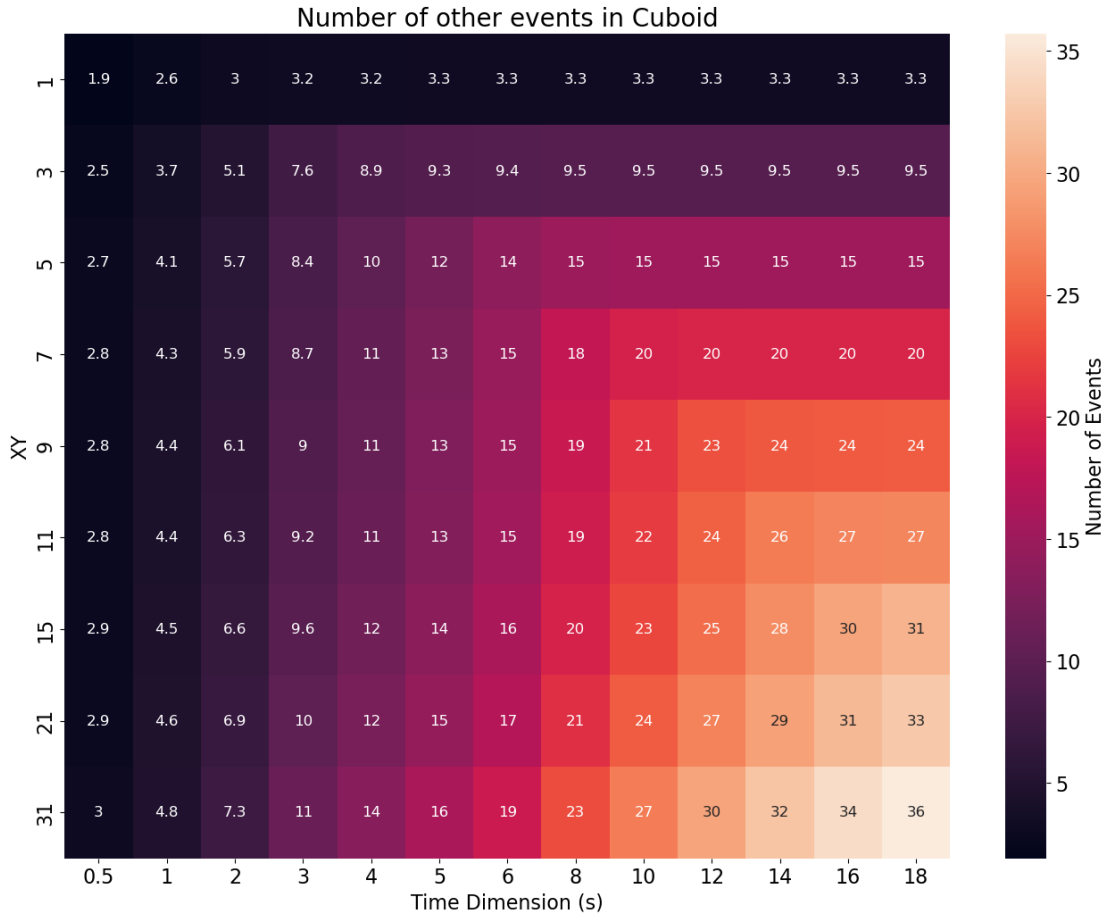


Figure 7.1: Average number of other events in cuboid. The darkest regions contain the fewest events. The x and y dimensions are equivalent on the y axis. The range of times considered are on the x axis. As the cuboid volume grows, the number of events inside the cuboid grows.

will contain more events.

Ideally, more than one event is inside the cuboid on average for line fitting purposes. To capture the prevalence of single event cuboids given a volumetric size, Figure 7.2 examines the proportion of cuboids containing only a single event where there is not enough information for a line to be formed. Logically, the Figure shows that a box with a small time dimension or radius dimension has a higher chance of only containing one event. Interestingly, the proportion of cuboids with a single event does not drastically change with an increase in the

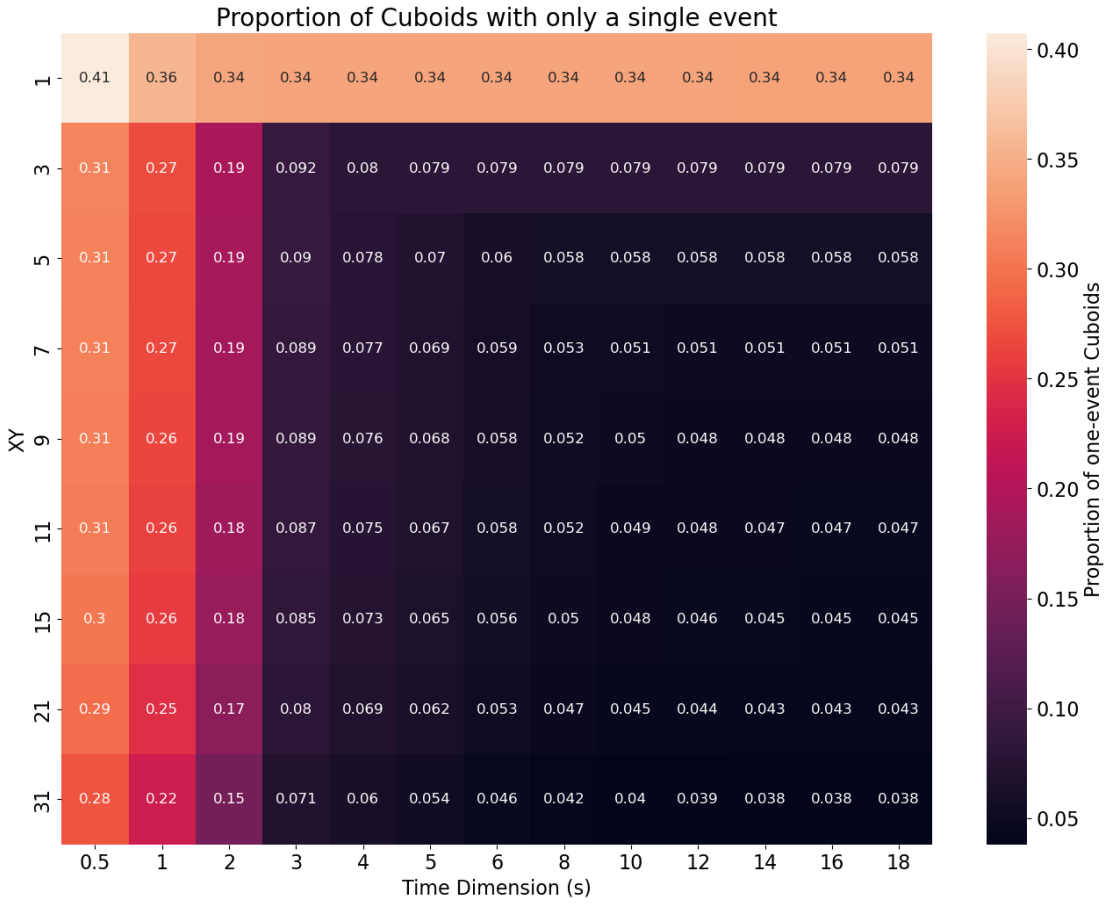


Figure 7.2: Proportion of cuboids with only a single event indicates that smaller cuboids have a greater chance of only containing one event. Cuboids with a small x and y dimension retain similar single event cuboids no matter the size of the time dimension for these data sets.

time dimension when x and y dimension size is small. At larger x and y sizes, the temporal dimension applies the most improvement at smaller time values and has diminishing returns at larger time values. It is important to note that I exclude the single event groups from all other cuboid analysis because they skew the data with information that the RANSAC grouper would not consider for fitting anyway. Instead, I capture the proportion of one event cuboids with this evaluation alone.

Next, Figure 7.3 shows the average proportion of events in the cuboid other

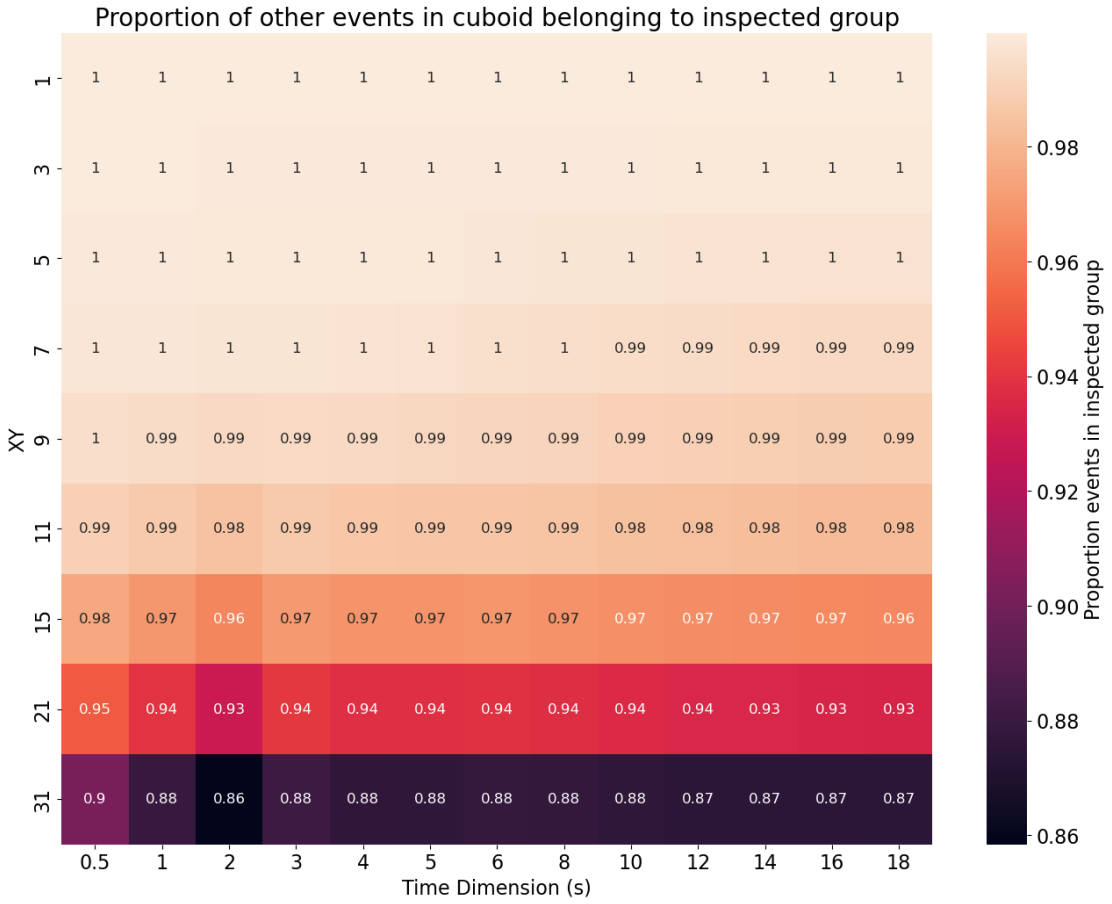


Figure 7.3: Average proportion of other events in cuboid belonging to group of an inspected event. As the cuboid grows and the number of events within the volume grows, there is a greater chance that the events inside the cuboid do not belong to the same source. The x and y dimensions have a greater impact on the proportion of events. The time axis can grow with nearly no impact for these data sets.

than the inspected event that belong to the same source as the inspected event. In this case, increasing the x and y size of the cuboid decreases the proportion of events belonging to the same group, while time has no noticeable effect. This lack of dependence on the temporal dimension for the proportion of events within the cuboid is a characteristic that is unique to these data sets. The minimal collection time in combination with the relatively slow motion of the sources relative to the sensor causes the sources to dwell on the same pixels for

significant periods of time. Additionally, the relative lack of noise also reduces the impact of time on this characteristic. It is important to note that the data set characteristics of low noise and minimal source overlap also influence the outcome of the cuboids with single events.

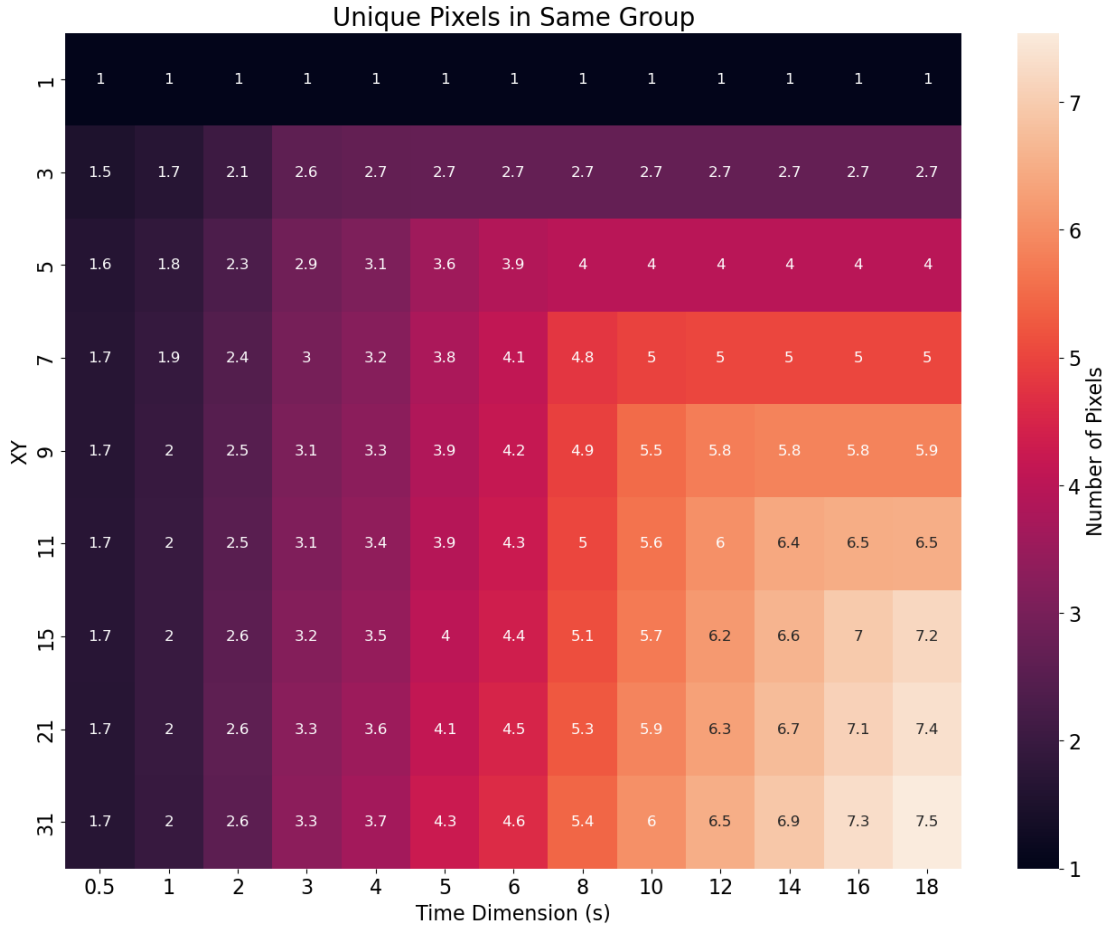


Figure 7.4: Unique pixels in same group as the initializing event grows as the size of the volume grows. Only a fraction of the pixels in the x and y projection contain events from a single source, so the unique pixel values do not follow the exponential growth of the total pixel count.

Finally, Figure 7.4 displays at the number of events on unique pixels from the inspected event's source in the cuboid. Intuitively, the larger the x and y dimension and the longer the time, the more unique pixels are represented. Of note, the values for each x and y dimension seem to plateau at increasing times

as the x and y dimension increases. The plateau does not occur at the maximum number of pixels in the defined cuboid. For example, the cuboid with a dimension of 7 in the x and y dimension, a total of 36 pixels, plateaus at approximately 5 unique pixels near $t = 10s$. Since most of the sources carve linear paths, the projection into the x and y plane should only contain a fraction of the unique pixels when they belong to a single source.

Taking these conditions into consideration, I construct an optimization function combining the data to inform the cuboid parameter selection

$$opt = \frac{(1 - \frac{1}{u}) + (1 - s) + \frac{1}{n!} + g}{4} \tag{7.1}$$

where s is the proportion of cuboids with a single event, u is the number of unique pixels belonging to the inspected group, n is the average number of events in the cuboid, and g is the proportion of events in the cuboid that belong to the inspected group. Figure 7.5 displays the output of the function.

The optimization function returns a maximum value at an x and y dimension of 11 pixels and a time of 18 seconds. Given this is the longest time I analyze, the resulting cuboid catches the most events and, therefore, reduces the proportion of cuboids formed with only a single event. The xY value is a balance between a larger number of pixels capturing more pixels associated with the inspected group and smaller number of overall pixels increasing the proportion of events belonging to the inspected group.

Practically, as the total number of events in the cuboids increases, the total RANSAC runtime also increases due to the many potential event combinations for RANSAC. Additionally, with more combinations there is a decreased chance of selecting a useful event pair. This is why I treat n in the optimality metric as $1/n!$ to reduce the n contribution to the optimization function, trending towards

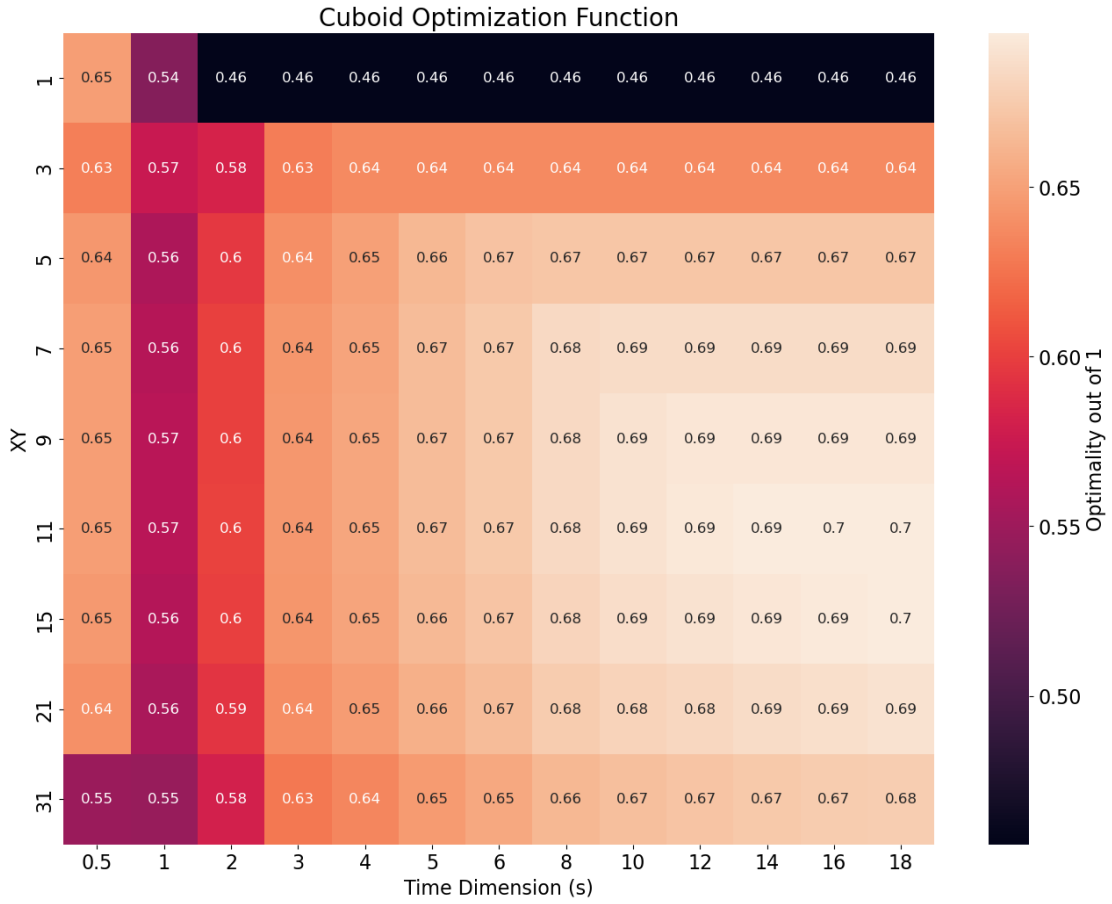


Figure 7.5: Cuboid optimization function visualization for all cuboid settings. The peak occurs at a value of 11 pixels in the x and y directions and 18 seconds in the time dimension.

0 for larger event counts. Another consideration is that the amount of time contained in the cuboid dictates a minimum desired history length of events to maintain the online grouping. I aim to minimize this history in order to keep only the most recent information most relevant to the output.

Given these considerations, I select two time dimension sizes and two x and y dimension sizes and compare them in the online grouper. I examine the optimality 1 metric for all four combinations with time values of 5 and 12 seconds and x and y dimensions of 5 and 9. I choose these points to contrast performance at the edge of the optimal region and at near the most optimal point in

7.5. The group optimality 1 for the RANSAC grouper output at these sampled points appears in Table 7.5.

Table 7.5: Comparison of grouping optimality 1 with chosen RANSAC cuboid dimensions of x and y and time. Darker green highlights higher optimality.

XY	t=5	t=12
5	0.94637	0.94769
9	0.9454	0.94728

Table 7.5 demonstrates that the grouper performance does not vary significantly over the sampled region, but demonstrates slightly better performance at x and y dimension of 5 pixels and prefers the larger time dimension. The overall difference in optimality 1 at that x and y dimension between time depths of 5 seconds and 12 seconds is 0.132%. This is only a marginal improvement. Given the considerations above regarding preference for limiting the size of the time dimension, subsequent runs of the RANSAC grouper use a cuboid with a x and y dimension of 5 pixels and a time dimension of 5 seconds.

In the initial cuboid analysis, I do not discriminate between satellite and star sources; I also conduct this analysis on satellite sources due to their underrepresentation when compared to stars. This analysis ensures the cuboid tuning is reasonable for the groups I'm most interested in identifying. The satellite-only analysis shows a nearly identical set of Figures to the results from inspecting all classes. Therefore, these Figures are not included in this dissertation. Additionally, the final optimization values have a 0.000761 mean difference and a standard deviation of 0.007551 from the satellite and star group analysis. With minimal differences in performance expected, I conclude the cuboid volume selected through examination of both star and satellite groups is sufficient for satellite groups alone.

Choosing Inlier Threshold and Time Scale

The next parameters I discuss are the inlier threshold to the RANSAC drawn lines and the value I use to scale the time axis. The inlier threshold is the maximum distance off a RANSAC line for an event to belong to the line. Since this value is in units of pixels, I scale the time axis to assist with association. I apply the same method with the batch clustering using the DBSCAN algorithm in Section 3.4.3 to adjust the time axis into an order of magnitude where the pixel radius catches associated events in the temporal dimension. The time values in the dataset are in units of microseconds, so unaltered 1 pixel distance is equivalent to 1 microsecond. If I scale the time axis by multiplying the time values by 1E-6, then 1 second is equivalent 1 pixel distance. Using this scale, if an event is 4 seconds away or 4 pixels away the algorithm treats it as the same distance. This scale may or may not be appropriate. To determine which temporal scale value that best serves different RANSAC inlier distances during grouping, I undertake an analysis by running the RANSAC grouper against the validation set measuring the optimality 1 of its groupings with varying inlier thresholds and time scales. I show the optimality values in Table 7.6.

Table 7.6: Inlier threshold and timescale vs. RANSAC grouper optimality 1. Blue indicates relatively high optimality, with red indicating low optimality.

Inlier Thresh	**1.00E-02**	**1.00E-03**	**1.00E-04**	**1.00E-05**	**1.00E-06**	**1.00E-07**
2	0.925422	0.922585	0.924368	0.926201	0.933137	0.929541
3	0.939892	0.940425	0.941209	0.941107	0.942294	0.902626
4	0.944499	0.944576	0.945074	0.94453	0.943165	0.902626
5	0.94345	0.943377	0.944039	0.943974	0.938239	0.78418
6	0.93821	0.938188	0.938797	0.938507	0.925922	0.709212

The group optimality shows a heavy dependence on the inlier threshold, with the grouping method performing best at an inlier threshold of 4. There is

not much dependence on time scale exhibited, except once the time scale reaches 1E-7. At this scale the events are compressed to 10 seconds in 1 pixel distance. Since most of the SDA data sets are approximately 20 seconds long, all events in the data sets are within two pixels at this scale. This significant compression of the time dimension relative to the pixel dimension causes the x and y pixel distance to determine primarily the events within the inlier distance. With this time scaling the RANSAC grouper misattributes many events, particularly at higher inlier thresholds. Given the limited sensitivity of the optimality to the time scale near the optimum value, the need to keep event times scaled as closely to their correct scale, and the desire to train the classifiers with time scale set to 1E-5 as discussed in Section 6.2.4, I select 1E-5 as the time scale with an inlier threshold of 4 pixels for subsequent runs of the RANSAC grouper.

Determining Negative event time offset

In Section 6.1.2, I describe that the RANSAC algorithm only fits ON events inside the cuboid. To fit OFF events, I offset another line in time with the same slope in order to find the OFF events associated with the inlier events of the ON line. Ideally, I want the fit lines close to the mean ON and OFF times to capture the most events both before and after the fit lines. The mean time offset between the ON and OFF event lines is not constant due to the comparison of the delayed pixel response. As a source leaves a pixel, the time to settle back to the nominal current value extends as the signal strength grows.

In order to choose values for the negative event line time offset, I inspect the training data sets' star and satellite groups. For each pixel in these groups, I count the total number of positive events and the time difference between the

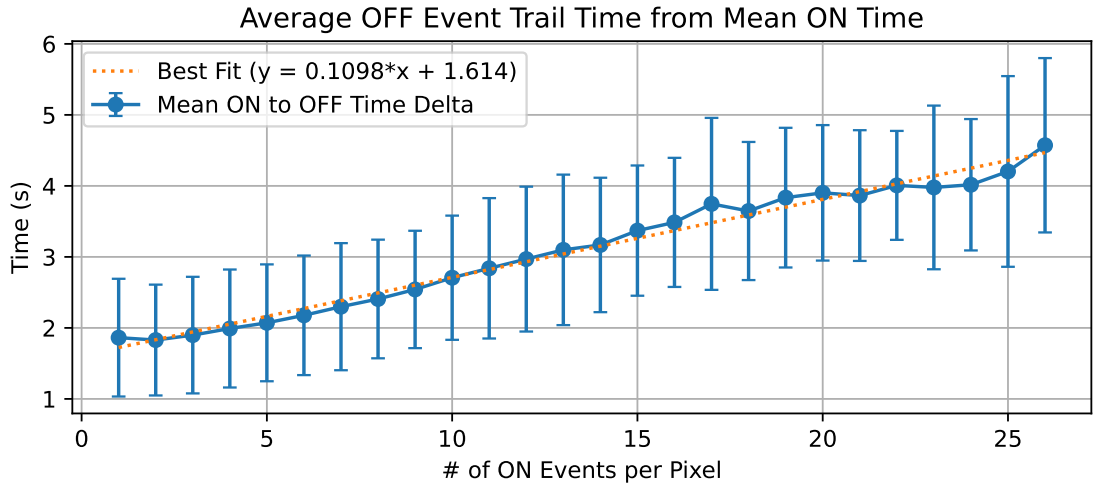


Figure 7.6: Negative event time difference from mean positive event time as a function of the number of positive events on a pixel demonstrates the linear growth in the time offset between the ON and OFF event lines for a single group of events. The bars show one standard deviation from the plot's mean time values.

average time of positive events and that of the negative events on the pixel to determine the time offset between the event lines. I organize the time offsets into bins unique to the number of positive events on that pixel. Figure 7.6 displays the mean and first standard deviation of the time offsets for each number of positive events bin. I fit a line to the mean values. I use the the linear fit parameters of slope, $m = 0.1098$, and intercept, $b = 1.614$, for setting the negative event line time offset for the RANSAC grouper.

Based on my RANSAC grouper construction, I also need to choose the time from the last positive event on each pixel threshold parameter that captures the furthest trailing OFF events if they are not inside the inlier threshold of the OFF line. To choose this parameter, I examine the time from the last positive event on each pixel to the time of the first and last negative event on that pixel associated with the same group label. I plot the mean time values and the corresponding first standard deviations for both the first OFF and last OFF event cases in Figure

7.7.

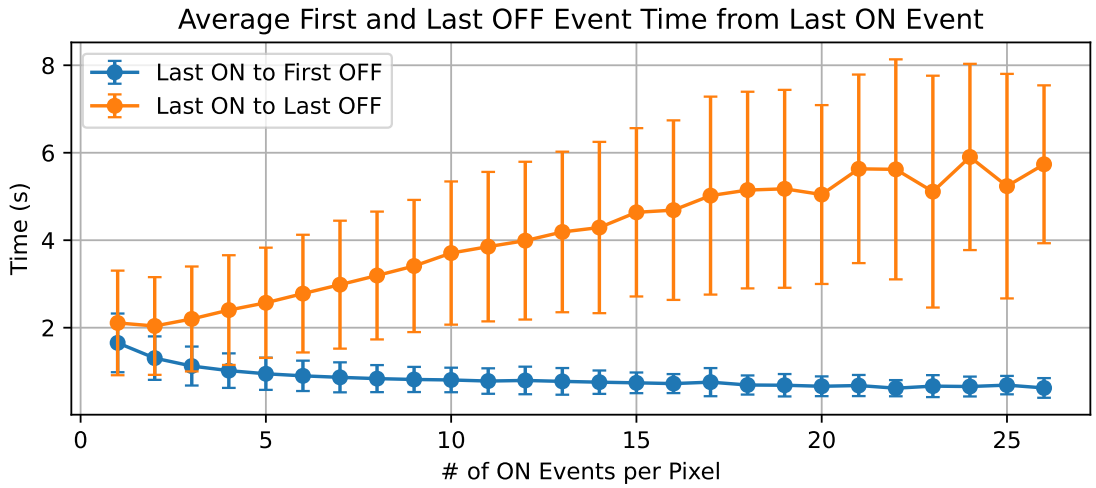


Figure 7.7: Time to the first and last negative events on pixel from the last positive event within the same group in the batch data as a function of the number of positive events on a pixel reveals a dependence of signal strength that follows the longer decay time caused by the low-pass filtration imposed by the EVS circuitry. For nosier data sets, picking a scaled negative event attribution time can assist with noise rejection.

Figure 7.7 demonstrates that the mean time from last positive to first negative decreases as the signal strength, the number of positive events on the pixel, grows. This phenomenon corresponds well with the application of a low-pass filter through the EVS circuitry. The current changes as a function of the difference between the set current and the background current as a point source moves off of a pixel. Stronger signals with a greater disparity have steeper changes in their current values initially. Therefore, the stronger signals lead to a faster occurrence of the initial OFF event after crossing the negative event logarithmic threshold.

The increase in average time between last positive and last negative event is also consistent with the low-passed nature of the signal change as the pixel returns to the nominal current value. No matter where the signal strength starts,

once it reaches another current value the decay time from the new current value is the same as a signal decay that starts from that new current value. Therefore, stronger signals take longer to decay and have longer negative event tails. It is important to note that the first standard deviation bars indicate a much higher variance amongst the timings of the last positive to last negative event than the first negative event. There are many contributors to this variance: the refractory period, noise on the signal, and arbiter readout. In order to capture all events belonging to a source in these data sets, I choose a time threshold of 10 seconds from last positive event for the RANSAC grouper. With this threshold, I cover two standard deviations for the worst cases in the data. I can only be liberal with this threshold due to the relative lack of noise within the data sets. Nosier data sets require a tighter threshold to reduce OFF noise events from attaching to real groups.

Choosing Star Group Percentile

The star group filter compares the median group slope to all groups, and designates groups within a percentile of difference from the median group as star groups, with other groups being sent to the classifier. This filter alleviates the number of groups I send to the classifier. There is one parameter to determine for the star group filter, the percentile of stars to assign as star groups. To determine an adequate percentile for designating star groups, I run the star group pre-classifier standalone against the validation set with the online RANSAC grouper output. In this configuration, the groups that the star group pre-classifier identifies I label as a star predicted class. Everything not selected by the pre-classifier, I label as "not a star". I apply no additional classification,

so this analysis only evaluates the ability of star group pre-classifier to identify star groups.

I compare the star and "not a star" class labels against the truth labels of the validation set and extract the star TPR and satellite FPR. As the percentile selected by the pre-classifier grows, more of the star groups, which should all have similar slopes, should be captured. As these groups correctly gain a star designation, the star TPR should increase. The increase in star TPR comes at a potential trade-off of satellite, hot pixel, or noise group also being classified as stars. Since I am treating stars as the true class, I want all the satellite events to be labeled as "not a star" events by the pre-classifier, so the satellite events go onto the full classifier. Therefore, I choose to examine the satellite FPR to determine when more satellites groups are incorrectly accepted as stars by the star group pre-classifier. If a satellite group gains a star label because the pre-classifier percentile is too high, the false positives increase and, as a result, the FPR increases.

Figure 7.8 captures both the star TPR and the satellite FPR versus chosen percentile to capture likely star groups. Starting with the star TPR, the figure shows a near linear increase in TPR with threshold, breaking the linear trend in the 90-100th percentiles. This matches the behavioral expectation of the TPR as the pre-classifier accepts more star groups and labels them as stars. A closer inspection of the 90-100th percentile range in Figure 7.9 reveals the downturn in slope occurs after the 95th percentile. Reaching the higher percentiles presents diminishing returns as the other objects in the frame gain the star class label. These additional objects do not lower the TPR for the true star class, but there is less improvement in the TPR metric when the pre-classifier reaches this condi-

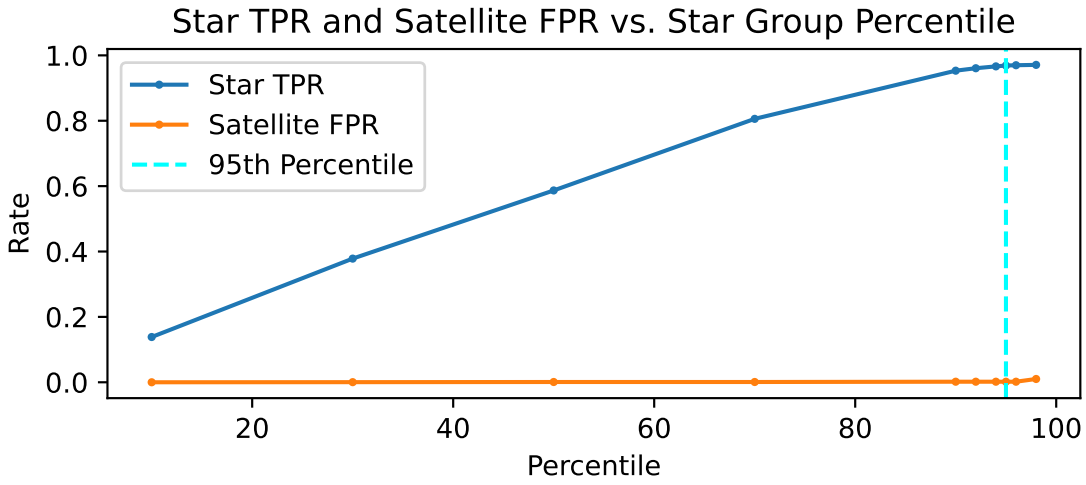


Figure 7.8: Star TPR and satellite FPR vs. star group percentile demonstrates the linear growth of the star TPR as the percentile increases and the pre-classifier labels more star groups as stars instead of "not a star". The satellite FPR remains stagnant until percentiles upwards of 90% because the satellites have enough variation in their tracks to not match the satellite tracks. The FPR increases when the pre-classifier accepts the furthest varying slopes and identifies the satellite groups as stars, increasing the false positive rate.

tion.

Taking a closer look at the satellite FPR in Figure 7.8, it remains static until the same 90-100th percentile range. Figure 7.9 shows the satellite FPR versus percentile for the 90-100th percentiles. The Figure shows the anticipated increase in FPR at the 98th percentile. While the FPR is still extremely low at the 98th percentile, I avoid rejection of the minimal satellite events in the data sets through selection of a smaller percentile. Specifically, I select the 95th percentile to balance the star TPR rate and leverage the minimal satellite FPR. The 95th percentile is the inflection point for the star TPR. Therefore, I am leveraging the power of the percentile filtering stars before diminishing returns while still minimizing the satellite FPR value. Another aspect to consider, is the computational time of the classifier. It predominates in the overall run time. Choosing a higher

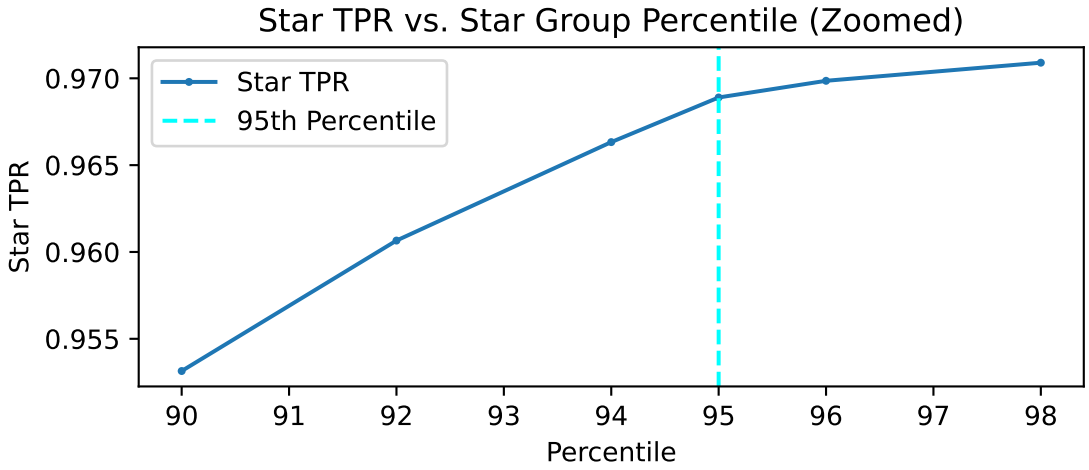


Figure 7.9: Star TPR versus the 90th through the 100th star group percentile show diminishing returns as the percentile approaches accepting all lines as stars. The slope has a point of inflection at the 95th percentile.

percentile decreases the algorithm run time by offloading the star classification to the percentile. Ultimately, I decide the 95th percentile balances these trade-offs. Therefore, I apply a 95th percentile for the star group pre-classifier for all subsequent runs of the RANSAC grouper in this dissertation.

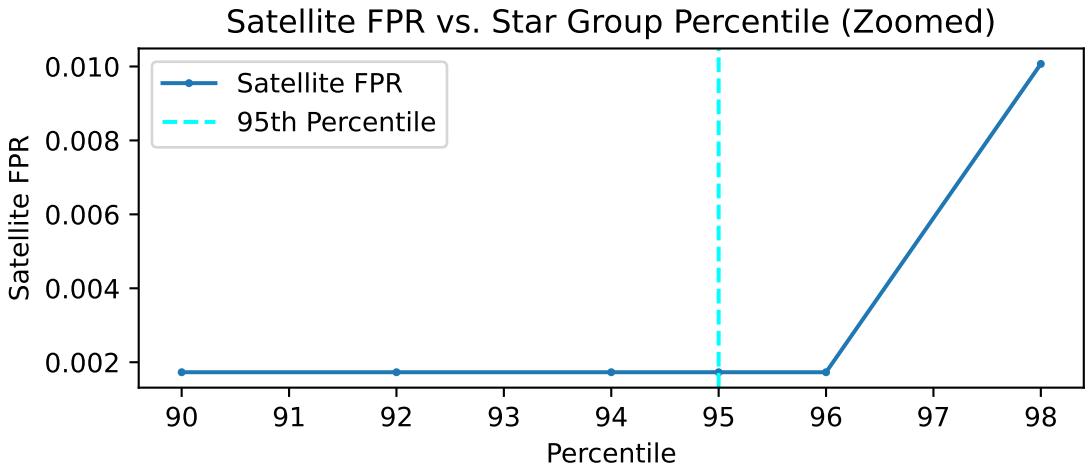


Figure 7.10: Satellite FPR versus the 90th through the 100th star group percentile show relatively low FPR until the 96th percentile. At this point of inflection, some satellite groups are close enough to the star slopes to be picked up in the star percentile. These groups, being classified as star groups, increase the satellite FPR.

Choosing RANSAC Iteration Count

I apply the RANSAC algorithm by selecting two points without replacement to fit a line and continues defining lines until it reaches a maximum iteration count. The RANSAC iteration count must be sufficient to ensure that there is a high probability of selecting lines with good fit to real sources. However, because RANSAC's output requires running for all iterations a high iteration count negatively impacts the algorithm's total runtime. My initial versions of RANSAC use a high number of iterations, N, where N is greater than 500. I conduct an analysis to determine the appropriate iteration count by examining the group optimality metrics at different RANSAC iteration count values. I start with the iteration count of 5, conduct 3 runs of the RANSAC grouper, and double the count until the grouping results appear mostly deterministic on subsequent RANSAC runs of the validation data sets. It is important to note, the application of RANSAC is not deterministic. It still randomly selects different points inside the cuboid for each run when not starting with the same random seed. However, if I get similar lines because I'm taking enough samples from the available events inside my tuned cuboid size, then the performance in optimality no longer increases. Table 7.7 displays the data from all runs.

Table 7.7 shows significant variation in group optimality when I use 5 and 10 iterations compared to 20 iterations. 20 iterations produces an output with the same optimality of 1E-4 precision. As aforementioned, absolute matching is not the goal of this exercise due to the random nature of RANSAC; consistency is desired instead amongst outputs with the same iteration count. Given the consistency of the 20 iteration result, I use 20 RANSAC iterations for the RANSAC grouper for all subsequent applications.

Table 7.7: RANSAC iterations vs. group optimality. Note consistency is best. The colors highlight similar optimality values.

Iterations	Group Optimality 1	Group Optimality 2
20	0.943826	0.961617
20	0.943848	0.961642
20	0.943822	0.961613
10	0.944828	0.962635
10	0.944767	0.962561
10	0.94476	0.962555
5	0.944711	0.962517
5	0.944594	0.962401
5	0.944786	0.96258

7.1.4 RANSAC Grouper Results

Table 7.8 displays a summary of the parameters I select via the analysis in Section 7.1.3. Ultimately, parameter selection is a process of balancing trade-offs to determine what parameters are sent to the final classifier. I want to accurately group the events together so that the classifier has the best chance at identifying the groups of events. In the case of the RANSAC grouper, it is also desirable to leverage the geometric information to reduce the classifier computational time. With these goals in mind, I evaluate the optimality of the parameters in Table 7.8. With these settings, the RANSAC Grouper produces groupings on the validation set with values of 94.5% for optimality 1 and 96.2% for optimality 2. Of the 5.5% of the improperly grouped events from optimality 1, 30.9% are noise in majority noise groups which may be caught by the classifier.

Table 7.9 shows the event and group metrics used to determine optimality with my parameter selections. As with the proximity-based grouping metrics, the detail of this event grouping information provides additional insight into the performance of the RANSAC-based grouper. Starting with real events,

Table 7.8: RANSAC grouper chosen parameters.

Parameter	**Setting**
Cuboid XY Size	5x5
Cuboid Time Size	5s
Inlier Threshold	4
Time Scale	1.00E-05
Negative Line Time Offset	y = 0.1098 * x + 1.614
Negative Event Max Time of Association	10s
Star Group Percentile	95
RANSAC Iterations	20

the RANSAC-based group maintains the same order of magnitude decrease of real events with the wrong real group and those belonging to a majority noise group as seen with the proximity-based grouping method. As mentioned in the proximity-based grouping analysis, minimizing the real events with a wrong group yields more accurate groups feeding into the classifier and should improve the classifier's performance. By orders of magnitude of events, the RANSAC-based grouping method performs as well as the proximity-based grouping method.

Table 7.9: RANSAC grouper metrics.

Metric	**Count**
Real event with correct group	413012
Real event with wrong group	2098
Real event in majority noise group	43
Real event labeled as noise	17778
Noise event in real group	2139
Noise event not grouped	139977
Noise event in majority noise group	10418
Duplicate groups	983
Duplicate groups that are noise	950
Majority Noise Groups	813

Taking a closer look at the groups with noise events in the Table, 79.5% of the grouped noise is within a majority noise group. As aforementioned in the

proximity-based analysis, this is the ideal condition of grouped noise events because they have a chance of being rejected by the classifier as a full group and they are not contaminating any real groups with additional information. The remaining 20.5% of noise in real groups may reduce classifier performance on the real group. It is important to note that the number of noise events in real groups is 2 orders of magnitude less than the real events in real groups, 0.5% of the total events in real groups, indicating minimal contamination.

Finally, I summarize the total duplicate groups via the RANSAC-based method in Table 7.9. Of the approximately 10,000 groups in the validation set, only 33 of the groups' events vote for the same real label in the truth data. Therefore, the RANSAC-based method does a reasonable job at grouping events through time due to its linear model.

7.1.5 Grouping Methods Comparison

Now that I have proximity-based and RANSAC-based tuned parameters, I compare their grouping performance by comparing the grouping metrics. First, I examine the overall optimalities in Table 7.10. The RANSAC grouper achieves best results with grouping optimality 1 with a value of 94.5%. The proximity-based grouper with pruning exhibits an optimality 1 value of 91% and without pruning a value of 73.4%. Grouping optimality 1 identifies the events that are properly grouped, real events with real groups and noise events not grouped. Therefore, the RANSAC-based grouper indicates a better match with the batch data than the proximity-based grouper and may perform better with the classifier. The optimality 2 metric, which adds noise grouped with majority noise

events, shows that the proximity-based grouper without pruning performs best if I accept the risk of rejecting the majority of noise through the classification of majority noise groups.

Table 7.10: Grouping Methods Optimality Compared

Grouper	**Optimality 1**	**Optimality 2**
Proximity-Based	91.0%	93.0%
Proximity-Based No Pruning	73.4%	98.9%
RANSAC	94.5%	96.2%

Next, I consider the event-level metrics. Table 7.11 lists these metrics for the proximity-based grouper with and without pruning and the RANSAC-based grouper. Starting with real events, the RANSAC-based grouper outperforms the proximity-based grouper (with pruning) by capturing 95.4% of the real events in the correct group as opposed to only 91.1%. The correct group is where an event agrees with the majority of events that belong to the same group in the batch truth data. The RANSAC-based grouper separates these real events from those grouped with the wrong group, majority noise groups, and those not grouped at all. Therefore, it outperforms the proximity-based grouper by reducing the sheer number of real events incorrectly grouped. In particular, the real events not in any group drops from 8.2% to 4.1% of the real events by switching from the proximity to RANSAC grouper. Overall, the real event metrics support the general conclusion of the RANSAC-based grouper being an improvement on the original proximity-based one.

As for noise event metrics, the RANSAC-based and proximity-based groupers perform on par with one another; The groupers do not group 91.8% and 90.7% of noise events respectively. The proximity-based grouper makes up for its lower noise rejection by capturing 7.8% of the noise events in majority noise groups as opposed to the 6.8% of the RANSAC-based grouper. Assuming

Table 7.11: Proximity-based and RANSAC-based grouper comparison.

Metric	**Proximity Count**	**Proximity No-Prune Count**	**RANSAC Count**
Real event with correct group	394193	429912	413012
Real event with wrong group	3336	2517	2098
Real event in majority noise group	66	502	43
Real event labeled as noise	35336	0	17778
Noise event in real group	2356	3608	2139
Noise event not grouped	138292	0	139977
Noise event in majority noise group	11886	148926	10418
Duplicate groups	5172	66599	983
Duplicate groups that are noise	4475	65012	950
Majority Noise Groups	4550	65087	813

these majority noise event groups will be rejected by a properly designed classifier leaves the two groupers with nearly identical 1.4% and 1.5% of noise events incorrectly applied to real groups for the RANSAC-based and proximity-based groupers respectively. While these metrics are nearly identical, the proximity-based grouping is more risky because it relies more heavily on the classifier rejection of noise to perform as well as the RANSAC-based grouper due to the number of majority noise groups.

Finally, I examine the duplicate groups of both methods. By subtracting the number of duplicate groups that map to noise from the total duplicate groups metric, I highlight the number of real event duplicate groups that share the same batch truth data label. The RANSAC grouper only produces 33 of these real duplicate groups. The proximity grouper, on the other hand, produces 697 real duplicates. This order of magnitude difference is possibly due to events that are associated with a group that lie just outside of the 2-dimensional threshold of the proximity-based grouper. These events, not being within range of a pixel of the appropriate group, start a duplicate hypothesis group. I do not have formal processes in the proximity-based grouper to merge duplicate groups, so this

issue compounds over time. In future iterations of the proximity-based grouper, merging groups should be taken into consideration.

7.2 Classifier Comparison and Tuning

After grouping the events, I use the collective event information of the groups to infer if the groups belong to either the satellite, star, noise, or hot pixel classes. While the groupers apply some preliminary classification, rejecting noise from the groups and the star group pre-classifier for the RANSAC-based grouping method, the remaining portion of the data rejection prior to the event utilization in state estimation or track development occurs in the classifier portion of the tracking algorithms. My first classifier operates on the pixel-level and thresholds pixels before track development. I discuss the performance of this method in Section 7.2.1. While the method proves point source classification through a Bayesian method, the performance motivates my research towards group-level classification.

Simultaneous to the pivot to the group-level classifications, I consider machine learning methods to replace my Bayesian method. Sections 7.2.2 and 7.2.3 cover the models' performance on batch and online validation data sets respectively prior to tuning the hypothesis test thresholds to accept the satellite class designation. From this analysis, I downselect to a few models to tune. I cover the tuning of these thresholds for each grouper and downselected classifier combination in Section 7.2.4. After tuning, I consider the number of events and time to convergence within each data set in Section 7.2.5 as well as the overall performance results of the online grouper and classifier combinations in Section

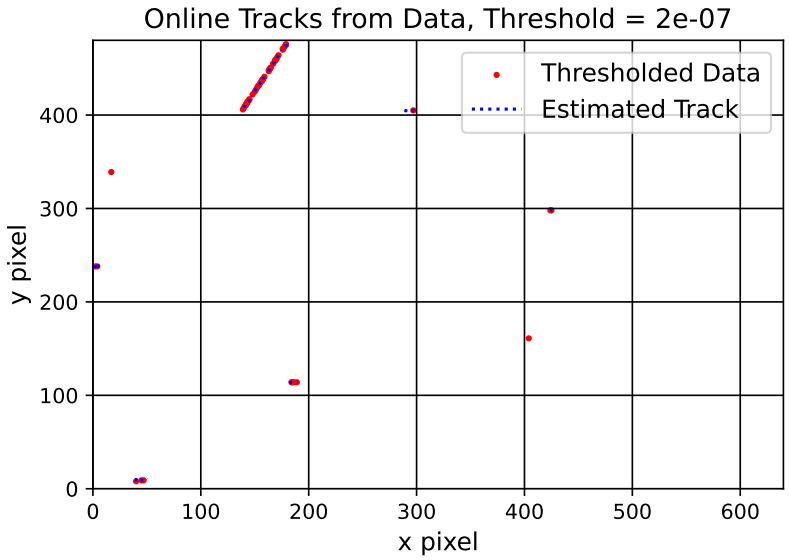
7.2.6.

7.2.1 Preliminary Pixel-Level Classifier

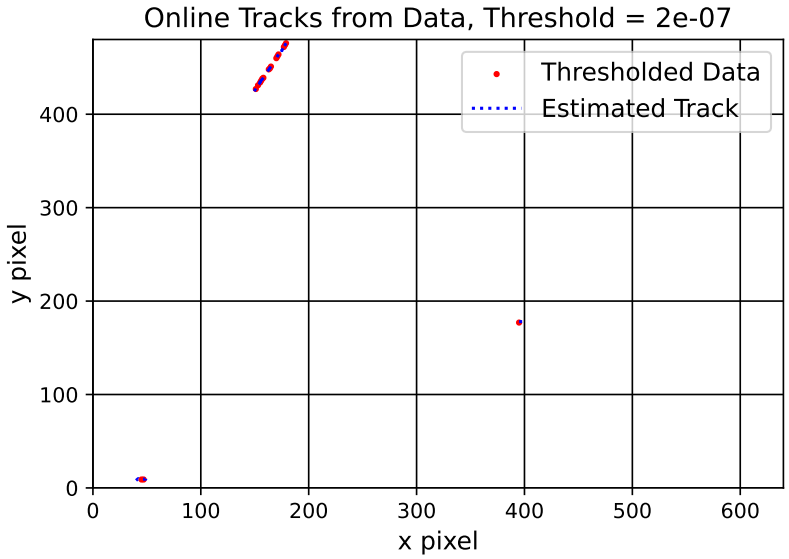
My initial classifier implementation evaluates the Bayesian conditional probability of pixel-level attributes. The pixels that pass the satellite hypothesis test add to the satellite tracks synthesized online. In this case, I use the track pixel leading event information to predict the next pixel in the track and timing of the first event on that pixel. This initial classifier implementation uses the proximity-based grouping method and I examine the performance with and without the noise filter that only passes events in close proximity in time to the grouping method.

Section 7.1.1 suggests a greater impact on the overall grouping performance through inclusion of pruning than the initial noise filter. The noise filter does have an impact, however. For example, I depict the final time for one data set of the pixel-level algorithm in Figures 7.11a and 7.11b for the non-filtered and filtered versions respectively. In these figures, I depict the estimated track in blue and plot the thresholded pixels in red. Ostensibly, the additional noise filter reduces the number of non-satellite pixels passing the satellite pixel-level hypothesis test. As a result, the noise filter reduces the number hypothesis tracks comprised of star signals in Figure 7.11 from 7 to 2 erroneous tracks. The reduction of the number of incorrect tracks is even more pronounced in nosier data sets, such as those in Figure 7.12b. In this Figure, the noise filter reduces the number of erroneous tracks from 68 to 10.

On the other hand, the satellite track, in the upper left of Figure 7.11, without

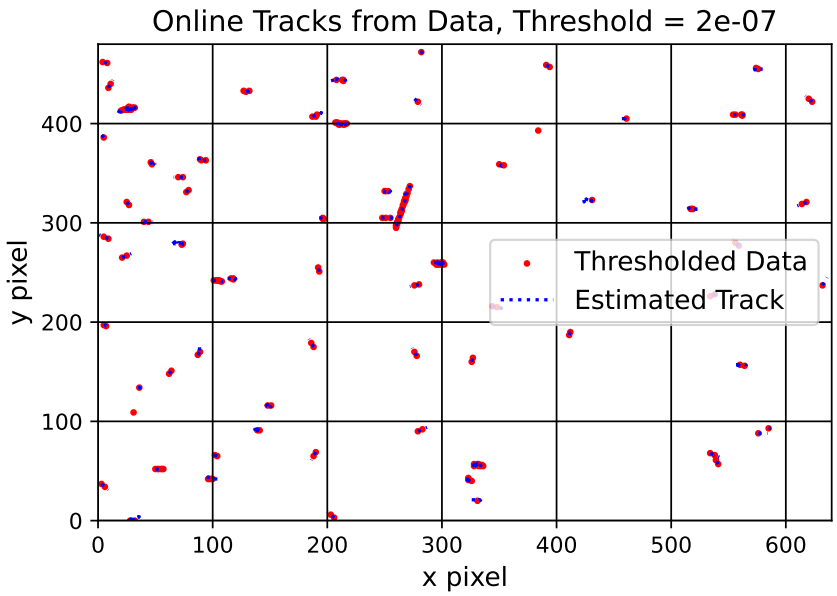


(a) Pixel-level with proximity-based grouper and Bayesian classifier data rejection without additional noise filter.

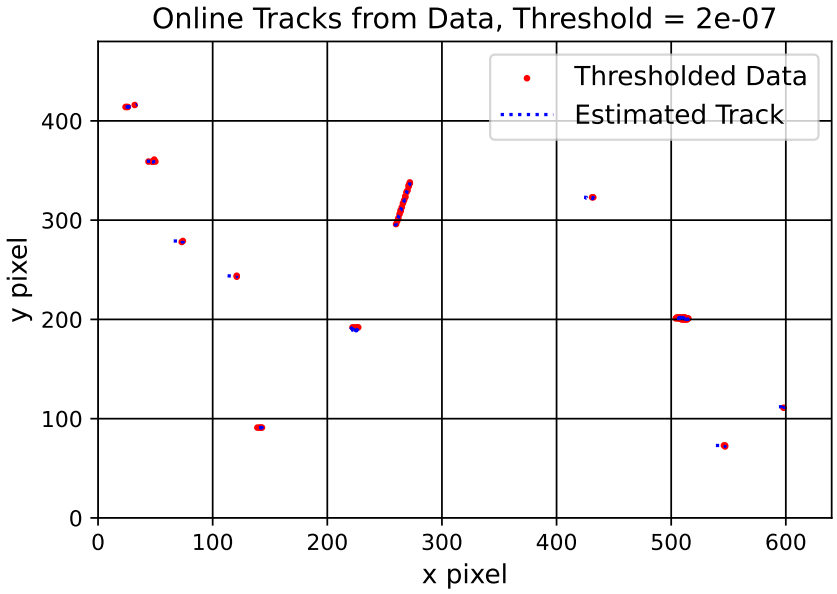


(b) Pixel-level with proximity-based grouper and Bayesian classifier data rejection with additional noise filter.

Figure 7.11: The proximity-based grouper and pixel-level Bayesian classifier at the final time without and with the initial noise filter removing lower frequency events demonstrates for a pixel-level classifier the additional noise filter reduces the number of erroneous satellite tracks from star pixels. a) There are 7 star tracks in the classifier output without the additional noise filter. b) There are 2 star tracks in the classifier output with the additional noise filter. However, the filter also removes portions of the satellite track.



(a) Pixel-level with proximity-based grouper and Bayesian classifier data rejection without additional noise filter on nosier data.



(b) Pixel-level with proximity-based grouper and Bayesian classifier data rejection with additional noise filter on nosier data.

Figure 7.12: The proximity-based grouper and pixel-level Bayesian classifier at the final time without and with the initial noise filter with nosier data demonstrates the utility of the noise filter with noise heavy data sets. a) There are 68 star tracks in the classifier output without the additional noise filter. b) There are 10 star tracks in the classifier output with the additional noise filter.

the additional noise filter covers a larger spatial extent than the one with the additional noise filter. Therefore, the noise filtered version is missing portions of of the track. The missing pixels are a direct consequence of the noise filter. While it is not apparent if the version without a noise filter is also missing pixels, I examine the ending satellite TPR values across all the validation data sets. The data sets without the additional noise filter have a satellite TPR of 0.473 and with the additional noise filter the value drops to 0.393. Therefore, over 50% of satellite data does not make it through the classifier stage of the data rejection process. Overall, the initial TPR indicates that the pixel-level online Bayesian classifier aggressively rejects satellite event data and the noise filter only increases the aggressive nature of this original algorithm.

For any subsequent algorithm utilizing the original pixel-level classifier implementation, the significant portion of missing satellite event data results in the subsequent algorithms having less information to exploit. For example, in this original implementation, rejecting a large amount of satellite event information lowers the likelihood of identifying all the satellite tracks within the data. The algorithm without the additional noise filter identifies the satellite track in only 92.1% of the validation data sets. In the version with the noise filter this metric drops even further with only 76.3% of the satellite tracks being identified.

The one advantage of the pixel-level classifier despite aggressive rejection is that it can still identify satellite tracks even if not all pixels contribute to the track generation. Since I evaluate the pixels within a group individually, sparse information passing the classifier can result in track generation. It is not required for the 92.1% of data sets with an identified track to have all the pixels represented. However, this also allows similarly resembling star pixels to generate erroneous

satellite tracks with only a couple pixels that resemble those of satellites. Knowing that some of the true satellite information is missing that could contribute to subsequent algorithms and the side-effect of non-satellite pixels generating erroneous tracks, I investigate group level classification to improve event level TPR and TNR as Section 6.2.4 describes.

Table 7.12: Pixel-wise Bayesian classifier online with proximity grouper TPR and TNR.

	Hot Pixel	**Noise**	**Star**	**Sat**
TPR	0	0.807086	0.908292	0.476286
TNR	1	0.90289	0.832465	0.959272

After improvements of the proximity-based grouper that I make alongside the RANSAC grouper development, I re-run the pixel-level classifier performance metrics without the noise filter to evaluate against the group-level performance. Table 7.12 highlights that the satellite TPR of 0.476 remains relatively similar to the original TPR prior to the grouper improvements. The TNR of the original, 0.901, improves to 0.959. Therefore, with implementation of the current proximity-based grouper as I outline in Section 6.1.1, the rejection of non-satellite pixels improves without the additional noise filter. Given the TNR is not 1, there are still non-satellite pixels that make it through the data rejection and can, subsequently, form tracks. Table 7.13 supports this conclusion with the confusion matrix of this grouper and classifier combination. In particular, of the events the classifier predicts to be satellites, 64.8% of them are not real satellite events as labeled in the batch truth data set. The noise pixels contribute 75.9% of the incorrect satellite predictions. This guarantees pixels pass through the classifier, but does not guarantee generation of incorrect satellite tracks since a track only results if two pixels in the same group pass the classifier. However, it is probable given the high percentage of non-satellite events passing the

threshold. While it is concerning that the non-satellite events outweigh the real satellite events, this disparity is partially due to the relative percentage of satellite events out of the total events in the validation data set. This aspect of the data reveals that an acceptable TNR needs to be upwards of 0.99 to ensure that real satellites are the predominant prediction after the classifier.

Table 7.13: Pixel-wise Bayesian classifier online with proximity grouper confusion matrix. White boxes in the diagonal highlight correctly classified results.

	Predict Hot Pix	**Pred Noise**	**Pred Star**	**Pred Sat**
Actual Hot Pix	0	5242	13601	558
Actual Noise	0	123108	12128	17298
Actual Star	0	30609	351974	4929
Actual Sat	0	6191	7435	12392

Table 7.12 also identifies the TPR and TNR of the other 3 classes within the data. Since the pixel-level hypothesis test only tests for satellite tracks, I assume events output from the online proximity-based grouper and pixel-level classifier not in groups are noise and in groups not assigned to satellites are stars. The pixel-level classifier, as it currently stands, cannot identify hot pixels from the other classes leaving that class with 0 events predicted on the event confusion matrix in Table 7.13. Therefore the hot pixel TPR is 0 and TNR is 1.

The noise and star classes perform better because I can rely on the proximity-based grouping to differentiate between these two classes of events that to do not make it through the hypothesis test. Both classes have mediocre performance without a formal hypothesis test; the noise class has a TPR of 0.807 and a TNR of 0.903 and the star class has a TPR of 0.908 and TNR of 0.832. The drop in the TNR for both classes is partially due to the dilution of the metric with additional false positives through non-thresholded satellite events and hot noise events that cannot be predicted. However, the primary contributor is the other

respective class, noise or star, and is, therefore, driven by the grouping method. The tendency to group noise events or not group star events also contributes to the lackluster TPR values. Of the false negative predictions for the noise events, the majority are satellite predictions. The grouped noise pixels have a 58.8% chance of classifying as satellites. On the other hand, the classifier does a better job at rejecting star events even with 7.9% of the star events not being grouped. Only 1.4% of the grouped star events passes through the pixel-level hypothesis test. The tendency to threshold grouped noise and not star events highlights the importance of the grouper for a pixel-level hypothesis test in overall performance of the events passing the pixel test threshold. Ultimately, the updated statistics support a pivot towards group-level hypothesis testing of all 4 classes to improve performance.

7.2.2 Untuned Classifier Results on Offline Validation Datasets

The preliminary proximity-based classifier indicates that pixel-level hypothesis testing has a better than random chance of identifying a satellite track. However, the performance of the proximity-based grouper and pixel-level Bayesian classifier in its current form simultaneously accepts a large number of non-satellite events and rejects a large portion of satellite events through the hypothesis test. In order to improve performance, I investigate an alternative of group-level classification with both the Bayesian classifier where pixels vote on the overall classification and a set of machine learning classifier models that I discuss in Section 6.2.4.

To downselect which models I use and tune, I compare the model classifica-

tion output on the labeled offline validation data after training. I use the offline validation data to remove the influence of the grouping methods. I select the maximum class output from the classifiers to assign a prediction to each group in the validation data. I run each model with various parameters, but I do not tune these parameters to optimize performance. For the KNN model, I use 5 neighbors equally weighted. I use unlimited depth and 30 trees for the Random Forest and Extra Trees. I also use 30 trees for the Gradient Boosting version of Random Forest. I keep all other model settings at the default for Scikit Learn 1.2.2 [93]. It is important to note that the Dense Network and CNN are custom implementations and I document their construction in Section 6.2.4.

Table 7.14: Accuracy of machine learning models based on offline validation data. Blue indicates higher relative accuracy, red indicates lower.

Model	**Accuracy**
KNN	0.9672
Gaussian NB	0.9142
Random Forest	0.9878
Extra Trees	0.9119
Gradient Boosting	0.9639
Dense Network	0.9955
CNN	0.9465

Table 7.14 details the accuracy

$$accuracy = \frac{TP_{sats} + TP_{stars} + TP_{noise} + TP_{hotpix}}{event_{tot}} \tag{7.2}$$

of each classifier as the fraction of the summation of the correct classifications for each class, TP_{class}, to the event total, $event_{tot}$ [74]. I use the top-1 accuracy metric to identify the models with the strongest potential performance. The Random Forest and Dense Network models are the top performing models, blue in the Table, with respect to the accuracy metric. Since I consider the proper classification of star events and satellite events the most important of the 4 classes, I

also inspect the star and satellite TPR and TNR of each classifier as Table 7.15 reports. Principally, all the group-level models have better satellite TPR performance and all but one model, Gaussian Naïve Bayes, have better star TPR performance than the initial algorithm pixel-wise Bayesian with the proximity-based grouper I cover in Section 7.2.1. Therefore, in the offline configuration group-wise classification, more of the star and satellite events classify to their truth classes. This is a promising indication that group-level classification will outperform the pixel-level classification in an online format.

In addition, the original pixel-level algorithm highlights the importance of maximizing the satellite TNR because of the limited number of satellite events with respect to the validation set as a whole. To avoid more events from other sources than true satellite events, the TNR metric should be greater than 0.99. Only Random Forest, Extra Trees, and the Dense Network achieve this metric. Extra Trees, however, has one of the poorest satellite TPR and star TNRs of the models I train. Since both of these metrics are lower, there are two conditions that are likely true; Either the model classifies true star events as satellites, increasing the satellite TPR denominator and decreasing the star TNR numerator, or the model classifies true satellite events as stars, increasing the star TNR denominator and decreasing the satellite TPR numerator.

Between the accuracy, TPR, and TNR metrics in the Tables 7.14 and 7.15, I select only the Random Forest and Dense Neural Network to tune and subsequently compare against the group-wise Bayesian classifier. After this initial implementation and down selection, I adjust Random Forest's number of trees to 100 after additional experimentation reveals that 100 trees with no depth limit provides better results than the 30 tree model, increasing accuracy by approxi-

Table 7.15: Machine learning model TPR and TNR for stars and satellites based on offline validation data. Blue indicates higher relative performance for an attribute, red indicates lower.

Model	Star TPR	Sat TPR	Star TNR	Sat TNR
KNN	0.96173	0.93700	0.97499	0.98393
Gaussian NB	0.85800	0.87008	0.95149	0.93451
Random Forest	0.98108	0.98016	0.99305	0.99200
Extra Trees	0.98081	0.76108	0.91053	0.99193
Gradient Boosting	0.92277	0.96291	0.98410	0.96504
Dense Network	0.98889	0.99642	0.99808	0.99526
CNN	0.97562	0.98092	0.98339	0.98643

mately 0.1%.

Despite not being selected, the KNN, Gradient Boosting, and CNN models are candidates for future development and tuning efforts. They should not be overlooked. Because KNNs use a distance calculation in n-dimensional space to find the nearest prior sample, they make good models for small training sets with consistent patterns. KNN's mediocre performance on the truth validation data sets indicates that new combinations of features are within the validation data and the incorrect class is sometimes the closest prior sample. A different feature set could change the performance. However, even with the current features, KNN is an easily trained model on minimal samples that performs better than the pixel-wise Bayesian. Therefore, I intend to consider it in future work. I will also consider Gradient Boosting in future work because research indicates it can exceed Random Forest classifier performance while taking less time to train and predict [94]. Since I am interested in reducing the complexity of the prediction, thereby the amount of run time, I intend to work with the Gradient Boosting more to attempt performance improvements. Finally, the CNN classifier suffers from less information due to the reduction of the time information stored in event indexes. I intend to conduct more research to restore the time

information and reduce model sizes. If I can redesign the model fitting these qualities, the CNN may perform on the level achieved with the Random Forest or Dense Network.

7.2.3 Initial Online Classifier Performance

The offline performance informs my down selection of the machine learning models to the Dense Neural Network and Random Forest models. Now, I integrate these models with the online RANSAC-grouper to compare their performance as they classify groups of events during active grouping. As part of the integration, I train additional models to classify event groups of 5, 10, 20, 40, and 80 events as I describe in Section 6.2.4. For this Section, I retrain the Random Forest model with 100 trees with no depth limit for a marginal performance improvement.

Table 7.16: Comparison of satellite TPR for models trained on different event quantities on offline validation set.

# Events Model	Random Forest	Dense Neural Net
5	0.9838	0.9641
10	0.9921	0.9866
20	0.9966	0.9964
40	0.9981	0.9978
80	0.9989	0.9998

With models for different group sizes, I re-run the offline metrics for the Random Forest and Dense Neural Network using the different model sizes. This analysis helps anticipate performance at different stages of the online algorithm. Table 7.16 provides the satellite TPR for each model on the offline validation dataset at varying model size. The model performance has a proportional relationship with the number of input features. Therefore, as more

information becomes available it is easier to discriminate satellites. Fortunately, there is a corresponding increase in satellite TNR as well. To take advantage of this, I combine models of various input feature sizes. I combine the models by selecting the appropriate model based on the number of input features and utilize the sliding-window method described in Section 6.2.4 to evaluate intermediary group sizes. This technique allows me to leverage the larger models, without needing a larger number of input features to begin inferencing.

Table 7.17: Selected classifier performance metrics on online with RANSAC grouper with threshold = 0.8.

	Accuracy	**Star TPR**	**Sat TPR**	**Star TNR**	**Sat TNR**
Dense	0.944559	0.968829	0.971635	0.961637	0.999389
Random Forest	0.945703	0.968912	0.969214	0.961234	0.999589

After the offline model size assessment informs my online implementation, I run the classifiers with the online RANSAC grouper and an initial threshold of 0.8 to assign a classification. Table 7.17 shows the initial classifier accuracy and the TPR and TNR metrics of the satellite and star classes. Starting the online star TPR, this metric aligns with the star TPR output when I select the 95 percentile star group threshold in Section 7.1.3. The RANSAC-based grouper's star preclassifier classifies most of the stars as I intend. On the other hand, the strong accuracy in the offline validation data for both models decreases slightly from 98.8% to 94.6% for the Random Forest and from 99.6% to 94.6% for the Dense Neural Network. This drop in performance is understandable as the grouped events from the RANSAC-based grouper are no longer ideal.

Since the star TPR is as expected, I look to the satellite TPR to explain the accuracy decrease. All the offline model sizes and machine learning models implemented in Table 7.16 have satellite TPRs greater than 0.964 with an average of 0.991. The online satellite TPRs are only 0.972 and 0.969 for the Dense

Table 7.18: RANSAC grouping trained classifier performance online with RANSAC grouper with threshold = 0.8.

	Accuracy	**Star TPR**	**Sat TPR**	**Star TNR**	**Sat TNR**
Dense	0.946716	0.968099	0.976209	0.962668	0.999494
Difference	0.002157	-0.00073	0.004574	0.0001031	0.000105
Random Forest	0.946601	0.967186	0.973441	0.962668	0.999513
Difference	0.000898	-0.00173	0.004228	0.001425	-7.7E-05

Network and Random Forest models respectively. Therefore, the online performance shows an improvement only over the model with the least information in the offline evaluation. I suspect that this performance decrease largely stems from the imperfect grouping performance of the grouper. Therefore, I retrain the models using labeled groups created by the RANSAC grouper. To create this alternative training data set, I take output groups of the RANSAC grouper and label the groups with the true class from the original dataset through majority event consensus. For groups with mixed labels from the original data set, the most represented group assigns the RANSAC group class. Table 7.18 shows the RANSAC-group trained model performance. The accuracy increases the accuracy by $\approx 0.1-0.2\%$, Sat TPR by $\approx 0.4\%$, and Sat TNR remains relatively unchanged. While this improvement is marginal compared to the decrease in performance from running the models online, I continue to use the RANSAC group trained machine learning models through the rest of this dissertation.

7.2.4 Choosing Classifier Thresholds

To further increase performance and tune specifically to improve satellite classification, I generate Receiver Operating Characteristic (ROC) curves by running both online grouping methods with my three down selected classifying meth-

ods, the Bayesian, the Dense Neural Network, and the Random Forest, on the validation data sets and compare only the satellite classifier output to the truth labels [60]. ROC curves plot the false positive rate, FPR, against the TPR for varying hypothesis test thresholds where I reject the null hypothesis of "not" the satellite class. For each ROC, I also plot the $x = y$ line which equates to a random classification. I use these ROC curves to select the maximum difference between the TPR and FPR given a different threshold for class acceptance. This maximum occurs at the corner of the ROC curve where the FPR is at its lowest value and the TPR is at its highest value. To visualize the absolute quantities, I also plot the TPR and FPR against the varying threshold values that produce the different TPR and FPR rates.

Since the grouper technique demonstrates a significant impact of the online grouping in Section 7.1.5 and the online groups are fed to the classifier, the grouping method also impacts the classifier output. I specifically design the RANSAC-based grouper to pre-classify stars and reject data from the final classifier, so I expect a dependence of overall classification on the grouper method used. I first address the consequences of the grouping method on the initial online classification in Section 7.2.3. Because the classification performance depends on grouping method, I examine the RANSAC-based and proximity-based grouping method ROC curves for each classifier to ensure the thresholds for each grouping and classifier combination yield the best performance.

RANSAC Grouper ROC Curves

I start with the RANSAC-based grouping method ROC curve analysis with the Bayesian, Random Forest, and then Dense Network classifier performance.

Throughout all of the ROC curves in this Section, the FPR and TPR here may appear inconsistent with the overall classification results reported in Section 7.2.6. This is largely because here I only evaluate the classifier output to generate the ROC curves. The star grouping pre-filter of the RANSAC-based grouper significantly reduces the number of star groups the classifier evaluates. Due to this, the number of potential False Positives by the classifier is reduced.

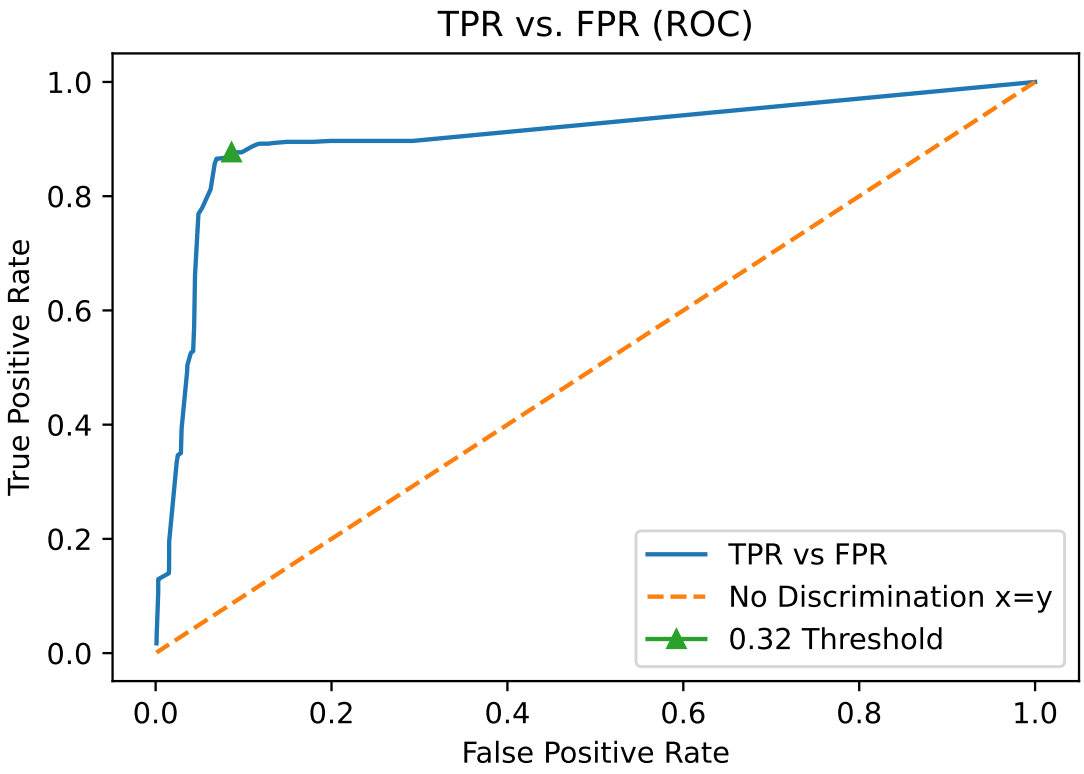


Figure 7.13: The RANSAC-based gouper with a Bayesian classifier ROC indicates reasonable separability between classes because the curve quickly ascends to a TPR slighly below 0.9 before a FPR of 0.1. Additional TPR gains require significant sacrifices with a higher FPR. The corner that maximizes the difference between TPR and FPR occurs at a hypothesis test threshold value of 0.32.

First, I examine the Bayesian Classifier ROC curve with the RANSAC Grouper in Figure 7.13. The curve is not ideal because the sensitivity, TPR, does not jump to a value of 1 when the specificity, $1 - FPR$, drops below 1. That condition maximizes the area under the ROC curve. Instead, Figure 7.13 shows

the TPR quickly ascends to a value slightly under 0.9 before the curve passes a FPR value of 0.1. Subsequent performance gains in TPR on the curve require significant sacrifice of the FPR rate. While not perfect, the final ROC curve still has a significant area under the curve and demonstrates adequate separability between classes.

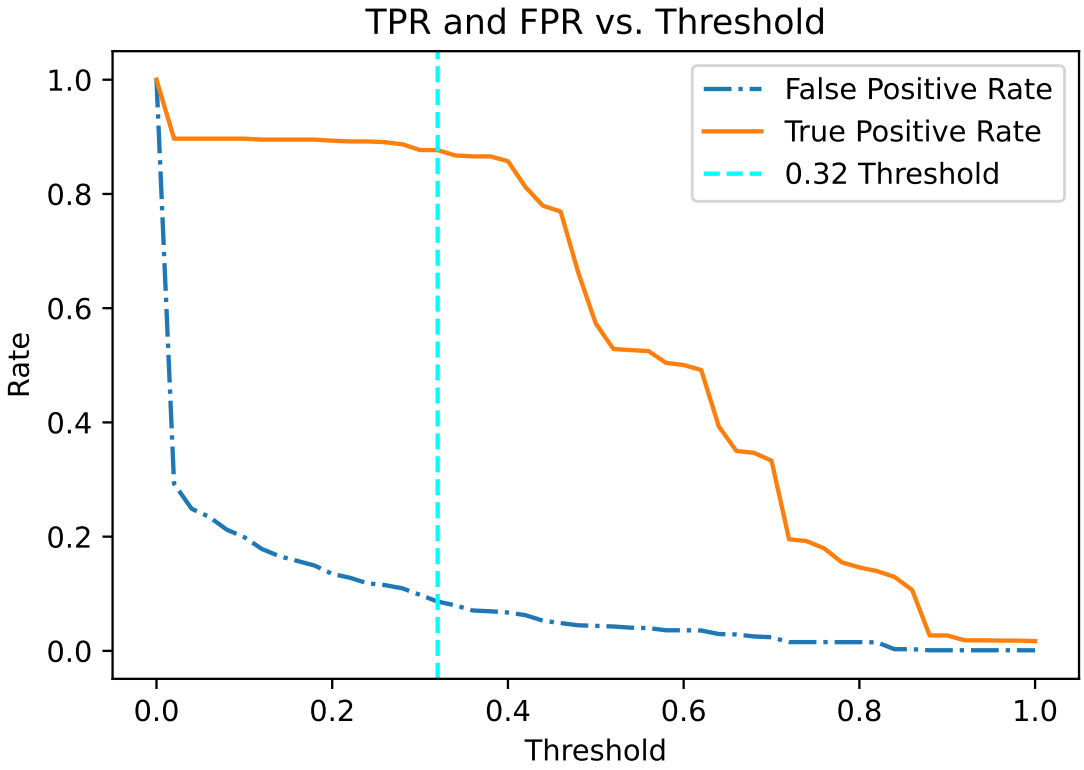


Figure 7.14: The RANSAC-based grouper with a Bayesian classifier satellite TPR and FPR as a function of threshold demonstrates a low threshold required to maximize separation between classes with the Bayesian classifier. The TPR gradually increases as I lower the threshold before plateauing. Thus the maximum TPR to FPR value occurs at a threshold of 0.32.

Figure 7.14 plots the TPR and FPR as a function of the threshold values. At a threshold value of 1, the classifier needs 100% confidence in the class label before allowing anything past the classifier. This does not occur, but as I lower the threshold the TPR metric rises faster than the FPR. Eventually, the TPR plateaus, and as the threshold decreases the FPR continues to increase. Therefore, the

TPR and FPR maximize at the beginning of the TPR plateau region. From this analysis, I select a threshold value of 0.32 to maximize Bayesian Classifier performance. A lower threshold value with a Bayesian method equates to being less confident in the class tested before accepting the class. As a result, I expect the separation applied by the Bayesian classifier to be imperfect. However, to choose the threshold via a ROC curve, I also expect the performance to exceed the pixel-wise Bayesian with a threshold I select through inspection.

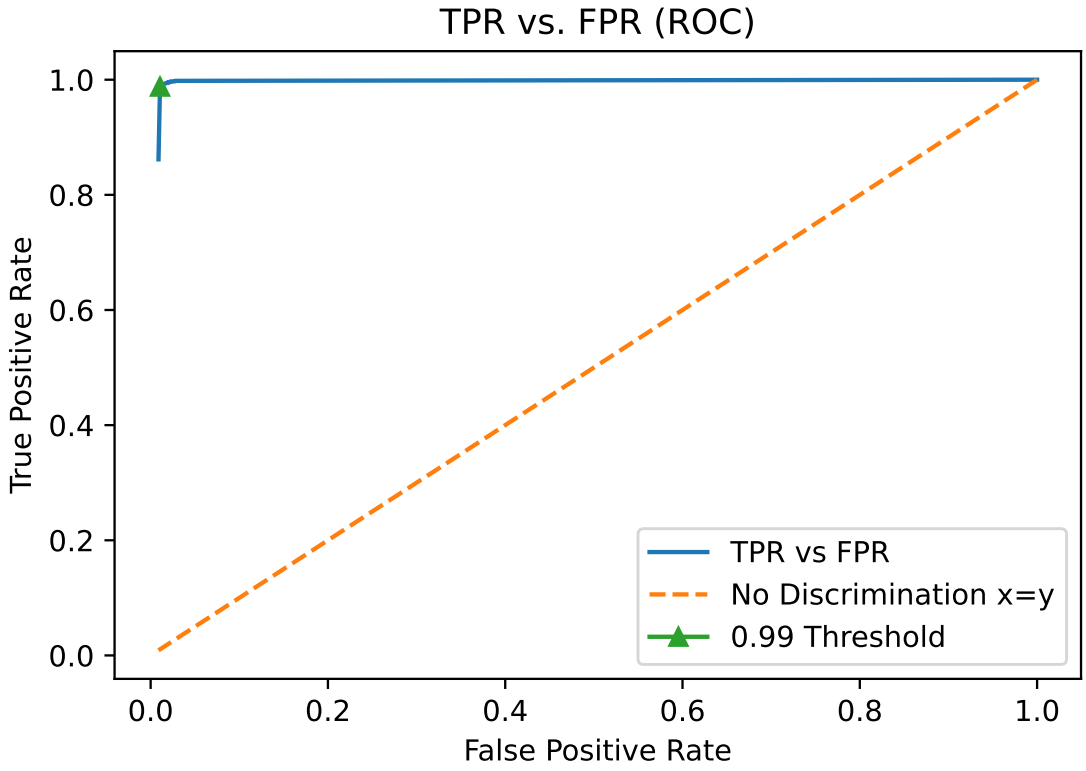


Figure 7.15: The RANSAC-based grouper with a Random Forest model ROC displays a high degree of separability. The TPR maximizes to a value of 1 with the slightest sacrifice in the FPR. Thus the area under the ROC curve is nearly maximized. These ROC curve attributes anticipate exceptional classifier performance.

Next, I apply the Random Forest classifier to generate the ROC curve with the RANSAC Grouper in Figure 7.15. Unlike the Bayesian classifier, the Random Forest classifier is nearly ideal with an initial TPR of nearly 1 without a sacrifice

of an increased FPR. The Random Forest trees actually enables cases where the satellite class has all of the output weight. Therefore, the TPR curve does not start at 0 when the threshold is 1. As the threshold decreases, only a small sacrifice in the FPR value will maximize the TPR value. As a result, the area under the curve is an almost perfect value of 1, which indicates the classifier is capable of a high degree of separability.

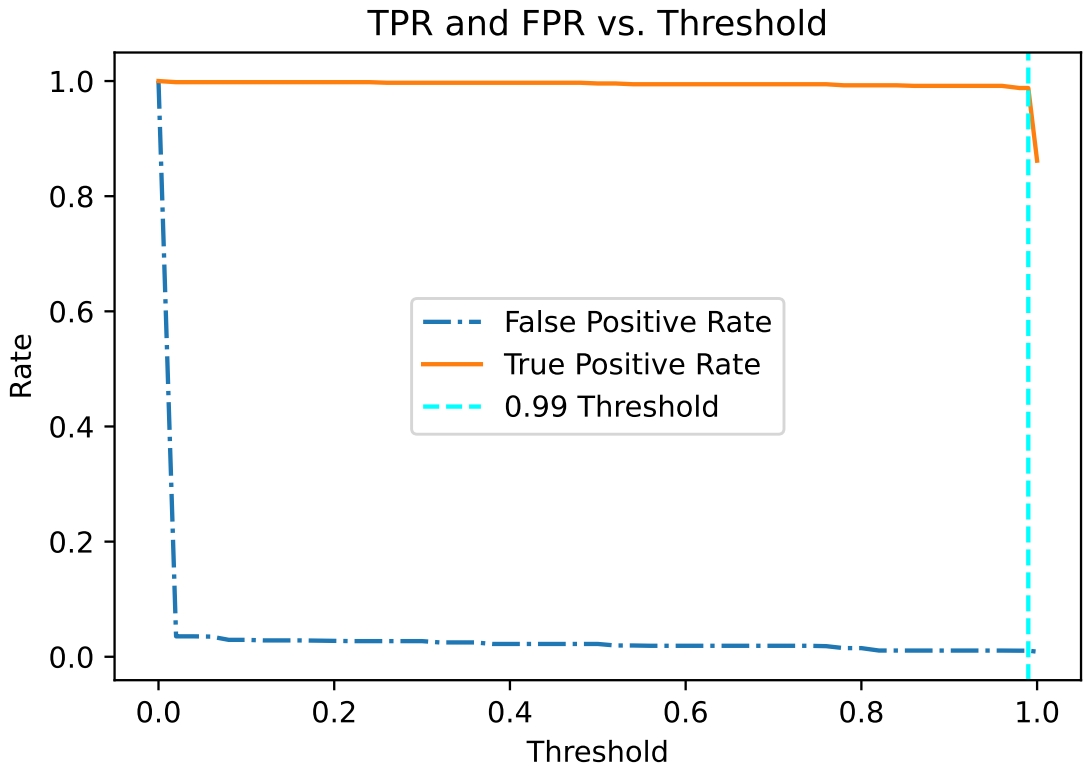


Figure 7.16: The RANSAC-based grouper with a Random Forest classifier satellite TPR and FPR as a function of threshold indicates I accept some loss with the initial online classifier performance using a threshold of 0.8. Because the TPR plateaus at such a high threshold value, choosing a threshold of 0.8 only increases the FPR. The maximum value between the TPR and FPR curves occurs at a threshold value of 0.99.

I choose the tuned threshold with the TPR and FPR as a function of threshold as Figure 7.16 displays. As anticipated, the TPR plateaus at a much higher threshold value and the FPR slowly increases as the threshold decreases. Since

the TPR levels out, there is no reason to accept the additional false positives of the original 0.8 threshold value applied in Section 7.2.3. The threshold of 0.8 I use in the initial online classifier analysis allows unnecessary false positives with no additional true positives. Therefore, I expect the tuned threshold to provide better performance. Instead, I choose a threshold of 0.99 to maximize the Random Forest classifier performance.

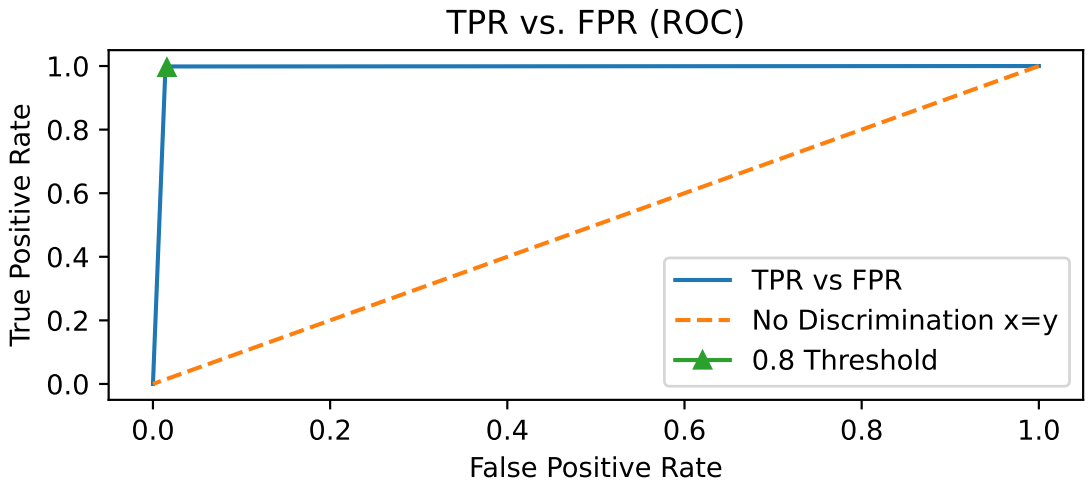


Figure 7.17: The RANSAC-based grouper with a Dense Neural Network classifier ROC curve is reminiscient of the Random Forest classifier ROC curve. The rise to a maximized TPR is a bit slower than the Random Forest model, but the Dense Network classifier still has a nearly ideal area under the ROC curve and, therefore, should display a high degree of separability.

Finally, I implement the Dense Neural Network classifier with the RANSAC-based grouper to generate the ROC curve in Figure 7.17. Like the Random Forest model, the performance is near ideal. However, the initial TPR value starts at 0, so there are no cases where the classifier's weight is only in the satellite class as with the Random Forest model. Additionally, the curve maximizes at a slightly higher FPR value. Once there, the TPR remains virtually unchanged.

The Dense Neural Network TPR and FPR versus threshold in Figure 7.18 highlights a slight increase in the difference between TPR and FPR at a thresh-

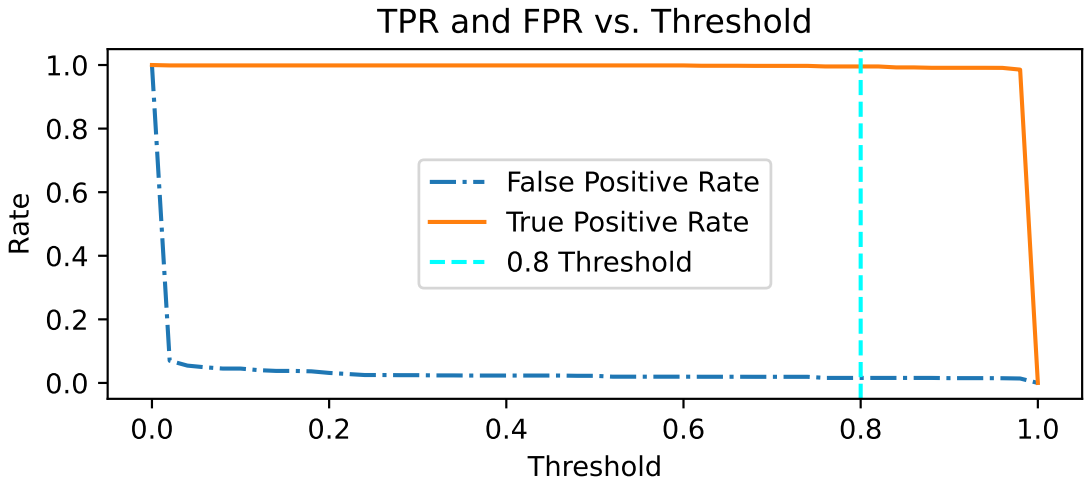


Figure 7.18: The RANSAC-based grouper with a Dense Neural Network satellite TP and FP vs. Threshold

old value of 0.8. This change is difficult to view in the ROC curve because of the minimal change in both TPR and FPR at this value. Therefore, the change occurs in the corner of the ROC curve. In order to maximize the TPR and FPR difference, I choose the 0.8 threshold for the Dense Neural Network using the RANSAC-based grouper.

Proximity Grouper ROC Curves

After creating the ROC curves for the RANSAC-based grouper, I separately create curves for the proximity-based grouper. The proximity-based grouper not only creates groups differently than the RANSAC-based method, but it does not apply the star pre-classifier. The different algorithmic construction sends all groups to the classifier and, therefore, requires different thresholds in order to optimize classifier performance for satellite identification. The following ROC curves and threshold analysis below allows me to select better performing thresholds.

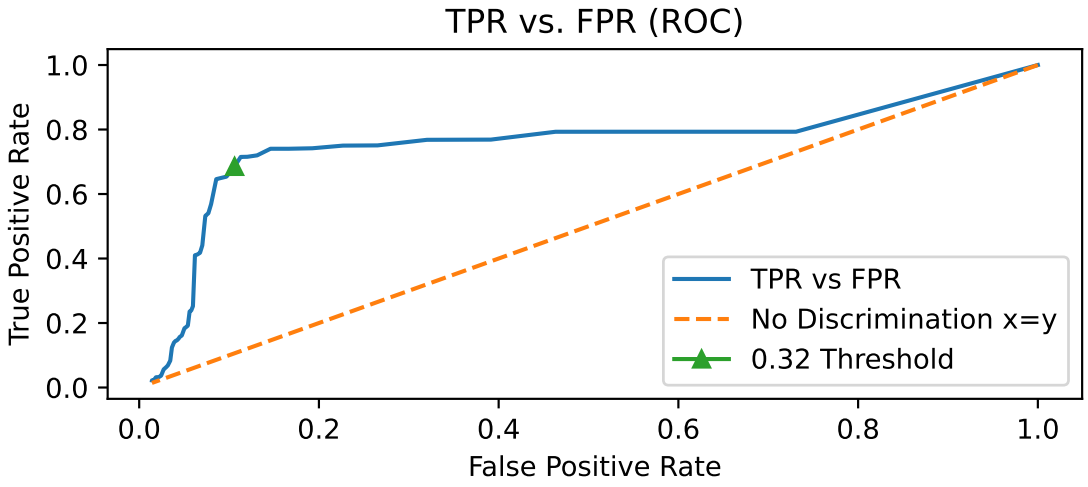


Figure 7.19: The proximity-based grouper with a Bayesian classifier ROC curve is less successful than the RANSAC-based grouping method when comparing the total area under the curve. Despite an initial increase in satellite TPR, the curve levels out at a lower percentage. This lower TPR value reduces the area under the curve and, therefore, the grouper and classifier combination cannot achieve as high a level of separability as seen with other combinations. At low enough thresholds the curve converges towards the $x = y$ line, so at too low a threshold the classifier does not perform better than random guessing.

I start with the Bayesian classifier as I did with the RANSAC-based clustering. Figure 7.19 shows the ROC curve of this classifier using the proximity-based grouper. The performance is underwhelming when directly compared to the ROC curves with the RANSAC grouper and demonstrates the importance of the grouping method. For a range of higher threshold values, the ROC curve moves away from the random guessing line. However, the TPR levels out around 0.7 as opposed to the 0.9 value of the Bayesian classifier - RANSAC-grouper combination. This approximate TPR remains steady until at low threshold values the ROC curve converges with the $x = y$ line. In this region the classifer is no better at identifying satellites than random guessing. Overall, there is less area under the ROC curve indicating the Bayesian classifier with the proximity-based grouper is less capable at separating satellite information

from the rest than the other combinations.

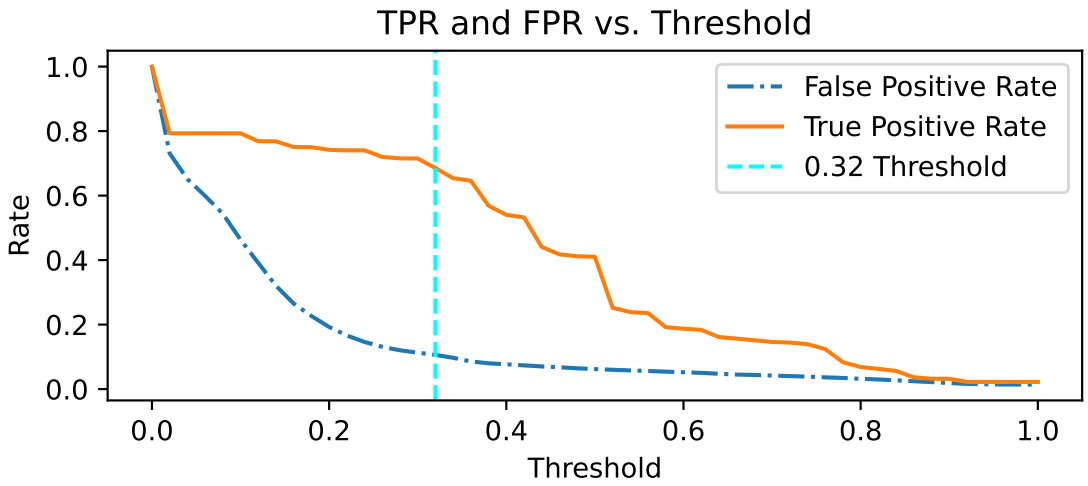


Figure 7.20: The proximity-based grouper with a Bayesian classifier satellite TPR and FPR as a function of threshold highlights the gentle rise of the TPR as compares to other grouper and combination metrics. Fortunately, the FPR remains relatively consistent for a portion of the threshold decreasing. Once the FPR starts to rise, the difference between FPR and TPR maximizes at a threshold value of 0.32.

Despite the weaker ROC curve performance, I still select an optimal threshold value. I do this to continue a fair comparison with the optimal outputs in Section 7.2.6. Using the same technique as with the RANSAC-based grouper, I examine the TPR and FPR as a function of threshold, shown in Figure 7.20, to select the threshold providing the maximum difference between the TPR and FPR values. The threshold of 0.32 maximizes the difference. Interestingly, this threshold is the same as the one I select for the Bayesian Classifier with the RANSAC grouper. This may be due to the Bayesian classifier operating on my selected data attributes. There may be less variation in these attributes despite a change in grouping method, meaning the Bayesian classifier still requires approximately the same threshold to maximize the difference in the TPR and FPR.

Next, I generate the Random Forest classifier with the proximity-based

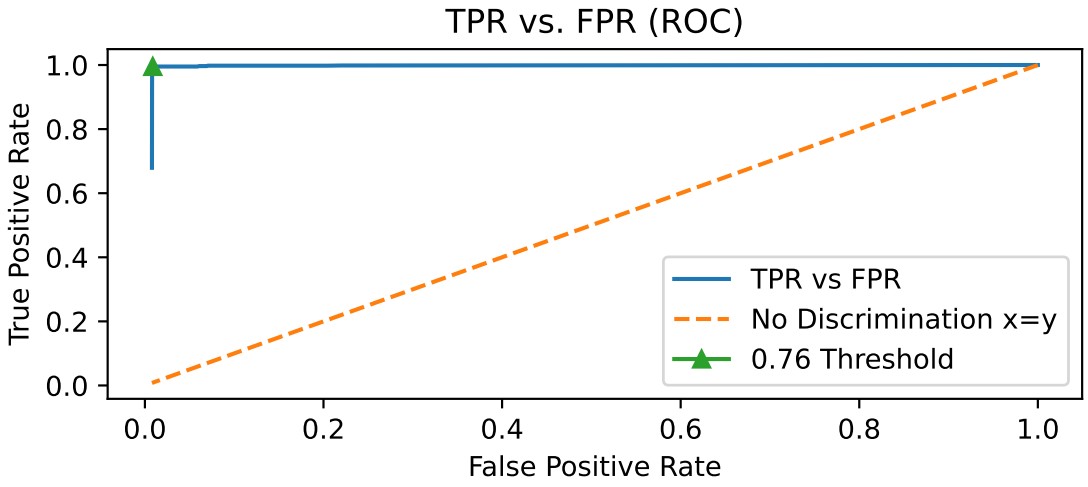


Figure 7.21: The proximity-based grouper with a Random Forest model ROC curve looks nearly identical to the RANSAC-based grouper ROC curve with the same model. Again, some group combinations have their entire weight into the satellite class as the TPR value at a threshold value of 1 is not 0. Additionally, the ROC curve nearly maximizes the area under the curve, so the combination provides high separability to the satellite events.

grouper ROC curve. The ROC curve in Figure 7.21 looks nearly identical to the ROC curve of the same model and the RANSAC-based grouping method. Therefore, the Random Forest model is resilient to changes in the grouping method and can handle star data rejection well without the star pre-classifier. As with the RANSAC grouper version, the TPR does not start at 0, so some combinations of grouped events pass the satellite hypothesis test at a threshold value of 1. Therefore, the Random Forest model is still capable of having all the weight in the satellite class output despite potential changes in grouping. The TPR at the threshold value of 1 is slightly lower for the proximity-based grouping method, approximately 0.9 versus 0.7, so fewer groups meet this condition. The ROC curve, however, quickly maximizes the TPR without much change in the FPR value, so the area under the curve is near maximal. Overall, the Random Forest model appears to be trained well to identify satellite objects.

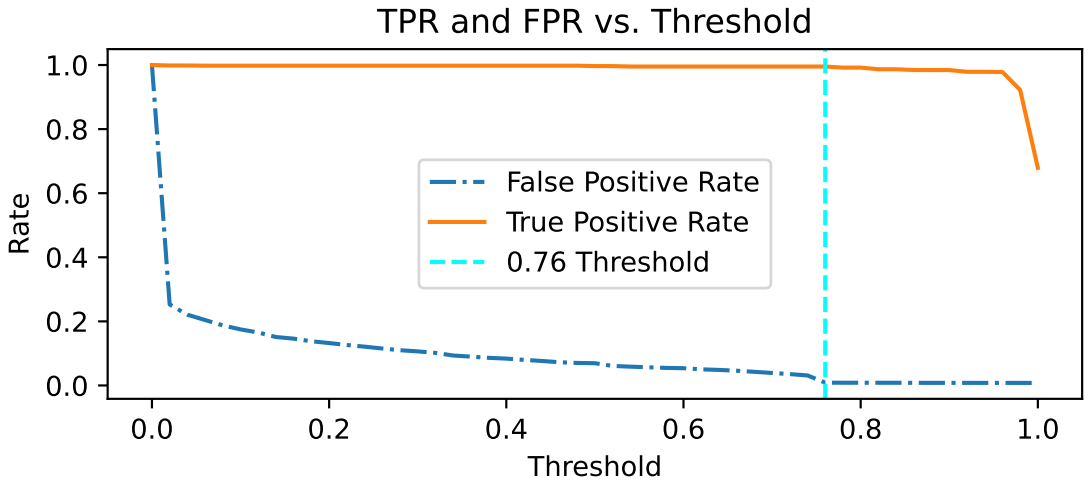


Figure 7.22: The proximity-based grouper with a Random Forest model satellite TPR and FPR against varying threshold values highlights a small change in performance between this combination and the Random Forest classifier with the RANSAC-based grouper. The TPR only maximizes with a lower threshold value. The cost of this lower threshold value is not significant because the FPR is essentially unchanged at the higher threshold values. The maximum difference between the TPR and FPR occurs at a threshold value of 0.76.

Again, I look at the TPR and FPR against the threshold values in Figure 7.22 to select an appropriate threshold for this combination of grouper and classifier. Unlike the ROC curve, the TPR and FPR curves in this Figure vary from their RANSAC-based grouping counterpart. In particular, it takes a lower threshold value for the TPR to reach a value of 1. Despite this requirement for a lower threshold value to achieve this performance, the FPR value does not increase during this drop in threshold. As a result, the difference between the TPR and FPR maximizes at a threshold value of 0.76 without much increase to the FPR.

The final combination of grouper and classifier I tune is the Dense Neural Network classifier with the proximity-based grouper. Figure 7.23 demonstrates that this combination of grouper and classifier also has similar high separability characteristics as the RANSAC-based grouper with this same classifying

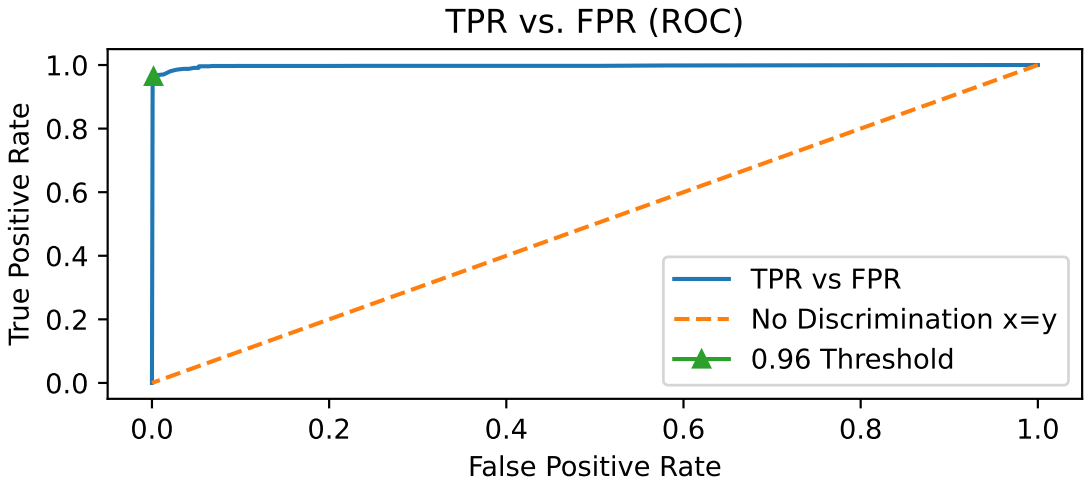


Figure 7.23: The proximity-based grouper with a Dense Neural Network ROC curve still exhibits a near optimal area under the curve indicating good separability of satellite events. The curve is not identical to its RANSAC-based counterpart. The initial jump in TPR occurs without much change in the FPR value. The jump is also not to a TPR value of 1. Instead, the TPR slowly converges to a value of one as the FPR increases.

model. However, the initial jump in TPR requires less of a FPR increase than the RANSAC grouped version. Ostensibly, this indicates more sensitivity to the satellite events without sacrificing specificity. However, the TPR does not maximize at 1 during the initial jump in the curve, making the area under the curve less than ideal and a trade-off must be made to choose the threshold value.

Despite the aforementioned characteristic in the ROC curve, I continue with my original threshold selection tactic and examine the results presented in Figure 7.24 to choose a threshold that creates the maximum difference between the TPR and FPR matrics. With this method, I select a threshold value of 0.96 to maximize the Dense Neural Network classifier performance with proximity-based grouping output. If I desire to find a higher percentage of satellite events for a subsequent algorithm, it may be worth the trade-off of more false positives through selection of a slightly lower threshold. This choice is not an option for

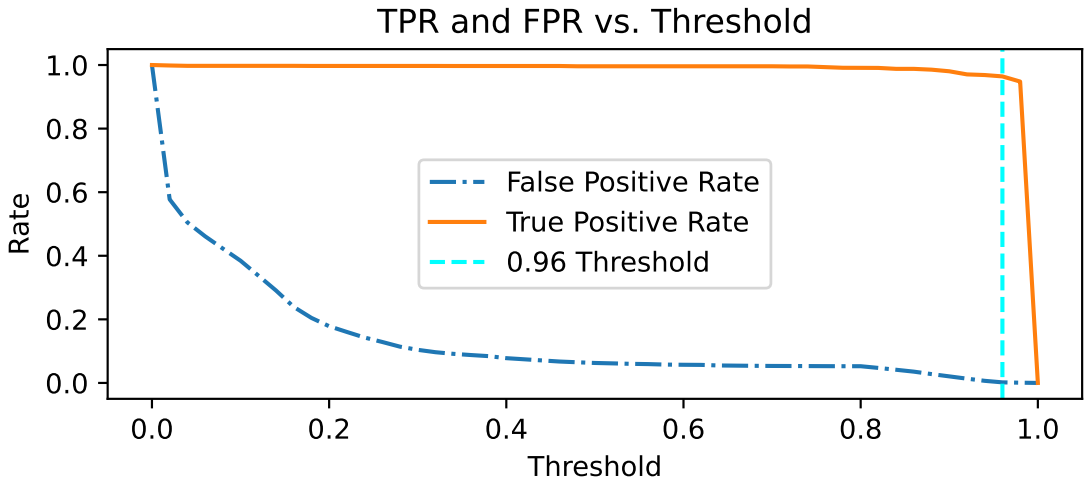


Figure 7.24: The proximity-based grouper with a Dense Neural Network satellite TPR and FPR as a function of threshold shows that despite the non-maximized TPR in the ROC curve, the region after the initial jump in TPR before the FPR begins to increase is still the maximum difference between the TPR and FPR values. The selection of the 0.96 threshold comes with the understanding that some satellite events will be rejected by the classifier.

the Bayesian classifer combinations, which do not have a TPR of 1 even at my chosen thresholds. This is because the Bayesian combinations only reach a TPR of 1 when the FPR is also 1. For now I maintain the tuned threshold of 0.96 that maximizes the difference to allow me to compare the model performance.

Chosen Thresholds

I tune 6 different combinations of grouper and classifier as discussed in the previous Section. Table 7.19 shows a synopsis of the thresholds I choose for each grouper and model combination, given the ROC analyses. This summary highlights the importance of tuning the models for any grouping method because the information reaching the classifier will change as a result. These thresholds should assist in my attempt to improve both the number of identified satellite events and reduce the number of non-satellite events passed by the classifier to

the next stage of the tracking algorithm.

Table 7.19: Chosen Thresholds from ROC Analyses

Model	**RANSAC**	**Proximity**
Bayesian	0.32	0.32
Random Forest	0.99	0.76
Dense Network	0.8	0.96

7.2.5 Classifier Timing Analysis

Given that the grouping and classifier combinations have optimally tuned thresholds from Section 7.2.4, I now explore the performance of these combinations during their online application. This exploration includes investigating the relationship between the number of events and the data set time duration against the satellite classification performance. Through this analysis, I evaluate satellite prediction accuracy with respect to event quantity in a grouping, as well as total duration of the grouping. This analysis will inform future work by finding required event quantities or durations necessary for classification.

By understanding how much information the classifiers require to make better predictions, I can potentially reduce the algorithmic complexity by only considering event groups with sufficient information. With this, I can build an argument to discard selected grouping sizes when there is not enough information for them to be useful. The temporal analysis informs how much prior event information I should use in a continuous online version of the tracking algorithm. Currently, as I run the grouping methods, I include all prior event information. This is possible in a limited 20 second collection. Continuous operation will consider a sliding window of events and through this analysis I can propose the duration of that window.

For this analysis, I inspect groupings that have a truth label of satellite in the validation set. I record the classification history and time of the groupings each time the classifier evaluates the grouping online. It is important to note, for the RANSAC-grouper, the star group pre-classifier also evaluates the satellite groups. After the online run of the grouper and model combinations for the validation data sets, I work with the classification history and time output. I iterate through each event in the grouping, and compare times of the event versus the predicted class of the grouping. With this technique, I establish both a quantitative and temporal relationship with the grouping's classification and its events.

For the accuracy versus event count, I evaluate the classification of every true satellite group at each binary event level in the group. Since not all satellite groups are the same length, I conduct a roll-up method for groupings that have fewer events than the largest grouping. With this method, I maintain the smaller groupings' classification accuracy at their final classification and average it in to the larger event counts. The final accuracy value for the event count should thereby converge to the overall TPR value of the satellite class' predictions.

For the timing analysis, I do not conduct a similar roll-up. Instead, I inspect the classification accuracy averaged only over groups that exist with at least a minimum duration, to better establish a relationship between a group's longevity and the accuracy of the class prediction of the grouper and classifier combination. To conduct this analysis, I group events in 0.1 second bins over the course of the group's duration and set the first event time to 0 seconds. Due to not applying a roll-up methodology on the timing analysis, the results between the event count and timing analyses are not directly comparable. Instead, I

mark on the timing analysis Figures where classification performance exceeds 80% and 90% for all subsequent time values, in order to examine event counts and durations that reliably inform classification decisions.

While I primarily focus on the performance regarding satellite events in this Section, I also explore the event count and timing relationships for star group accuracy in the same manner for the RANSAC-based grouper. I inspect these features to draw conclusion on the star grouping pre-filter influence on the RANSAC-based grouper overall classification results.

RANSAC Grouper Class Timing

I start the class timing analysis with the RANSAC-based grouper in combination with my three classifiers. The first combination I examine is the RANSAC-based grouper with the Bayesian classifier. Figure 7.25 demonstrates the roll-up accuracy as a function of event number. The accuracy of the classes improves with more event information which follows classification performance increases seen in Section 7.2.3 when I increase the model sizes. This same principle also applies to the Bayesian classification. In the Figure I mark when the accuracy plateaus and no longer changes. This final accuracy value is close to the final satellite TPR in Section 7.2.6. It is not an exact match because the accuracy metric does not include false positive events in the denominator. While the combination achieves the plateau at 1481 events, the accuracy nears this final accuracy value for the first time with groups around 30 events.

Next I assess the accuracy as a function of the time during which a satellite group exists. Around 2 seconds of satellite events, the classification plateaus at

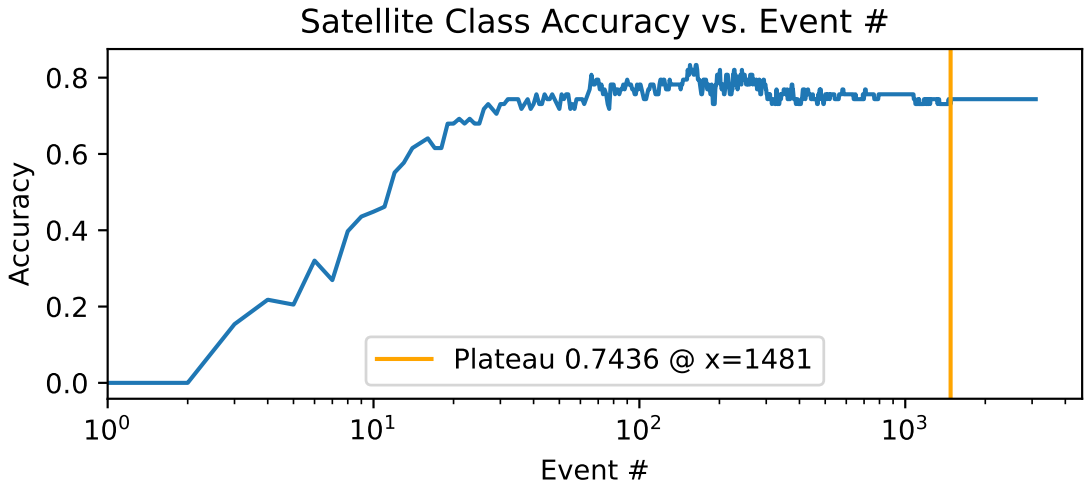


Figure 7.25: The RANSAC-based grouper with a Bayesian classifier satellite class accuracy versus group event count demonstrates that the accuracy converges to approximately 0.7436 satellite accuracy with only 30 events. The accuracy stops changing after 1481 events and remains consistent.

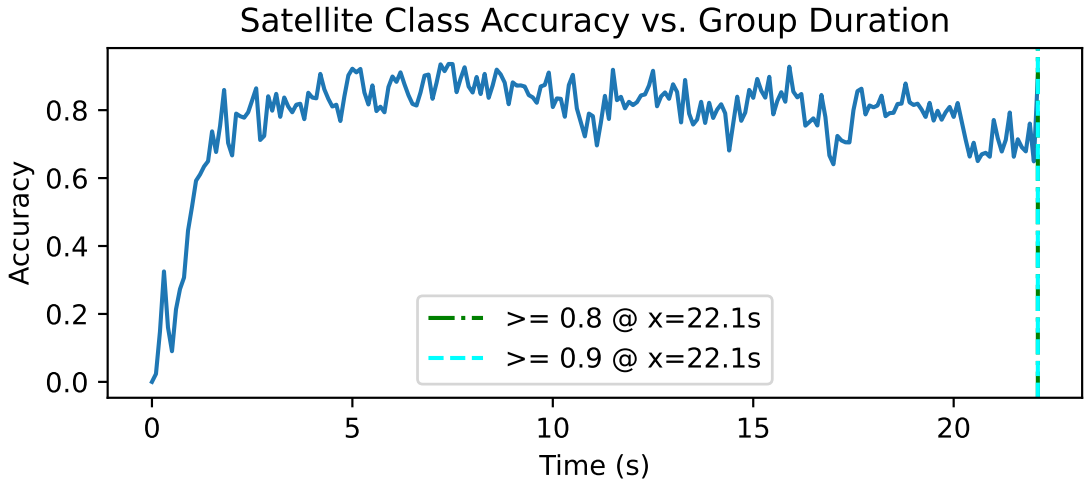


Figure 7.26: The RANSAC-based grouper with a Bayesian satellite class accuracy versus group time shows that the accuracy spikes after only 2 seconds. Once around the accuracy of 0.75, the classification accuracy holds until approximately 10 seconds. After 10 seconds the accuracy declines.

its approximate accuracy of 0.75 as Figure 7.26 shows. While the satellite accuracy occasionally passes 80%, it does not hold that value, but it does surpass it at the final time in the plot. Therefore, the lines marking the past 80% and 90% thresholds are at the final time bin. I do not depict individual group classification histories, so it is important to note that after the initial jump in accuracy at the start of each group, there is generally not a significant increase with more events or longer represented time periods. In fact, in the case of the event number, the accuracy increases until about 100 events. Then it begins to decline. The decline is also visible with the group duration. If the time duration of a group increases beyond 10 seconds, it appears to lead to a steady decline in predictive performance.

Now I look at star classification for the combination of the Bayesian classifier with the RANSAC grouper. With the RANSAC grouper, I pre-filter a large portion of the stars and apply the star class label. Therefore, I only need a few events that generate the approximate right slope for the group to reach the star classification. As a result, the star group filter exceeds 80% accuracy at only 7 events, followed by exceeding 90% at 8 events as shown in Figure 7.27. The star filter curve is also much smoother than the satellite classifier curve. This indicates the star group pre-filter functions well. It identifies most stars and they rarely need to be reevaluated when additional events add to the group.

The consistency from not reevaluating the classification is also visible in the accuracy versus the group duration as Figure 7.28 depicts. There is less variation in the class accuracy over time because fewer star groups see the formal classifier. In this Figure, I mark where the accuracy exceed the 80% threshold and the 90% threshold consistently corresponding approximately to the 7 and

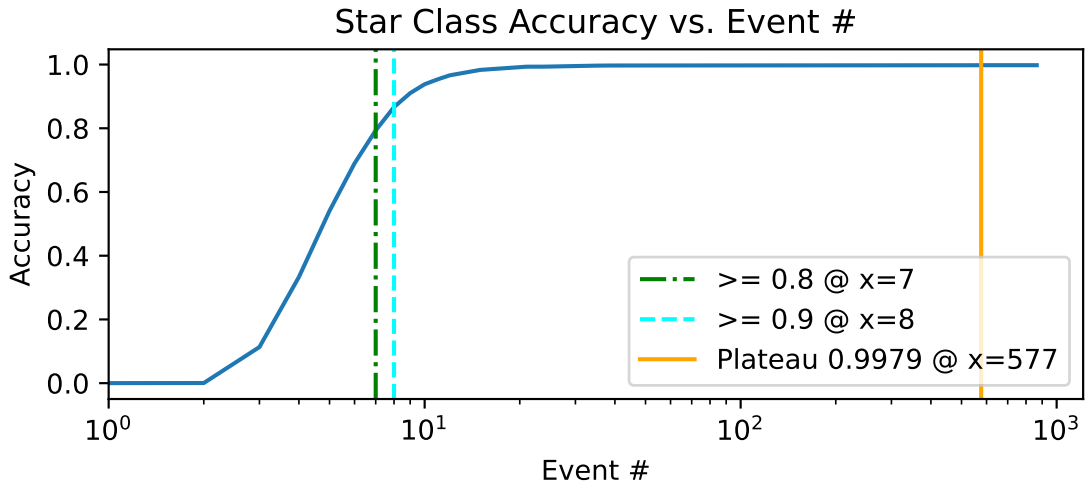


Figure 7.27: The RANSAC-based grouper with a Bayesian star class accuracy versus group event count only takes 8 events to converge to the 90% accuracy and before 20 events the curve settles around the final accuracy measurement.

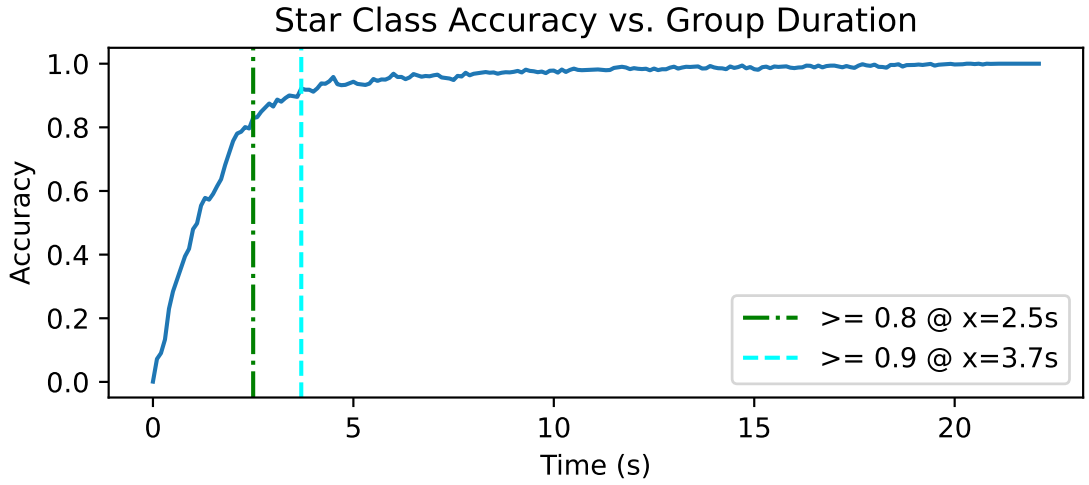


Figure 7.28: The RANSAC-based grouper with a Bayesian star class accuracy versus group event time demonstrates the consistency in the class accuracy offered by the star group pre-filter method. Less variation occurs over time. The curve surpasses the 80% and 90% accuracies at 2.5 and 3.7 seconds respectively.

8 events of the group event count graph. The curve passes the 80% accuracy at 2.5 seconds and the 90% accuracy at 3.7 seconds which matches similarly to the convergence of the satellite class despite a different method conducting most of the classification.

The next model combination I examine is the Random Forest Model with the RANSAC Grouper. Figure 7.29 shows the satellite accuracy as a function of event count. The accuracy exceeds 80% at group sizes of 39 events and never decreases. This follows the performance increase the larger model sizes can provide. The jump occurs at the 40-event model size which suggests model size thresholds are an important tunable factor of the machine learning model implementation. In fact, there are the jumps in this graph where the larger event models replace the smaller ones. It does not predict below 5 events as I design. Then at 10 events, 20 events, 40 events, and 80 events there are increases in accuracy, albeit with diminishing returns each time.

Considering the group duration in Figure 7.30, the curve passes the 0.80 accuracy at 1.4 seconds. Interestingly, without the roll-up averaging the lower accuracies of the smaller event groups reveal the higher accuracies of the larger groups, the accuracy over group duration exceeds 90% at 2.1 seconds. Comparing the performance to the Bayesian classification, the convergence is not only to a higher accuracy, but the resulting accuracy is more stable over the duration of the groups. The Random Forest model does not have the decrease in accuracy the Bayesian model demonstrates. This indicates the Random Forest model sliding window is a better method to handle the classification over time. In the future, it is worth examining similar methods to limit the pixels voting on the Bayesian classification over time. Not only will that construction be more

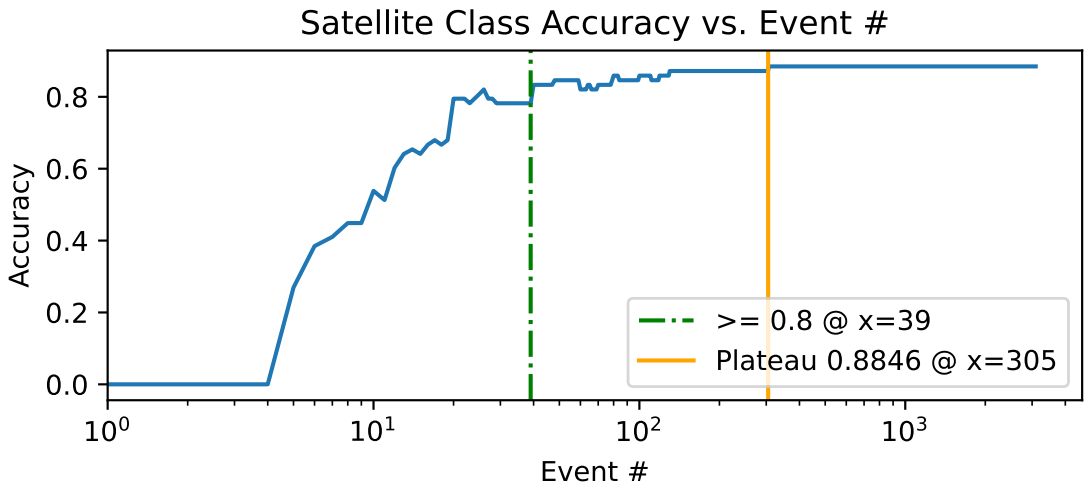


Figure 7.29: The RANSAC-based grouper with a Random Forest satellite class accuracy versus group event count highlights the importance of the classifier model sizes. Every class size increase, 5, 10, 20, 40, and 80 events each see an increase in accuracy. The event accuracy exceeds 80% at the 40 event model jump.

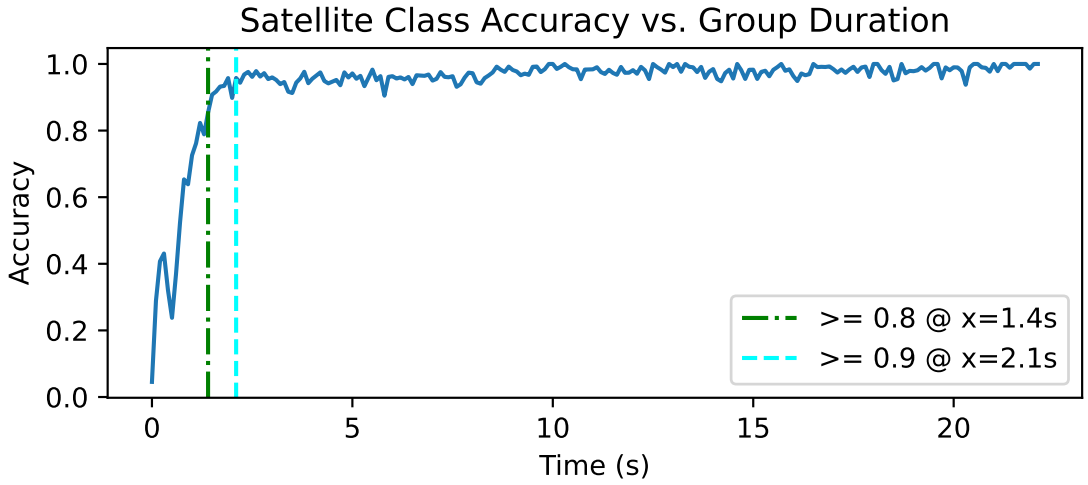


Figure 7.30: The RANSAC-based grouper with a Random Forest satellite class accuracy versus group time demonstrates the machine learning sliding window classifies long duration groups more successfully than the Bayesian model. After a quick convergence to an accuracy of 90% at 2.1 seconds, the classifier holds throughout the duration of the rest of the group lengths.

manageable for online use because the data sets will no longer be finite, but potentially it can improve the Bayesian classification over time to mimic what works well with the machine learning models.

I consider the stars classification for the Random Forest Model with the RANSAC grouper. In Figure 7.31, the accuracy exceeds 80% at 8 events and 90% at 12 events, slower than the Bayesian classifier. While the smooth slope still indicates the star group pre-classifier handles most of the groups, the change in number of events required to accurately classify satellites has its foundation in the high star TPR the Bayesian classifier achieves. I discuss the star TPR performance yielding this change in Section 7.2.6. Since the Random Forest classifier has lower TPR for stars, the star groups that make it through to the classifier are not as likely to be identified by the classifier. This is particularly pronounced with smaller group sizes, shown by a drop in satellite TPR with smaller group sizes discussed in Section 7.2.3.

Figure 7.32 depicts the impact of the lower Random Forest star TPR performance on the time to convergence. In the run with the Random Forest model, the star group accuracy curve reaches 80% at 3.7 seconds and 90% at 5.5 seconds. These values are also slower than the Bayesian's 2.5 and 3.7 seconds respectively.

The Dense Neural Network with the RANSAC grouper is the next model I examine. I depict the accuracy as a function of group events in Figure 7.33. In this case, the accuracy exceeds 80% at 19 events and 90% at 99. This classifier is the first to exceed 90% accuracy in the satellite event graph, and achieves the 80% accuracy in nearly half the events as the Random Forest model. The accuracy versus event curve also exhibits the same accuracy jump phenomenon of

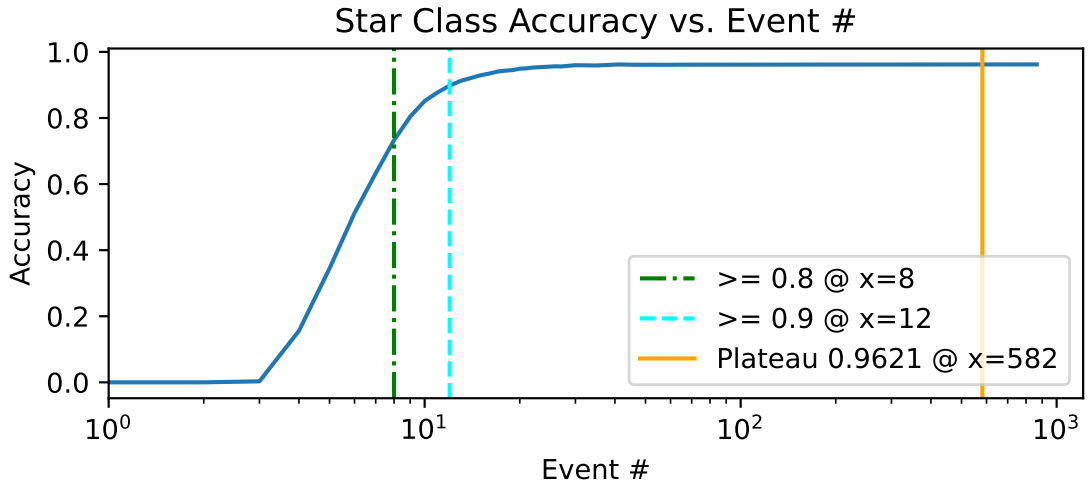


Figure 7.31: The RANSAC-based grouper with a Random Forest star class accuracy versus group event count shows the influence of classifier choice still influences the star classification performance despite the RANSAC star grouper pre-classifier. The convergence to 80% and 90% accuracy occurs at 8 ad 12 events respectively, more events than the Bayesian classifier.

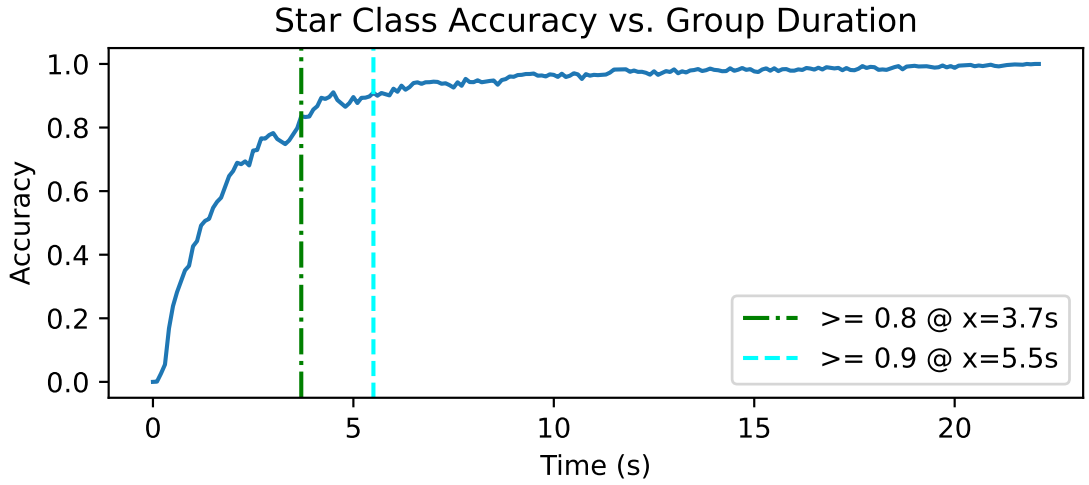


Figure 7.32: The RANSAC-based grouper with a Random Forest star class accuracy versus group event time also displays a slower response time to convergence than the Bayesian classifier reaching accuracy of 80% at 3.7 seconds and 90% at 5.5 seconds. This is a consequence of the lower star TPR with the Random Forest model than the Bayesian classifier.

the Random Forest model when event numbers are sufficient enough to proceed to the next model size.

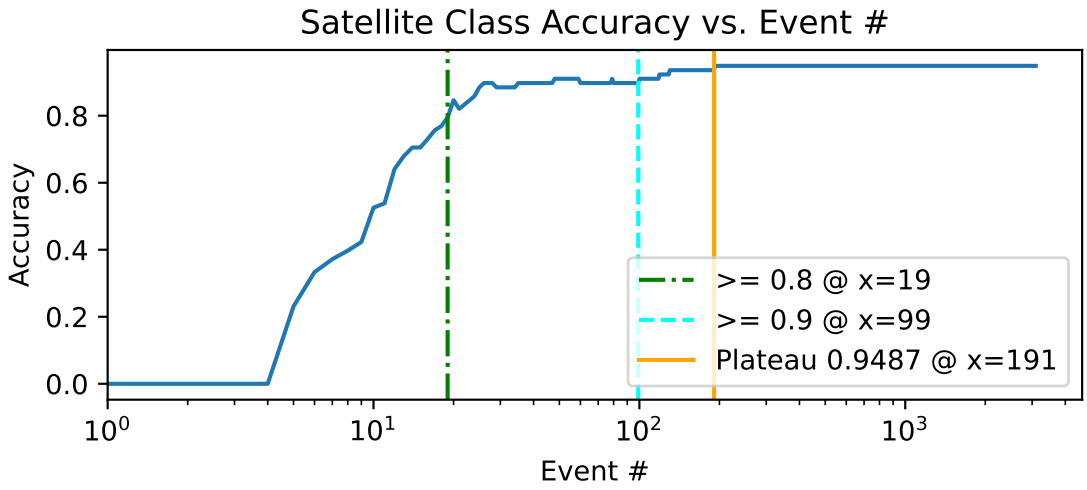


Figure 7.33: The RANSAC-based grouper with a Dense Network satellite class accuracy versus group event count lowest number of events, 19, to converge to the 80% accuracy and is the only satellite classifier to exceed an accuracy of 90% at 99 events. Like the Random Forest model, the accuracy jumps at changes in the model sizes.

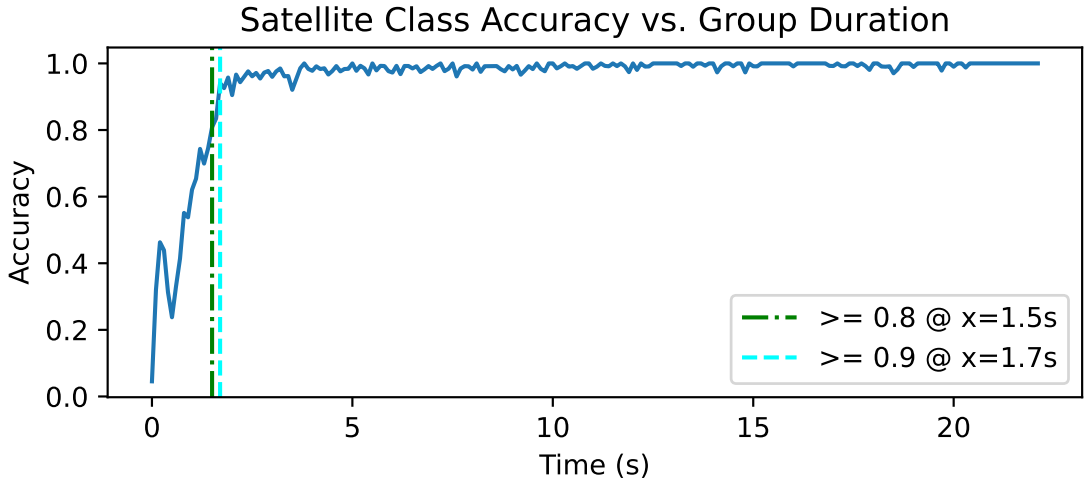


Figure 7.34: The RANSAC-based grouper with a Dense Network satellite class accuracy versus group time takes longer to converge to the 80% accuracy value but is faster to the 90% accuracy than the Random Forest model. This combination of grouper and classifier demonstrates the most stable output for groups of increasing duration.

When I compare the Random Forest satellite classification performance to

group time in Figure 7.34, the Dense network requires 1.5 seconds of events to exceed an accuracy of 80%. This is 0.1 seconds slower, or one time bin, than the Random Forest classifier. This additional 0.1 second is likely inconsequential, especially considering the Dense network quickly achieves an accuracy greater than 90%, at 1.7 seconds. This is 0.4 seconds faster than the Random Forest model. One additional benefit of the Dense Neural Network is the improved stability of the classification over even the Random Forest classifier. The classification barely deviates from the value over 90% for all satellite group durations after the initial 1.7 seconds.

Finally, I consider the star classification performance for the Dense Network with the RANSAC-based grouper. The profile of the event count versus accuracy curve in Figure 7.35 is essentially identical to the curve the Random Forest classifier produces. Both the Random Forest and Dense Network models have lower star TPR performance than the Bayesian classifier. However, the consistency indicates that both models primarily rely on the star group pre-classifier to capture most of the star groups. The Bayesian classifier provides more initial support to groups that have passed the star group pre-classifier. However, the Random Forest and Dense Network cannot evaluate the classification until 5 events are available and with only 5 events the classification TPR is poor.

Unsurprisingly, if the star group pre-classifier drives both the Random Forest and Dense Network star classifications, the group duration versus accuracy in Figure 7.36 also looks identical to the curve with the Random Forest classifier. This further solidifies the idea that the two models primarily rely on the star group pre-classifier to find the star groups.

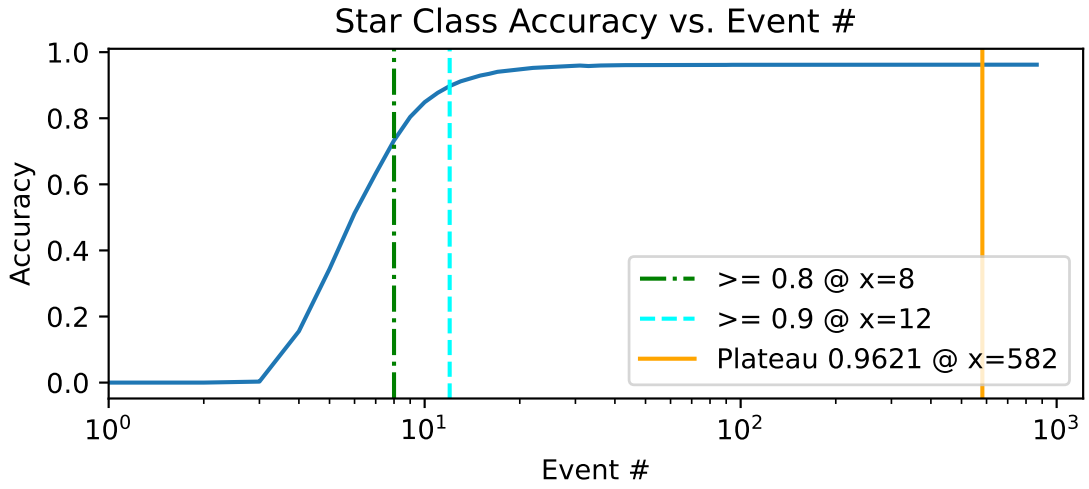


Figure 7.35: The RANSAC-based grouper with a Dense Network Star class accuracy versus group event count is identical to the Random Forest classifier curve of the same metric. These two models have lower star TPR compared to the Bayesian model. Therefore, both implementations rely on the star grouper pre-classifier with early groups and the curves come out identical.

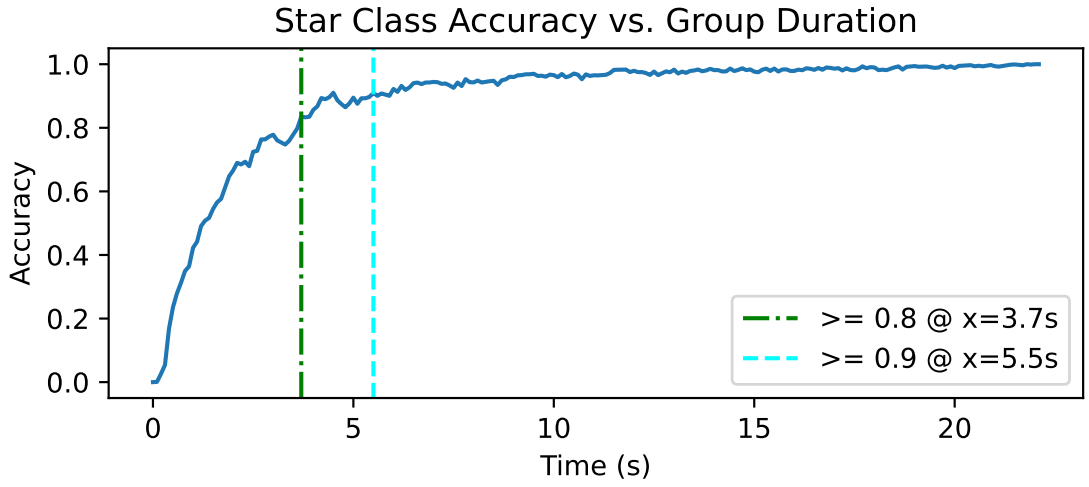


Figure 7.36: The RANSAC-based grouper with a Dense Network star class accuracy versus group event time is also identical to the Random Forest classifier curve of the same metric. Since the event count curves are the same, it is unsurprising that the time duration curve is also the same.

Proximity Grouper Class Timing

The RANSAC-based grouper demonstrates promising online timing results, particularly with the Dense Neural Network. Now, I compare the RANSAC-based performance to that of the proximity-based grouper. I adjust the thresholds for these combinations as chosen in Section 7.2.4. While, the lack of the star group pre-classifier will affect performance, I only observe satellite classification accuracy in this Section. I start, again, with the Bayesian classifier, but this time with the proximity-based grouper. Observing the satellite class timing for this combination, I notice very similar behavior as with the corresponding RANSAC grouper. In Figure 7.37, the classifier initially increases performance until around 100 events, then declines some and plateaus. Additionally, the accuracy reaches its approximate plateau value for the first time around 30 events. Unlike the RANSAC-based grouper, however, the initial spike in accuracy occurs at only 4 events.

As with the event count versus accuracy, the group duration versus accuracy also resembles the performance of the corresponding RANSAC classification model. The accuracy over time in Figure 7.38 initially spikes within the first 2 seconds. Groups of duration longer than 10 seconds see a decrease in accuracy. The identical behavior further solidifies the need to rework the Bayesian before implementing true online processing. As it currently stands, the accuracy drop is less for the groups the RANSAC-based grouping method creates. As a result, I would only recommend the Bayesian classifier using the RANSAC-based grouper.

I next consider the Random Forest model with the proximity-based grouper. Figure 7.39 shows the satellite classification accuracy as a function of event

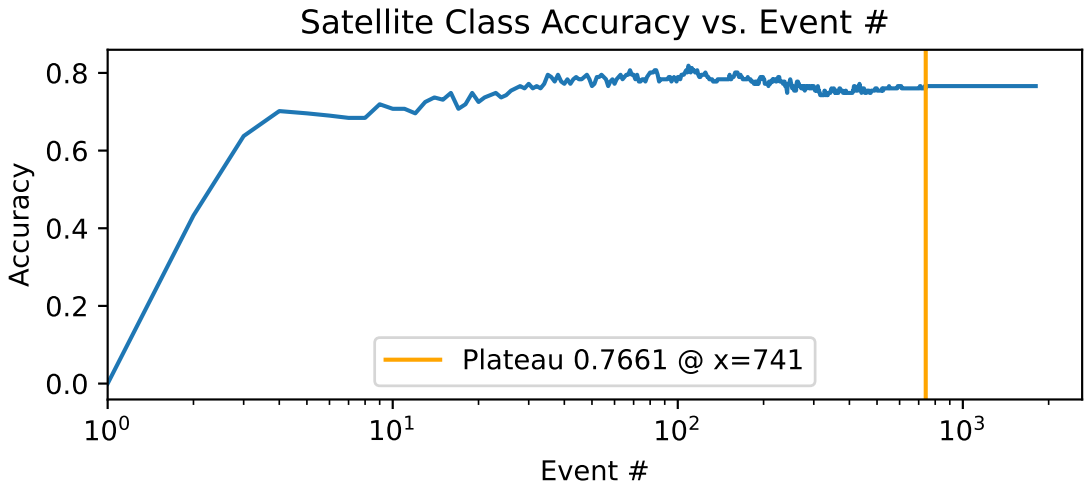


Figure 7.37: The proximity-based grouper with a Bayesian satellite class accuracy versus group event count has similar behavior as the RANSAC version. Around 30 events the accuracy reaches the final plateau value, exceeds it temporarily, peaks in accuracy around 100 events, and, finally, settles to the final plateau value.

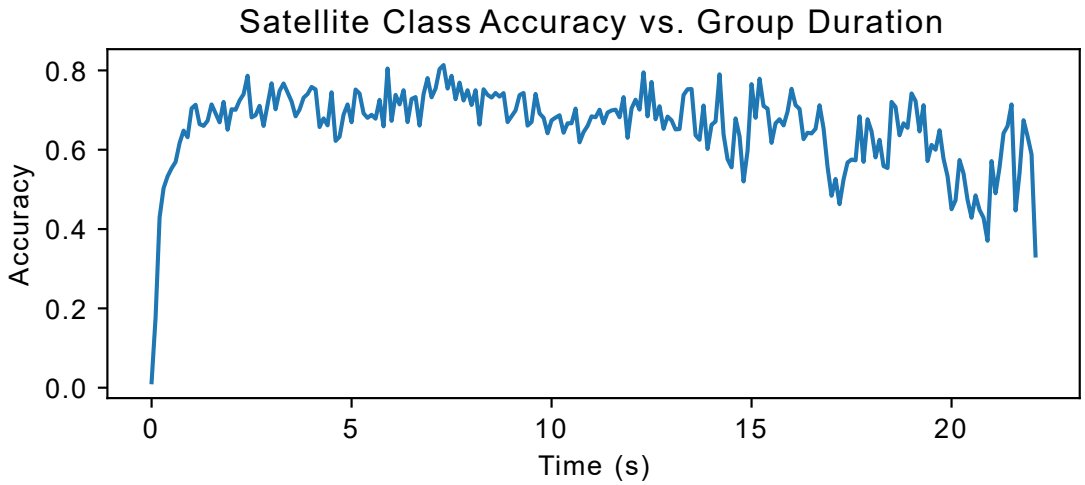


Figure 7.38: The proximity-based grouper with a Bayesian satellite class accuracy versus group time also shares its behaviors with the RANSAC grouper version. The accuracy initially spikes at 2 seconds and drops as the group duration increases. The severity of the decrease, however, is greater than that of the version with the RANSAC-based grouper.

count. The accuracy exceeds 80% on satellites at 11 events and 90% at 29 events. These are the earliest event numbers observed so far for any grouping and classifier combination. This combination also has the highest initial spike in accuracy of the machine learning classifiers after the online method reaches the 5th event. The spike to 0.6 accentuates the promising performance of the Random Forest model with the proximity-based grouper.

In the group duration Figure 7.40 depicts, the Random Forest classifier with the proximity-based grouping continues to preform strongly with respect to early satellite identification. The combination of grouper and classifier exceeds the 80% accuracy at only 0.7 seconds. Despite the temporal advantage, however, the classifier has similar variance as the RANSAC grouper version in the satellite accuracy when the group duration grows. With a lower peak accuracy, this grouper and classifier combination does not sustain greater than 90% accuracy until 20.6 seconds when only a few groups' classifications influence the accuracy metric. In addition, as the number of groups of longer duration decrease, less accurate classifications gain traction. This lowers the overall accuracy in a similar manner to the accuracy decreases seen with the Bayesian classifier combinations, though less severe.

The final grouper and classifier combination I address is the Dense Neural Network with the proximity-based grouper. In this combination's event number versus accuracy curve in Figure 7.41 exhibits an interesting behavior tied to the model sizes. As with previous machine learning models, the accuracy steps up at each increasing model size at 5, 10, 20, 40, and 80 events. However, instead of satellite classification accuracy remaining relatively unchanged before the next step, the accuracy decreases with more events. This indicates that aver-

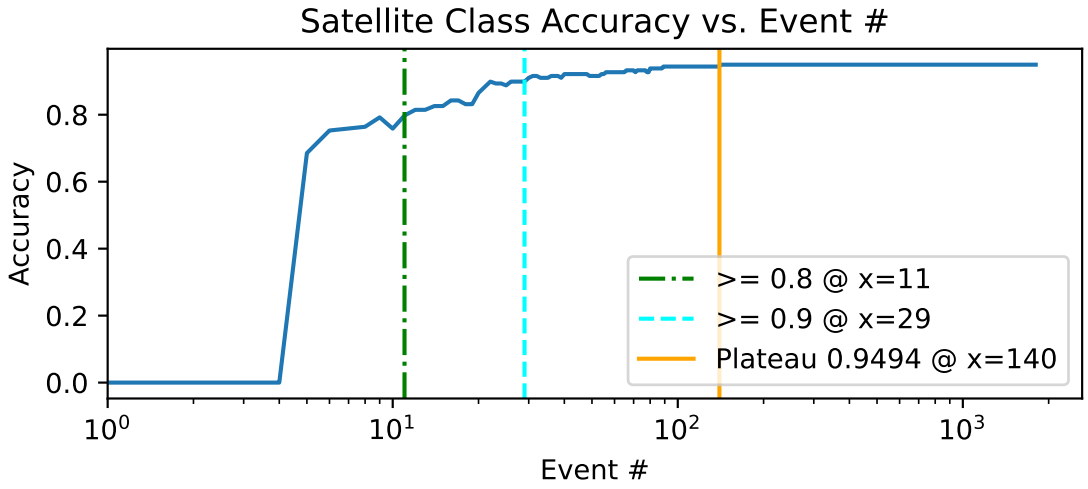


Figure 7.39: The proximity-based grouper with a Random Forest satellite class accuracy versus group event count demonstrates strong accuracy values even with small models. The accuracy jumps to 0.6 with the smallest model and passes the 80% and 90% accuracies at 11 and 29 events respectively.

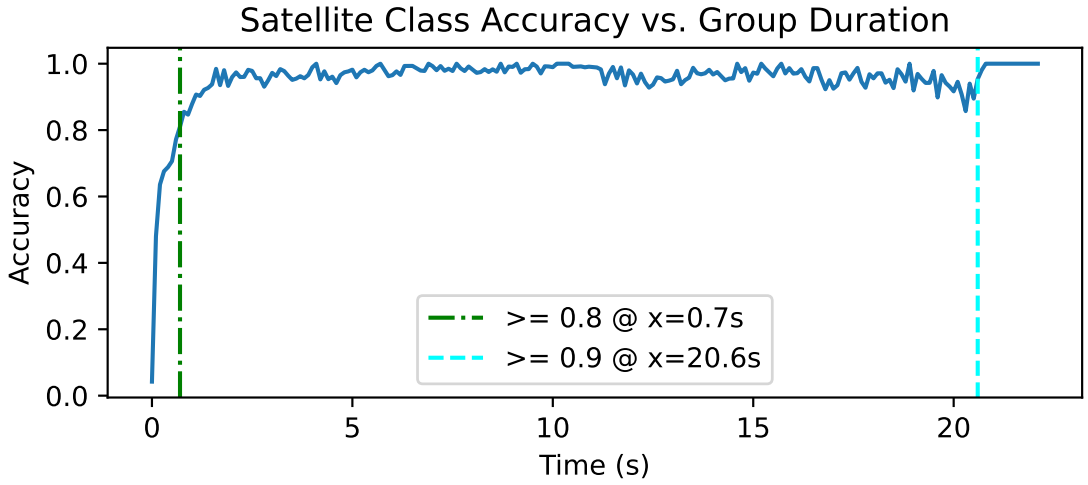


Figure 7.40: The proximity-based grouper with a Random Forest satellite class accuracy versus group time responds extremely quickly averaging an accuracy of 80% at only 0.7 seconds. The accuracy of the group and classifier combination is slightly lower than the version with the RANSAC grouping method, so the classifier does not sustain an accuracy over 90% until 20.6 seconds.

aging results from rolling windows between model sizes may decrease performance when I apply them to the proximity grouper-created groups. As a consequence of the induced sawtooth, the Dense network exceeds the accuracy of 80% at 76 events, significantly slower than the performance of the Dense network with RANSAC groupings. The disparity between these two performances may be a result of the network being trained specifically against RANSAC grouped groupings.

Despite the different behavior at the event count level, the accuracy as a function of group duration resembles the performance of the Random Forest model with a similar curve profile in Figure 7.42, albeit with slower time to the 80% accuracy mark at 1.9 seconds. If I compare the accuracy between this grouper and classification combination and Dense Network with the RANSAC-based grouper, the accuracy experiences more variance as a function of duration. Overall, training models with batch data created with the same method I use to group the data online improves the performance of the encompassing data rejection method. As I discuss in Section 8.2, I intend to explore the performance of classifiers trained on proximity-based grouped truth data to improve classification with proximity-based online grouping.

Classifier Timing Analysis Results

In general, group accuracies increase as the event count and time increases and the classifier has more information available, except in the case of the Bayesian classifier which exhibits a slight decrease in accuracy beyond 100 events and 10 seconds. The fastest accuracy versus event and time, within the grouper and classification combinations, is the Random Forest with a proximity-based

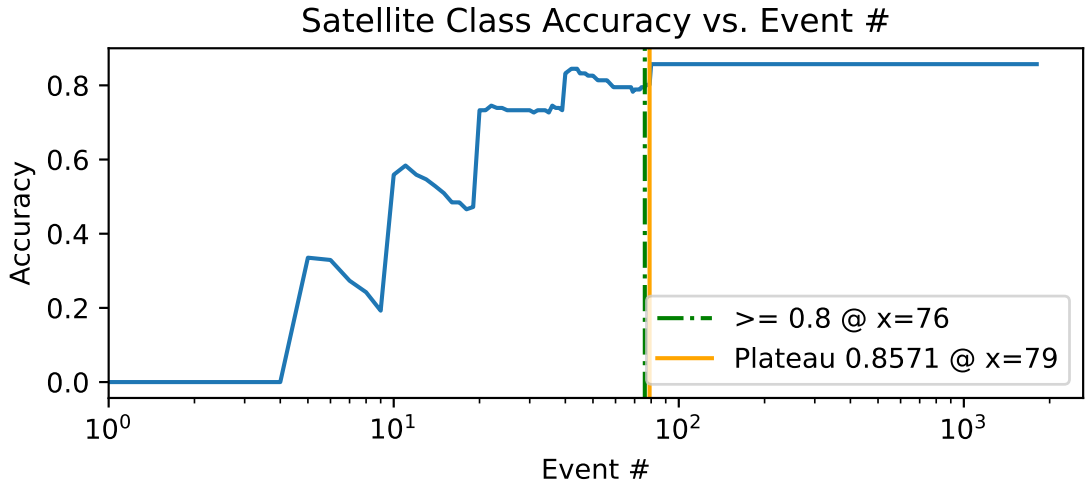


Figure 7.41: The proximity-based grouper with a Dense Network satellite class accuracy versus group event count has a unique sawtooth shape. As the groups surpass a model size the accuracy improves, but when averaging over the rolling windows between model sizes the accuracy drops. The sawtooth prevents sustained passing of the 80% accuracy until 79 events.

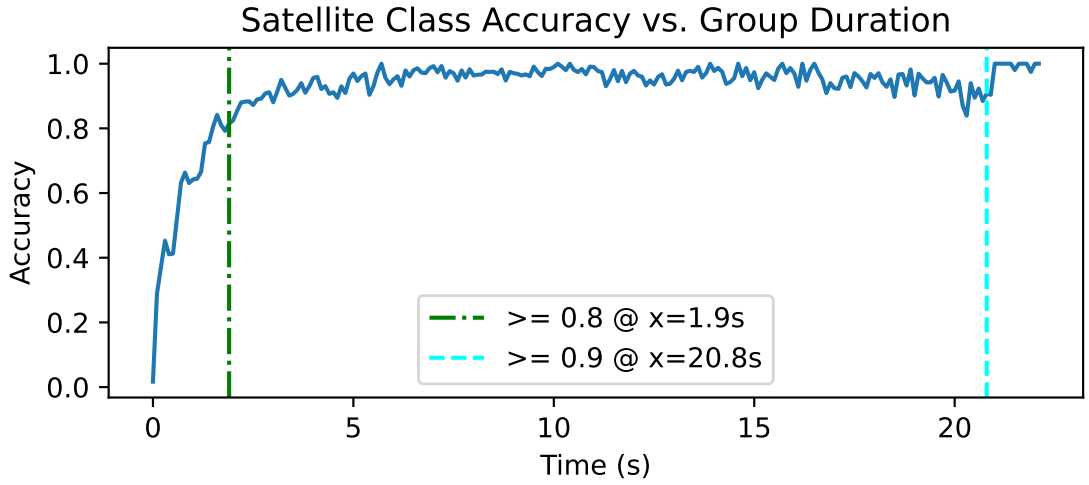


Figure 7.42: The proximity-based grouper with a Dense Network satellite class accuracy versus group time resembles the Random Forest performance despite the sawtooth curve seen per event count. The rise to 80% accuracy is slower at 1.9 seconds. However, like the Random Forest, the higher accuracy variance prevents rising to 90% consistently until the longest duration groups at 20.8 seconds.

grouper combination. However, the Dense Network with a RANSAC-based grouper yields final higher accuracy in the satellite classification. Tables 7.20 and 7.21 summarize the number of events required and the duration required, respectively, to exceed the 80% and 90% accuracies.

Taking a closer look at the event count versus accuracy summary of Table 7.20, the fastest accuracy increase is achieved by the Random Forest model when paired with the proximity-based grouper as previously mentioned. It achieves the 80% accuracy at 11 events and the 90% accuracy at 29 events. The next highest performer is the Dense Network with the RANSAC-base grouper. It achieves the 80% accuracy at 19 events and the 90% accuracy at 99 events. While not all the models reach the 80% accuracy in the satellite classification, most model and grouping combinations are relatively close to their final plateau accuracy by 40 events. Even the Dense Network with the proximity-based clustering, which exhibits a sawtooth curve, is within the 2nd to last model size I implement and does not have as severe degradation of accuracy. Therefore, I conclude, for these data rejection algorithm combinations, that accuracy performance settles with approximately 40 events. For the RANSAC grouper classifiers in their current configuration, I can provide a more definitive conclusion that 40 events reliably produce 80% prediction accuracy for satellite events.

Table 7.20: All Classifier-Grouper Combos, Event # vs. Accuracy Exceeding 80 and 90%

Model	**>=0.8**	**>=0.9**
Bayesian RANSAC	NA	NA
Bayesian Prox	NA	NA
Random Forest RANSAC	39	NA
Random Forest Prox	11	29
Dense Net RANSAC	19	99
Dense Net Prox	76	NA

In the group time versus accuracy summary in Table 7.21, more models achieve the 80% and 90% accuracy measurements. With the RANSAC grouper, both the Random Forest and Dense network models exceed and hold the accuracy values of 80% around 1.5s and 90% around 2s. However, the Bayesian only achieves this accuracy near the longest group time present in the dataset and is unlikely to sustain the value given the trend to its accuracy curve. The timing results vary somewhat between subsequent runs of the RANSAC grouper due to the non-deterministic nature of the RANSAC point selection. Despite this, the Random Forest and Dense network models perform very similarly between iterations when paired with the RANSAC grouper, with minor differences in final accuracy. With the Proximity grouper, the Random Forest model achieves 80% accuracy with one third the time it takes the Dense network with 0.7 seconds and 1.9 seconds respectively. However, both take a very long time to achieve 90% accuracy, about 20.7 seconds, because the classifier does not sustain its classification over longer duration events. Finally, the Bayesian does not achieve 80% accuracy with the proximity-based grouper. Generally, the machine learning models outperform the Bayesian classifier on consistent satellite accuracy. Despite not reaching the same accuracy levels, the initial accuracy peaks under 2 seconds. Therefore, with any grouper and classifier combination I use on this data, I can expect a reasonable satellite classification to take 2 seconds. Between the 2-second and the 40 events from before, I recommend these inform the minimal time window and events retention to provide useful classification for fully online algorithms.

Table 7.21: All Classifier-Grouper Combos, Group Time vs. Accuracy Exceeding 80 and 90%

Model	**>=0.8**	**>=0.9**
Bayesian RANSAC	22.1s	22.1s
Bayesian Prox	NA	NA
Random Forest RANSAC	1.4s	2.1s
Random Forest Prox	0.7s	20.6s
Dense Net RANSAC	1.5s	1.7s
Dense Net Prox	1.9s	20.8s

7.2.6 Online Classifier Results

As a culminating analysis for the classifier, I run the RANSAC and proximity groupers with the Bayesian, Random Forest, and Dense Network models against the validation data set. For each of the runs, I examine the classifier assigned final predicted class of each grouping for each group of events. I compare the performances of the different grouper and classification data rejection combinations at the event-level organizing the class predictions against their true classes in confusion matrices. Unlike the classifier timing analysis in Section 7.2.5 that primarily examines satellite accuracy, I return to the TPR and TNR of each class to evaluate each data rejection combination.

Online RANSAC Grouper Classifier Results

Starting again with the RANSAC-based grouper and applying my three classification methods, I inspect each class' TPR and TNR, and present the full confusion matrix to evaluate their performance. I first look at the satellite TPR and TNR of the three classification methods when I apply this grouping method. Table 7.22 shows the classifier results when I run them online with the RANSAC grouper. For the Bayesian, Random Forest, and Dense Neural network models,

the Dense Neural Network obtains the best TPR results, with the Random Forest classifier close behind. The Bayesian classifier exhibits similarly high TNR to the other models, but shows a significant decrease in TPR versus the other models. However, the Bayesian performance almost doubles that of the online pixel-level Bayesian classifier. Therefore, the group-level classification generally improves performance. It is important to note that the classifier TNR results include the classification the RANSAC star group pre-classifier filter. The generally consistent removal of star groups contributes greatly to the satellite TNR.

Table 7.22: RANSAC Grouper Online Classifier Satellite TPR/TNR Results

Model	**Satellite TPR**	**Satellite TNR**
Bayesian	0.7474	0.9970
Random Forest	0.9688	0.9997
Dense Network	0.9762	0.9995

Table 7.23 shows the Bayesian classifier with RANSAC-based grouper's overall class TPR & TNRs and Table 7.24 displays the corresponding confusion matrix. This data rejection combination achieves 74.74% TPR and 99.7% TNR for satellites. As such, it can reasonably identify satellites and rejects most other classes. It is unable to identify hot pixels, with a TNR near 1 but a TPR near 0. Instead, it classifies most hot pixel groups as noise, with 90.7% of the hot pixel events labeled as noise in the confusion matrix. This result is acceptable for our desired satellite and star prediction performance. It is able to identify most noise, and reject most other real groups from being noise. It predicts 2 orders of magnitude less star and satellite events as noise than the actual noise events. The Bayesian is also able to identify stars well when paired with the Star Group filter, with a 97.36% TPR. However, it does not reject other groups from the star class as well. It classifies 13.3% of the satellite events and 7.2% of

the noise events as star events which lowers the star TNR to 91.88%. The low TNR for stars may be a result of the Bayesian probabilities being built with a class imbalance, where star groups are the vast majority of groups, resulting in overprediction. Overall, the Bayesian does perform adequately in identifying satellites. Despite the high TNR, the other classes represented are predicted to be satellites with some frequency, resulting in about 8.7% of satellite predictions accepting the wrong class, as seen in the confusion matrix.

Table 7.23: Online RANSAC Grouper Bayesian TPR & TNR

Bayesian	**Hot Pixel**	**Noise**	**Star**	**Satellite**
TPR	0.0010	0.9226	0.9736	0.7474
TNR	1.0000	0.9302	0.9188	0.9970

Table 7.24: Online RANSAC Grouper Bayesian Confusion Matrix. Reference the color scale in 7.13.

	Predict Hot Pix	**Pred Noise**	**Pred Star**	**Pred Sat**
Actual Hot Pix	20	17598	1157	626
Actual Noise	3	140721	10982	828
Actual Star	0	9989	377295	228
Actual Sat	0	2640	3933	19445

Next, I evaluate the Random Forest classifier with RANSAC-based grouper's preformance. Table 7.25 lists the overall class TPR & TNRs and Table 7.26 displays the confusion matrix. The Random Forest classifier has a very high satellite TNR, rejecting nearly all other events successfully. This is coupled with a high satellite TPR of 96.88% with the remaining groupings mostly being characterized as noise. The combination results in good predictive performance for satellites with few false positives. For stars, the star group pre-classifier primarily dictates the Random Forest model's performance as the online class timing results in Section 7.2.5 indicate. Relying on this method results in a high star TPR and respectable star TNR, both around 96%. The star TNR is acceptable at a lower value than the satellite TNR because stars are a larger proportion of

the events. While the Random Forest model is able to predict hot pixels, it does so with a low TPR of near 52%. Most other hot pixel groups classify as noise, which again is a satisfactory result for star and satellite predictions. Overall, the Random Forest model paired with the RANSAC grouper performs very well in identifying satellites and stars.

Table 7.25: Online RANSAC Grouper Random Forest TPR & TNR

Rand Forest	**Hot Pixel**	**Noise**	**Star**	**Satellite**
TPR	0.4896	0.9482	0.9671	0.9688
TNR	0.9989	0.9464	0.9626	0.9997

Table 7.26: Online RANSAC Grouper Random Forest Confusion Matrix. Reference the color scale in 7.13.

	Predict Hot Pix	**Pred Noise**	**Pred Star**	**Pred Sat**
Actual Hot Pix	9498	9715	152	36
Actual Noise	598	144629	7208	99
Actual Star	0	12702	374758	52
Actual Sat	0	773	38	25207

Finally, I consider the Dense Neural Network classifier with RANSAC-based grouper's performance. Table 7.27 covers the overall class TPR & TNRs and Table 7.28 lists the corresponding confusion matrix. The Dense network achieves a slightly lower satellite TNR than the Random Forest model, but it still rejects the vast majority of other classes. Other event classes only make up 1.1% of the total satellite predictions. The Dense Network's satellite TPR is higher than the Random Forest model by 0.74%. Star performance is extremely similar to the Random forest model, likely due to the star group filter, with similar results for noise and hot pixel predictions. Overall, the Dense Neural network appears to provide the best model, using the RANSAC-based grouping, at both predicting satellites and rejecting other classes from being predicted as satellites.

Table 7.27: Online RANSAC Grouper Dense Network TPR & TNR

Dense Net	Hot Pixel	Noise	Star	Satellite
TPR	0.5195	0.9425	0.9670	0.9762
TNR	0.9971	0.9488	0.9627	0.9995

Table 7.28: Online RANSAC Grouper Dense Network Confusion Matrix. Reference the color scale in 7.13.

	Predict Hot Pix	Pred Noise	Pred Star	Pred Sat
Actual Hot Pix	10078	9137	148	38
Actual Noise	1456	143769	7204	105
Actual Star	151	12479	374742	140
Actual Sat	32	549	38	25399

Online Proximity Grouper Classifier Results

After exploring the RANSAC-based grouper and classifier combinations, I now evaluate the performance of the proximity-based grouper and classifier combinations. For this analysis, I inspect each class' TPR and TNR and present the full confusion matrix for each classifier. Table 7.29 shows the classifier results when run online with the proximity-based grouper. With the proximity grouper the Random Forest model obtains the best TPR results. It switches with the Dense network, which has the best satellite TPR performance with the RANSAC-based grouper. However, the Random Forest model's satellite TNR is low enough that more than 10% of its satellite predictions are actually other classes. The primary contributor is noise events which make up 86.6% of the additional predictions. The Dense Network model has a higher satellite TNR, making it perform the best for both identifying stars and satellites by better rejecting information not a part of those classes. While the Random Forest and Dense Networks vie for best performance with the proximity-based grouper, the Bayesian model is barely able to outperform random chance in satellite TPR, predicting many noise groups as satellites.

Table 7.29: Proximity Grouper Online Classifier Satellite TPR/TNR Results

Model	**Satellite TPR**	**Satellite TNR**
Bayesian	0.3517	0.9539
Random Forest	0.9817	0.9924
Dense Network	0.9516	0.9987

First, I examine The Bayesian classifier with proximity-based grouper's classification output to gain better insight into its poor performance compared to the other two models. Table 7.30 lists the overall class TPR & TNRs and Table 7.31 depicts the confusion matrix. As aforementioned, the Bayesian model has a very low satellite TPR at 35.17%. While it exhibits a TNR that appears high at 95.39%, inspecting the confusion matrix reveals that actual satellites still make up a minority of the predictions for satellite as they did for the pixel-level implementation of the Bayesian classifier. 54.5% of the predicted satellite events are actually noise. In fact, the both the satellite TPR and TNR are lower than the pixel-level implementation I describe in Section 7.2.1. This grouper and classifier combination also has a lower Star TNR than other combinations for which I evaluate the group-level hypotheses. The confusion matrix reveals that the classifier assigns most groups to a star label. This aligns somewhat with the training methodology for the Bayesian approach. I do not address the class imbalance with stars being the most represented objects as I do with the machine learning methods. Therefore, groups have a high probability of being a star simply by being a group. Overall, the Bayesian classifier paired with the proximity-based grouper does not yield acceptable results to be used to reliably update satellite tracks. This performance is disappointing after great lengths to improve upon the initial pixel-level implementation's performance. Possibly through augmentation of the training data, correction to a rolling window as I suggest in Section 7.2.5, and refinement of the attributes utilized in classification, the Bayesian clas-

sifier can perform on the same level as the machine learning models. However, I leave the improvement of the Bayesian model to future work.

Table 7.30: Online Proximity Grouper Bayesian TPR & TNR

Bayesian	**Hot Pixel**	**Noise**	**Star**	**Satellite**
TPR	0.0073	0.7236	0.9100	0.3517
TNR	0.9940	0.8897	0.8168	0.9539

Table 7.31: Online Proximity Grouper Bayesian Confusion Matrix Threshold = 0.32. Reference the color scale in 7.13.

	Predict Hot Pix	**Pred Noise**	**Pred Star**	**Pred Sat**
Actual Hot Pix	142	15184	3302	773
Actual Noise	2956	110380	20151	19047
Actual Star	436	28508	352619	5949
Actual Sat	2	4056	12809	9151

Next, I look at the Random Forest classifier with the proximity-based grouper's performance. Table 7.32 provides the class TPR & TNRs and Table 7.33 details the confusion matrix. This grouper and classifier combination exhibits the highest TPR for satellites of all data rejection combinations I examine at 98.17%. However, the satellite TNR result balances the satellite TPR performance. While the TNR is high, it still accepts several other classes, resulting in about 14.3% of predicted satellites not being true satellites. Examining the ROC curve in Section 7.2.4, if I increase the classifier threshold it would not result in improved TNR performance because the FPR is already minimized. For stars, the Random Forest model performs best of the models used with the proximity-based grouper, but falls short of the performance of all classifiers paired with the RANSAC-based grouper. Overall, the results indicate that this would be a useful combination to identify stars and satellites, but it is unable to do as well as some of the other combinations due to the lower satellite TNR. It is yet another demonstration of the power of the RANSAC-based approach with the star

group pre-classifier, improving performance on satellite groups to such extent that the proximity-based grouping methods cannot compete.

Table 7.32: Online Proximity Grouper Random Forest TPR & TNR

Rand Forest	**Hot Pixel**	**Noise**	**Star**	**Satellite**
TPR	0.3510	0.8953	0.9229	0.9817
TNR	0.9984	0.9111	0.9230	0.9924

Table 7.33: Online Proximity Grouper Random Forest Confusion Matrix Threshold = 0.76. Reference the color scale in 7.13.

	Predict Hot Pix	**Pred Noise**	**Pred Star**	**Pred Sat**
Actual Hot Pix	6810	8708	3765	118
Actual Noise	923	136560	11359	3692
Actual Star	3	29422	357636	451
Actual Sat	0	359	116	25543

The final grouper and classifier combination I examine is the Dense Neural Network classifier with the proximity-based grouper. Table 7.35 provides the overall class TPR & TNRs and Table 7.35 lists the corresponding confusion matrix. In contrast to the Random Forest model, the Dense Neural Network model has a slightly lower satellite TPR of 95.16% but an increased TNR, with only 2.8% of predicted satellites not being real satellite groups. For stars, the Dense Network does not do quite as well as the Random Forest model. The classifier, unfortunately, classifies the majority of actual hot pixel event groupings as stars. This issue is something to focus on if the data rejection output switches from satellite to star identification for star tracking purposes. While the hot pixel predictions are not of use, conflating them to stars could prove to be an issue in future star tracker development. However, with satellite identification as the primary focus of this work, the Dense Network's balance of satellite TPR and TNR are best suited to identifying satellites with the Proximity grouper.

Table 7.34: Online Proximity Grouper Dense Network TPR & TNR

Dense Net	Hot Pixel	Noise	Star	Satellite
TPR	0.2293	0.9070	0.9233	0.9516
TNR	0.9984	0.9181	0.8836	0.9987

Table 7.35: Online Proximity Grouper Dense Network Confusion Matrix Threshold = 0.96. Reference the color scale in 7.13.

	Predict Hot Pix	Pred Noise	Pred Star	Pred Sat
Actual Hot Pix	4448	5511	9379	63
Actual Noise	729	138348	13046	411
Actual Star	157	29314	357791	250
Actual Sat	0	642	617	24759

Comparison Between Combined Grouper Classifier Performance

I now compare the final classifier results when combined with the online groupers. Table 7.36 summarizes the Satellite TPR and TNR for all combinations of grouper and classifiers. The Random Forest classifier in conjunction with the proximity-based grouper achieves the highest Satellite TPR of 98.17%. This combination, however, exhibits a relatively low TNR, at 99.24%, resulting in a significant number of satellite false positives. While the percentage is high, this is due to the under-representation of satellites in the datasets. I present the raw numerical values in Table 7.37. This data rejection combination results in 4261 false positives in contrast to 25543 true positives. The lower TNR performance compared to the Random Forest model in combination with the RANSAC-based grouper may also result from the lack of star group pre-classifier. I attribute the overwhleming success of the RANSAC-based grouper at least partially to the inclusion of the star group pre-classifier. While easier to implement with the RANSAC due to the lines it fits, the star group pre-classifier precluded the proximity-based grouper. With additional manipulation, it could be added to the proximity-based grouper in the future. Addition-

ally, as the classifier models benefit from being trained on the RANSAC-based groupings, the classifiers may perform better on the proximity-based groupings if they are trained specifically against labeled groupings from the proximity-based grouper.

Table 7.36: Final Classifier Performance Satellite TPR & TNR. Blue indicates better relative performance for the metrics, with red indicating worse performance.

Satellite	**RANSAC w/Star Filter**		**Proximity**	
Model	**TPR**	**TNR**	**TPR**	**TNR**
Bayesian	0.7474	0.9970	0.3517	0.9539
Random Forest	0.9688	0.9997	0.9817	0.9924
Dense Network	0.9762	0.9995	0.9516	0.9988

The Dense Neural Network with the RANSAC-based Grouper achieves the next highest TPR. Not only does it achieve a TPR of 96.62%, it also has a much higher TNR of 99.95% when compared against the Random Forest with the proximity-based grouper. Overall, this combination yields the highest performance of both detecting satellites and rejecting non-satellite groupings. The Dense Neural Network also performs well with the proximity-based grouper. The proximity-based grouper version has a slightly lower TPR and TNR. However, the TNR is the highest of all the proximity-based classifiers and it is sufficiently high such that this data rejection combination could be useful to provide events reasonably sanitized of other classes to any subsequent processing. It is important to highlight the other reasonably successful processing options because they may be important pieces to build resilient final online algorithms. For example, one possible solution to deal with online observer pointing changes, like those of space-based observers, could require switching between or blending a RANSAC-based and a proximity-based method. The fact that the Dense Network works well with both grouping options may make such a blended algorithm easier to implement.

Next, I consider the Random Forest model with a RANSAC-based grouper. This model has very similar performance to the Dense Network with a RANSAC-based grouper combination. The Random Forest only has a slightly higher TNR but lower TPR. Therefore, this combination may also be useful in future applications. In fact, it may be preferable due to the run-time speed of the Random Forest, which is observed to run more quickly than the Dense Network for inference. Unlike the Dense Network, however, its proximity-based counterpart does not maintain the competitive TNR. The proximity-based TNR requires improvement prior to blending or switching between the two classifiers, should that the best path forward for changing observer motion.

The last classifier, the Bayesian, generally underperforms with lower TPRs and TNRs when compared the the machine learning classifiers. The only Bayesian classifier result that is remotely adequate occurs with the RANSAC Grouper; The classifier and grouper combination produces a reasonably high satellite TNR of 99.7%, so it maintains the ability to reject the majority of non-satellite events from the satellite class. Considering the same classifier with the proximity-based method drops to a TNR of 95.7%, the adequate performance likely relies on the success of the star group pre-classifier. Despite the reasonable TNR, it has a lower TPR than the other classifiers at 74.74%. This TPR is an improvement over the pixel-level Bayesian classifier implementation I report in Section 7.2.1. This improvement validates the pivot to evaluating group-level probabilities. However, the group-level proximity-based grouper implementation has both lower TPR and TNR than the pixel-level implementation particularly for longer duration groups.

The machine learning models demonstrate higher inferencing performance

is possible than what the Bayesian achieves. I suspect with tweaks to the Bayesian implementation, using methods I apply during development of the machine learning models, its performance could improve. Through this analysis, I amass the following adjustments to further investigate. Many of these suggestions modify the training data. First, I recommend balancing out the star versus satellite information through augmentation to mimic the successful training of the machine learning models. The machine learning models also benefit from training on data subjected to the same grouping method as I use during the online classification. I can implement this same process to build the probability tables based on group attributes of the same style. Additionally, I recommend augmenting the training data with synthetically generated data sets to better develop the probability tables. It is also possible to convert the tables to continuous probability density functions which may also improve resiliency to attributes not in the training group. Next, I propose refinement of the attributes utilized in classification. This refinement may move away from pixel-level voting and, instead, only evaluate the group-level attributes. The primary difficulty with switching to group-level statistics is finding a way to leverage the profiles of events which are not uniform for one source across all pixels. Possible profile attributes to capture the pixel profile at the group-level could include the strongest pixel, with the most ON events, representing the group profile or averaging the number of ON and OFF events per pixel over the full group. Finally, I suggest limiting evaluation to a rolling window of events at the group-level to potentially prevent the decrease in performance with longer duration groups. Fundamentally, there are more adjustments to try on the Bayesian classifier to, hopefully, improve its performance.

Looking beyond the satellite TPR and TNR values, I present the final clas-

Table 7.37: Final Classification Raw Satellite-related Event Numbers. Blue indicates better relative performance for the metrics, with red indicating worse performance.

Satellite	**RANSAC w/Star Filter**				**Proximity**			
Model	**TP**	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**
Bayesian	19445	557765	1682	6573	9151	533678	25769	16867
Random Forest	25207	559260	187	811	25543	555186	4261	475
Dense Network	25399	559164	283	619	24759	558723	724	1259

sifier Star TPR and TNR in Table 7.38. These metrics indicate that classifiers paired with the RANSAC grouper perform better at identifying stars than the proximity grouper. As aforementioned, I expect that this is primarily due to the presence of the star group pre-classifier included in the RANSAC Grouper, which achieves an average of 96.89% TPR for star groupings. The TPRs in Table 7.38 are slightly lower than in the Section 7.1.3 where I choose the star group percentile. This phenomenon is due to groups that fall out of the filter and are re-evaluated by the classifier.

Table 7.38: Final Classifier Performance Star TPR & TNR. Blue indicates better relative performance for the metrics, with red indicating worse performance.

Star	**RANSAC w/Star Filter**		**Proximity**	
Model	**TPR**	**TNR**	**TPR**	**TNR**
Bayesian	0.9736	0.9188	0.9100	0.8168
Random Forest	0.9671	0.9626	0.9229	0.9230
Dense Network	0.9670	0.9627	0.9233	0.8836

Overall, I discover that only the current proximity-based grouper with Bayesian classifier combination is not useful for identifying satellites. Table 7.39 ranks the utility of the grouper and classifier combinations summarizing my conclusions from conducting this grouper and classifier combination analysis. Primarily this ranking considers the satellite TPR and TNR metrics. The TNR metric needs to be particularly high to indicate the number of non-satellite events classifying as satellite events is acceptably low. Therefore, I heavily

Table 7.39: Grouper-Classifier Combination Utility for Satellite Identification and Ranking

Grouper	**Classifier**	**Useful (Y/N)**	**Rank**
RANSAC	Dense Net	Y	1
RANSAC	Random Forest	Y	2
Proximity	Dense Net	Y	3
Proximity	Random Forest	Y	4
RANSAC	Bayesian	Y	5
Proximity	Bayesian	N	N/A

weigh TNR performance in this overall ranking.

CHAPTER 8

EVENT-BASED TRACKING CONCLUSIONS

By exploring various grouping and classification methods to accomplish the data rejection step of my overall online tracking algorithm concept, I prove that reasonable filtering to mostly satellite events is possible with features within the address event representation of event data for point source objects. Through the development process, I create grouping methods that attain a maximum of 94.5% group agreement with the truth labeled clustered data. Then I create and tune classifiers that achieve a maximum 97.6% TPR and 99.9% TNR for the satellite class during online classifying on the validation data set. While the maximum performance I achieve classifying stars is slightly lower at a 96.7% TPR and 96.5% TNR, the star group pre-classifier proves to be a pivotal component in isolation of the satellite events. Overall, the data rejection I achieve on the validation data sets through development of these grouping and classifying methods promises decently reliable satellite information feeding into subsequent portions of the tracking algorithm. This success suggests that EVS are well suited to augment future SDA operations.

8.1 Grouping Methods

The two grouping methods, RANSAC-based and proximity-based groupers demonstrate online grouping of the event-based data groups after I select their operational parameters. For both techniques the online methods attribute over 90% of the validation data into either real events being in real groups or noise events not being grouped. While both methods reject a similar number of the noise events, the proximity-based grouper relies more on grouping noise into

majority noise event groups. Using this method, I rely more on the classifier to reject these groups and prevent an erroneous satellite from being inferred. Another issue with the proximity-based grouper is the tendency to produce duplicate groups that map back to the same real group in the batch truth data sets. This disparity identifies a lack of a merging process within the proximity-based grouper which the RANSAC-based grouper contains. Moving forward, the proximity-based grouper requires a merging group process to perform on the level of the RANSAC-based grouper.

While the RANSAC-based grouper outperforms the proximity-based grouper in nearly every metric, there are a couple reasons to maintain and iterate on the proximity-based grouper going forward into more advanced algorithm development. The first thing to consider is that the RANSAC-based grouper's development has only been on staring data sets thus far. The EVS points at one azimuth and elevation during a collection and the induced motion in the frame is from the rotation of the Earth. I expect the complexity of motion from a space-based observer to induce non-linear motion of the star field through the frame. RANSAC's strength is in the consistent linear motion through the frame. If the motion changes direction, the RANSAC group lines will no longer help associate the new events. I anticipate some creative algorithms to capture the change in the linear motion and that may require elements of the proximity-based grouping method to work around these challenges.

The next thing to consider is the algorithmic complexity. I walk through determining 8 RANSAC-grouper parameters in Section 7.1.3 as opposed to the 4 parameters of the proximity-based grouping method. While some of these parameters may be resilient to different sensing scenarios, many require tuning

for each use case. For space-based sensing in particular, different slew rates will yield different parameters such as cuboid size. I suggest further exploration of the parameter requirements of different data sets, relying on the generation of synthetic data for space-based applications through the simulation framework I develop in Chapter 3. Parameter settings may be scheduled for particular EVS collection situations such as satellite bus motion. Ideally, however, future models will capture the parameter settings to make the grouping algorithms more resilient to any operational use case.

8.2 Grouping Classification

All of the full data rejection techniques I develop in this dissertation use the original pixel-level Bayesian classifier with the proximity-based grouping technique as a baseline. This combination serves as the proof of concept of data rejection and, in this implementation, reduces the pixels I consider for developing track estimates. Evaluating the hypothesis test at the pixel-level proves to be advantageous to build satellite tracks. The classifier version without the additional noise filter finds the proper satellite track in 92.1% of the validation data sets. However, I can not rely on all the pixels in a group to pass the test threshold because the satellite TPR in only 0.476. Additionally, despite the deceiving TNR performance of 0.959, the pixel-wise test allows a greater number of star and noise events than the satellite events to pass the threshold and classify as satellite events. Ultimately, the pixel-wise technique serves as a demonstration that the data rejection process can be done on point source generated event attributes. I use the lessons learned about pixel-level classification as inspiration to examine point source events at a larger group-level to improve the classifica-

tion metrics and the overall data rejection process.

After the pixel-level demonstration, I pivot my focus to the development of a Dense Neural Network and Random Forest machine learning models to compare against the Bayesian classifier with group-level evaluation through pixel voting. Comparing performance across these classifiers and with both groupers, I find the Dense Neural Network and Random Forest models achieve better results in identifying satellites when paired with the RANSAC-based grouper, with TPRs of 0.976 and 0.969 and TNRs of 0.9995 and 0.9997 respectively. Therefore, both of these grouper and classifier combinations demonstrate that they are able to differentiate satellite groupings to inform updating tracks. The same classifiers are also useful for this purpose when I use them with the proximity-based grouper, but do not achieve the same TPR and TNR. Additionally, I discover that the Bayesian group-level model is able to perform sufficiently well with the RANSAC grouper that it could be useful to inform satellite track updates, but falls short of the TPR performance of the Dense Network and Random Forest models; rejecting many real satellite groupings. I find that pairing the Bayesian with the proximity-based grouper exhibits too low a TPR and TNR to be useful in identifying satellites. There is room to improve upon the Bayesian performance. I suggest modifications of the training data and online application of the Bayesian statistics to leverage what I discover through use of the machine learning methods and, hopefully, improve the Bayesian classifier performance. Of note, the Bayesian classifier may improve with more information to use in its training sets. I will utilize the outputs of my synthetic event generation simulation to build robust classifiers, augmenting my current data sets and providing new ones for different sensing scenarios without needing to observe with a real EVS first.

One major success in the grouper and classifier combination for data rejection is for specifically identifying stars. The RANSAC grouper's star group pre-classifier appears to have a significant positive impact to the star TPR across all classifiers, with the Random Forest and Dense Network models also having TNRs around 0.96. The Bayesian model exhibits a slightly higher TPR and lower TNR, but is still useful in finding stars. With the proximity-based grouper all of the star TPRs and TNRs decrease. The Random Forest model exhibits a star TPR of 0.923 and TNR of 0.923, performant enough for star identification, but this leaves more events that may classify as satellites. The Dense Network has a similar TPR but lower TNR, resulting in it not being as useful for star identification. Finally, the Bayesian classifier has the lowest star TPR and TNR, which I find not useful for identifying stars. Overall, I find all of the RANSAC grouper classifier pairings and the proximity-based group with the Random Forest classifier perform well enough to pick out stars to inform downstream decisions. Since, consistent removal of the star data assists in limiting the events that identify as satellites to primarily true satellite events, I recommend continued use of a star group pre-classifier method to aid filtering to true satellite events for subsequent processing. This consistent identification through a geometric process will also aid future star trackers leveraging the event data.

Overall, the classifier performance is promising for my intended application. With the proper training and tuning, the combination of the grouping and classification methods successfully identifies non-resolved satellite events with reasonable consistency. Therefore, future algorithmic development using EVS data derived from point source objects can rely on data rejection to operate on the sources of interest.

8.3 Future Work

8.3.1 Star Filter on Proximity grouper

The RANSAC grouper's creation of 3-dimensional lines for each group allows for trivial group slope comparison, which is the underlying foundation of the star group pre-classifier. While the proximity-based grouper itself does not innately generate these slopes they can be extracted from the group data. In fact, as part of the pixel-wise Bayesian classifier, I wrote a method in the proximity-based grouper which created a 3-dimensional slope for each hypothesis for the rudimentary track estimation. In the future, I plan to utilize this slope to apply a star group pre-filter on the proximity-based grouper's hypotheses as I do with the RANSAC grouper to attempt to increase performance.

8.3.2 Training Machine Learning Models on Proximity Grouper Groupings

In the same way that I trained classifier models with labeled output from the RANSAC grouper in Section 7.2.3, I plan to train another set of models off of labeled output of the proximity-based grouper. Training with the RANSAC grouper output improves the online classification of the validation data sets, particularly with online RANSAC implementations. Therefore, training with the proximity-based grouper output may yield improved classification performance with the Dense Network and Random Forest models when paired with the proximity-based grouper.

8.3.3 Bayesian Training Data Improvement

One of the primary aspects I learn through the machine learning implementation, is the makeup of the training data affects the final classification performance. I did not treat my data in any particular manner prior to building the probability tables that the Bayesian leverages. Because of the lack of treatment, the Bayesian classifier has some disadvantages compared to the machine learning models.

I can treat the data intelligently, informed by the machine learning model training data augmentations. This could include augmentation to add variations of the satellite information to balance out the classes. These augmentations can be synthetically generated using my simulator or could be modifications of the existing satellite groups. I can also assemble the training groups with the expected grouping method to align the probability tables with modified attributes conditional on the grouping process. Finally, I intend to reassess the attributes at a group-level to determine which features contribute useful information for inferencing.

8.3.4 Refinement with Synthetic Data

One of the difficulties with the algorithm development in this dissertation is the lack of available data, especially the variety and number of satellite detections. With the simulation I develop in Chapter 3, I can expand the diversity of the training and validation data sets. Additionally, I can use the synthetic data to test the robustness of the algorithms to new sensing paradigms, such as space-based sensing.

8.3.5 Kalman Filtering

In the future, I plan to implement the next step in the tracker after data rejection. The output from my version of the data rejection model with the right grouper and classifier combination is primarily a list of grouped satellite events. With the filtered event information, I know the x and y pixel location and the time the source first arrives on that pixel. Using the gnomonic projection discussed in Section 3.1, I can map the satellite back to a (RA, DEC) and a rate which the satellite traverses the angular space. Taking the information with some level of uncertainty informed by the expected accuracy of the grouper and classifier combination and stochasticity of the measurement process, I can apply this position and velocity information to update my expectation of a satellite state. Kalman filtering is broken into two steps, modifying the state estimate with the measurement and updating the state in time before the next measurement [3]. Research shows that asynchronous Kalman filtering can use event-based data [101], so applying the filtered data to update an orbital state should be possible. Alternatively, I may try the AURORAS approach to use the filtered data for orbit fitting[6].

APPENDIX A

PROBABILITY TABLES

A.1 Class Probability

Table A.1: Probability of a class of detection occurring in the training data set.

Hot Pixel	Noise	Star	Satellite
0.005	0.510	0.460	0.026

A.2 Conditional Probability Tables

Table A.2: Profiles that have probabilities greater than 0.01 given a satellite detection has occurred. 1 is a positive threshold change. 0 is a negative threshold change. The highest probabilities for satellites and stars are in blue. The lowest probabilities are in red.

Profile	Hot Pixel	Noise	Star	Satellite
(0,)	0.92174	0	0.225666	0.087371
(1, 0, 0)	0.001269	0.008594	0.049406	0.016056
(1, 1, 1, 0, 0, 0)	0.000132	0	0.021979	0.024112
(1, 1)	0.001349	0.004911	0.015015	0.021616
(1, 0)	0.018461	0.049724	0.116239	0.169749
(1,)	0.024375	0	0.060413	0.179961
(1, 1, 0)	0.001763	0.01504	0.040381	0.117951
(1, 1, 0, 0, 0)	0.000118	0.000307	0.023043	0.010269
(1, 1, 1, 1, 0, 0)	4.02E-05	0.000307	0.008522	0.026949
(1, 1, 1, 1, 1, 0, 0, 0, 0)	2.58E-05	0.000307	0.00871	0.01356
(1, 1, 0, 0)	0.000626	0.001842	0.04724	0.057642
(1, 1, 1, 1, 1, 0, 0, 0)	3.16E-05	0.000307	0.006875	0.018042
(1, 1, 1, 0)	0.000121	0.00399	0.00872	0.030807
(1, 1, 1, 1, 0, 0, 0, 0)	4.88E-05	0.001228	0.012039	0.011744
(1, 1, 1, 0, 0)	0.000187	0.000614	0.024557	0.059401
(1, 1, 1, 1, 0, 0, 0)	3.45E-05	0.001228	0.014097	0.027289
(1, 1, 1, 1, 1, 1, 0, 0, 0, 0)	2.01E-05	0	0.004801	0.011517

Table A.3: Average distance between pixels for satellite conditional probabilities greater than 0.001. The highest probabilities for satellites and stars are in blue. The lowest probabilities are in red.

Average Distance	Hot Pixel	Noise	Star	Satellite
1	0	0	0.456469	0.359809
1.414214	0	0	0.324007	0.286962
2	0	0	0.131537	0.156246
2.236068	0	0	0.046042	0.071542
2.828427	0	0	0.019204	0.040054
3	0	0	0.003144	0.003631
3.162278	0	0	0.008283	0.012879
3.605551	0	0	0.004772	0.014184
4	0	0	0.00159	0.00556
4.123106	0	0	0.001341	0.008056
4.472136	0	0	0.000449	0.001759
5	0	0	0.00109	0.009304
5.09902	0	0	0.000315	0.003177
5.385165	0	0	0.000217	0.001589
5.656854	0	0	0.000299	0.002042
6.082763	0	0	0.000124	0.001532
6.324555	0	0	6.37E-05	0.001078
6.708204	0	0	0.000143	0.001589
8.246211	0	0	2.55E-05	0.001135
Above Max Bin	0	0	0.000131	0.003064

Table A.4: Average time between events for satellite conditional probabilities greater than 0.001. The highest probabilities for all classes are in blue. The lowest probabilities are in red.

Average Time	Hot Pixel	Noise	Star	Satellite
0	0.946116	0	0.28608	0.267332
10000	0.012636	0.095764	0.086974	0.141949
110000	8.61E-06	0.000614	9.56E-05	0.012425
120000	1.72E-05	0.000614	0.000198	0.012425
130000	1.15E-05	0.003376	0.000347	0.014694
140000	1.44E-05	0.004297	0.000395	0.015659
150000	1.72E-05	0.005832	0.000656	0.01668
160000	4.02E-05	0.008901	0.000819	0.015375
170000	2.58E-05	0.00706	0.001067	0.014808
180000	3.16E-05	0.003069	0.001176	0.013673
190000	2.87E-05	0.007366	0.001258	0.013106
200000	2.58E-05	0.005218	0.001577	0.012595
210000	5.46E-05	0.004911	0.001864	0.012141
220000	4.59E-05	0.006139	0.002157	0.011914
230000	3.16E-05	0.001228	0.002609	0.013106
240000	2.30E-05	0.001842	0.00323	0.011347
250000	4.31E-05	0.001842	0.003992	0.01095
260000	4.59E-05	0.001535	0.004661	0.011006
270000	6.03E-05	0.003683	0.005362	0.010155
280000	5.17E-05	0.001535	0.006356	0.010836
290000	4.88E-05	0.001842	0.006786	0.010836
300000	5.74E-05	0.001228	0.007382	0.010326

Table A.5: New pixel rate for satellite conditional probabilities greater than 0.001. The highest probabilities for satellites and stars are in blue. The lowest probabilities are in red.

New Pixel Rate	Hot Pixel	Noise	Star	Satellite
5.00E-06	0	0	0.796173	0.352491
1.00E-05	0	0	0.033079	0.201747
1.50E-05	0	0	0.012909	0.091853
2.00E-05	0	0	0.006971	0.042608
2.50E-05	0	0	0.004556	0.023999
3.00E-05	0	0	0.003138	0.01424
3.50E-05	0	0	0.002173	0.012368
4.00E-05	0	0	0.001937	0.009475
4.50E-05	0	0	0.001472	0.005163
5.00E-05	0	0	0.001125	0.004936
5.50E-05	0	0	0.000924	0.004028
6.00E-05	0	0	0.000806	0.003574
6.50E-05	0	0	0.0008	0.003858
7.00E-05	0	0	0.000749	0.002326
7.50E-05	0	0	0.000577	0.001702
8.00E-05	0	0	0.000459	0.002099
8.50E-05	0	0	0.000456	0.002156
9.00E-05	0	0	0.000392	0.001589
0.0001	0	0	0.00029	0.001872
0.000105	0	0	0.000261	0.001191
0.00011	0	0	0.000293	0.001135
0.000115	0	0	0.000271	0.001305
Above Max Bin	0	0	0.100023	0.18467

Table A.6: Total time of profile on pixels for satellite conditional probabilities greater than 0.006. The highest probabilities for satellites and stars are in blue. The lowest probabilities are in red.

Total Time	Hot Pixel	Noise	Star	Satellite
0	0.946116	0	0.28608	0.267332
10000	0.012636	0.095764	0.086974	0.141949
680000	2.58E-05	0.000614	0.000812	0.006354
720000	4.88E-05	0	0.000659	0.006695
750000	3.16E-05	0.000921	0.000596	0.006354
760000	2.01E-05	0.000614	0.000714	0.007602
770000	3.16E-05	0.000307	0.000736	0.006922
780000	5.46E-05	0.000614	0.000707	0.006524
790000	3.45E-05	0.000614	0.000688	0.006411
800000	3.73E-05	0.000921	0.000695	0.007659
820000	7.46E-05	0	0.000714	0.006354
840000	6.32E-05	0.001228	0.00064	0.007035
850000	5.17E-05	0.001228	0.000672	0.008113
860000	5.46E-05	0.000614	0.000679	0.006922
870000	5.17E-05	0.001535	0.000589	0.007149
900000	4.02E-05	0.000307	0.000599	0.006865
910000	3.45E-05	0.000614	0.000612	0.007092
920000	3.45E-05	0.000921	0.000672	0.006014
930000	3.16E-05	0.000614	0.000624	0.006922
940000	4.31E-05	0.000614	0.000605	0.006298
970000	5.74E-05	0.001228	0.000602	0.006922
990000	4.88E-05	0	0.000698	0.006808
1090000	6.03E-05	0	0.000739	0.006071

APPENDIX B

TRAINING-VALIDATION FILES

Filename	Validation Set
2021_01_03T02_04_50.000event0109.csv	FALSE
2021_01_03T02_26_31.000event0141.csv	TRUE
2021_01_03T02_27_07.000event0142.csv	FALSE
2021_01_03T03_17_51.000event0217.csv	TRUE
2021_01_03T03_42_14.000event0253.csv	FALSE
2021_01_03T03_42_50.000event0254.csv	FALSE
2021_01_03T03_50_38.000event0265.csv	TRUE
2021_01_03T03_51_14.000event0266.csv	FALSE
2021_01_03T03_52_31.000event0268.csv	FALSE
2021_01_03T04_02_30.000event0283.csv	FALSE
2021_01_03T04_03_05.000event0284.csv	FALSE
2021_01_03T04_16_57.000event0305.csv	FALSE
2021_01_03T04_44_34.000event0347.csv	FALSE
2021_01_03T04_45_10.000event0348.csv	FALSE
2021_01_03T09_47_15.000event0803.csv	FALSE
2021_01_03T09_47_51.000event0804.csv	FALSE
2021_01_03T10_28_54.000event0864.csv	TRUE
2021_01_03T11_24_38.000event0947.csv	FALSE
2021_01_03T11_47_39.000event0981.csv	FALSE
2021_01_03T11_48_15.000event0982.csv	FALSE
2021_01_04T01_55_12.000event0094.csv	FALSE
2021_01_04T04_03_24.000event0288.csv	FALSE
2021_01_04T04_36_13.000event0338.csv	TRUE

Filename	Validation Set
2021_01_04T04_50_06.000event0359.csv	FALSE
2021_01_04T04_50_43.000event0360.csv	FALSE
2021_01_04T05_11_51.000event0391.csv	FALSE
2021_01_04T05_12_27.000event0392.csv	FALSE
2021_01_04T10_26_45.000event0859.csv	TRUE
2021_01_04T11_38_03.000event0964.csv	FALSE
2021_01_04T12_27_07.000event1037.csv	FALSE
2021_01_05T03_59_24.000event0279.csv	FALSE
2021_01_05T04_00_00.000event0280.csv	FALSE
2021_01_05T05_30_25.000event0417.csv	FALSE
2021_01_05T05_31_01.000event0418.csv	TRUE
2021_01_05T08_38_36.000event0701.csv	FALSE
2021_01_05T09_36_33.000event0789.csv	FALSE
2021_01_05T09_37_09.000event0790.csv	FALSE
2021_01_05T10_43_04.000event0889.csv	FALSE
2021_01_05T10_43_40.000event0890.csv	FALSE
2021_01_05T10_49_34.000event0899.csv	TRUE
2021_01_06T02_11_50.000event0117.csv	FALSE
2021_01_06T02_12_26.000event0118.csv	FALSE
2021_01_06T02_16_00.000event0123.csv	FALSE
2021_01_06T02_18_34.000event0127.csv	FALSE
2021_01_06T02_31_42.000event0147.csv	TRUE
2021_01_06T02_32_19.000event0148.csv	FALSE
2021_01_06T03_22_58.000event0225.csv	FALSE
2021_01_06T03_23_33.000event0226.csv	TRUE

Filename	Validation Set
2021_01_06T05_38_32.000event0431.csv	FALSE
2021_01_06T05_39_08.000event0432.csv	FALSE
2021_01_06T05_40_25.000event0434.csv	FALSE
2021_01_06T09_06_41.000event0747.csv	FALSE
2021_01_06T09_25_02.000event0775.csv	FALSE
2021_01_06T10_32_37.000event0877.csv	FALSE
2021_01_06T10_33_13.000event0878.csv	FALSE
2021_01_06T10_51_46.000event0906.csv	FALSE
2021_01_07T01_14_49.000event0028.csv	FALSE
2021_01_07T04_45_17.000event0343.csv	FALSE
2021_01_07T04_45_52.000event0344.csv	FALSE
2021_01_07T05_07_26.000event0377.csv	TRUE
2021_01_07T05_14_08.000event0387.csv	FALSE
2021_01_07T05_14_44.000event0388.csv	TRUE
2021_01_07T06_01_28.000event0459.csv	FALSE
2021_01_07T06_02_04.000event0460.csv	FALSE
2021_01_11T01_34_49.000event0052.csv	FALSE
2021_01_11T02_32_32.000event0138.csv	TRUE
2021_01_11T02_34_49.000event0141.csv	FALSE
2021_01_11T02_35_25.000event0142.csv	FALSE
2021_01_11T02_42_21.000event0152.csv	TRUE
2021_01_11T02_43_03.000event0153.csv	FALSE
2021_01_11T04_01_39.000event0269.csv	FALSE
2021_01_11T04_19_13.000event0295.csv	FALSE
2021_01_11T04_19_49.000event0296.csv	FALSE

Filename	Validation Set
2021_01_11T04_55_33.000event0347.csv	FALSE
2021_01_11T04_56_10.000event0348.csv	TRUE
2021_01_11T05_16_59.000event0379.csv	TRUE
2021_01_11T07_46_51.000event0603.csv	FALSE
2021_01_11T07_47_27.000event0604.csv	FALSE
2021_01_11T09_18_55.000event0742.csv	FALSE
2021_01_11T09_32_47.000event0763.csv	FALSE
2021_01_11T09_33_23.000event0764.csv	TRUE
2021_01_11T11_21_14.000event0924.csv	TRUE
2021_01_11T12_14_53.000event1004.csv	TRUE
2021_01_12T01_40_27.000event0061.csv	FALSE
2021_01_12T01_41_02.000event0062.csv	FALSE
2021_01_12T01_54_55.000event0083.csv	TRUE
2021_01_12T01_55_31.000event0084.csv	FALSE
2021_01_12T02_39_31.000event0143.csv	TRUE
2021_01_12T02_40_07.000event0144.csv	FALSE
2021_01_12T02_53_55.000event0163.csv	TRUE
2021_01_12T03_31_12.000event0215.csv	FALSE
2021_01_12T03_31_48.000event0216.csv	FALSE
2021_01_12T03_34_23.000event0220.csv	FALSE
2021_01_12T03_42_20.000event0231.csv	TRUE
2021_01_12T03_42_56.000event0232.csv	FALSE
2021_01_12T03_47_29.000event0239.csv	FALSE
2021_01_12T03_48_06.000event0240.csv	TRUE
2021_01_12T04_20_50.000event0287.csv	FALSE

Filename	Validation Set
2021_01_12T04_22_07.000event0289.csv	FALSE
2021_01_12T04_51_56.000event0333.csv	FALSE
2021_01_12T04_52_31.000event0334.csv	FALSE
2021_01_13T01_23_20.000event0033.csv	FALSE
2021_01_13T01_23_56.000event0034.csv	FALSE
2021_01_13T01_39_19.000event0055.csv	FALSE
2021_01_13T01_57_49.000event0081.csv	FALSE
2021_01_13T01_58_25.000event0082.csv	TRUE
2021_01_13T02_46_47.000event0151.csv	TRUE
2021_01_13T02_47_23.000event0152.csv	FALSE
2021_01_13T02_55_24.000event0163.csv	TRUE
2021_01_13T03_59_41.000event0256.csv	FALSE
2021_01_13T04_00_23.000event0257.csv	TRUE
2021_01_13T04_07_38.000event0267.csv	FALSE
2021_01_13T04_08_14.000event0268.csv	FALSE
2021_01_13T04_46_10.000event0323.csv	FALSE
2021_01_13T04_46_46.000event0324.csv	FALSE
2021_01_13T04_48_44.000event0327.csv	TRUE
2021_01_13T04_49_20.000event0328.csv	TRUE
2021_01_13T05_16_37.000event0367.csv	FALSE
2021_01_13T05_17_13.000event0368.csv	FALSE
2021_01_13T05_17_54.000event0369.csv	FALSE
2021_01_13T05_18_30.000event0370.csv	FALSE
2021_01_13T07_42_28.000event0582.csv	FALSE
2021_01_13T08_59_17.000event0695.csv	FALSE

Filename	Validation Set
2021_01_13T09_23_14.000event0729.csv	TRUE
2021_01_13T09_23_50.000event0730.csv	TRUE
2021_01_13T09_33_05.000event0743.csv	FALSE
2021_01_13T09_33_41.000event0744.csv	FALSE
2021_01_13T10_05_37.000event0791.csv	FALSE
2021_01_13T11_35_38.000event0921.csv	TRUE
2021_01_13T11_52_14.000event0944.csv	TRUE
2021_01_15T02_14_15.000event0105.csv	FALSE
2021_01_15T02_14_50.000event0106.csv	TRUE
2021_01_15T03_21_43.000event0205.csv	TRUE
2021_01_15T03_22_19.000event0206.csv	FALSE
2021_01_15T04_30_21.000event0305.csv	FALSE
2021_01_15T08_06_25.000event0623.csv	FALSE
2021_01_15T08_07_01.000event0624.csv	TRUE
2021_01_15T09_27_32.000event0741.csv	FALSE
2021_01_15T09_28_08.000event0742.csv	FALSE
2021_01_15T10_01_10.000event0790.csv	TRUE
2021_01_15T10_01_51.000event0791.csv	TRUE
2021_01_15T10_12_05.000event0805.csv	TRUE
2021_01_15T10_12_41.000event0806.csv	FALSE
2021_01_15T10_20_23.000event0818.csv	FALSE
2021_01_15T10_21_05.000event0819.csv	TRUE
2021_01_15T10_41_08.000event0847.csv	TRUE
2021_01_15T10_41_44.000event0848.csv	FALSE
2021_01_15T11_32_21.000event0921.csv	FALSE

Filename	Validation Set
2021_01_15T11_32_57.000event0922.csv	FALSE
2021_01_15T12_00_11.000event0961.csv	TRUE
2021_01_15T12_00_47.000event0962.csv	TRUE
2021_01_15T12_32_50.000event1006.csv	FALSE
2021_01_15T12_33_31.000event1007.csv	FALSE
2021_01_15T12_34_06.000event1008.csv	TRUE
2021_01_16T02_23_43.000event0115.csv	TRUE
2021_01_16T02_24_18.000event0116.csv	FALSE
2021_01_16T02_32_13.000event0127.csv	FALSE
2021_01_16T02_32_49.000event0128.csv	FALSE
2021_01_16T02_35_23.000event0132.csv	FALSE
2021_01_16T02_36_00.000event0133.csv	FALSE
2021_01_16T02_36_35.000event0134.csv	FALSE
2021_01_16T03_19_33.000event0197.csv	FALSE
2021_01_16T03_20_10.000event0198.csv	FALSE
2021_01_16T04_03_54.000event0262.csv	TRUE
2021_01_16T04_35_05.000event0307.csv	TRUE
2021_01_16T04_35_42.000event0308.csv	FALSE
2021_01_16T05_03_29.000event0347.csv	TRUE
2021_01_16T05_04_05.000event0348.csv	FALSE
2021_01_16T05_04_45.000event0349.csv	FALSE
2021_01_16T05_05_21.000event0350.csv	FALSE
2021_01_16T06_34_38.000event0479.csv	FALSE
2021_01_16T09_08_40.000event0703.csv	FALSE
2021_01_16T09_09_17.000event0704.csv	TRUE

Filename	Validation Set
2021_01_16T09_39_12.000event0748.csv	FALSE
2021_01_17T02_09_09.000event0095.csv	FALSE
2021_01_17T02_09_46.000event0096.csv	FALSE
2021_01_17T03_04_32.000event0175.csv	FALSE
2021_01_17T03_05_08.000event0176.csv	FALSE
2021_01_17T03_14_20.000event0189.csv	TRUE
2021_01_17T03_14_56.000event0190.csv	TRUE
2021_01_17T03_58_17.000event0255.csv	TRUE
2021_01_17T03_58_53.000event0256.csv	TRUE
2021_01_17T04_15_04.000event0279.csv	FALSE
2021_01_17T05_02_48.000event0349.csv	FALSE
2021_01_17T05_03_23.000event0350.csv	FALSE
2021_01_17T09_19_27.000event0723.csv	FALSE
2021_01_17T09_50_31.000event0768.csv	FALSE
2021_01_17T10_32_18.000event0829.csv	TRUE
2021_01_17T10_32_53.000event0830.csv	FALSE
2021_01_17T10_55_19.000event0863.csv	TRUE
2021_01_17T10_55_56.000event0864.csv	FALSE
2021_01_17T12_40_27.000event1012.csv	FALSE
2021_01_18T01_33_12.000event0041.csv	TRUE
2021_01_18T01_33_48.000event0042.csv	FALSE
2021_01_18T02_35_48.000event0132.csv	FALSE
2021_01_18T02_58_39.000event0165.csv	TRUE
2021_01_18T02_59_15.000event0166.csv	FALSE
2021_01_18T03_04_21.000event0174.csv	FALSE

Filename	Validation Set
2021_01_18T03_21_03.000event0199.csv	FALSE
2021_01_18T03_21_39.000event0200.csv	FALSE
2021_01_18T03_36_57.000event0222.csv	FALSE
2021_01_18T05_35_19.000event0391.csv	TRUE
2021_01_18T05_35_55.000event0392.csv	FALSE
2021_01_18T09_03_21.000event0701.csv	FALSE
2021_01_18T09_07_12.000event0707.csv	FALSE
2021_01_18T09_07_48.000event0708.csv	FALSE
2021_01_18T09_24_41.000event0733.csv	TRUE
2021_01_18T09_25_16.000event0734.csv	FALSE
2021_01_22T03_48_56.000event0227.csv	FALSE
2021_01_22T03_49_33.000event0228.csv	FALSE
2021_01_22T03_50_14.000event0229.csv	FALSE
2021_01_22T04_57_31.000event0329.csv	FALSE
2021_01_22T04_58_07.000event0330.csv	FALSE
2021_01_22T08_44_03.000event0672.csv	TRUE
2021_01_22T08_44_44.000event0673.csv	FALSE
2021_01_31T02_01_42.000event0063.csv	TRUE
2021_01_31T02_48_47.000event0132.csv	FALSE
2021_01_31T02_59_53.000event0148.csv	FALSE
2021_01_31T03_30_33.000event0193.csv	TRUE
2021_01_31T03_31_09.000event0194.csv	TRUE
2021_01_31T03_31_50.000event0195.csv	FALSE
2021_01_31T04_24_03.000event0271.csv	FALSE
2021_01_31T04_24_38.000event0272.csv	TRUE

Filename	Validation Set
2021_01_31T05_20_40.000event0354.csv	FALSE
2021_01_31T09_18_45.000event0703.csv	FALSE
2021_02_05T01_25_25.000event0001.csv	FALSE
2021_02_05T02_38_04.000event0107.csv	TRUE
2021_02_05T02_43_49.000event0116.csv	FALSE
2021_02_05T02_53_04.000event0129.csv	FALSE
2021_02_05T04_59_46.000event0313.csv	FALSE
2021_02_05T05_00_22.000event0314.csv	TRUE
2021_02_05T08_58_24.000event0662.csv	TRUE
2021_02_05T10_03_23.000event0757.csv	FALSE
2021_02_05T10_03_59.000event0758.csv	TRUE
2021_02_05T10_15_48.000event0775.csv	FALSE
2021_02_05T10_54_13.000event0831.csv	TRUE
2021_02_05T10_56_05.000event0834.csv	TRUE
2021_02_05T11_14_04.000event0861.csv	FALSE
2021_02_05T13_02_44.000event1022.csv	FALSE
2021_02_06T01_27_19.000event0004.csv	FALSE
2021_02_06T02_19_25.000event0080.csv	TRUE
2021_02_06T04_49_26.000event0305.csv	FALSE
2021_02_06T04_50_02.000event0306.csv	FALSE
2021_02_06T04_55_09.000event0314.csv	FALSE
2021_02_06T07_39_45.000event0562.csv	TRUE
2021_02_06T12_20_31.000event0981.csv	FALSE
2021_02_07T02_49_25.000event0123.csv	TRUE
2021_02_07T02_50_01.000event0124.csv	FALSE

Filename	Validation Set
2021_02_07T03_34_41.000event0189.csv	FALSE
2021_02_07T03_35_18.000event0190.csv	FALSE
2021_02_07T04_57_22.000event0307.csv	TRUE
2021_02_07T04_57_58.000event0308.csv	FALSE

BIBLIOGRAPHY

[1] Saeed Afshar, Andrew Peter Nicholson, Andre Van Schaik, and Gregory Cohen. Event-based object detection and tracking for space situational awareness. *IEEE Sensors Journal*, 20(24):15117–15132, 2020.

[2] Samya Bagchi and Tat-Jun Chin. Event-based star tracking via multiresolution progressive hough transforms. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2143–2152, 2020.

[3] Yaakov Bar-Shalom, X Rong Li, and Thiagalingam Kirubarajan. *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2004.

[4] Emmanuel Bertin. Sextractor user manual. `https://www.gnu.org/licenses/gpl.html`, 2024. Accessed 30-07-2024.

[5] Samuel S Blackman. Multiple hypothesis tracking for multiple target tracking. *IEEE A&E Systems Magazine*, 19(1), January 2004.

[6] Jeffery Bloch, Lynda Liptak, David Briscoe, Suzzanne Falvey, and Tasha Adams. AURORAS: The Next Evolution of Orbit Determination Using Passive Optical Observations. In *Proceedings of the Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference*, September 2022.

[7] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[8] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[9] T. Brewer. A comparative evaluation of the fast optical pulse response of event-based cameras. Master's thesis, Air Force Institute of Technology, 2021.

[10] Liesl Burger, Igor A Litvin, and Andrew Forbes. Simulating atmospheric turbulence using a phase-only spatial light modulator. *South African Journal of Science*, 104:129–134, March 2008.

[11] Sergei S Chesnokov, Valerii P Kandidov, Victor I Shmalhausen, and Vladimir V Shuvalov. Numerical/optical simulation of laser beam propagation through atmospheric turbulence. Technical Report N68171-94-C-9147, United States Army, London, Dec 1995.

[12] Brian Cheung, Mark Rutten, Samuel Davey, and Greg Cohen. Probabilistic multi hypothesis tracker for an event based sensor. *IEEE Sensors Journal*, 20:15117–15132, December 2020.

[13] Tat-Jun Chin, Samya Bagchi, Anders Eriksson, and Andre Van Schaik. Star tracking using an event camera. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[14] Gregory Cohen. The astrosite: A mobile neuromorphic space domain awareness observatory. `https://greg-cohen.com/project/astrosite/`, 2019. Accessed: 2024-07-24.

[15] Gregory Cohen, Saeed Afshar, Brittany Morreale, Travis Bessell, Andrew Wabnitz, Mark Rutten, and André van Schaik. Event-based sensing for space situational awareness. In *Proceedings of the Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference*. Maui Economic Development Board, September 2017.

[16] Gregory Cohen, Saeed Afshar, and André Van Schaik. Approaches for astrometry using event-based sensors. In *Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference*, page 25, 2018.

[17] GAIA Collaboration, T. Prusti, J. H. J. de Bruijne, and et al. The gaia mission. *Astronomy and Astrophysics*, 595(A1), September 2016.

[18] Sony Semiconductor Solutions Corporation. Sony to release two types of stacked event-based vision sensors with the industry's smallest 4.86 μm pixel size for detecting subject changes only delivering high-speed, high-precision data acquisition to improve industrial equipment productivity. `https://www.sony-semicon.com/en/news/2021/2021090901.html?utm_source=PROPHESEE+Website&utm_medium=Web&utm_campaign=PROPHESEE+IMX636-637+launch&utm_id=IMX636-637+launch`, Sep 2021. Accessed 31-07-2024.

[19] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[20] Phan Dao and Dave Monet. Geo optical data association with concurrent metric and photometric information. In *Proceedings of the AMOS Technologies Conference, Maui, HI, USA*, pages 19–22, 2017.

[21] Tobi Delbruck, Yuhuang Hu, and Zhe He. V2e: From video frames to realistic dvs event camera streams. *arXiv e-prints*, pages arXiv–2006, 2020.

[22] SciPy developers. Two-sample kolmogorov-smirnov test. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ks_2samp.html, 2023. Accessed 31-07-2024.

[23] DoD. United states in space: National security strategy and national space policy. Technical report, Department of Defense, 2011.

[24] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1), January 1972.

[25] ESA. The hipparcos and tycho catalogues, 1997.

[26] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD-96*, number 34, pages 226–231, 1996.

[27] D. W. Evans, M. Riello, F. De Angeli, J. M. Carrasco, P. Montegriffo, C. Fabricius, C. Jordi, L. Palaversa, C. Diener, G. Busso, C. Cacciari, F. van Leeuwen, P. W. Burgess, M. Davidson, D. L. Harrison, S. T. Hodgkin, E. Pancino, P. J. Richards, G. Altavilla, L. Balaguer-Núñez, M. A. Barstow, M. Bellazzini, A. G. A. Brown, M. Castellani, G. Cocozza, F. De Luise, A. Delgado, C. Ducourant, S. Galleti, G. Gilmore, G. Giuffrida, B. Holl, A. Kewley, S. E. Koposov, S. Marinoni, P. M. Marrese, P. J. Osborne, A. Piersimoni, J. Portell, L. Pulone, S. Ragaini, N. Sanna, D. Terrett, N. A. Walton, T. Wevers, and Ł. Wyrzykowski. Gaia data release 2. *Astronomy and Astrophysics*, 616(A4), August 2018.

[28] Howard Evans, Taylor Renner, and Ronald Tuttle. *The Phenomenology of Intelligence-focused Remote Sensing*. Riverside Research, 2014.

[29] Evelyn Fix. *Discriminatory analysis: nonparametric discrimination, consistency properties*, volume 1. USAF school of Aviation Medicine, 1951.

[30] Framos. Prophesee event-based packaged sensor gen3m vga cd. https://www.framos.com/wp-content/uploads/media/pdf/e5/0e/50/SENSOR_PROPHESEE-Packaged-Sensor-Product-Brief_PUBLICAhRoliTmPTl72.pdf, 2023. Accessed 31-07-2024.

[31] David L Fried. Statistics of a geometric representation of wavefront distortion. *Journal of the Optical Society of America*, 55(11):1427–1435, November 1965.

[32] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[33] Kiyoshi Fukushima, Yoshio Yamaguchi, Mitsuru Yasuda, and Shigemi Nagata. An electronic model of the retina. *Proceedings of the IEEE*, 58(12), December 1970.

[34] Kunihiko Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position-neocognitron. *IEICE Technical Report, A*, 62(10):658–665, 1979.

[35] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J Davison, Jörg Conradt, Kostas Daniilidis, et al. Event-based vision: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[36] Walter Gautschi. Efficient computation of the complex error function. *SIAM Journal on Numerical Analysis*, 7(1):187–198, 1970.

[37] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63:3–42, 2006.

[38] Joseph W Goodman. *Introduction to Fourier optics*. Roberts and Company publishers, 2005.

[39] Rui Graca and Tobi Delbruck. Unraveling the paradox of intensity-dependent dvs pixel noise. *arXiv preprint arXiv:2109.08640*, 2021.

[40] Rui Graca, Brian McReynolds, and Tobi Delbruck. Optimal biasing and physical limits of dvs event noise. *arXiv preprint arXiv:2304.04019*, 2023.

[41] Robin M Green. *Spherical Astronomy*. Cambridge University Press, November 1985.

[42] Shasha Guo and Tobi Delbruck. Low cost and latency event camera background activity denoising. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):785–795, 2022.

[43] Russell C Hardie, Jonathan D Power, Daniel A LeMaster, Douglas R Droege, Szymon Gladysz, and Santasri Bose-Pillai. Simulation of anisoplanatic imaging through optical turbulence using numerical wave propagation with new validation analysis. *Optical Engineering*, 56(7), 2017.

[44] Arne Henden and Ronald Kaitchuck. *Astronomical Photometry : a text and handbook for the advanced amateur and professional astronomer*. Willmann-Bell Inc., 1990.

[45] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

[46] Benne W Holwerda. Source extractor for dummies v5, 2005.

[47] Felix R Hoots, Paul W Schumacher Jr, and Robert A Glover. History of analytical orbit modeling in the u.s. space surveillance system. *Journal of Guidance, Control, and Dynamics*, 27(2), March 2004.

[48] inIvation. inivation specifications - current models. `https://inivation.com/wp-content/uploads/2023/11/2023-11-iniVation-devices-Specifications.pdf`, 2023. Accessed 31-07-2024.

[49] Alexey Grigorevich Ivakhnenko. Polynomial theory of complex systems. *IEEE transactions on Systems, Man, and Cybernetics*, SMC-1(4):364–378, 1971.

[50] M Jah and Ronald A Madler. Satellite characterization: angles and light curve data fusion for spacecraft state and parameter estimation. In *Proceedings of the advanced Maui optical and space surveillance technologies conference*, volume 49, 2007.

[51] C Jordi, M Gebran, JM Carrasco, J De Bruijne, H Voss, C Fabricius, J Knude, A Vallenari, R Kohley, and A Mora. Gaia broad band photometry*. *Astronomy and Astrophysics*, 523:A48, 2010.

[52] TS Kelso. Celestrak. `https://celestrak.com/`, 2021. Accessed 30-07-2024.

[53] Chanho Kim, Fuxin Li, Arridhana Ciptadi, and James M Rehg. Multiple hypothesis tracking revisited. In *Proceedings of the IEEE international conference on computer vision*, pages 4696–4704, 2015.

[54] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *Advances in neural information processing systems*, 30, 2017.

[55] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128x128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2):556–576, February 2008.

[56] Shih-Chii Liu, Tobi Delbruck, Giacomo Indiveri, Adrian Whatley, and Rodney Douglas. *Event-Based Neuromorphic Systems*. Wiley, 2014.

[57] R. Bolles M. Fischler. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the Association for Computing Machinery*, 24(6), Jun 1981. Accessed 30-07-2024.

[58] Misha Mahowald. *VLSI analogs of neuronal visual processing: a synthesis of form and function*. PhD thesis, California Institute of Technology Pasadena, 1992.

[59] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annual of Mathematical Statistics*, 18(1):50–60, March 1947.

[60] Jess Marcum. A statistical theory of target detection by pulsed radar. *IRE Transactions on Information Theory*, 6(2):59–267, 1960.

[61] Matthew G McHarg, Richard L Balthazor, Brian J McReynolds, David H Howe, Colin J Maloney, Daniel O'Keefe, Rayomand Bam, Gabriel Wilson, Paras Karki, Alexandre Marcireau, et al. Falcon neuro: an event-based sensor on the international space station. *SPIE Optical Engineering*, 61(8), August 2022.

[62] Matthew G McHarg, Ryan K Haaland, Dana Moudry, and Hans C Stenbaek-Nielsen. Altitude-time development of sprites. *Joournal oof Gephysical Research*, 107(A11), 2002.

[63] Peter N McMahon-Crabtree and David G Monet. Commercial-off-the-shelf event-based cameras for space surveillance applications. *Applied Optics*, 60(25):G144–G153, September 2021.

[64] Brian McReynolds, Rui Graca, and Tobi Delbruck. Exploiting alternating dvs shot noise event pair statistics to reduce background activity. *arXiv preprint arXiv:2304.03494*, 2023.

[65] Brian McReynolds, Rui Graca, Rachel Oliver, Masashi Nishiguchi, and

Tobi Delbruck. Demystifying Event-based Sensor Biasing to Optimize Signal to Noise for Space Domain Awareness. In *Proceedings of the Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference*, page 142, September 2023.

[66] Brian J McReynolds, Rui Graca, Lucas Kulesza, and Peter McMahon-Crabtree. Re-interpreting the step-response probability curve to extract fundamental physical parameters of event-based vision sensors. In *Unconventional Optical Imaging IV*, volume 12996, pages 127–140. SPIE, 2024.

[67] Yonhon Ng, Yasir Latif, Tat-Jun Chin, and Robert Mahony. Asynchronous kalman filter for event-based star tracking. In *European Conference on Computer Vision*, pages 66–79. Springer, 2022.

[68] Robert J Noll. Zernike polynomials and atmospheric turbulence. *Journal of the Optical Society of America*, 66(3):207–211, March 1976.

[69] Yuji Nozaki and Tobi Delbruck. Temperature and parasitic photocurrent effects in dynamic vision sensors. *IEEE Transactions on Electron Devices*, 64(8):3239–3245, August 2017.

[70] U.S. Department of Commerce. Kolmogorov-smirnov goodness-of-fit test. `https://www.itl.nist.gov/div898/handbook/eda/section3/eda35g.htm`, 2023. Accessed 31-07-2024.

[71] Department of Defense. Space Based Space Surveillance — spoc.spaceforce.mil. `https://www.spoc.spaceforce.mil/About-Us/Fact-Sheets/Display/Article/2381700/space-based-space-surveillance`. Accessed 30-07-2024.

[72] ESA Space Debris Office. Esa's annual space environment report. `https://www.esa.int/Space_Safety/Space_Debris/ESA_s_Space_Environment_Report_2023`, September 2023. Accessed 30-07-2024.

[73] US Government Accountability Office. Gao-15-342sp defense acquisitions assessments of selected weapon programs. `https://www.gao.gov/assets/gao-15-342sp.pdf#page=133`, 2015. Accessed 31-07-2024.

[74] Markus Ojala and Gemma C Garriga. Permutation tests for studying classifier performance. *Journal of machine learning research*, 11(6), 2010.

[75] Lisa Poyneer, Marcos van Dam, and Jean-Pierre Véran. Experimental verification of the frozen flow atmospheric turbulence assumption with use of astronomical adaptive optics telemetry. *Optical Society of America*, 26(4):833–846, April 2009.

[76] Prophesee. Event-based vision sensor (evs). `https://www.sony-semicon.com/en/products/is/industry/evs.html`, 2023. Accessed 31-07-2024.

[77] Prophesee. Genx320 - dice. `https://www.prophesee.ai/wp-content/uploads/2024/02/GENX320-Product-Brief-2024-Feb-DICE-OK-1.pdf`, 2023. Accessed 31-07-2024.

[78] Prophesee. Introducing the world's smallest and most power-efficient event-based vision sensor ever released. `https://www.prophesee.ai/event-based-sensor-genx320/`, 2023. Accessed 31-07-2024.

[79] Prophesee. Sensor's quantum efficiency and spectral response. `https://support.prophesee.ai/portal/en/kb/articles/sensor-s-quantum-efficiency-spectral-response`, Jun 2024. Accessed 31-07-2024.

[80] Isaac B Putnam and Stephen C Cain. Modeling a temporally evolving atmosphere with zernike polynomials. In *Proceedings of the Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference*. Maui Economic Development Board, September 2012.

[81] Nicholas Owen Ralph, Alexandre Marcireau, Saeed Afshar, Nicholas Tothill, André Van Schaik, and Gregory Cohen. Astrometric calibration and source characterisation of the latest generation neuromorphic event-based cameras for space imaging. *Astrodynamics*, 7(4):415–443, 2023.

[82] John W Raymond. Space capstone publication, spacepower (scp) headquarters united states space force june 2020 dedicated to past, present, and future spacepower pioneers. *Space Force Publications*, 2020.

[83] Donald Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6), December 1979.

[84] Nicolas A Roddier. Atmospheric wavefront simulation using zernike polynomials. *Optical Engineering*, 29(10):1174–1180, October 1990.

[85] Craig J Rodger. Red sprites, upward lightning, and vlf perturbations. *Reviews of Geophysics*, 37(3):317–336, August 1999.

[86] Seth Roffe, Himanshu Akolkar, Alan D George, Bernabé Linares-Barranco, and Ryad B Benosman. Neutron-induced, single-event effects on neuromorphic event-based vision sensor: A first step towards space applications. *IEEE Access*, May 2021.

[87] Sam Roweis, Dustin Lang, Keir Mierle, David Hogg, and Michael Blanton. Making the sky searchable: Fast geometric hashing for automated astrometry. `https://cosmo.nyu.edu/hogg/research/2006/09/28/astrometry_google.pdf`, 2024. Accessed 30-07-2024.

[88] SAIC. Space-track. `https://www.space-track.org/auth/login`, 2024. Accessed 30-07-2024.

[89] Jason D Schmidt. *Numerical Simulation of Optical Wave Propagation with examples in MATLAB*. SPIE, 2010.

[90] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited and revisited: Why and how you should (still) use dbscan. *ACM Transactions on Database Systems*, 42:1–21, 2017.

[91] scikit-image developers. Straight line hough transform. https://scikit-image.org/docs/dev/auto_examples/edges/plot_line_hough_transform.html, 2021.

[92] scikit-learn authors. Scikit learn gaussian naive bayes. `https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes`. Accessed: 2024-07-08.

[93] scikit-learn developers. Scikit learn 1.2.2 settings. `https://scikit-learn.org/1.2/modules/generated/sklearn.ensemble.RandomForestClassifier.html`, 2021. Accessed 30-07-2024.

[94] scikit-learn developers. Comparing random forests and histogram gradient boosting models. `https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_hist_grad_boosting_comparison.html`, 2023. Accessed 30-07-2024.

[95] Sony. Event-based Vision Sensor (EVS) — Products & Solutions — Sony

Semiconductor Solutions Group — sony-semicon.com. https://www.sony-semicon.com/en/products/is/industry/evs.html, 2022. Accessed 31-07-2024.

[96] Remi Soummer, Laurent Pueyo, Anand Sivaramakrishnan, and Robert J Vanderbei. Fast computation of lyot-style coronagraph propagation. *Opt. Express*, 15, Nov 2007.

[97] Luc Tinch, Nitesh Menon, Keigo Hirakawa, and Scott Mc-Closkey. Event-based Detection, Tracking, and Recognition of Unresolved Moving Objects. In *Proceedings of the Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference*, September 2022.

[98] David A Vallado. *Fundamentals of astrodynamics and applications*, volume 12. Springer Science & Business Media, 2001.

[99] David A Vallado, Paul Crawford, Richard Hujsak, and TS Kelso. Revisiting spacetrack report #3: Rev 2. *AIAA*, 2006.

[100] Antonella Vallenari, Anthony GA Brown, Timo Prusti, Jos HJ De Bruijne, F Arenou, Carine Babusiaux, Michael Biermann, Orlagh L Creevey, Christine Ducourant, Dafydd Wyn Evans, et al. Gaia data release 3. summary of the content and survey properties. *Astronomy and Astrophysics*, 674(A1), June 2023.

[101] Ziwei Wang, Yonhon Ng, Cedric Scheerlinck, and Robert Mahony. An asynchronous kalman filter for hybrid event cameras. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 448–457, 2021.

[102] Jonathan Weisberg. Multiple conditions. https://jonathanweisberg.org/vip/multiple-conditions.html, 2021.

[103] Donald Carson Wells and Eric W Greisen. Fits : A flexible image transport system. *Astronomy & Astrophysics*, pages 363–370, 1981.

[104] David Frank Winkler. *Searching the skies: the legacy of the United States Cold War defense radar program*. www. Militarybookshop. CompanyUK, 1997.

[105] Mofreh R Zaghloul and Ahmed N Ali. Algorithm 916: computing the faddeyeva and voigt functions. *ACM Transactions on Mathematical Software (TOMS)*, 38(2):1–22, 2012.

[106] Michał Zołnowski, Rafal Reszelewski, Diederik Paul Moeys, Tobi Delbruck, and Krzysztof Kamiński. Observational evaluation of event cameras performance in optical space surveillance. In *NEO and Debris Detection Conference, Darmstadt, Germany*. ESA Space Safety Programme Office, 2019.

This work has made use of data from the European Space Agency (ESA) mission *Gaia* (`https://www.cosmos.esa.int/gaia`), processed by the *Gaia* Data Processing and Analysis Consortium (DPAC, `https://www.cosmos.esa.int/web/gaia/dpac/consortium`). Funding for the DPAC has been provided by national institutions, in particular the institutions participating in the *Gaia* Multilateral Agreement.