CCAT-prime Wall Climbing Robot

Project Report

Cornell University
Master of Engineering
Department of Mechanical and Aerospace Engineering

Xiaotian Liu
12/04/2019
xl425@cornell.edu
415-419-0489

Abstract

This is a final report from the MAE 6900 independent research project of Xiaotian Liu. The main purpose of this project is to design a wall-climbing robot capable of performing metrology on the CCAT-prime telescope in Cerro Chajnantor, Chile. The work covered in this report is completely or partially developed by the author through Jan 2019 to Dec 2019. Main topics in Chapter 1 of this paper involve simulation algorithm, Matlab Simulink model introduction, toolbox required for simulation, defects of simulation compared with real operation, current simulation status. For Chapter 2, implemented matlab code, TCP communication, GUI introduction are involved. All hardware layout, pin configurations, wire strategies are described in Chapter 3 with figures. It also summarizes the future efforts that may take place to ensure successful project completion. We worked alongside other Cornell University students on the Controls Team, as well as students on the Mechanical and Testing group.Overseeing the project was our MAE faculty advisor, Professor Dmitry Savaransky as well as Stephen Parshley, the Project Engineer and Terry Herter, the Project Director and a faculty member of the Cornell University Department of Astronomy.

Table of Contents

List of Terms

- **TCP:** (Transmission Control Protocol) is a standard that defines how to establish and maintain a network conversation through which application programs can exchange data. TCP works with the Internet Protocol (IP), which defines how computers send packets of data to each other.
- **Differential Drive:** consists of 2 drive wheels mounted on a common axis, and each wheel can independently be driven either forward or backward.
- **Simulink:** a MATLAB-based graphical programming environment for modeling, simulating and analyzing multi domain dynamical systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries.
- **Raspberry Pi:** a low cost, credit-card sized computer that can plug into a computer monitor or TV, and uses a standard keyboard and mouse. It programs in languages like Scratch and Python.
- **PWM:** Pulse width Modulation signal, to control speed of fans
- **Serial Communication:** the process of sending data one bit at a time, sequentially, over a communication channel or computer bus.
- **ESC:** Electronic Speed Controller for fan
- **ECS:** Eddy Current Sensor
- **ADC:** Analog to Digital Converter

Acknowledgement

Acknowledge to the support from Prof. Savaransky who served as our project advisor for providing us with many resources and technical guidance, as well as Stephen and Terry, the Project Engineer and the Project Director, for giving us this opportunity to work
on this great project. We would also like to thank the other Cornell University students alongside whom we worked.

Introduction

Over the summer of 2018, two M.Eng. students from Cornell University developed a prototype autonomous mobile robot to assist in the process of deformation analysis on the primary and secondary mirrors of the CCAT-p observatory in Cerro Chajnantor, Chile. This robot proved to be an adequate solution to the problem of positioning a retro-reflective puck at specific measurement nodes across the mirror panels. Used in tandem with a laser measurement system from Etalon AG, a complete deformation analysis of the mirrors will result.

Upon approval of the robot's capabilities, a project group was formed to design, assemble and test a robot capable of meeting a stringent set of requirements. These requirements are derived from the harsh environmental conditions of the observatory, the necessity of precision measurement and the damage risks associated with an autonomous wall-climbing robot. This group is split into two subgroups of Cornell University students, Mechanical and Controls, and is overseen by Professor Dmitry Savransky. At the beginning of the 2018-2019 school year, the project was in its initial stages. The major deadline for this project is August 2019, when a second prototype will need to be completed in order to conduct tests in Germany in a simulated operational environment.

Entering the second year of the project, the fundamental structure of the robot is developed and general control algorithm is produced. The second prototype was built from CAD drawing to real components and sent to Germany for a measurement test. Both mechanical and control groups work together to make a functionable prototype which can perform all angle climbing with two 500 W suction fans. Testing group performed a pressure chamber test to simulate half pressure atmosphere conditions. A second generation of prototype that solves current problems is expected to be built at the beginning of 2020 and full close loop control should be implemented.

Chapter 1 Simulation

1.1 Matlab Simulink Introduction

Simulink is a block diagram environment for multi domain simulation and Model-Based Design. It supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems. Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB, enabling users to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis. For this project, Simulink is used to demonstrate possible moving trajectory of designed robot under particular circumstances. Some basic blocks and general procedure of building a model is introduced for more information, check MATLAB Simulink page https://www.mathworks.com/help/simulink/getting-started-with-simulink.html.

Component-Based Modeling Guidelines

Consider componentization for large models and multiuser development teams.

STEP 1: **Choose Among Types of Model Components**
STEP 2: **Compare Capabilities of Model Component Types**
STEP 3: **Define Interfaces of Model Components**

Basic Modeling Workflow

Model a simple mechanical system, then scale the model for a collaborative component-based modeling project.

STEP 1: **Model a System Algorithm**
STEP 2: **Create Model Components**
STEP 3: **Manage Signal Lines**
STEP 4: **Manage Model Data**
STEP 5: **Reuse Model Components from Files**
STEP 6: **Create Interchangeable Variations of Model Components**

STEP 7: **Set Up a File Management System**

Configuration Management

Projects can help you work with configuration management tools for team collaboration.

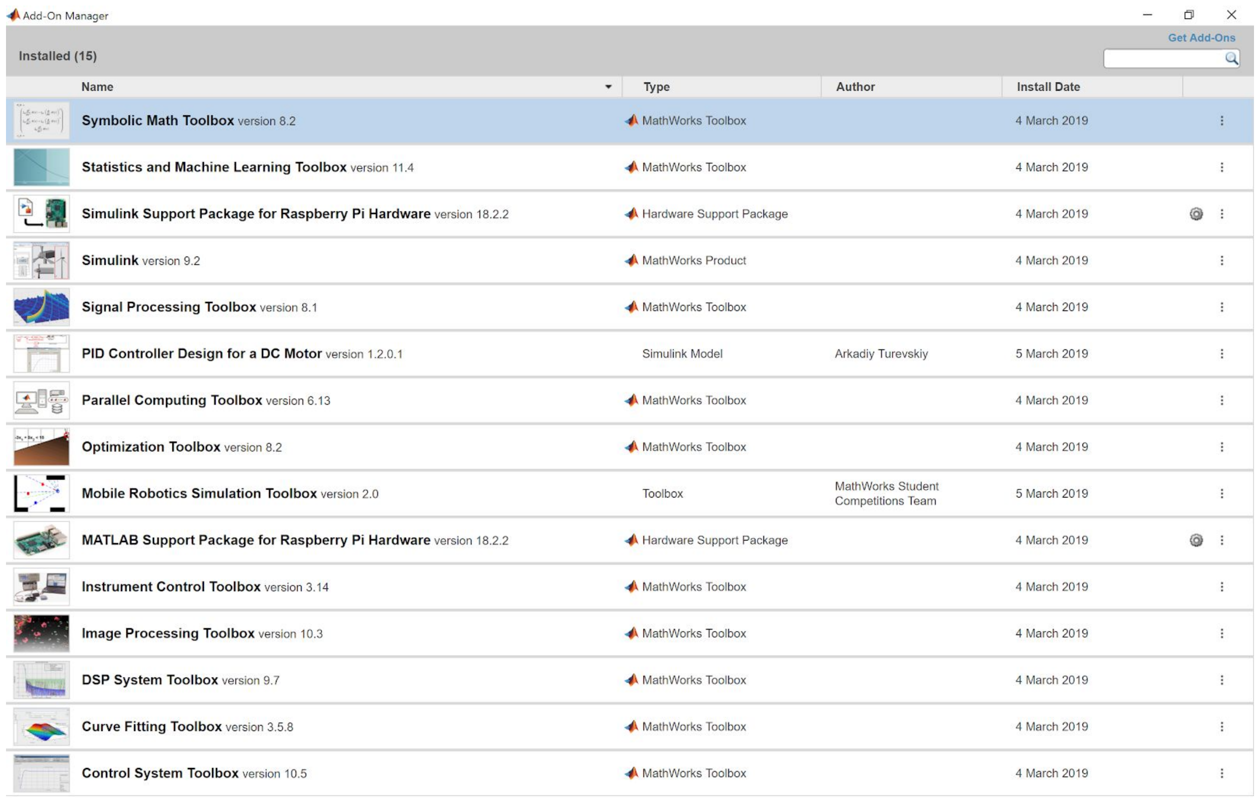
Preview Content of Model Components

Display representation of block contents on the face of a block.

Navigate Model Hierarchies

Navigate model hierarchy of subsystems and referenced models.

Besides the general model set up procedure, additional toolbox can provide external help with existing modules such as Raspberry Pi Compiler or Differential Drive Motor Simulator. Some of them are free to use but some of them might need to be purchased. For this specific project, the following tool boxes shown in Figure 1 are used.



Name	Type	Author	Install Date
Symbolic Math Toolbox version 8.2	MathWorks Toolbox		4 March 2019
Statistics and Machine Learning Toolbox version 11.4	MathWorks Toolbox		4 March 2019
Simulink Support Package for Raspberry Pi Hardware version 18.2.2	Hardware Support Package		4 March 2019
Simulink version 9.2	MathWorks Product		4 March 2019
Signal Processing Toolbox version 8.1	MathWorks Toolbox		4 March 2019
PID Controller Design for a DC Motor version 1.2.0.1	Simulink Model	Arkadiy Turevskiy	5 March 2019
Parallel Computing Toolbox version 6.13	MathWorks Toolbox		4 March 2019
Optimization Toolbox version 8.2	MathWorks Toolbox		4 March 2019
Mobile Robotics Simulation Toolbox version 2.0	Toolbox	MathWorks Student Competitions Team	5 March 2019
MATLAB Support Package for Raspberry Pi Hardware version 18.2.2	Hardware Support Package		4 March 2019
Instrument Control Toolbox version 3.14	MathWorks Toolbox		4 March 2019
Image Processing Toolbox version 10.3	MathWorks Toolbox		4 March 2019
DSP System Toolbox version 9.7	MathWorks Toolbox		4 March 2019
Curve Fitting Toolbox version 3.5.8	MathWorks Toolbox		4 March 2019
Control System Toolbox version 10.5	MathWorks Toolbox		4 March 2019

Figure 1: Toolbox installed for MATLAB Simulink

Since the control algorithms are relatively complicated, block diagrams are not sufficient to perform simulation. One important technique used to provide a graphical language that includes state transition diagrams, flow charts, state transition tables, and truth tables is Stateflow. One demonstration of the usage of Stateflow is motion control algorithm based on differential drive principle. As shown in Figure 2, a waypoint tracking logic is implemented with a PID controller. Note that two different states are created and if/else statements are used to compute interconnected calculations and decide when to enter different states. Simulink Functions and MATLAB Functions can be specified individually in Stateflow and used as normal functions. Stateflow is commonly used in control diagrams, localization algorithm, danger loop creation and other blocks in this simulation.

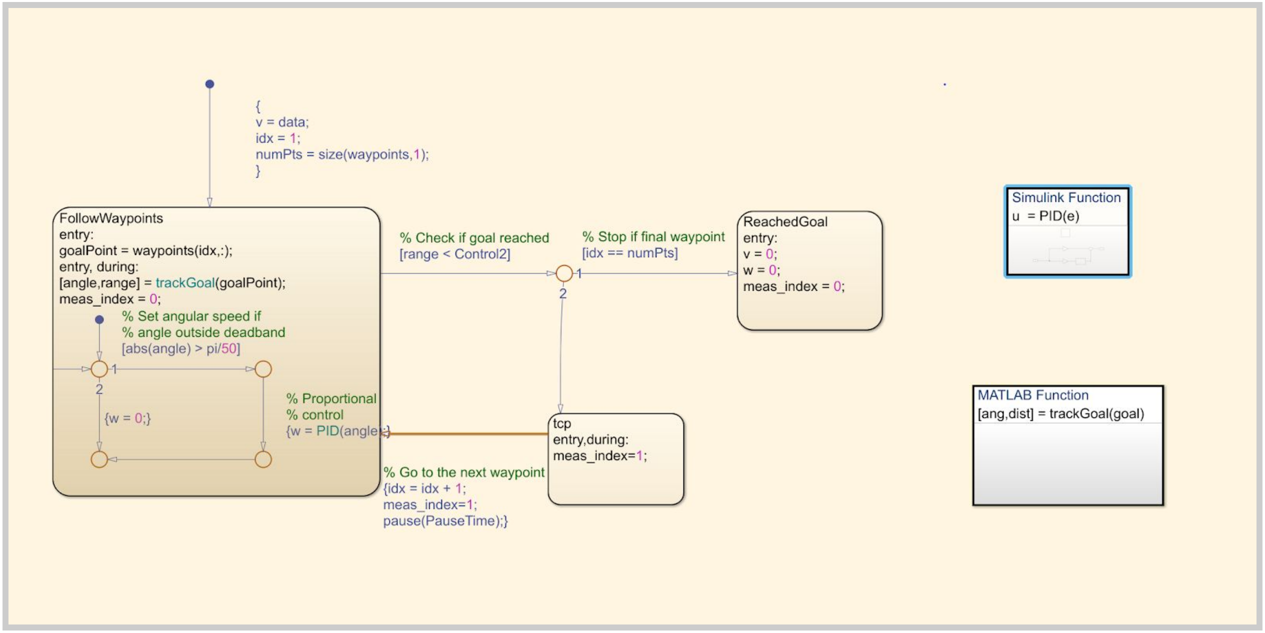


Figure 2: Stateflow in Simulink Control Block

1.2 Simulation Modules and Algorithm

The whole simulation is based on a general simulation scheme outline. Besides the fundamental waypoint tracking method, several additional subsystems are constructed to satisfy the requirements of CCATp robot: TCP/IP communication system with external device; Auto-generated path plan to map the entire mirror surface; Localization subsystem to minimize the cumulative error; and danger loop to prevent damage under extreme conditions such as power shutdown or communication lost. Because most control algorithms need sensor output, simulation also requires generation of signal output (often booleans) as input of subsystems. Figure 3 demonstrates overall control outlines and all developed Simulink simulations are based on this outline.

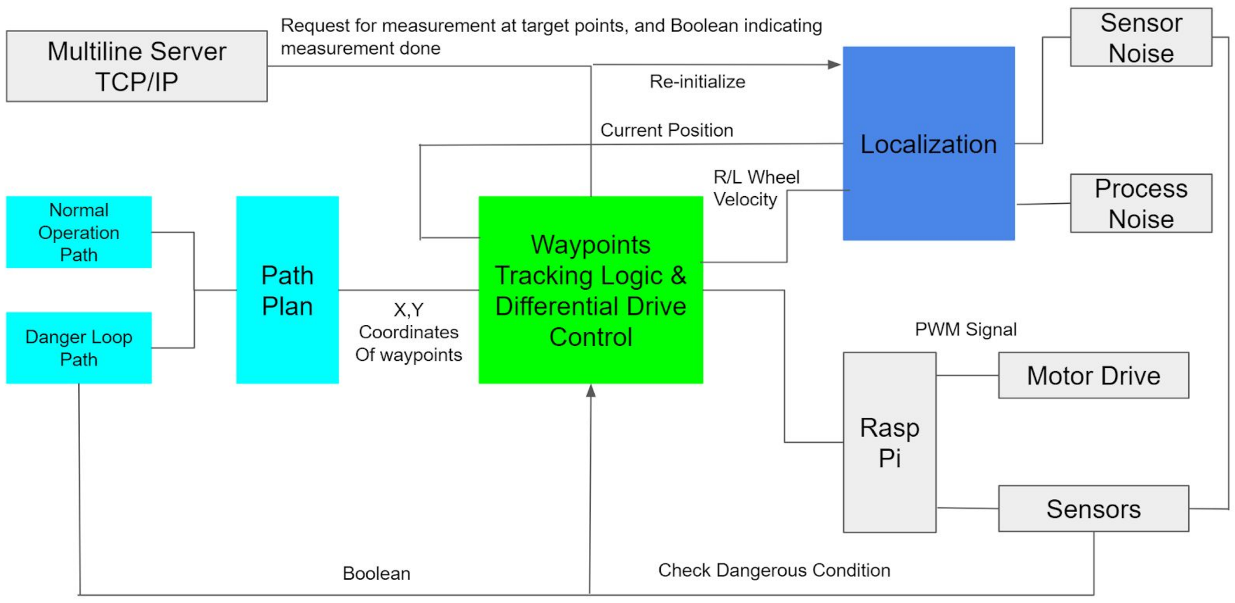


Figure 3: Simulation Control Scheme Outline

The rest of the content in this section will be a close description of how each module is constructed and what are the inputs and outputs of each module. All related files can be found in shared Google Drive - Control Group - Simulink Simulation folder. Please use the following description as a reference while using related Simulink files. Figure 4 is the overview of the whole simulation diagram, notice that two TCP/IP inputs are required to make the entire simulation operable. One TCP/IP inputs are simulated as Etalon Measurement and another indicates laboratory emergency signal. The entire simulation generates 10 waypoints as moving path, the robot moves along the path with continuously updated position and auto generated small adjustment. Noises are actively inserted in both moving process and IMU simulator to simulate real world errors to check the capability of the relocalization algorithm.

All blocks in the Simulink diagram involve one or more matlab functions. The inputs and outputs of each function is documented in the script itself. For the purpose of clarity , they are not going to be explained in detail. For more information, please refer to other students' reports in their specific fields. For example, all matlab scripts used in frame transformation are explained in detail in Ruohan Gao and Seth McCall Final Report in google drive.

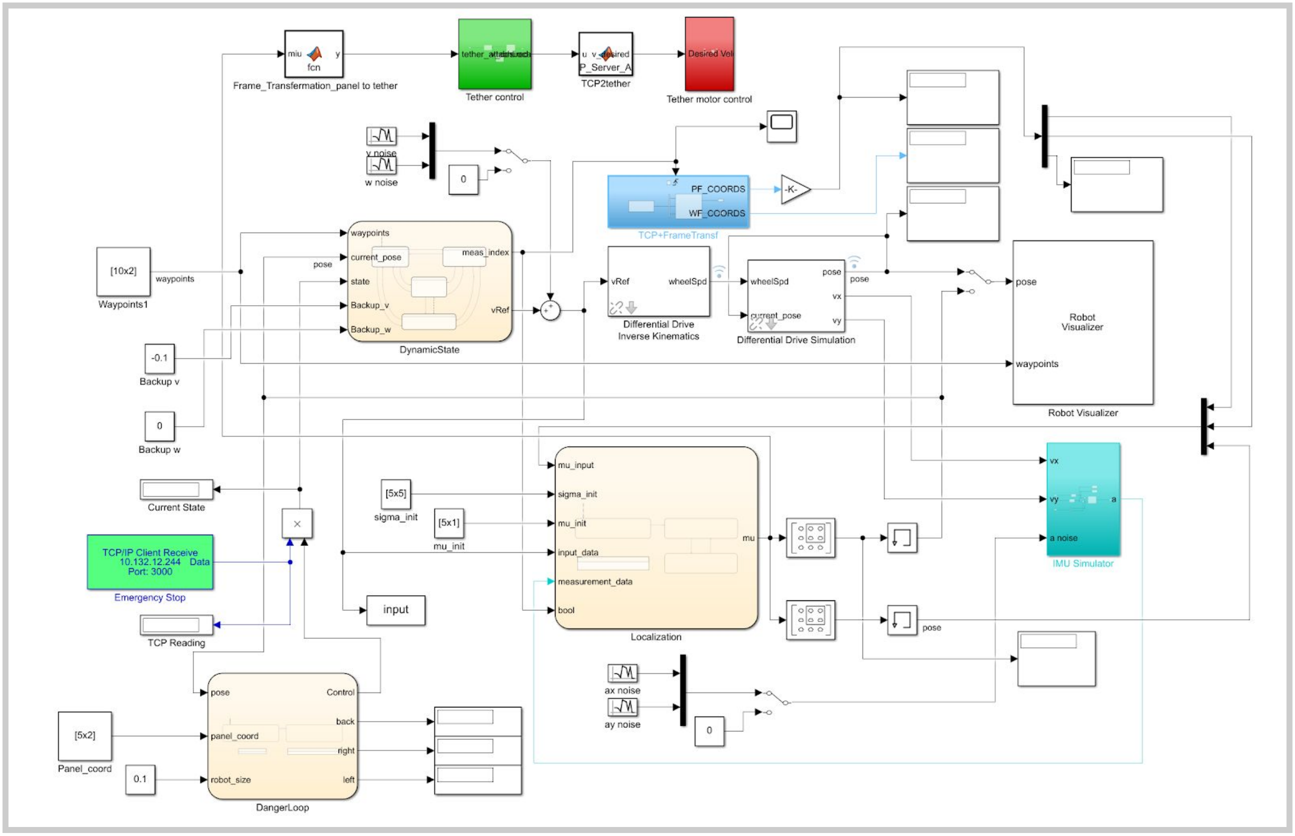


Figure 4: Simulation Diagram Overview

Dynamic States/Motion Control

As shown in Figure 5, there are 4 states in total in the outer stateflow hierarchy, each state represents the following moving conditions: normal operation, return to base, emergency stop and back up. Each state has its own embedded control algorithm that adjust the wheel velocities based on position update. The decision made on which state is chosen depends on one of the input: *state*, which is an integer between 1 and 4. The number is determined as a combination of both TCP/IP danger signal and danger loop subsystem output. The other inputs are *waypoints, current_pose, Backup_v, Backup_w* while the outputs are *meas_index* and *vRef*. *Meas_index* determines when to take IMU data and used for relocalization and *vRef* outputs the desired wheel velocities for left and right wheels. For more information regarding the inputs and outputs, please refer to Appendix A MATLAB Simulink Input and Output.

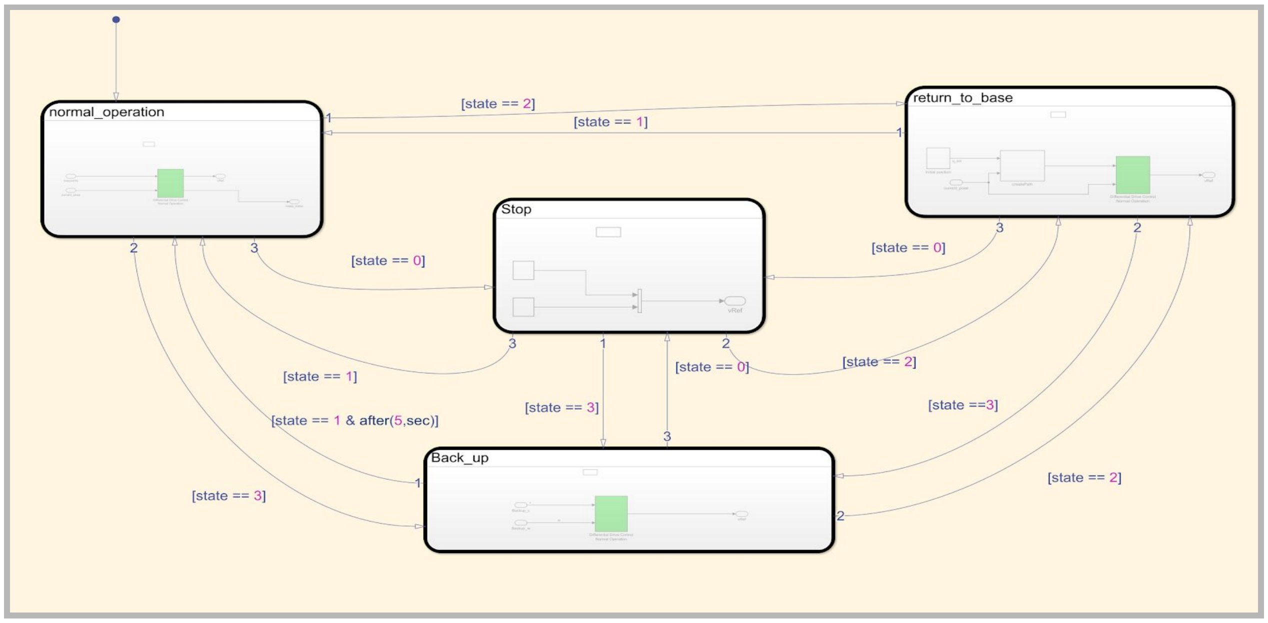
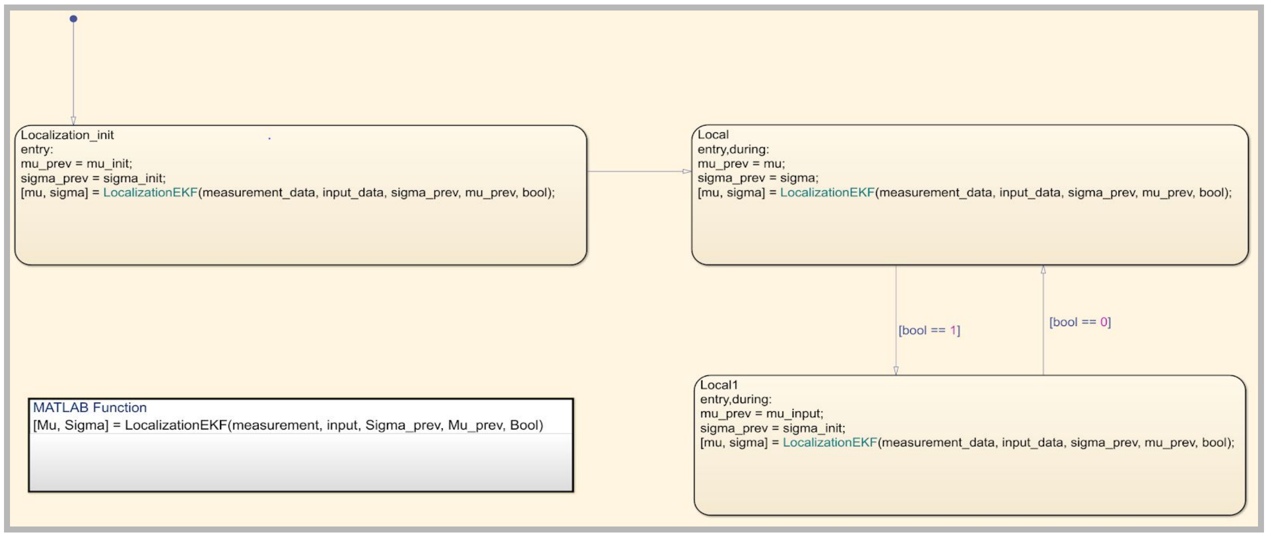


Figure 5: Dynamic State Subsystem Overview

Localization

Localization algorithm is based on updating current position (x,y,w) based on IMU data and using Kalman Filter to filter out the noises. *Sigma_init* and *Mu_init* are inputs that initialize the function and *measurement_data* is output of IMU simulator. The *meas_index* from dynamic state subsystem determines when to triggle the position update. The output is the current position *mu* which will be used as current position input. For more information regarding the inputs and outputs, please refer to Appendix A MATLAB Simulink Input and Output.



```
Localization_init
entry:
mu_prev = mu_init;
sigma_prev = sigma_init;
[mu, sigma] = LocalizationEKF(measurement_data, input_data, sigma_prev, mu_prev, bool);
```

```
Local
entry,during:
mu_prev = mu;
sigma_prev = sigma;
[mu, sigma] = LocalizationEKF(measurement_data, input_data, sigma_prev, mu_prev, bool);
```

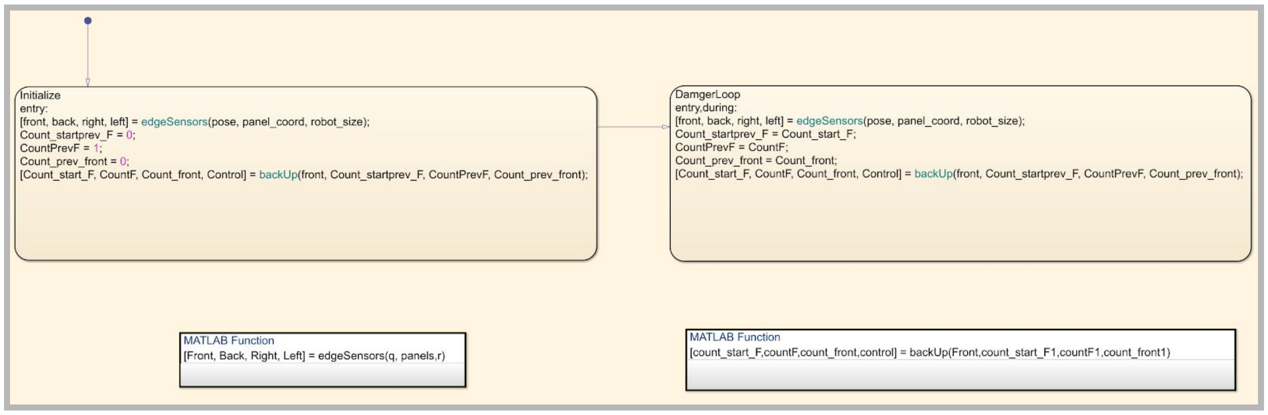
```
Local1
entry,during:
mu_prev = mu_input;
sigma_prev = sigma_init;
[mu, sigma] = LocalizationEKF(measurement_data, input_data, sigma_prev, mu_prev, bool);
```

```
MATLAB Function
[Mu, Sigma] = LocalizationEKF(measurement, input, Sigma_prev, Mu_prev, Bool)
```

Figure 6: Localization Subsystem Overview

Danger Loop

As shown in Figure 7, Danger Loop subsystem is similar to localization but has only one output that partially determines which dynamic state should robot operate in. The *size* of robot and *panel_coordinate* determine whether the robot is approaching the edge of panel and ready to switch to *back up* mode. The output *Control* will combine with laboratory danger signal to give a *state* input into the dynamic state. In real world situations, the input of the danger loop will be switched to edge sensor signal input. For more information regarding the inputs and outputs, please refer to Appendix A MATLAB Simulink Input and Output.



```
Initialize
entry:
[front, back, right, left] = edgeSensors(pose, panel_coord, robot_size);
Count_startprev_F = 0;
CountPrevF = 1;
Count_prev_front = 0;
[Count_start_F, CountF, Count_front, Control] = backUp(front, Count_startprev_F, CountPrevF, Count_prev_front);
```

```
DamgerLoop
entry,during:
[front, back, right, left] = edgeSensors(pose, panel_coord, robot_size);
Count_startprev_F = Count_start_F;
CountPrevF = CountF;
Count_prev_front = Count_front;
[Count_start_F, CountF, Count_front, Control] = backUp(front, Count_startprev_F, CountPrevF, Count_prev_front);
```

```
MATLAB Function
[Front, Back, Right, Left] = edgeSensors(q, panels,r)
```

```
MATLAB Function
[count_start_F,countF,count_front,control] = backUp(Front,count_start_F1,countF1,count_front1)
```

Figure 7: Danger Loop Subsystem Overview

Frame Transformation

Fram transformation is required when triggle Etalon Measurement. Since the received robot position is its 3D position in global coordinate, transformation is required to compute 3D coordinates into 2 dimensional coordinates in panel frame. Notice in Figure 9,unlike other subsystems, all 4 blocks are matlab functions. For more information regarding the inputs and outputs, please refer to Ruohan Gao and Seth McCall Final Report in google drive.

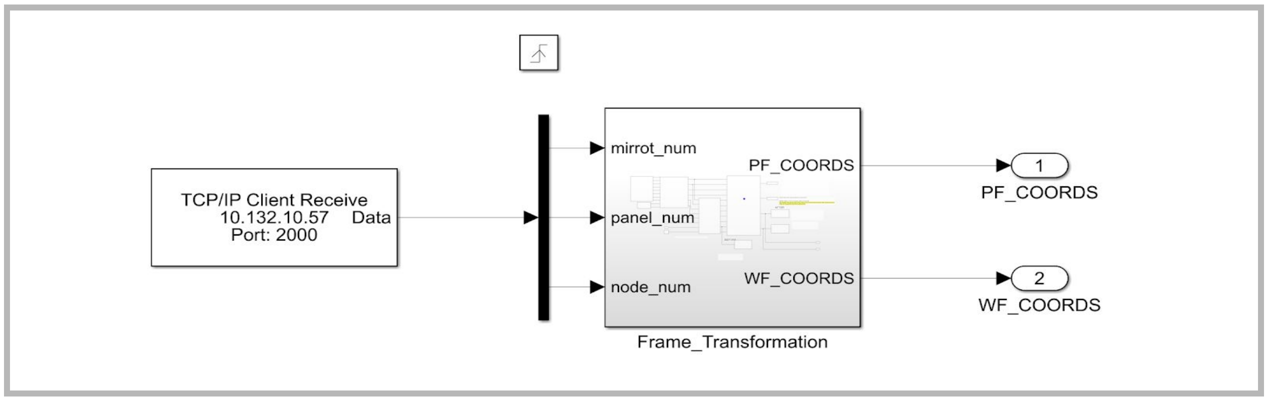
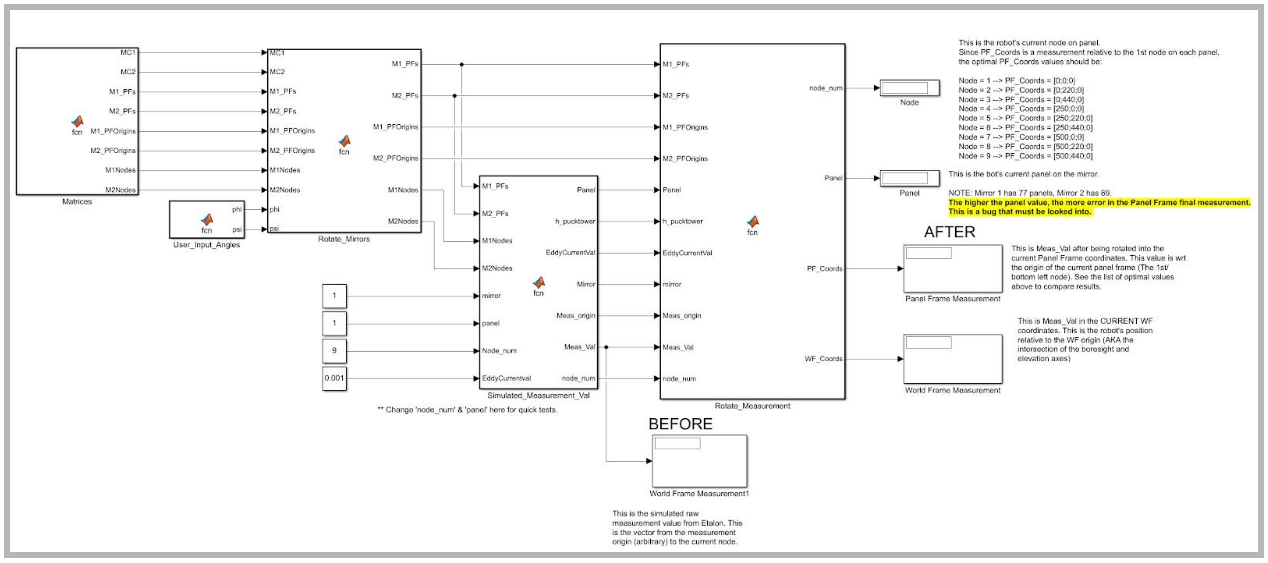


Figure 8: Frame Transformation Subsystem Overview



This is the robot's current node on panel.
Since PF_Coords is a measurement relative to the 1st node on each panel, the optimal PF_Coords values should be:

Node = 1 --> PF_Coords = [0;0;0]
Node = 2 --> PF_Coords = [0;220;0]
Node = 3 --> PF_Coords = [0;440;0]
Node = 4 --> PF_Coords = [250;0;0]
Node = 5 --> PF_Coords = [250;220;0]
Node = 6 --> PF_Coords = [250;440;0]
Node = 7 --> PF_Coords = [500;0;0]
Node = 8 --> PF_Coords = [500;220;0]
Node = 9 --> PF_Coords = [500;440;0]

This is the bot's current panel on the mirror.

NOTE: Mirror 1 has 77 panels, Mirror 2 has 69.
The higher the panel value, the more error in the Panel Frame final measurement. This is a bug that must be looked into.

This is Meas_Val after being rotated into the current Panel Frame coordinates. This value is wrt the origin of the current panel frame (The 1st/ bottom left node). See the list of optimal values above to compare results.

This is Meas_Val in the CURRENT WF coordinates. This is the robot's position relative to the WF origin (AKA the intersection of the boresight and elevation axes)

** Change 'node_num' & 'panel' here for quick tests.

This is the simulated raw measurement value from Etalon. This is the vector from the measurement origin (arbitrary) to the current node.

Figure 9: Detailed Block Diagram inside Frame Transformation Block

1.3 Simulation Results & Future Work

The overview of the final version of Simulink simulation is Figure 4, however, several different generations of similar models are generated to simulate robot trajectory under different circumstances. The fundamental differences focus on whether outer TCP/IP inputs are needed and whether noises are required to take into consideration. Some videos are recorded for different Simulink models and placed in google drive. In general, a 10 waypoints trajectory is shown in the video. Figure 10 exhibits the complete results of model without TCP/IP inputs and Figure 11 demonstrates the trajectory with backup motion and emergency stop. More tests can be performed based on the requirement and some issues are popping up during the implementation.

- Sensor Data is hard to simulate because some of them are booleans and limited by sampling frequency
- Frictions and Sliding can only be simulated by manually adding noise into the result, this method may not correctly generate errors in real tests.
- The current version of Simulink model is based on one panel waypoint tracking. Although a new cross panel algorithm is developed and demonstrated, it needs to be implemented into the general model.
- At this point, the moving method is based on differential drive. However, due to the limitation of the torque provided by motor, the maximum angular velocity it

can achieve is limited. Deeper evaluations of this problem need to be assessed to ensure it can turn successfully in sharp angles.

- Further simulation is needed, but at this point, testing should be based more on actual test in similar environment of Chili facility. Working closely with the testing group for future implementation is recommended.

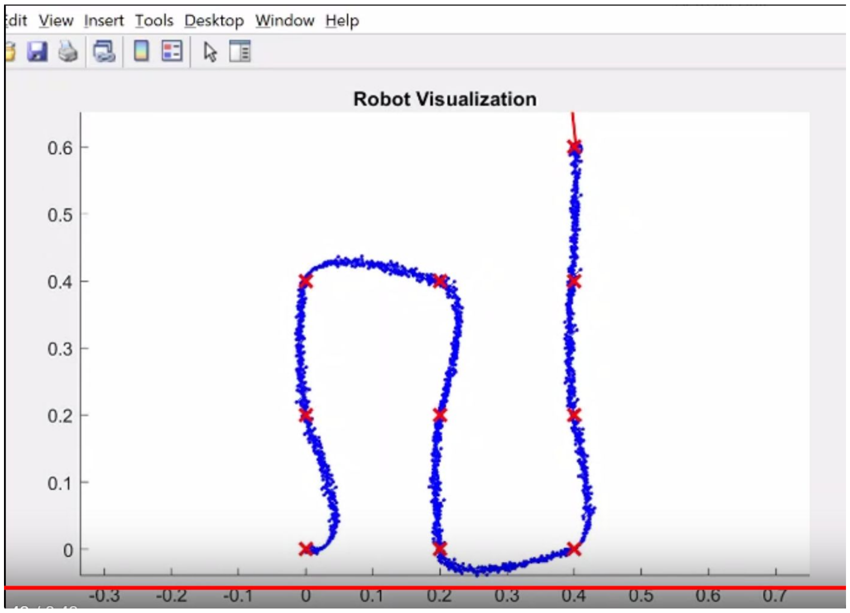


Figure 10: Simulation results with noises

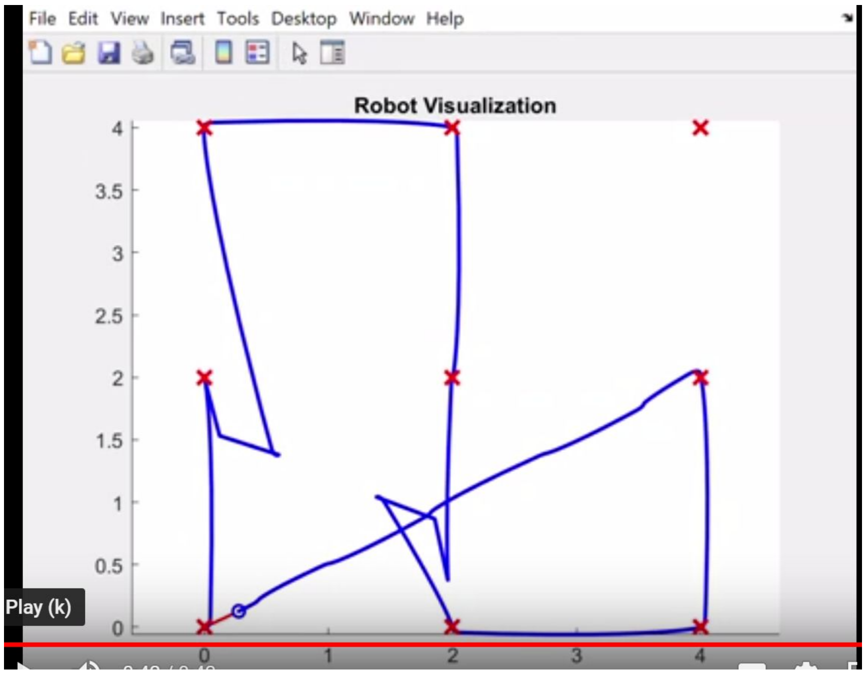


Figure 11: Simulation Result with Backup and Return-to-base Motion

Chapter 2 Software Development

2.1 Software Platform Introduction

The entire control algorithm architecture is described in section 2.2. Since the requirement restricts the amount of computation on board, three levels of computation is developed: Teensy 3.5 for data acquisition, Raspberry Pi 3B+ for on-board computation, data storage and transmission, and off-board computer for close loop control and user input. Different softwares are used for coding in different platforms. This section will introduce some basic knowledge and command in three software platforms to get readers familiar with them.

Teensy 3.5 & C Language

Teensy 3.5 & 3.6 are high-end offerings in the new line of Teensys. The main CPU is the Freescale/NXP Kinetis K66 ARM Cortex-M4 running at 180 MHz. There's 1 Megabyte of Flash on board, 256K of RAM, and 4K of EEPROM in this chip, 32 DMA channels, two CAN bus ports, and a USB High Speed (480 Mbit/sec) port. The microcontroller also supports 25 analog inputs with 13-bit resolution, two analog outputs with 12-bit resolution, a native SD card port, Ethernet MAC capable of 100 Mbps (you'll need a 'shield' for this), I2S audio, a crypto acceleration unit, random number generator, six serial ports, three SPI ports, four I2C ports, and a real-time clock. . There are 62 IO pins available as 0.1″ headers and as SMD pads on the back.

Similar to Arduino, the coding software is C language based, the library needs to be installed before using. The 13 bit resolution of the analog pins can support higher accuracy with Eddy Current Sensor installed on board. In the current version of Teensy code, it takes ECS data and transmitted to pi for temporary storage and encoder readings from four motors into PID controller in Pi. For Detailed Pin Playout, please refer to Appendix B Hardware Pin Layout. Serial connection is used to communicate with Pi for data transmission. Therefore, the ECS data acquisition frequency is limited. All commands to motors, including fan motors are transformed into PWM signals here and send to ESCs or Motor Drive.

Raspberry Pi 3B+

The Raspberry Pi is a fully-fledged mini computer, capable of doing whatever you might do with a computer. It comes with 4x USB, HDMI, LAN, built-in Bluetooth/WiFi support, 1GB RAM, 1.2GHz quad-core ARM CPU, 40 GPIO (General Purpose Input Output) pins, audio and composite video output, and more. Rather than not having many choices, instead, your options are staggeringly large. With pre-programed python scripts, the Pi can auto start the script once it boots up and set up all TCP/IP connections with off-board control computers. It can also support SSH, VNC connections and get remote control from any laptop that knows its IP address. For this project, the python scripts development is processed by using display with HDMI cable, keyboard and mouse. For Detailed Pin Playout, please refer to Appendix B Hardware Pin Layout.

The current script in the Pi contained serial connection with Teensy and TCP/IP connection with control computer. For security purposes, the TCP/IP communication is setup in a separate router. For detailed information please refer to section 2.3 TCP/IP communication. The script will store ECS data and send it back to the control computer through the socket and receiving control command. PID controller is developed to tune the motor torques.

Control Computer & MATLAB
Any computer with WIFI access and MATLAB subscription is capable of performing off-board control computing. All related files and functions can be found in google drive - control group - Matlab Code. Basically, the off-board computer will give out measurement commands and motor speed commands based on sensor output received. The overall control algorithm is similar to the MATLAB Simulink model described in section 1 but in pure function form. Off-board computers should establish TCP/IP communication to Raspberry Pi, Etalon measuring environment, and Laboratory server.

At the end of Fall 2019, an open loop control script is implemented for Germany Test on 5th December. It takes desired linear velocity, angular velocity, travelling distance as input. The user can choose percent of fan throttle and triggle ECS measurement. A Graphical User Interface (GUI) is created for clarity and convenience and will be described in detail in section 2.4 MATLAB GUI.

2.2 Overall Control Architecture

This project requires communication between off-board computers between not only on-board hardwares, but also Multiline Server (provided by Etalon Measurement Environment) and Chili Telescope Laboratory server. To corporate the communication and according to the testing environment provided by Etalon, TCP/IP socket is used for data transmission. However, the limitation of TCP/IP reading frequency limits the data transmission speed and related sensor reading frequency. Further work is needed to accomplish this problem.

As shown in Figure 12, besides the three level control platform described above in section 2.1, additional control required to adjust tether length. Tether is used to support power for all electronics and motors on board and prevent potential damage to mirror while power shut down. The length of the tether needs to be adjusted along with the movement of the robot. This particular part has not been developed yet but needed to be noticed since this command will be directly controlled from the off-board computer. Since Teensy 3.5 cannot store large amounts of data by itself, ECS data must be transmitted to Pi immediately, this limits the reading frequency. Better control architecture might be created to solve this problem later on. Edge sensors are tested individually but have not been implemented into the entire script since the danger loop is not finished.

Multiline Server Testing Environment is tested with TCP/IP communication in Spring 2019. Command sending and receiving are success. However, since the testing environment is not fully functionable, the format of real measurement results are not clear at this point. The detailed command format received from Multiline Server can be found in drive.

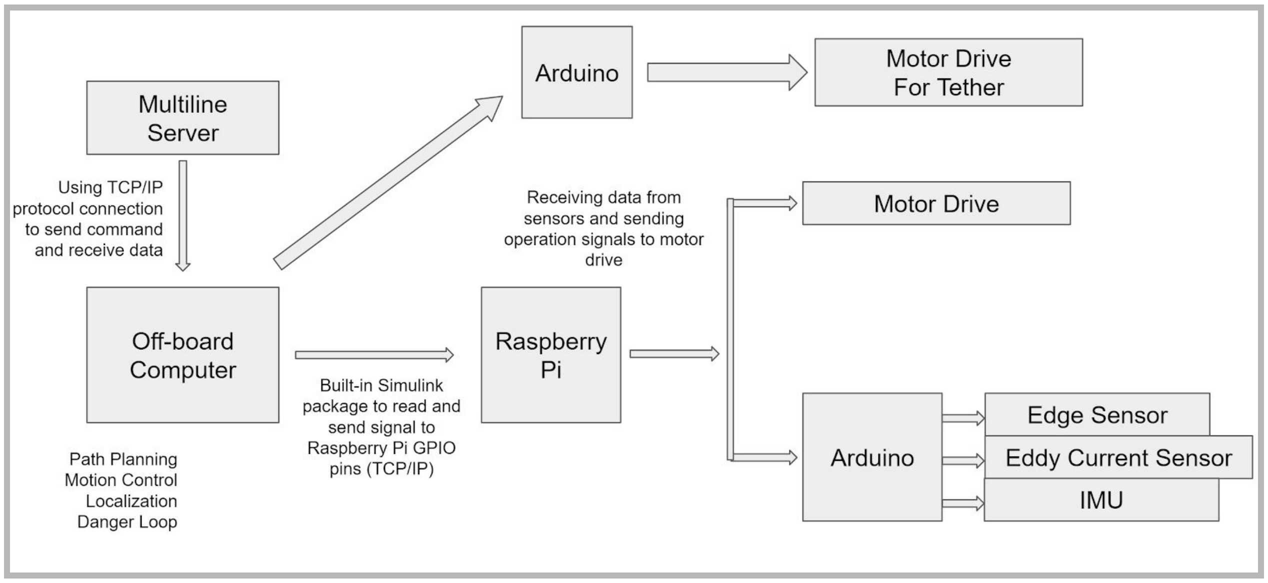

Multiline Server
Using TCP/IP protocol connection to send command and receive data
Off-board Computer
Path Planning
Motion Control
Localization
Danger Loop
Built-in Simulink package to read and send signal to Raspberry Pi GPIO pins (TCP/IP)
Arduino
Motor Drive For Tether
Receiving data from sensors and sending operation signals to motor drive
Raspberry Pi
Motor Drive
Arduino
Edge Sensor
Eddy Current Sensor
IMU

Figure 12: Simulation Result with Backup and Return-to-base Motion

2.3 TCP/IP Communication

TCP/IP is the communication protocol for communication between computers on the Internet. TCP/IP stands for Transmission Control Protocol / Internet Protocol. It defines how electronic devices should be connected to the Internet, and how data should be transmitted between them. TCP is for communication between applications. If one application wants to communicate with another via TCP, it sends a communication request. This request must be sent to an exact address. After a "handshake" between the two applications, TCP will set up a "full-duplex" communication between the two applications. The "full-duplex" communication will occupy the communication line between the two computers until it is closed by one of the two applications.

This communication protocol is used in CCATp project to establish communication between Etalon Multiline Server, Off-board Computer, On-board Raspberry Pi and Laboratory Server. The off-board computer is treated as TCP Client while others are treated as TCP Client. Before establishing the protocol, all applicants must under the same Local Area Network (LAN) and know IP addresses. To establish TCP/IP protocol in both MATLAB and Python, please refer to the following procedures.

Configure Properties of TCP/IP In MATLAB

- **Address:** Remote host name or IP address for connection. Specify address as the first argument when you create the tcpclient object. In this example Address is '172.28.154.231'. t = tcpclient('172.28.154.231', 4012)
- **Port:** Remote host port for connection. Specify port number as the second argument when you create the tcpclient object. The Port must be a positive integer between 1 and 65535. In this example Port is 4012.
- **BytesAvailable:** Read-only property that returns the number of bytes available in the input buffer.
- **Timeout:** Waiting time in seconds to complete read and write operations, specified as a positive value of type double. The default is 10. You can change the value either during object creation, or after you create the object.
- **ConnectTimeout:** Maximum time in seconds to wait for a connection request to the specified remote host to succeed or fail, specified as a positive value of type double. If not specified, the default value is Inf. You can change the value only during object creation.

Functions:

- fopen(t) will start the TCP/IP

- fscanf(t) will read out bytes available in TCP Buffer as text
- fread(t) will read out bytes available in TCP Buffer as numbers
- fwrite(t, data) will write data into buffer and ready to be read
- t.BytesAvailable can check bytes of data in buffer that is ready to be read
- t.InputBufferSize will specify the number of bytes can be written in

Here is the code used in GUI for TCP/IP set up:

t = tcpip('192.168.1.105', 20000, 'NetworkRole', 'client');

t.InputBufferSize = 40000;

t.Timeout = 30;

TCP/IP Client in Python

- Import socket is necessary to
- Specify Ip address: TCP_IP = '127.0.0.1'
- Specify Port number: TCP_PORT = 5005
- Specify BUFFER_SIZE = 1024
- Create Socket: s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- Start the socket as client: s.connect((TCP_IP, TCP_PORT))
- Start the socket as server: s.bind((TCP_IP, TCP_PORT))

TCP/IP Functions in Python

- s.send(MESSAGE)
- s.close()
- s.recv(BUFFER_SIZE)
- conn, addr = s.accept()
- conn.send(data)
- conn.recv(BUFFER_SIZE)
- conn.close()

Notice that for this project, TCP server is created in python script in Pi and client is created in MATLAB in off-board computer. The input buffer size is increased to 40000 to allow large amounts of ECS data to be transfered. The timeout range is 30 seconds to avoid a long waiting period. However, the whole data processing time and bytes reading time is roughly 50 seconds from measuring initialized until data has been plotted. This is not ideal in terms of measuring continuity. Further development is needed to allow faster data transmission time.

2.4 MATLAB GUI

GUIs (also known as graphical user interfaces or UIs) provide point-and-click control of software applications, eliminating the need to learn a language or type commands in order to run the application. MATLAB apps are self-contained MATLAB programs with GUI front ends that automate a task or calculation. The GUI typically contains controls such as menus, toolbars, buttons, and sliders. Many MATLAB products, such as Curve Fitting Toolbox, Signal Processing Toolbox, and Control System Toolbox include apps with custom user interfaces. In this project, for simplicity, GUIDE is used to help create simple UI with callback functions. Figure 13 is the GUI created for Germany Test in Dec. 2019. Notice there are several buttons, user input text fields and a slider for fan throttle control. The empty plot area is for ECS reading visualizer. The whole GUI can be found in Google Drive - Control Group - Control_script_Matlab folder.

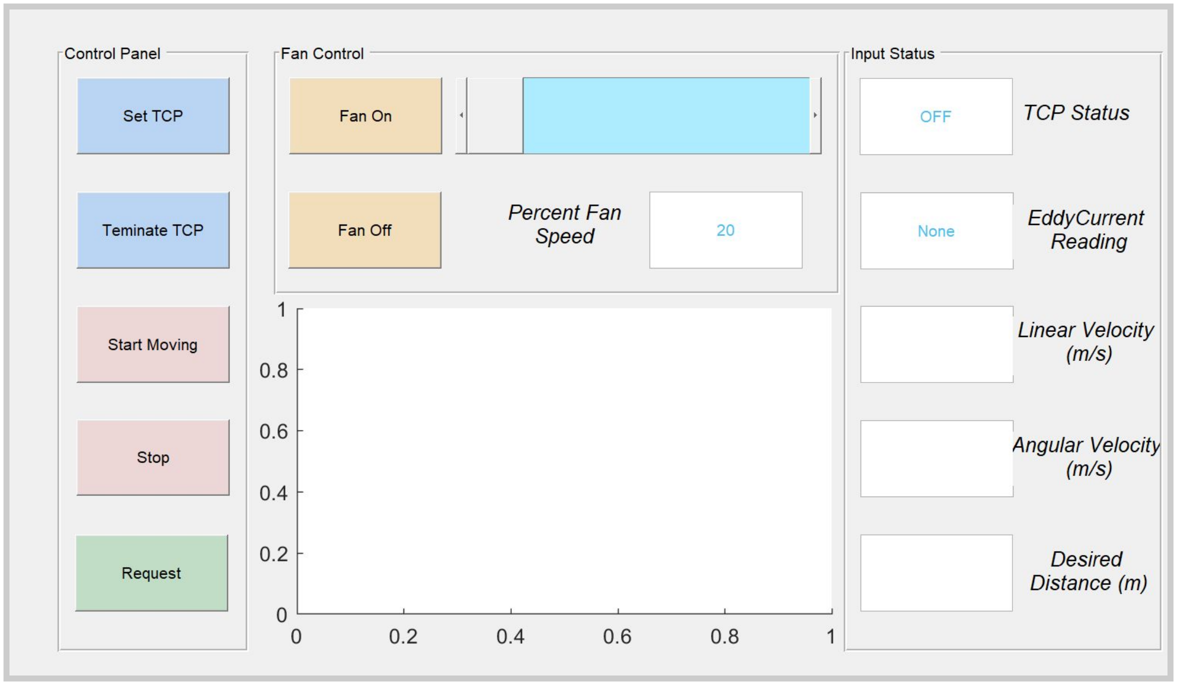


Figure 13: Graphical User Interface Created in MATLAB

For a detailed tutorial of how to create GUI with guide, please refer to MATLAB tutorial. https://blogs.mathworks.com/videos/2013/02/06/introduction-to-gui-building-with-guide-in-matlab/. The rest of this section will summarize some important points that are worth noting in GUI created above.

- Each GUI includes two parts: a script file with all callback functions, and a GUI template that determines how the interface will look like.

- The *tag* property of each handle corresponds to the name of each callback function. To read or change the values of the handle, the following function can be used
    - *handles.TCP_IND.String = 'ON' set*s handle's string text
    - *set(handles.slider1, 'Value', speed)* sets handle's value
    - *str2double(handles.Linear_Vel.String)* reads out user input string as numbers
- For variables created in one callback function and need to be used in other functions, please create it as a global object first and call the object in function that need to be used. For example, global t create tcp objects as a global object.
- If some buttons or slider should not be activate until an event happen, the following function can be used:
    - *set(handles.Move_Forward,'enable','on')*
    - *set(handles.Move_Forward,'enable','off')*

2.5 Current Control Status and Limitations

So far until Dec 2019, the focus is on open loop control and device communications. This section will summarize the work that has been done in the past year and work might need to be finished in future. Note that all the following content is based on control algorithm decided in previous sections and further adjustment is needed alongside any changes.

Current Status

- Motor drive script that controls 4 motors with PWM signal is finished.
- Encoder reading is successfully received by Teensy 3.5 and sent to Pi through serial communication.
- Teensy is able to receive commands from an off-board computer passed by Pi to triggle ECS readings.
- Pi is capable of performing PID control on motor speed based on user input desired speed and received encoder data.
- TCP/IP communication between off-board computer and Pi is stable, timeout and inputbuffersize are specified particular for Germany Test, further modification required.
- ECS reading can be read from ADC and past to Pi, the readings are noisy caused by 5V limitation on current ADC.
- IMU data acquisition script is finished but has not been implemented in the entire algorithm.

- To ensure data safety, a particular rotor is used to provide Local Area Network which is separate from Ethernet. The password and name is shown below:
    - *Name: CCATprobot*
    - *Passward: IM4CCATp#*
- MATLAB GUI is created with functions to set fan speed, pass desired motor speed, receiving and processing ECS readings, establishing and disconnecting TCP/IP protocol. All relevant functions are located in the same folder described in section 2.4 .

Future Work

- Solve reading frequency problems, possibly find a better way to read and store data.
- Test command sending and data receiving with real Multiline Server.
- Working with a mechanical team to design new chassis that fit all electronic devices and have space for wiring .
- Implement control algorithms that adjust tether length.
- Purchasing new ADC that is able to take 10V analog signal.
- The current motor cannot provide enough torque to make sharp turns. Either differential drive motion needs to be changed or, motor with larger torque needs to be considered.
- Close loop Control
    - Implement IMU into the control script
    - Test EKF accuracy
    - Get Dangerloop working with edge sensors

Chapter 3 Hardware Connection

3.1 On-board/Off-board Hardware Introduction

This section will evaluate electronic devices and mechanical structures installed on the robot or will be potentially installed in the Chili facility. Electronic devices include Teensy 3.5, Raspberry Pi 3B+, edge sensors, Eddy Current Sensor, Analog to Digital Converter, motors, 2 suction fans, 12V to 5V transformer and IMU. Puck tower will be used at the center of robot to support measuring devices as well as secure ECS sensor. For detailed description of Teensy and Pi, please see section 2.1 and for puck tower design and thermal expansion results, please refer to project report from mechanical team. The rest of this section will provide a brief introduction of other electronics mentioned above.

- **Edge sensors:** SparkFun Line Sensor Breakout - QRE1113 (Digital) - This version of the QRE1113 breakout board features a digital output, using a capacitor discharge circuit to measure the amount of reflection. This tiny board is perfect for line sensing when only digital I/O is available, and can be used in both 3.3V and 5V systems. The power input and output pins are brought out to a 3-pin, 0.1" pitch header. The board also has a single mounting hole to screw it onto the chassis.
- **ADC:** ADS1115 contains a system that converts an analog signal, such as a sound picked up by a microphone or light entering a digital camera, into a digital signal. This ADC may also provide an isolated measurement such as an electronic device that converts an input analog voltage or current to a digital number representing the magnitude of the voltage or current. Typically the digital output is a two's complement binary number that is proportional to the input, but there are other possibilities.
- **12-5V Transformer:** SolarSynthesis DC DC converter 12V/24V input voltage, output voltage range is 3.7-12V, max output current is 4.2A. Note: the output voltage is Automatically detected, cannot be adjusted manually. The waterproof car power regulator can convert DC 12V/24V down to DC 5V 9V 12V; with max 95 percent conversion efficiency. Protective function includes over current protection, short circuit protection, over voltage protection, reverse connection protection, and over temperature protection.
- Other electronics will be introduced later in sections between 3.2 and 3.4. The overall wiring diagram is shown in Figure 14.

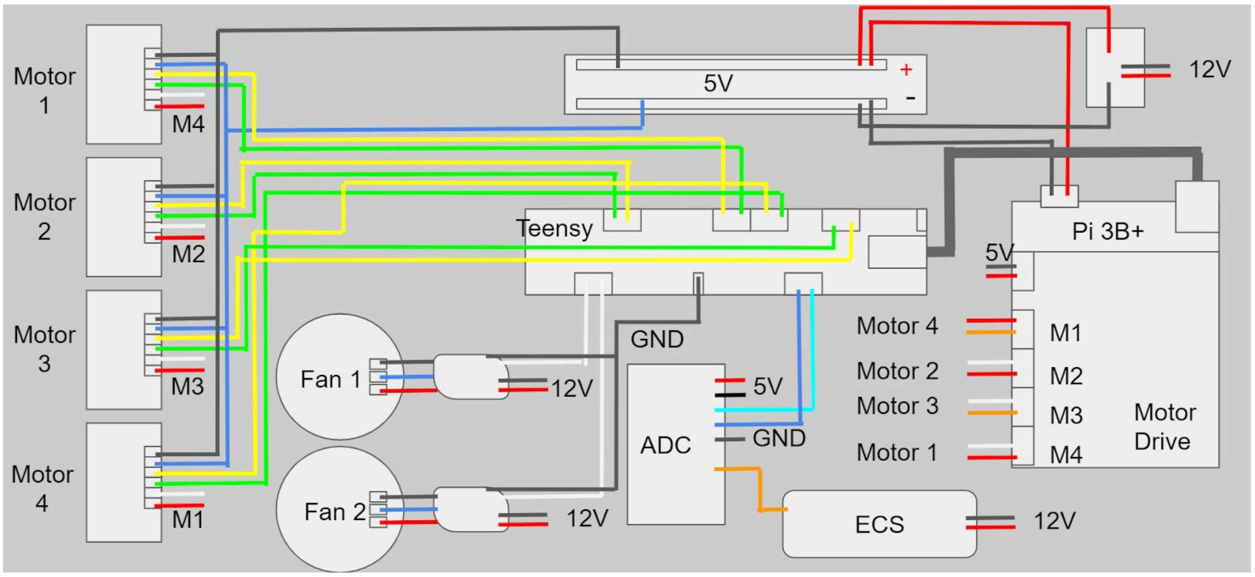


Figure 14: Wiring Diagram of Electronic Devices

3.2 Motor Connections

The motors currently used are 75:1 Micro Metal Gearmotor HP 6V with Extended Motor Shaft. These tiny brushed DC gearmotors are available in a wide range of gear ratios—from 5:1 up to 1000:1—and with five different motors: high-power 6 V and 12 V motors with long-life carbon brushes (HPCB), and high-power (HP), medium power (MP), and low power (LP) 6 V motors with shorter-life precious metal brushes. The 6 V and 12 V HPCB motors offer the same performance at their respective nominal voltages, just with the 12 V motor drawing half the current of the 6 V motor. The 6 V HPCB and 6 V HP motors are identical except for their brushes, which only affect the lifetime of the motor. For more information, please check the MEL file in Google drive and follow the website link of this motor. This motor has 2 wires for voltage input, 2 wires for PWM input and 2 wires for encoder output. The following connections are made to set up each motors and motor drive:

- PWM Signal
  - To Motor Drive
- Encoder Signal
  - Green and Yellow to Teensy Pins
- Black and Blue to 5V breadboard
  - Black Positive
  - Blue Negative
- Motor 3: M3 Port on Drive: White and Orange from Left to right
- Motor 4: M1 Port on Drive: Red and Orange from Left to right
- Motor 1: M4 Port on Drive: White and Red from Left to right
- Motor 2: M2 Port on Drive: White and Red from Left to right
- Voltage inlet: from 5V breadboard

3.3 Fan Connections

The choices of fans are developed throughout the year and the final version is Powerfun EDF 50mm 11 Blades Ducted Fan with RC Brushless Motor 4900KV with ESC 40A(2~4S). 2~4S SKYWALKER HOBBYWING high voltage version ESC can take up to 40A and Burst current(≤10s) can be up to 50A. Output is 5V/3A and battery type can be Lipo/NiMH. This fan supports 2S,3S,4S Lipo or 5~12cells NiMH. However, each fan will draw 500W power under full throttle and in total 1kW power is required from the facility. This leads to the demand of new transformers and tether that can safely pass that much amount of power under 12V. The connections of fans and ESCs are shown below:

- Power and PWM
    - Black to ESC Top
    - Red to ESC Bot
    - Blue to ESC middle
- ESC Power
    - To 12V from teather
- 3 continuous beep indicates that TCP are ready to be connected

3.4 Eddy Current Sensor Connections

Eddy Current Sensor is a product from Lion Precision, providing high resolution measurement for even the dirtiest environments. These advantages have made eddy-current sensors indispensable for many machine builders, production managers, or precision metrology applications. However, since it is super sensitive, it requires re-calibration every year and the wire of sensors must avoid any bending. For detailed information: https://www.lionprecision.com/products/eddy-current-sensors/ . It is connected to ADC to provide digital reading to Teensy 3.5. The connections are shown below:

- ADC 5V VDD to 5v Power
- ADC GND to GND
- ADC ADDR to GND
- ADC A0 to ECS 10V Output
- ADC SCL to Teensy SCL Pin 19
- ADC SDA to Teensy SDA Pin 18
- ECS Power to 12V tether

3.5 Other Connections

Teensy as fundamental control device connecting with all electronic devices are connected with more than 10 wires. The detailed pin connections are shown below. And other connections between transformers and Raspberry Pi is also shown in this section.

- Motor 3: Yellow Pin 3; Green Pin 4
- Motor 4: Green Pin 7; Yellow Pin 8
- Motor 1: Green Pin 9; Yellow PIN 10
- Motor 2: Yellow 29; Green 30
- Pin 37&38: 2 ESC White
- Pin 18&19: I2C on ADC (Eddy Current)
- Teensy powered from Pi USB port

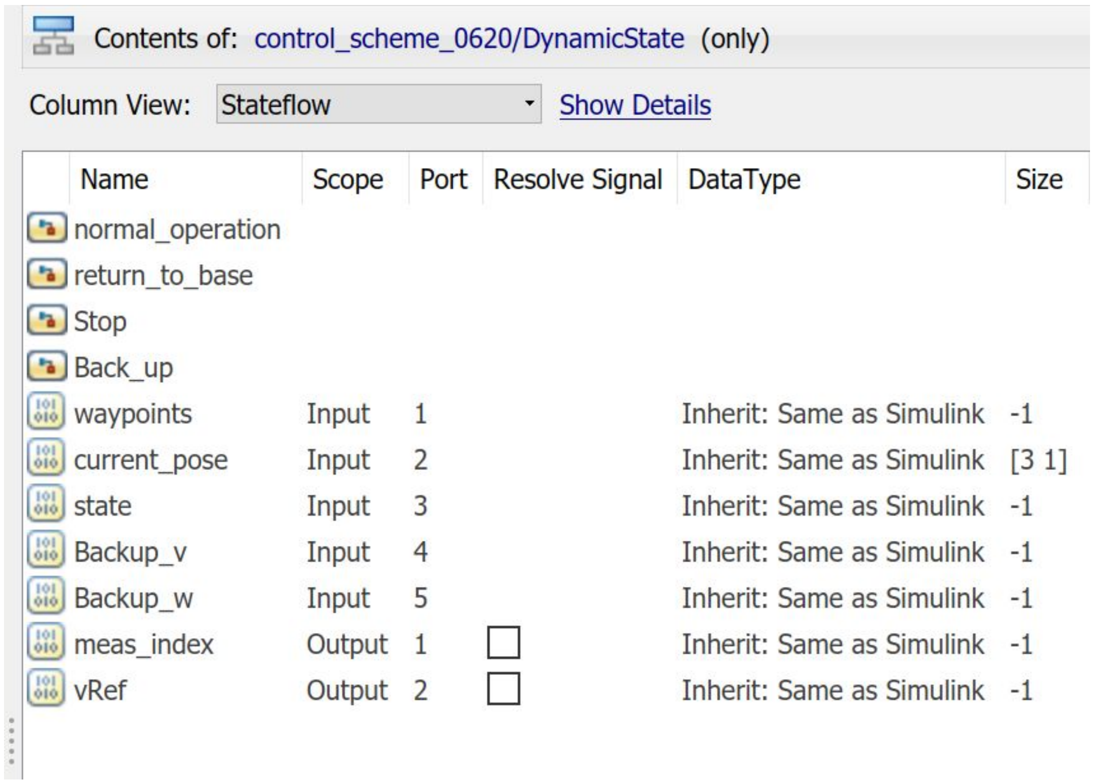
- 12V to 5V transformer connect teather and 5V breadboard
- Pi is powered from 5V breadboard

Conclusion

From the beginning of January to the end of December, this project enters stage 2 of its process. Based on the initial design blueprint, a self-constructed prototype is built by the project team. As a member of the control group, the main task was creating control algorithms, simulating the result in MATLAB Simulink, and writing scripts implemented in electronic devices. As shown in previous chapters. The simulation is pretty much finished with possible further development. The control architecture was constructed into 3 levels and interconnected by TCP/IP protocol and Serial communication. All control scripts built so far are limited to open loop control.

Notice all materials covered above indicate the work student has been focused on, which does not cover all work done by the whole team. For detailed information not covered in this project, please refer to other students' project reports.

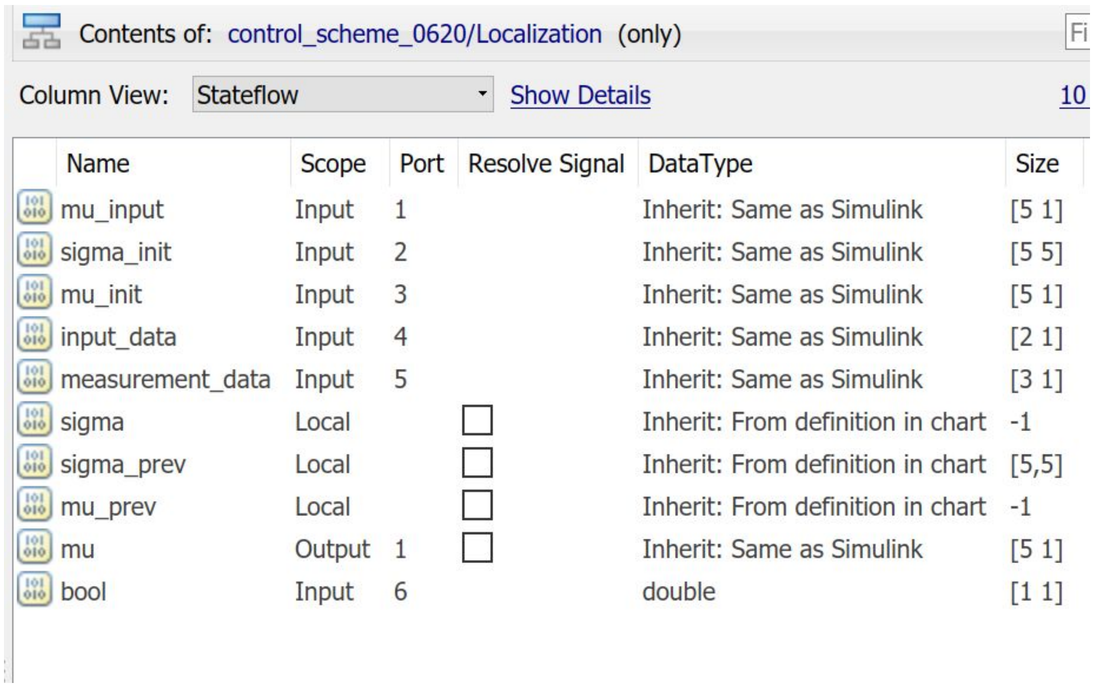
Appendix A MATLAB Simulink Input and Output



Contents of: control_scheme_0620/DynamicState (only)

Column View: Stateflow | Show Details

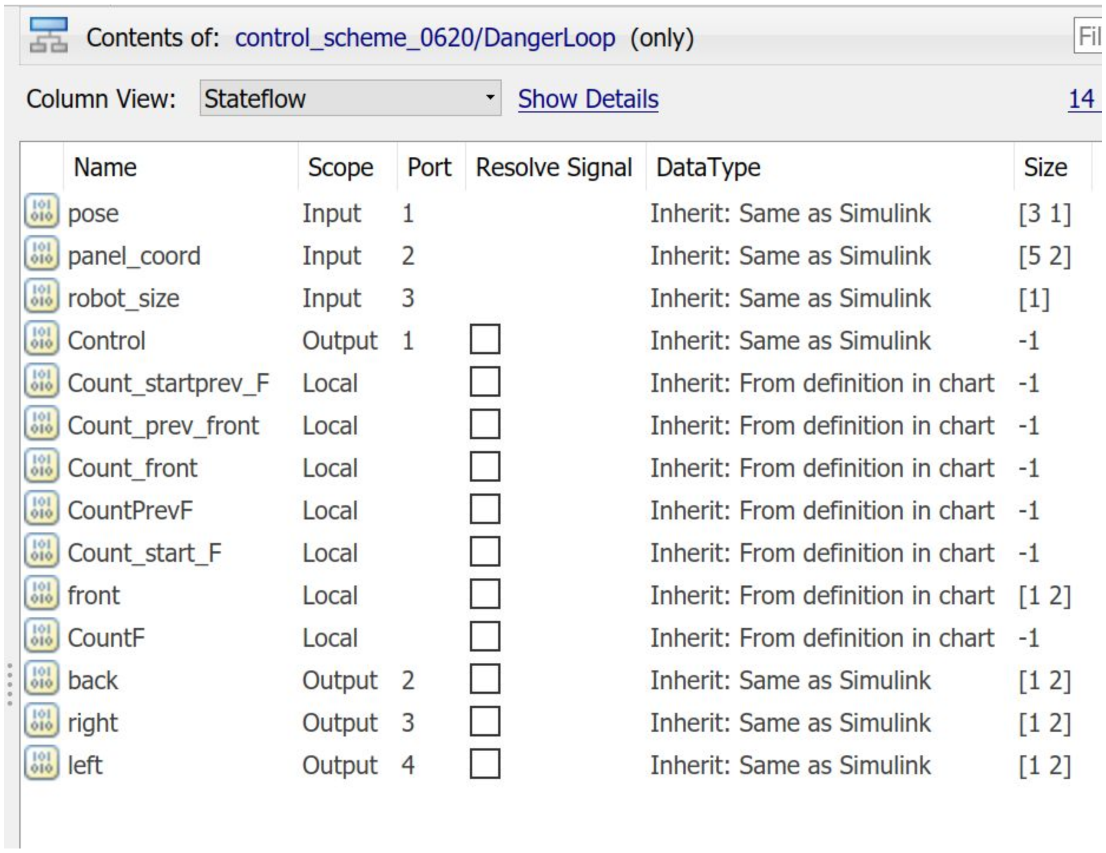
Name	Scope	Port	Resolve Signal	DataType	Size
normal_operation					
return_to_base					
Stop					
Back_up					
waypoints	Input	1		Inherit: Same as Simulink	-1
current_pose	Input	2		Inherit: Same as Simulink	[3 1]
state	Input	3		Inherit: Same as Simulink	-1
Backup_v	Input	4		Inherit: Same as Simulink	-1
Backup_w	Input	5		Inherit: Same as Simulink	-1
meas_index	Output	1	☐	Inherit: Same as Simulink	-1
vRef	Output	2	☐	Inherit: Same as Simulink	-1



Contents of: control_scheme_0620/Localization (only)

Column View: Stateflow | Show Details

Name	Scope	Port	Resolve Signal	DataType	Size
mu_input	Input	1		Inherit: Same as Simulink	[5 1]
sigma_init	Input	2		Inherit: Same as Simulink	[5 5]
mu_init	Input	3		Inherit: Same as Simulink	[5 1]
input_data	Input	4		Inherit: Same as Simulink	[2 1]
measurement_data	Input	5		Inherit: Same as Simulink	[3 1]
sigma	Local		☐	Inherit: From definition in chart	-1
sigma_prev	Local		☐	Inherit: From definition in chart	[5,5]
mu_prev	Local		☐	Inherit: From definition in chart	-1
mu	Output	1	☐	Inherit: Same as Simulink	[5 1]
bool	Input	6		double	[1 1]

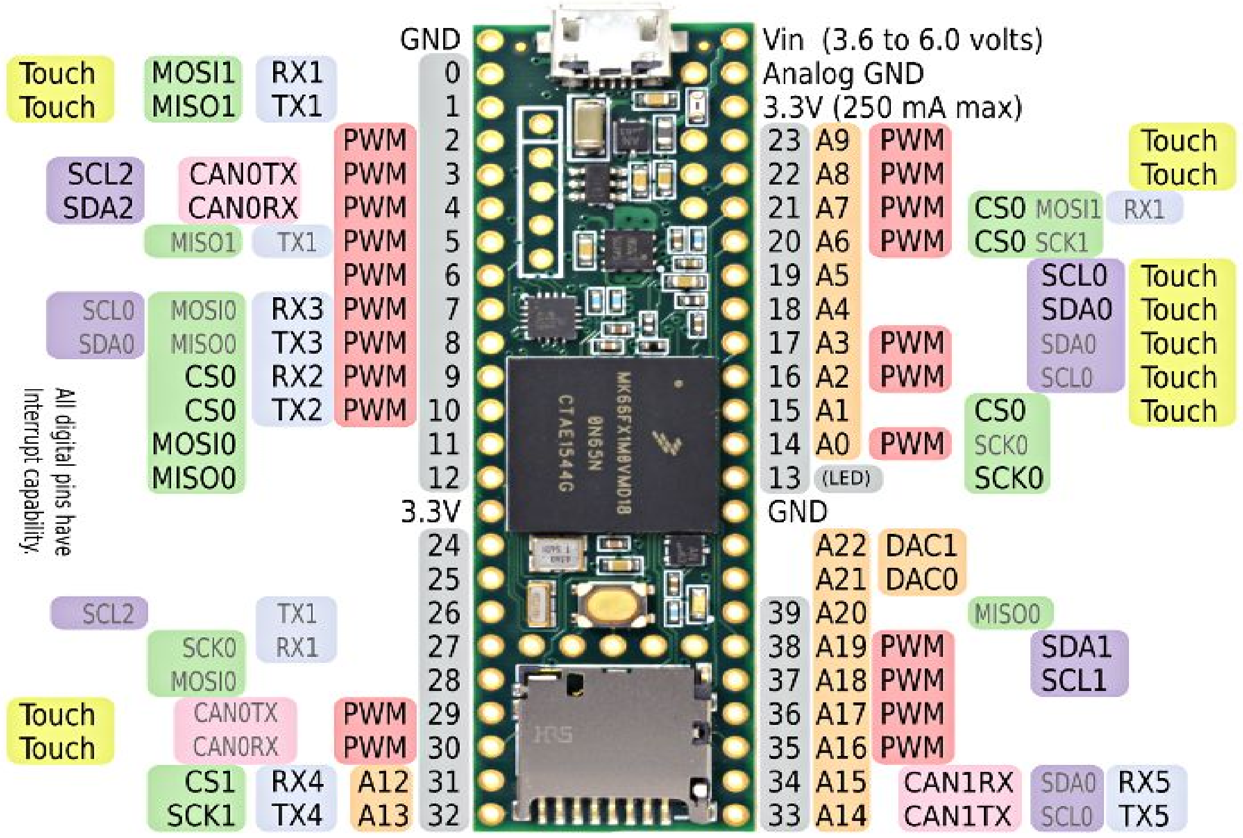


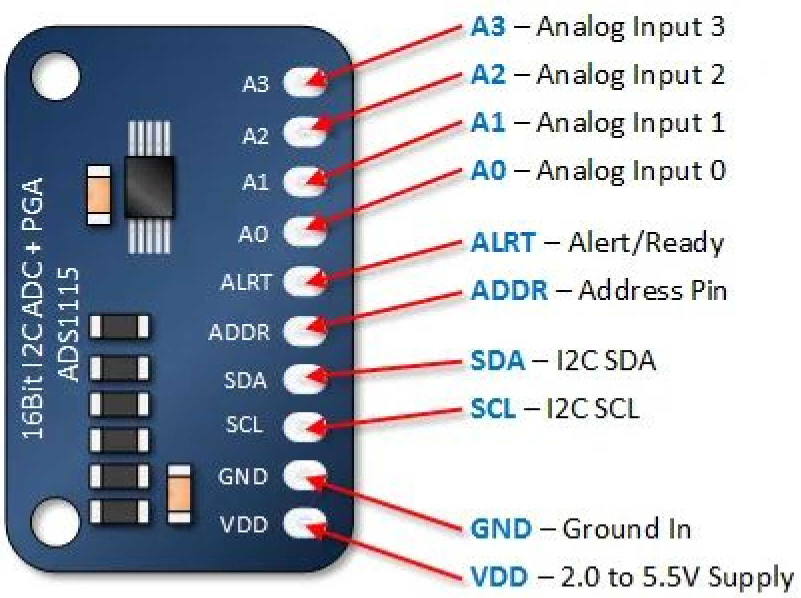
Contents of: control_scheme_0620/DangerLoop (only)

Column View: Stateflow Show Details 14

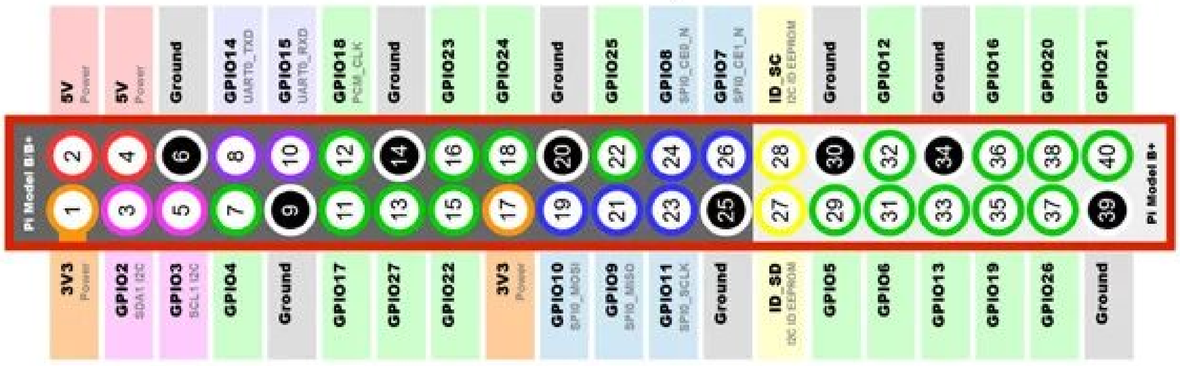
Name	Scope	Port	Resolve Signal	DataType	Size
pose	Input	1		Inherit: Same as Simulink	[3 1]
panel_coord	Input	2		Inherit: Same as Simulink	[5 2]
robot_size	Input	3		Inherit: Same as Simulink	[1]
Control	Output	1	☐	Inherit: Same as Simulink	-1
Count_startprev_F	Local		☐	Inherit: From definition in chart	-1
Count_prev_front	Local		☐	Inherit: From definition in chart	-1
Count_front	Local		☐	Inherit: From definition in chart	-1
CountPrevF	Local		☐	Inherit: From definition in chart	-1
Count_start_F	Local		☐	Inherit: From definition in chart	-1
front	Local		☐	Inherit: From definition in chart	[1 2]
CountF	Local		☐	Inherit: From definition in chart	-1
back	Output	2	☐	Inherit: Same as Simulink	[1 2]
right	Output	3	☐	Inherit: Same as Simulink	[1 2]
left	Output	4	☐	Inherit: Same as Simulink	[1 2]

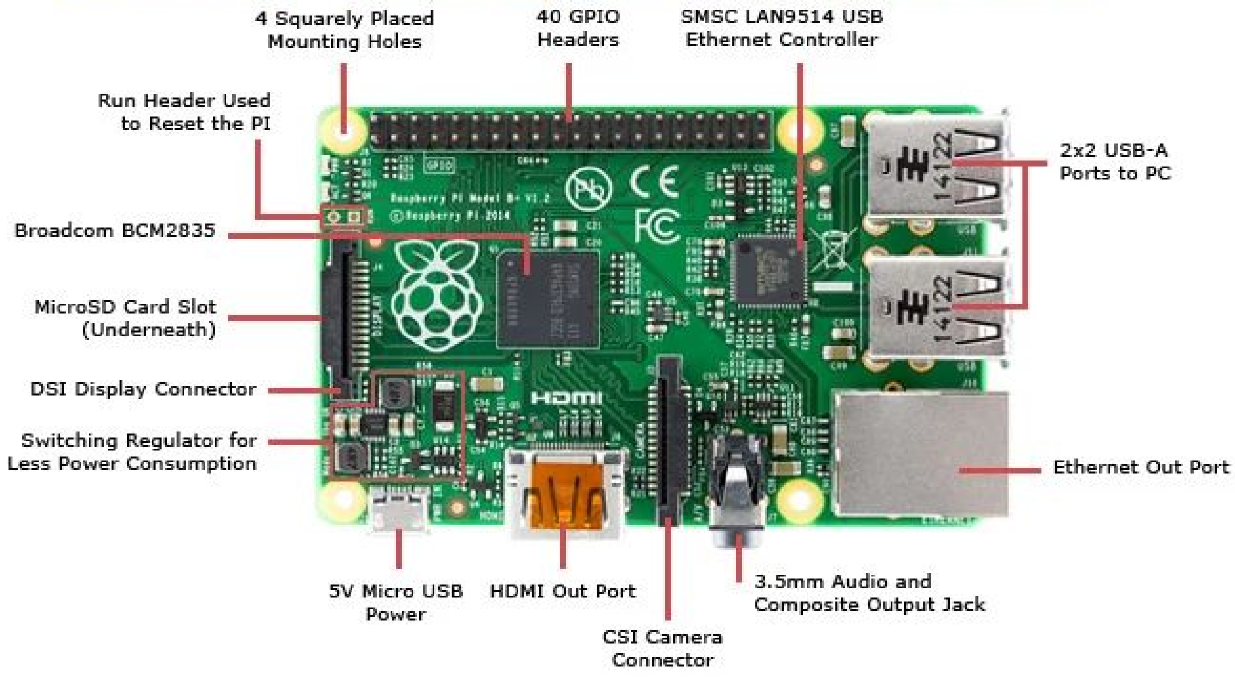
Appendix B Hardware Pin Layout





GPIO Pinout Diagram




4 Squarely Placed Mounting Holes
40 GPIO Headers
SMSC LAN9514 USB Ethernet Controller
Run Header Used to Reset the PI
2x2 USB-A Ports to PC
Broadcom BCM2835
MicroSD Card Slot (Underneath)
DSI Display Connector
Switching Regulator for Less Power Consumption
Ethernet Out Port
5V Micro USB Power
HDMI Out Port
CSI Camera Connector
3.5mm Audio and Composite Output Jack