

6-15-2017

Multi-bot Easy Control Hierarchy

Ryan Cooper

Santa Clara University, rcooper@scu.edu

Marton Demeter

Santa Clara University, mdemeter@scu.edu

Jonathan Ho

Santa Clara University, jho@scu.edu

Alan Nguyen

Santa Clara University, aknguyen@scu.edu

Follow this and additional works at: https://scholarcommons.scu.edu/cseng_senior



Part of the [Computer Engineering Commons](#)

Recommended Citation

Cooper, Ryan; Demeter, Marton; Ho, Jonathan; and Nguyen, Alan, "Multi-bot Easy Control Hierarchy" (2017). *Computer Engineering Senior Theses*. 75.

https://scholarcommons.scu.edu/cseng_senior/75

This Thesis is brought to you for free and open access by the Engineering Senior Theses at Scholar Commons. It has been accepted for inclusion in Computer Engineering Senior Theses by an authorized administrator of Scholar Commons. For more information, please contact rsroggin@scu.edu.

SANTA CLARA UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

DEPARTMENT OF MECHANICAL ENGINEERING

Date: June 14, 2017

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Ryan Cooper

Marton Demeter

Jonathan Ho

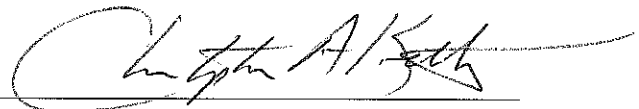
Alan Nguyen

ENTITLED

Multi-bot Easy Control Hierarchy

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING



Thesis Advisor



Department Chair

Multi-bot Easy Control Hierarchy

by

Ryan Cooper

Marton Demeter

Jonathan Ho

Alan Nguyen

Submitted in partial fulfillment of the requirements
for the degree of

Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 15, 2017

Multi-bot Easy Control Hierarchy

Ryan Cooper

Marton Demeter

Jonathan Ho

Alan Nguyen

Department of Computer Engineering

Santa Clara University

June 15, 2017

ABSTRACT

The goal of our project is to create a software architecture that makes it possible to easily control a multi-robot system, as well as seamlessly change control modes during operation. The different control schemes first include the ability to implement on-board and off-board controllers. Second, the commands can specify either actuator level, vehicle level, or fleet level behavior. Finally, motion can be specified by giving a waypoint and time constraint, a velocity and heading, or a throttle and angle. Our code is abstracted so that any type of robot - ranging from ones that use a differential drive set up, to three-wheeled holonomic platforms, to quadcopters - can be added to the system by simply writing drivers that interface with the hardware used and by implementing math packages that do the required calculations. Our team has successfully demonstrated piloting a single robots while switching between waypoint navigation and a joystick controller. In addition, we have demonstrated the synchronized control of two robots using joystick control. Future work includes implementing a more robust cluster control, including off-board functionality, and incorporating our architecture into different types of robots.

Acknowledgements

The Multi-bot Easy Control Hierarchy team would like to acknowledge these individuals and organizations for their continued help and support.

- Dr. Christopher Kitts
- Santa Clara University School of Engineering
- Santa Clara University's Robotics Systems Laboratory
- Scot Tomer
- Mike Rasay
- Anne Mahacek

Table of Contents

1	Introduction	1
1.1	Background	1
1.2	Project Objective	4
2	System Overview	5
2.1	Requirements	5
2.2	Use Cases	5
2.2.1	Use Case: Manual Control of Robot	5
2.2.2	Use Case: Set Waypoint for Robot	5
2.2.3	Use Case: Switch Between Manual and Waypoint Modes	8
2.3	Technologies Used	10
2.4	Physical Depiction	11
2.5	Flow Diagram	12
2.6	Ground Control System Architecture	13
2.7	Robot Software Architecture	14
2.8	Matlab Controller	15
2.9	Dataturbine	16
2.10	Risk Table	16
3	User Interface	18
3.1	Description	18
3.2	Front-end Design	18
3.2.1	Settings	18
3.2.2	Robots	21
3.2.3	Graphics	22
3.3	Back-end Design	23
3.3.1	Server	23
3.3.2	Logger	23
3.4	Testing and Results	25
3.4.1	Robots	25
3.4.2	Server	25
4	Robot Prototype	26
4.1	Overview	26
4.2	Requirements	27
4.3	Components of Prototype	28
5	Switching Architecture	31
5.1	Design	31
5.2	Implemented Controllers	31
5.2.1	Manual Control	31
5.2.2	Waypoint Navigation	32
5.2.3	Switching Implementation	32

6	Pozyx	35
6.1	Testing	35
6.2	Results	35
7	Societal Issues	40
8	Summary	43
8.1	Project Overview	43
8.2	Future Work	43
	References	45
A	Appendix	46
A.1	Annotated Bibliography	46
A.2	Literature Review	48
A.2.1	Proposed Objectives	48
A.2.2	Previous Work	48
A.3	Source Code	51
A.3.1	Robot	51
A.3.2	WebUI	86
A.3.3	Ground Control	117

List of Figures

1.1	Quadcopter	1
1.2	Joystick Controller	2
1.3	MARV	2
1.4	Eight Channel Controller	3
1.5	Pixhawk Flight Controller	3
2.1	Use Case for Multi-bot Easy Control Hierarchy	9
2.2	Physical Depiction	12
2.3	Data Flow Diagram	12
2.4	Architecture Diagram for Ground Control System	13
2.5	Architecture Diagram for Robot Software	14
2.6	Command Packet	15
3.1	Initial Design	19
3.2	Model Overlay to Configure Initial Robot Settings	19
3.3	Overview of the User Interface	20
3.4	Single and Cluster Configuration Options	20
3.5	Populated Controller and Robot Dropdowns	21
3.6	Waypoint Settings	21
3.7	Connected Robot	22
3.8	Disconnected Robot	22
3.9	Unselected Robot with Heading	22
3.10	Selected Robots with Center of Cluster	23
3.11	Waypoint	23
3.12	Data Flow Diagram of the User Interface	24
3.13	Log file	24
4.1	Robot Prototype	26
4.2	Edison Component Block Diagram	27
4.3	Raspberry Pi Component Block Diagram	28
5.1	Implemented Controller Switching Overview	34
6.1	Pozyx Position Distribution Before Filtering	38
6.2	Pozyx Position Distribution After Filtering	38
6.3	Pozyx Anchor Layout	39
6.4	Pozyx Waypoint Error	39

List of Tables

2.1	Ryan Cooper's AHP Input	6
2.2	Marton Demeter's AHP Input	6
2.3	Alan Nguyen's AHP Input	7
2.4	Jonathan Ho's AHP Input	7
2.5	Final Requirements Ranking	8
2.6	Requirements	8
2.7	Use Case: Manual Control of Robot	9
2.8	Use Case: Set Waypoint for Robot	10
2.9	Use Case: Switch Between Manual and Waypoint Modes	10
2.10	Software Technology Used	11
2.11	Hardware Technology Used	11
2.12	Risk Analysis Table	17
3.1	User Interface Throughput Test	25
4.1	Bill of Materials	29
6.1	Position Accuracy	36
6.2	Grid Offset	36
6.3	Waypoint Navigation Accuracy	37
6.4	Increasing Height of Pozyx Anchor	37

Chapter 1

Introduction

1.1 Background

As robots become more widespread and the variety of autonomous tasks increases, the ability to accommodate different control specifications becomes increasingly important. Robots nowadays have a wide variety of benefits and are used for a multitude of applications, which often present a benefit to society in various applications, such as manufacturing, search and rescue, and convenience [1]. As automation becomes more feasible, robots have the potential of changing the way industry operates. Because the variety of applications is endless, it is unlikely that these robots share the same actuators and operating platforms. Robots that travel on land, water, and air all have contrasting control specifications.



Figure 1.1: Quadcopter

We break these control specifications into three categories, which we call “the dimensions of control.” The first dimension is switching between different levels of control such as actuator-, vehicle-, and fleet-level. The Robotics Systems Lab (RSL) has previously developed a set of robotic kayaks that have the ability to switch between these levels [4]. Autonomous applications such as locating environmental pollution and escorting targets [5] are made possible when operating in fleet-level. Precision control of a single kayak

is achievable through vehicle- or actuator-level control by specifying joystick based velocity commands as shown in Figure 1.2.



Figure 1.2: Joystick Controller

Our second dimension is switching between the aforementioned velocity control and waypoint position control. A good example of a system capable of such switching is a commercial drone autopilot system called the 3DR PixHawk as seen in Figure 1.5. Utilizing a graphical user interface, the user has the ability to develop a flight plan, consisting of a series of waypoints, which will then be executed on the autopilot. The RSL has already utilized this piece of technology in the project MARV shown in Figure 1.3[2], which uses the PixHawk's waypointing capability to map the bottom of various lakes. When manual velocity control is needed, that functionality is also available on the PixHawk.



Figure 1.3: MARV

The third dimension of switching establishes where the calculations occur - onboard or offboard the robot. Most of the RSL projects focused on cluster control are built with an offboard controller designed in Simulink in Matlab. The RSL has previously built a controller that switches between these two levels of

computation [6].



Figure 1.4: Eight Channel Controller



Figure 1.5: Pixhawk Flight Controller

The RSL is currently developing advanced robotic missions that require the flexibility to switch from one control configuration to another. For one application, it may be beneficial to control the robot to move at a specific velocity in a specific direction. For another, it would be better to set up a path for the robot travel to autonomously, and it is always a safe idea to be able to quickly switch to manual control when doing an autonomous mission. The existing systems have no quick way to switch between different controllers, as the switching requires reprogramming, rebooting, and reinitializing the robot. This can be a tedious process that wastes a lot of time for researchers when they are testing new control algorithms. The current way of switching can be a potential risk for the field robots. If the robots are too distant to physically interact, such as underwater or in space, the robot could be damaged or abandoned. The RSL needs a better way to change how their robots are being controlled. It would provide them greater flexibility and a potentially greater degree of

control if the modes of control were easily switched. For example, in some cases, waypoint navigation is preferred over manual control. The RSL requires an architecture that allows for a seamless switch between control by synchronizing active controllers and rerouting controller communication. We will save the RSL their time and money when switching between configurations and platforms.

1.2 Project Objective

The objective of the project was to create a system that allows for a seamless switch between control configurations by synchronizing the activation of controllers and by rerouting controller communications. To achieve this, our team has integrated the technologies preferred by the Robotic Systems Lab, such as Matlab and Dataturbine, to handle robot controllers and communication within our architecture. We created a user interface that keeps track of deployed robots and allows for the user to configure which controller the robots obey. We also built three differential-drive robots utilizing the Intel Edison and Raspberry Pi microprocessors to show cross-platform compatibility and act as a testbed to prove our system in a real-world scenario. The result of our efforts produced a system that allows roboticists to configure their cross-platform robots and controllers on the fly using an intuitive interface.

Chapter 2

System Overview

2.1 Requirements

After speaking with our advisor, Dr. Christopher Kitts, our team gathered a list of requirements, which are presented in Table 2.6 for our project. We split the requirements into different categories such as functional, non-functional, and design constraint. These were further divided into critical and recommended. Because of the limited time scope of the project, our team needed to rank certain features to determine our priorities and allow us to allocate work effectively. The team input their opinions by comparing different features in an analytic hierarchy process table. These inputs are shown in Table 2.1, Table 2.2, Table 2.3, and Table 2.4. The inputs were compiled and the features were ranked based on the data in Table 2.5.

2.2 Use Cases

Figure 2.1 shows the use cases that define the tasks that a user can perform to achieve a certain goal within our system. The use cases have two primary components: a detailed description of each use case and a summary diagram. The use case descriptions include the preconditions and postconditions needed to achieve a certain goal, the type of user involved, what steps the user needs to perform to reach the use case goal, as well as exceptions that the system may encounter.

2.2.1 Use Case: Manual Control of Robot

Our project revolves around the ability to switch between controllers when operating a robot. The first type of controller that we discussed revolved around controlling the robot manually with a physical controller. This is shown in Table 2.7.

2.2.2 Use Case: Set Waypoint for Robot

As shown in Table 2.8, the next use case is waypoint navigation. Dr. Kitts suggested that we implement this algorithm first due to its relative simplicity. It utilizes our graphical user interface.

Table 2.1: Ryan Cooper's AHP Input

Ryan	GUI to display info	Different levels of control	Switching onboard-/offboard	Different kinds of platforms	Multiple control tools	Expandable and interoperable
GUI to display info	1	0.5	2	3	3	2
Different levels of control	2	1	2	3	3	2
Switching onboard-/offboard	0.5	0.5	1	2	3	2
Different kinds of platforms	0.3333	0.3333	0.5	1	2	0.5
Multiple control tools	0.3333	0.3333	0.3333	0.5	1	0.5
Expandable and interoperable	0.5	0.5	0.5	2	2	1

Table 2.2: Marton Demeter's AHP Input

Marton	GUI to display info	Different levels of control	Switching onboard-/offboard	Different kinds of platforms	Multiple control tools	Expandable and interoperable
GUI to display info	1	0.5	3	2	3	3
Different levels of control	2	1	2	2	3	2
Switching onboard-/offboard	0.3333	0.5	1	2	2	2
Different kinds of platforms	0.5	0.5	0.5	1	2	2
Multiple control tools	0.3333	0.3333	0.5	0.5	1	0.5
Expandable and interoperable	0.3333	0.5	0.5	0.5	2	1

Table 2.3: Alan Nguyen's AHP Input

Alan	GUI to display info	Different levels of control	Switching onboard-/offboard	Different kinds of platforms	Multiple control tools	Expandable and interoperable
GUI to display info	1	0.5	0.5	0.5	0.5	0.5
Different levels of control	2	1	2	1.5	3	2
Switching onboard-/offboard	2	0.5	1	2	2	1
Different kinds of platforms	2	0.666	0.5	1	2	1
Multiple control tools	2	0.3333	0.5	0.5	1	0.5
Expandable and interoperable	2	0.5	1	1	2	1

Table 2.4: Jonathan Ho's AHP Input

Jonathan	GUI to display info	Different levels of control	Switching onboard-/offboard	Different kinds of platforms	Multiple control tools	Expandable and interoperable
GUI to display info	1	0.5	2	2	3	3
Different levels of control	2	1	3	2	3	3
Switching onboard-/offboard	0.5	0.3333	1	1	2	2
Different kinds of platforms	0.5	0.5	1	1	2	2
Multiple control tools	0.3333	0.3333	0.5	0.5	1	0.5
Expandable and interoperable	0.3333	0.3333	0.5	0.5	2	1

Table 2.5: Final Requirements Ranking

Final Rankings	
GUI to display info	0.2038619662
Different levels of control	0.294287738
Switching onboard/offboard	0.1665858258
Different kinds of robots and domains	0.1328613009
Multiple control tools	0.07895176963
Expandable and interoperable	0.1234513994

Table 2.6: Requirements

Functional Requirements	
Critical	Have a GUI to show information on robots such as speed, distance, location, and mode
	Able to support robots of different specifications, domains, or mobility
	Convert between different ways of controlling the system onboard or offboard control
Recommended	Accommodate specifying velocity commands vs position commands
	Have multiple ways of controlling all connected robots such as controller, joystick, or computer program
Non-Functional Requirements	
Critical	An expandable, interoperable architecture which has guidelines and documentation to allow users to add robots
	Interoperability between Raspberry Pi, Intel Edison, and other processors with Linux
Recommended	Achieve least amount of latency and quick response times
Design Constraints	
	Must integrate with Matlab, Pozyx, and Datarbine

2.2.3 Use Case: Switch Between Manual and Waypoint Modes

The user should be able to switch between the controllers that we have implemented in our system. We developed this use case to allow the system to switch between manual and waypoint modes. It is important

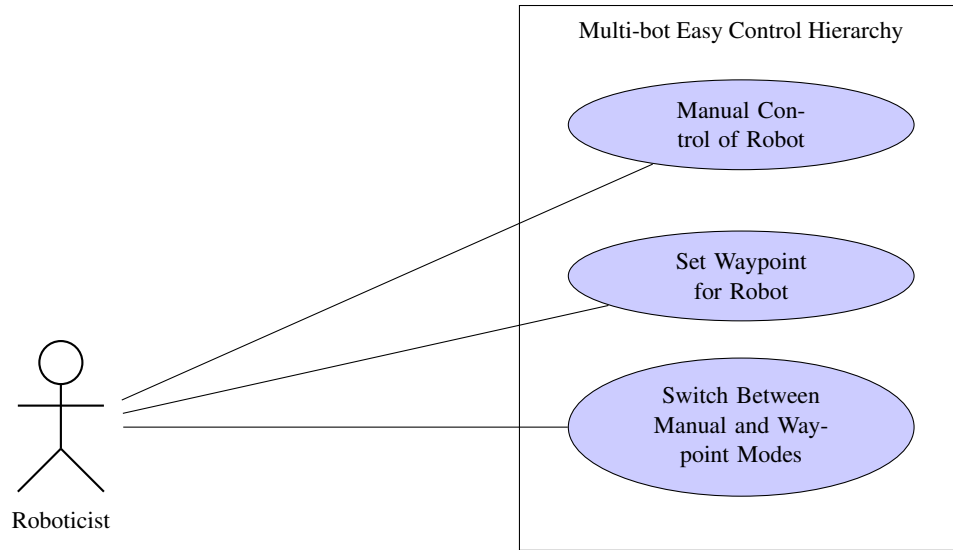


Figure 2.1: Use Case for Multi-bot Easy Control Hierarchy

for the user to have to ability to change control configurations on the fly in order to effectively utilize the switching architecture.

Table 2.7: Use Case: Manual Control of Robot

Use Case Name	Manual Control of Robot
Goal	Be able to control the robot with a controller.
Actor(s)	Robotician
Precondition(s)	Setting is on manual mode.
Postconditions(s)	Robot moves according to the input depending on the controller configuration.
Step(s)	Input direction and velocity using controller.
Exception(s)	Robot does not follow directions from controller

Table 2.8: Use Case: Set Waypoint for Robot

Use Case Name	Set waypoint for robot
Goal	Robot moves to the desired location of the waypoint specified by the UI
Actor(s)	Roboticist
Precondition(s)	Setting is on waypoint mode
	Location is reachable by robot
Postconditions(s)	Robot arrives at the specified location
Step(s)	Input desired location on the communication system
Exception(s)	Robot's path is blocked by obstacle

Table 2.9: Use Case: Switch Between Manual and Waypoint Modes

Use Case Name	Switch between manual and waypoint modes
Goal	Be able to switch between manual control and waypoint control
Actor(s)	Roboticist
Precondition(s)	Function is assigned in the controller configuration
	Have different types of movement information depending on the current setting
Postconditions(s)	Robot switches between the modes
Step(s)	Click assigned button on user interface
Exception(s)	N/A

2.3 Technologies Used

Table 2.10 shows the various software technologies that were used to create the system. Table 2.11 shows the hardware technologies present on our prototype robot.

Table 2.10: Software Technology Used

GitHub	Version Control
Matlab	Controllers
Java	Dataturbine, Jmatlab
Python	Robot Software
HTML	UI Layout
CSS	UI Styling
Javascript	UI Logic
Node.js	UI Backend

Table 2.11: Hardware Technology Used

Raspberry Pi Zero	Robot Controllers
Intel Edison	Robot Controllers
Pozyx UWB	Indoor Positioning System

2.4 Physical Depiction

Figure 2.2 depicts what a typical use of Multi-bot Easy Control Hierarchy (MECH) would look like. Figure 2.2 shows the different methods of control in our system. Navigation mode A is manual drive with joystick control. Mode B is waypoint navigation by specifying points through the user interface. Mode C is waypoint navigation with cluster control. The user, depicted by the stick figure, has configured MECH and is monitoring its operation. The computer has the ground control version of MECH that allows for the user to configure the robots with a GUI. The computer is also connected to a communication device that allows it to connect to the robots. This communication device receives information about the positions of the robots and routes the control information to the robots. The computer runs a control algorithm in Matlab or is connected to a controller to control the robots. Each of the robots can be controlled by their own unique control algorithm, or one algorithm can make a group of robots move in a formation.

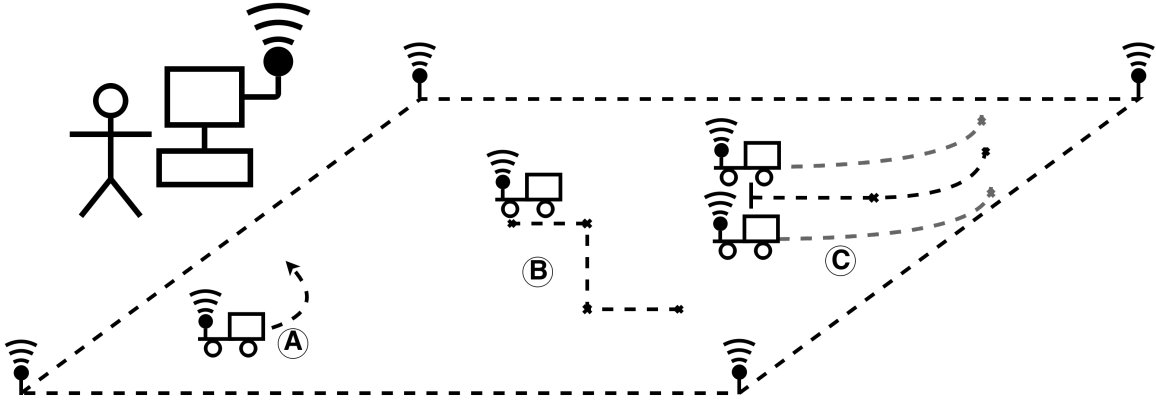


Figure 2.2: Physical Depiction

2.5 Flow Diagram

Figure 2.3 depicts how data travels through MECH, and how onboard calculations mode differs from offboard mode. Data is generated at the ground control through the user interface. This information makes its way to the controller through several communication layers. The boxes with disconnected sinusoidal waves show that information is transmitted wirelessly in that mode. The onboard mode also depicts the controller to be part of the robot. This is different from offboard mode, where the controller is part of ground control. When in offboard mode, data is sent to the robot for the actuators through the communication layers, and sensor data is transmitted back to ground control the same way.

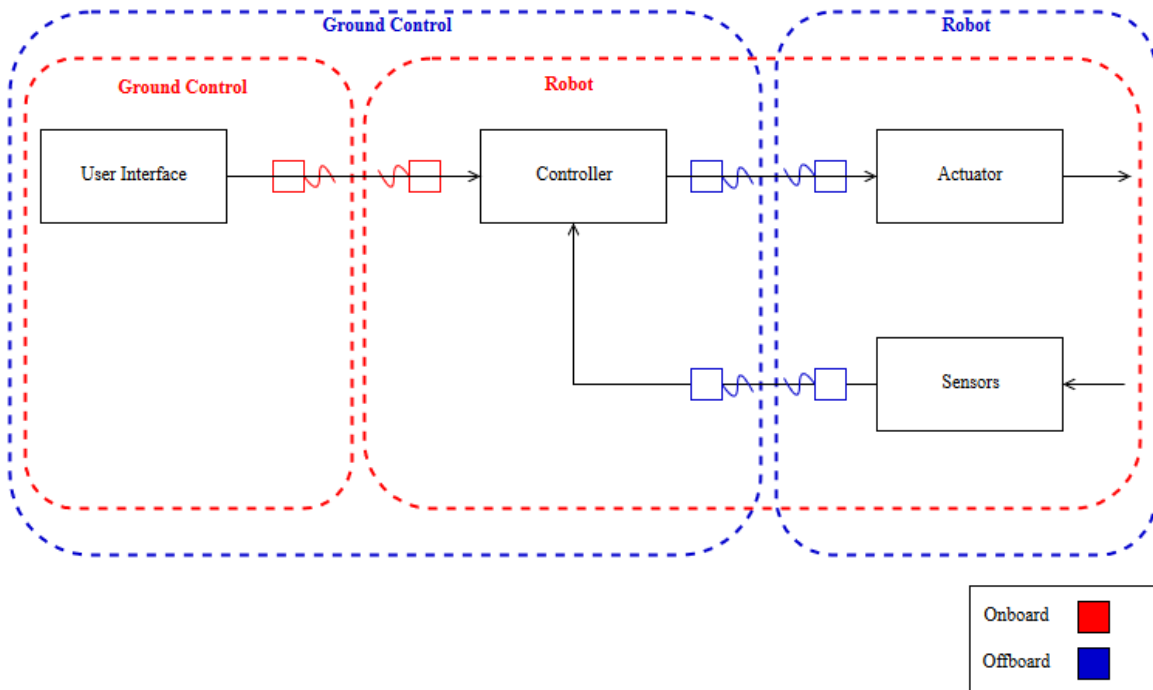


Figure 2.3: Data Flow Diagram

2.6 Ground Control System Architecture

Figure 2.4 depicts what MECH looks like on the ground control server. The controller is generated from Simulink and fed to Matlab. Matlab handles the parsing of the control data and sends it to JMatlab. The controller could be a joystick of some kind or an algorithm made by researchers. The only control scheme we were able to implement was a joystick controller that converted the raw joystick values into motor level commands for the robot. Even though we only implemented one controller, our software architecture is able to support different Simulink controllers.

The control data is sent to Dataturbine via JMatlab. Since Matlab is a single threaded process, it could not natively support asynchronous communication. JMatlab is necessary for asynchronous communication for Matlab and Dataturbine. Sensor data from the robot is sent to Matlab via Dataturbine. The sensor data is published to a Dataturbine channel that follows the naming convention of "robot_X_source", where X is the robot number. The sensor data, such as position and heading, allows a researcher to create closed loop control algorithms to operate the robot.

When sensor data is sent through Dataturbine, the user interface picks up on it and displays it to the user. The UI also allows the user to remotely configure the robots to listen to a connected controller. This reconfiguration is done on a specific UI channel called "_UI_source". Every robot and controller subscribes to this channel for when the UI sends out a reconfiguration request. A reconfiguration request can ask a specific controller or robot to reconfigure its sinks to listen to a specific channel.

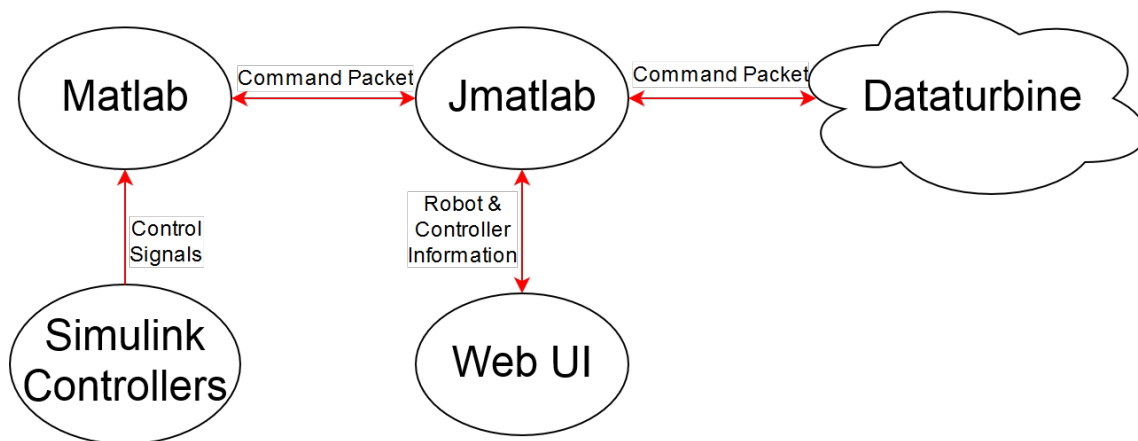


Figure 2.4: Architecture Diagram for Ground Control System

2.7 Robot Software Architecture

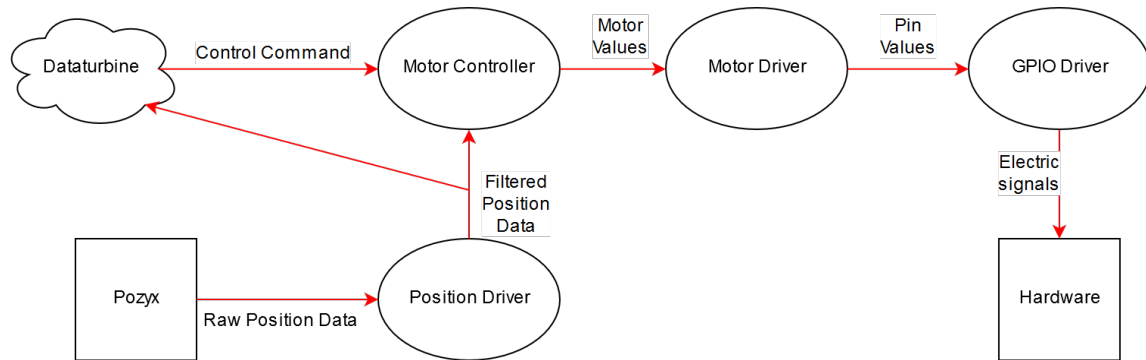


Figure 2.5: Architecture Diagram for Robot Software

Figure 2.5 depicts the high level software architecture that handles the data flow of MECH on the robot. The robot runs a Linux operating system that has Python 2.7 and Java 8 installed. A Java process handles the communication with Dataturbine and reconfiguration of which controller the robot is listening to when the UI requests the robot to reconfigure. Java also communicates with the Python code via a unix domain socket. Python handles all of the data parsing and control of the hardware. Each ellipse in Figure 2.5 represents a python process that runs on the robot. The python process are started by DDStarter.py. DDStarter.py and the Java process on the robot are started with StartRobot.sh

Dataturbine sends control data to the motor controller which determines how to handle the data. The motor controller parses the command packet to determine what control scheme the packet requests the robot be controlled in and then uses the rest of the packet accordingly.

The command packet consists of four fields as shown in Figure 2.6. Src is where the packet came from. The Dst field is where the packet is intended to be decoded. The robot checks this field to see if the packet was for it, and then it checks the Src field to see if the packet came from the UI or from the controller it is listening to. If the packet was not for the robot or is not from UI or the controller the robot is listening to, it discards the packet. This reduces the amount of time wasted on packets that aren't for the robot. The control field is used to determine how to interpret the data field to control the robot. The control field is discussed in greater detail in Section 5.2.3. Depending on the control field, the data field may have further divisions that signify the left motor and right motor values, the steering and throttle values, the velocity and heading values, or a waypoint. The size of each field and the packet as a whole was optimized to transmit the minimum required information to have the robot execute the correct action. To allow the software to be used reliably with communication infrastructures that have a low bandwidth, the packet size was minimized.

The motor controller has support for vehicle-level control specified by a velocity and heading, or throt-

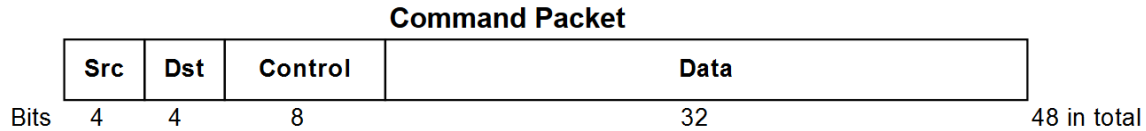


Figure 2.6: Command Packet

tle and steering values. It also supports actuator level commands which are represented by raw PWM values for the motors, and waypoint navigation commands which are represented as a list of Euclidean coordinate waypoints.

Onboard controllers can operate on a set of waypoints from the waypoint navigator or vehicle-level commands, whereas off-board controllers can be implemented to handle either vehicle-level commands or actuator-level commands. Onboard waypoint navigation is achieved by using the position data from the position driver to create a closed-loop control algorithm.

The position driver filters the raw Pozyx data and sends it to Dataturbine and the motor controller. The motor driver and the GPIO driver are abstraction layers that facilitate the code to run on multiple platforms (the Intel Edison and Raspberry Pi for this project). These abstraction layers convert generalized control signals into the specific pin values required to operate the robot as desired. These specific pin configurations and specifications are defined in a configuration file. It is expected that a user will update the configuration file with their pin configurations when they want to use our software. The hardware shown in Figure 2.5 is an L298 dual H-bridge motor driver.

2.8 Matlab Controller

Matlab is one of the most used software suites in the field of control systems. As such, it is the preferred language used at Santa Clara University’s Robotic Systems Lab. The RSL students, staff, and professors have created over fifteen years worth of control algorithms in Matlab.

Matlab has a block diagram environment for facilitating the development and simulation of control algorithms called Simulink. Simulink is able to interface with various types of controllers such as a joystick. The Robotic Systems Lab also has been developing control algorithms in Simulink for many years. Our project hopes to integrate the technology that the Robotic Systems Lab deems familiar.

Using JMatlab, an interface between Java and Matlab, we connected the Matlab component of our system with Dataturbine. This allowed for asynchronous communication with the Matlab component and the rest of our system.

When testing the Matlab / Simulink controller, we found that the latency between the controller and

the robot executing the command was high. The commands coming from the controller were flooding the robot. By reducing the message rate with the introduction of a 100ms delay, we were able to resolve the latency issue.

2.9 Dataturbine

Dataturbine is a publisher/subscriber software architecture, and for this project robots, controllers, and the UI are all publishers and subscribers on Dataturbine. Dataturbine can be polled for information about who is on the turbine. The UI uses this information to reconfigure the robots and controllers. We integrated with Dataturbine by communicating with it over a TCP connection.

Dataturbine is the communication manager for our project. It was a design constraint for our project because the SCU RSL has been using it in projects for the last ten years and wanted to continue using it. It also kept track of what was connected to the network and allowed for us to change the routing of data on the fly.

Software integrated with Dataturbine and made use of its utilities to get information to the UI and to reconfigure controllers and robots. This was done using TCP connections for inter-process communication between processes that communicated with Dataturbine.

2.10 Risk Table

In order to gain awareness of some problems that might arise during development, we created Table 2.12, to analyze such risks. It tells us the consequences of each risk as well as its impacts. The impacts are calculated by multiplying the probability and severity of each event. Probability is represented by “P” and is on a scale of 0 to 1. Severity and impact is represented by “S” and “I” respectively. Severity and impact are on a scale of 1 to 10. We also created mitigation strategies to help us avoid each event.

Table 2.12: Risk Analysis Table

Name of Risk	Consequences	Probability [0-1]	Severity [1-10]	Impact [1-10]	Mitigation Strategies
Destruction of equipment	Rebuild/reorder robot	0.6	7	4.2	Be careful with robot. Do not exceed limits
Conflicting Schedules	Cannot meet to work together to meet deadlines	1	4	4	Schedule meetings well in advance. Be open to inconvenient times. Schedule meetings with only part of the group present
Illness	fall behind, reassign tasks	0.7	4	2.8	Maintain proper health. Sleep well
Too many features	Necessary features are overlooked	0.3	8	2.4	Prioritize features in order of importance. Limit the number of features
Miscommunication	Improper implementations, missed expectations	0.6	3	1.8	Use descriptive words. make sure both parties understand what is expected before they part ways
Run out of time	Cut features	0.2	8	1.6	Follow development timeline. Ask for help before deadlines

Chapter 3

User Interface

3.1 Description

This project required an interface to support all of the functionality of the architecture, while allowing the user to easily interact with, and configure various settings for the system. The solution was a graphical user interface (GUI) implemented as a website written for Google Chrome utilizing HTML, CSS, and Javascript. No frameworks were used, all the code was written by hand from the ground up.

3.2 Front-end Design

As the system evolved, and as more features got added, the design of the GUI changed substantially to better accommodate these new elements.

Our initial design shown in Figure 3.1, consisted of a total of three panels: one containing a list of the robots, one for a visual display, and one for the system settings. At that time, our robots had to be added manually to the system, so a model overlay was used to configure the initial settings of the robots as is shown in Figure 3.2. The design later was updated as the architecture evolved to utilize Dataturbine to automatically add robots to the system.

After the controllers and waypoint navigation were added to the architecture, the design changed yet again, into its final version as shown in Figure 3.3.

3.2.1 Settings

Configuration

The settings panel was split into three parts to account for the changes. The leftmost area became the configuration menu, where the user is able to opt to either group selected robots into clusters, or make them single as shown in Figure 3.4. The state of the robots is saved until they are disconnected.

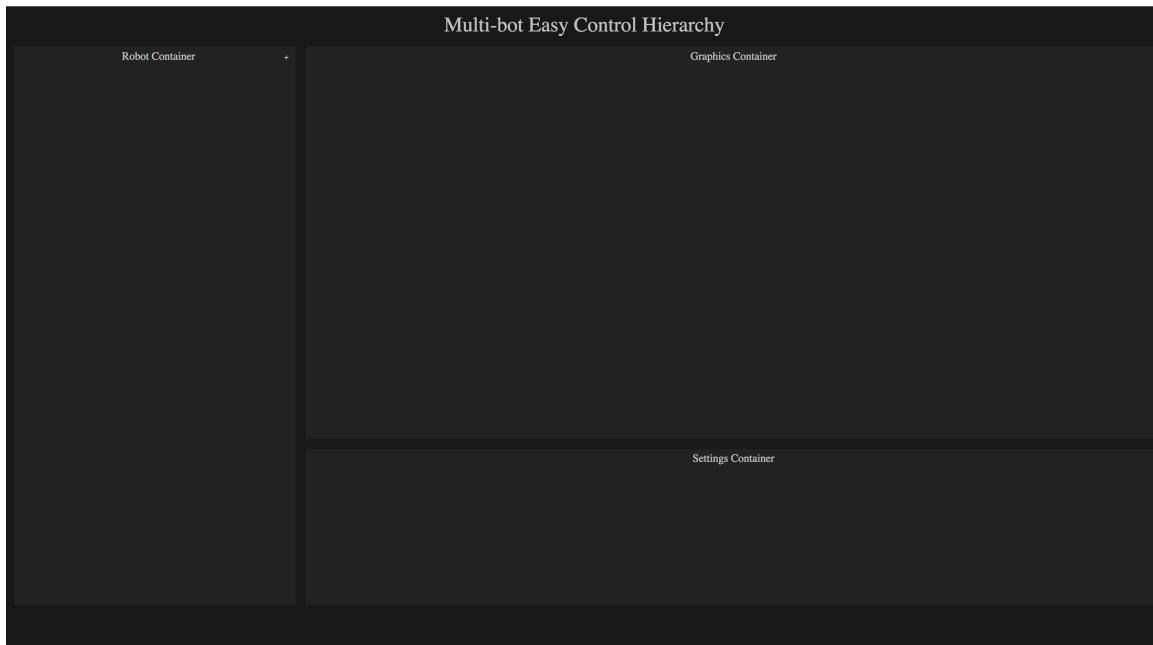


Figure 3.1: Initial Design

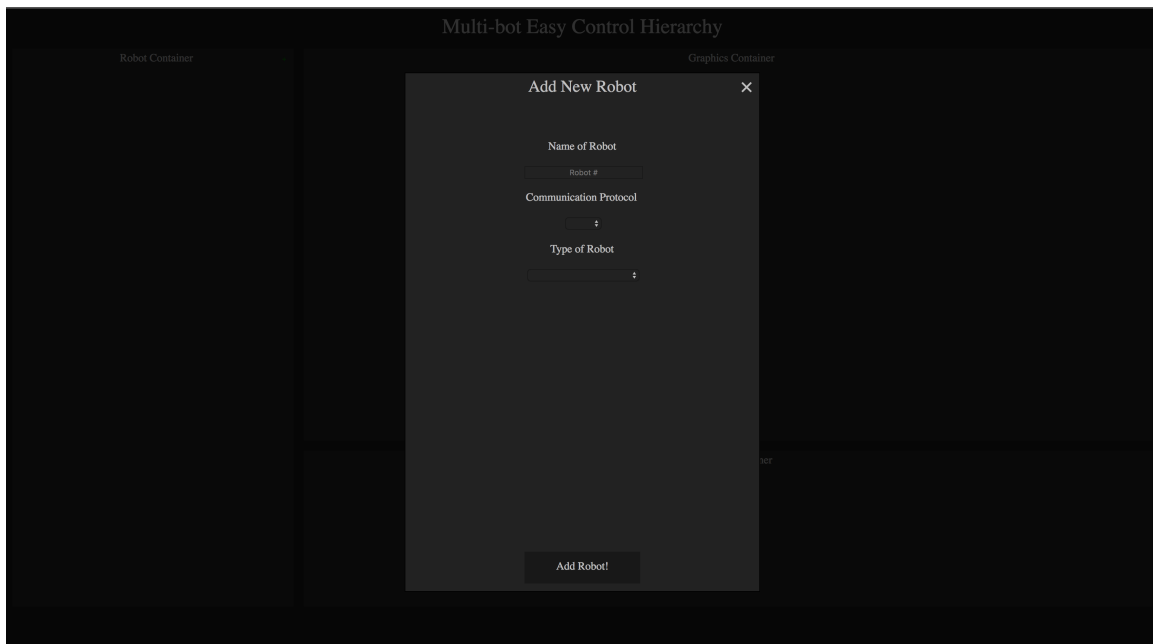


Figure 3.2: Model Overlay to Configure Initial Robot Settings

Controllers

The middle area became the Controllers menu. The request button in the top right corner sends a packet to JMatlab requesting all available Dataturbine channel information:

requestDTConfig()

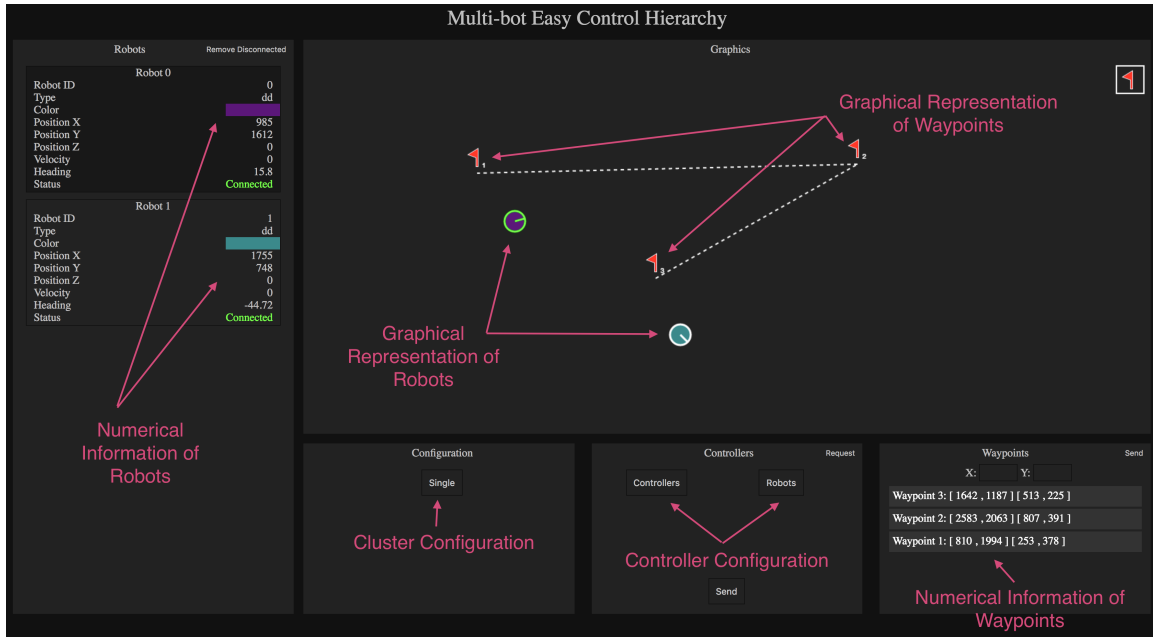


Figure 3.3: Overview of the User Interface



Figure 3.4: Single and Cluster Configuration Options

The return packet is the following ASCII string:

DTConfig: robot1_name type, controller1_name type, robot2_name type, controller2_name type;

The javascript code listening to the websocket communication filters out the necessary information and populates the dropdowns accordingly as shown in Figure 3.5. The robots are then assignable to controllers as the user wishes.

Currently waypoint and Simulink joystick controllers are implemented. Waypoint information originates from the GUI itself, and is set the default controller for any new robot. All the active joysticks' information is received from Dataturbine through JMatlab, and shows up visually in the dropdown, ready to be paired with the robots.

Waypoints

The rightmost panel is the home of waypoint coordinates. It has three notable features: the two text boxes to specify x and y coordinates, the coordinates list, and the send button as shown in Figure 3.6. The text boxes offer an alternative to specifying waypoints through the graphical area, and allow for the selection

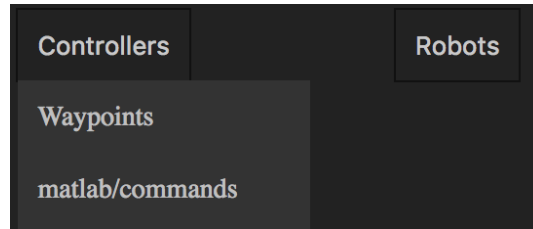


Figure 3.5: Populated Controller and Robot Dropdowns

of precise coordinates. Whenever a waypoint coordinate is added on either the graphical area, or through the text boxes, it shows up in the list below the text boxes. It displays the most recently added waypoint first, so that the user is able to keep track of the waypoints they added. Finally, the send button sends all the currently active waypoints to all the selected robots. The button only appears when there is at least one robot selected that has its controller type set to waypoints.

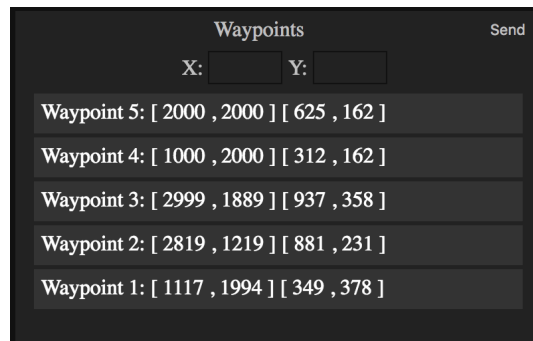


Figure 3.6: Waypoint Settings

3.2.2 Robots

The panel for the robots displays numeric information received through Dataturbine, from the robots. The information we're currently providing the user with includes: Name, ID, Type, Color, Position X, Position Y, Position Z, Velocity, Heading, Status, as shown in Figure 3.7. Each one of the robots has its own unique card with the aforementioned information. The card values update whenever a new packet is received through the websocket that corresponds to the appropriate robot. If no packet is received for a certain amount of time, the status of the robot changes to disconnected automatically as shown in figure 3.8. A javascript timer function keeps track of the last received packet for each one of the robots.

The robots are also selectable through these cards. By holding down the SHIFT key, the user is able to select more than one.

Robot 0		
Robot ID		0
Type		dd
Color		
Position X		1670
Position Y		2121
Position Z		0
Velocity		0
Heading		29.64
Status		Connected

Figure 3.7: Connected Robot

Robot 0		
Robot ID		0
Type		dd
Color		
Position X		461
Position Y		2584
Position Z		0
Velocity		0
Heading		44.32
Status		Disconnected

Figure 3.8: Disconnected Robot

3.2.3 Graphics

Robots

The graphical area consists of an HTML5 canvas. The canvas can be used to generate graphical primitives, such as circles and squares. It is also possible to create custom shapes. The robots are represented as little circles, with a line through it to mark the heading as can be seen in Figure 3.9. A robot can be selected either through the robot panel as mentioned earlier or by clicking on the little circles representing them in the graphical area. Whenever a robot is selected, its outline changes color from white to green to indicate the selection. If multiple robots selected, their outline changes, and a dashed line connects them all to indicate a cluster, as shown in Figure 3.10. The center of a cluster is marked with a smaller yellow circle.



Figure 3.9: Unselected Robot with Heading

Waypoints

Waypoints can also be specified in the graphical area. If the user clicks the waypoint flag located in the top right corner, then waypoint selection will activate, and a little flag will be displayed above the cursor of the user. Wherever a user clicks while waypoint selection is active, a waypoint coordinate will be marked

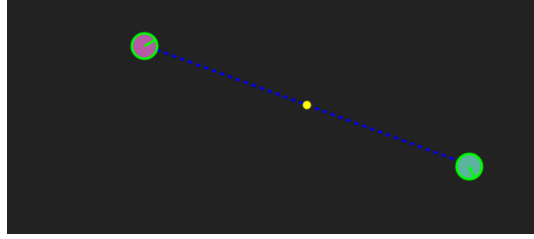


Figure 3.10: Selected Robots with Center of Cluster

with a flag in that spot as shown in Figure 3.11. Waypoint selection can also be activated by pressing the key "F". Waypoint selection can be deactivated by clicking the waypoint flag in the top right corner or by pressing the escape key.

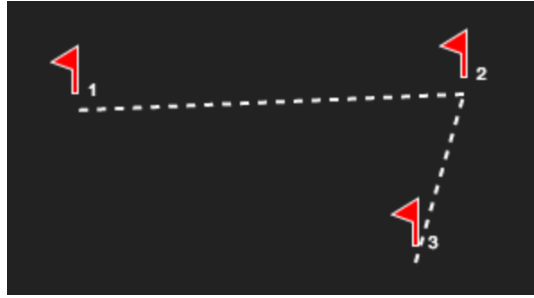


Figure 3.11: Waypoint

3.3 Back-end Design

3.3.1 Server

The backend of the user interface consists of a server written in Node.js. Node.js is the perfect programming language for such a server, as it allows for bidirectional and asynchronous flow of information. The purpose of the back-end server is to relay information between the website and JMatlab. The server communicates with the website through a websocket. A websocket library called "ws" was used. The connection to JMatlab is through TCP, which is natively supported by Node.js, so no external libraries were utilized. The information forwarding and flow is shown in Figure 3.12.

3.3.2 Logger

The server utilizes a logger to keep track of information about the server. The logger is also written in Node.js, and has both synchronous and asynchronous versions. The synchronous version of the logger has to be used to log information when the server is shutting down, otherwise the program would quit before the data is saved, however it is used all throughout. The kind of information saved is presented in the list below.

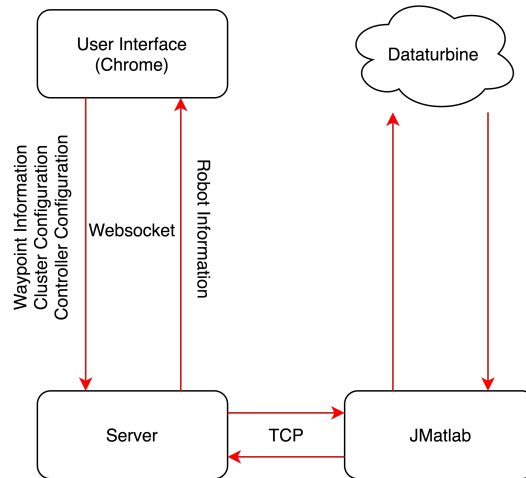


Figure 3.12: Data Flow Diagram of the User Interface

- Time and date of when the back-end server started
- Time and date of when a TCP client connected
- Time and date of when a TCP client disconnected
- Time and date of when a websocket client connected
- Time and date of when a websocket client disconnected
- Time and date of when the back-end server shut down

It is also possible to save data of varying degrees with the logger. It supports four levels:

- Debug
- Info
- Warning
- Error

```

Mon Jun 05 2017 10:04:14 - INFO: Server started.
Mon Jun 05 2017 10:04:18 - INFO: TCP client has connected.
Mon Jun 05 2017 10:04:20 - INFO: Websocket client has connected.
Mon Jun 05 2017 10:43:18 - INFO: Websocket client has disconnected.
Mon Jun 05 2017 10:43:49 - INFO: TCP client has disconnected.
Mon Jun 05 2017 10:44:36 - INFO: Server shut down.
  
```

Figure 3.13: Log file

The level of data saved is also recorded in the log file, along with the time and date, as shown in Figure 3.13.

3.4 Testing and Results

3.4.1 Robots

The GUI had to be tested when there weren't any robots attached, so that consistent progress could be made. A program was developed to imitate information provided by robots, as well as respond to the information given by the website. The program connects to the TCP connection that normally JMatlab connects to. It has the ability to randomly generate heading for each robot, and it stores their current and future positions. It can also store waypoint information given to a robot, which is then able to complete the course if given the instruction. It has been a significant contribution to the development of the website.

3.4.2 Server

The bandwidth and packet handling ability of the server also had to be benchmarked initially. A program was written to provide a varying throughput of data to test the capabilities of the server. Some of the packets were sent with such little interval that the server received the packets merged. To combat such situations, packets are parsed, and only the first part of a merged packet is used. The tests below show on average how many packets are usable out of ten thousand. It should be kept in mind that the program sends packets very quickly, and that the real robot only sends its information about 3 times a second, as opposed to 40 times a second. As can be seen from Figure 3.1, about 88% of the packets are usable from the test

Table 3.1: User Interface Throughput Test

Test Number	Usable Packets	Total Packets
1	8957	10000
2	8796	10000
3	8746	10000

program.

The throughput of the test program is around 2.1025 bytes / millisecond, which is achieved by sending an average of 52 byte strings every 25 milliseconds. The actual robot sends an approximately 52 byte string every 333 milliseconds, which is a throughput of 0.1562 bytes / millisecond. Due to maintaining its performance under heavy load in the benchmark tests, we concluded that the GUI will be able to perform with larger fleets under normal use.

Chapter 4

Robot Prototype

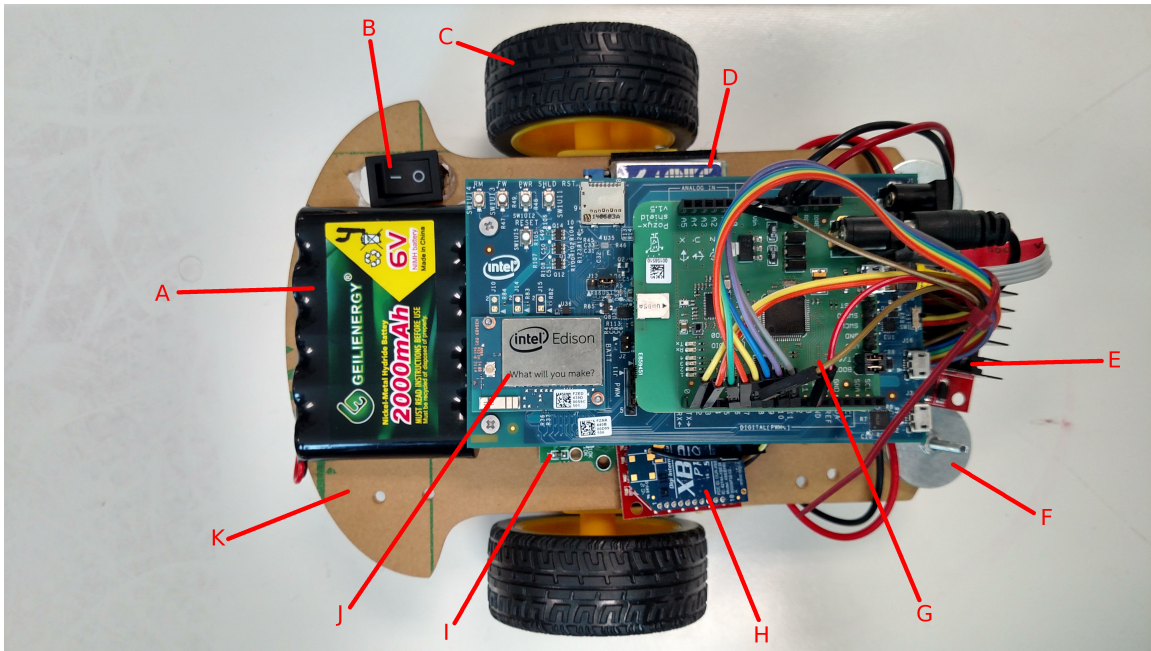


Figure 4.1: Robot Prototype

4.1 Overview

We designed the robot that we used to test our software. Most of the components on the robot are depicted in Figure 4.1. The components were a 6V 2000mAh NiMH battery(A), power switch(B), two powered wheels(C), a 5V 3A UBEC voltage regulator(D), a L298 dual H-bridge motor controller(E), 8 washers for counterweight(F), Pozyx position tag(G), XBee communication module(H), optical motor encoders(I), a microprocessor, in this case an Intel Edison with Arduino breakout board(J), and an acrylic chassis(K). The components missing from Figure 4.1 are a caster wheel, and a 6V to 9V boost converter. Even though only one set of components was used, the software on the robot was designed to abstract with a hardware abstrac-

tion layer. Although we did not demonstrate that the software works for a different mode of communication or using a different motor driver, it was written to be extensible to different hardware.

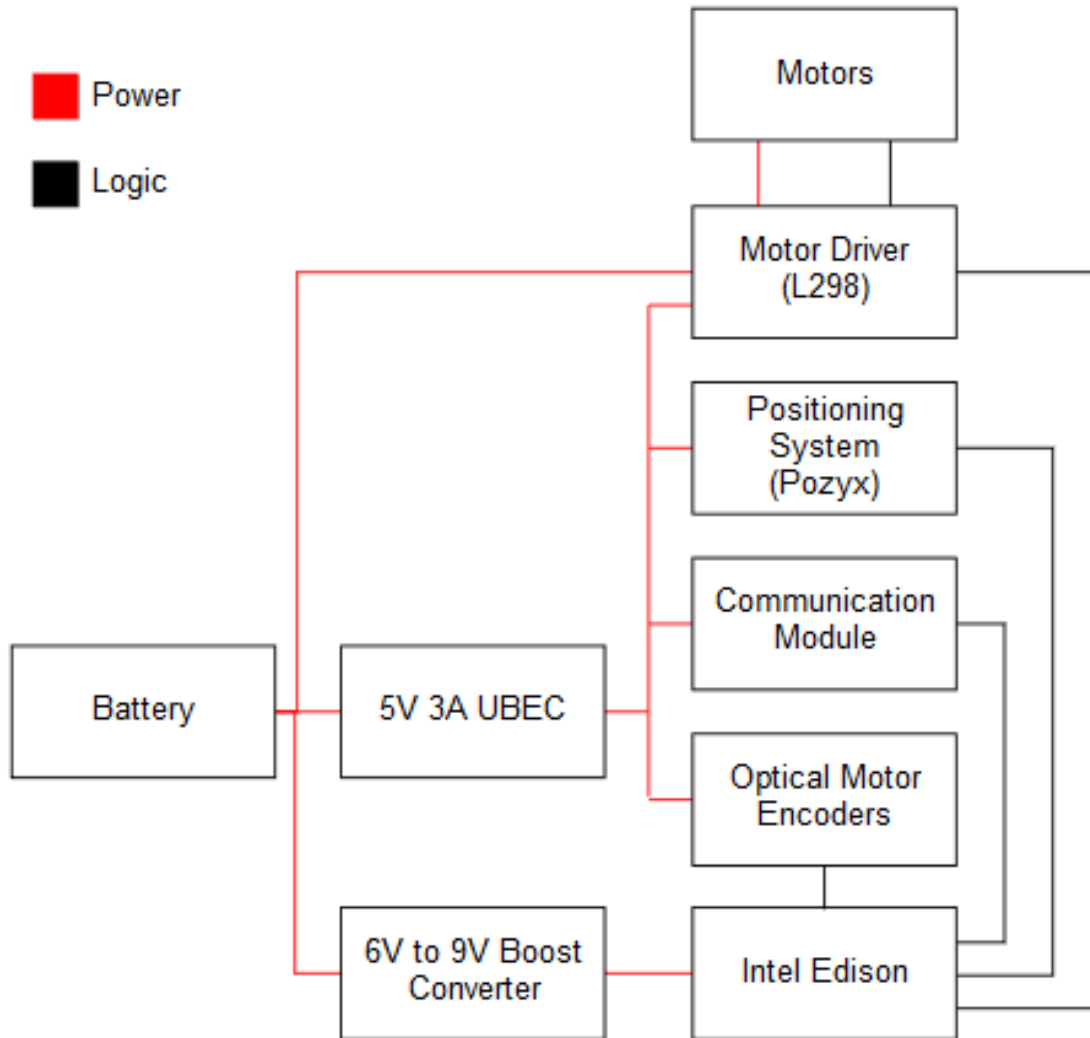


Figure 4.2: Edison Component Block Diagram

4.2 Requirements

We set out to build a robot that would be able to support all of the functionality that we planned our system would have. Since we needed to support a variety of different robot designs and purposes, the robot needed to support different communication, localization, and motor control hardware. Additionally, we needed the robot to be big enough to mount all of the hardware in an easily serviceable way, so that if something were to break we could easily replace it. The robot also needed a method of actuation that was

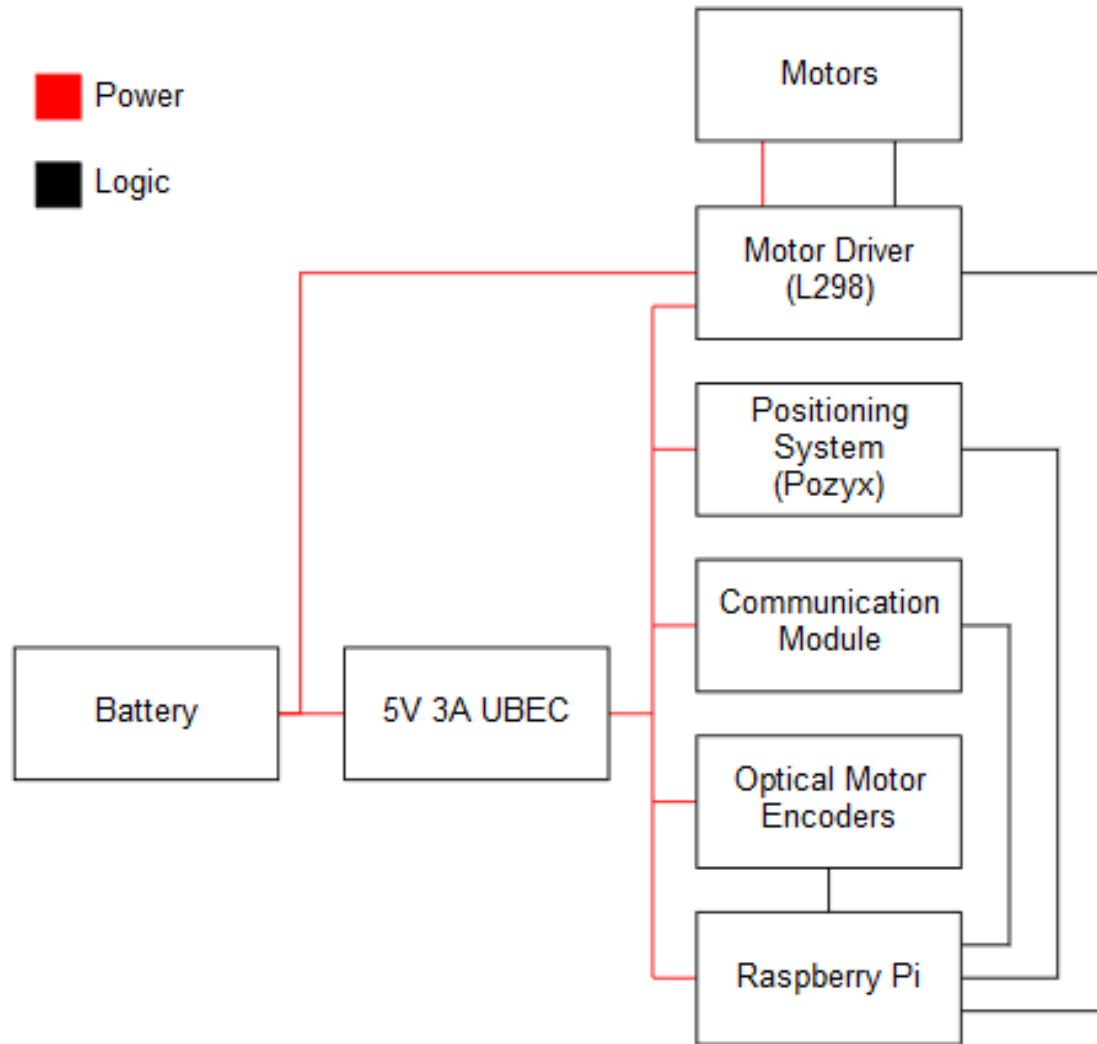


Figure 4.3: Raspberry Pi Component Block Diagram

capable of demonstrating different controllers. The microprocessors needed to be able to run the same code so that we did not waste time on different versions of the code for the different robots. Our components were selected to fulfill these requirements.

4.3 Components of Prototype

The XBee module and optical motor encoders were unused and removed from the scope of the project, as we determined they were not critical components for the success of our project and our time was better spent tackling other problems. Instead of communicating via the XBee, we used the built-in WiFi that came on the Raspberry Pi and the Intel Edison. The XBee would have been our proof that the software supports different communication hardware and protocols.

One voltage regulator was used to provide 5V to the L298, XBee module, and Raspberry Pi, while the other was used to provide 9V to the Intel Edison, which powered Pozyx over a USB connection. They were chosen for their efficiency, price, and form factor.

The 2000mAh battery was used to power the system. It was chosen because it was estimated that the battery would power the system for roughly an hour if the robot was moving at full throttle the whole time, and it was relatively cheap compared to other battery options.

The L298 was the hardware motor driver, which was chosen because of its ease of use, price, and our familiarity with how to use it. Pozyx was used for determining the location and orientation of the robot. Pozyx was chosen for this project because the technology was new to the lab and they wanted to see how well it could perform.

Table 4.1: Bill of Materials

Item	Quantity	Unit Cost
Intel Edison and Arduino Breakout	1	\$100
Raspberry Pi 2	1	\$35
Geilienergy 2000 mAh NiMH Battery	4	\$9.99
L298N Motor Driver	3	\$3.23
Switch Mode UBEC DC-DC Regulator	3	\$5.80
Boat Rocker On/Off Switch	3	\$0.94
2.1 x 5.5 mm DC Power Converter	2	\$0.47
Photoelectric Encoders	3	\$6.78
Robot Car Chassis Kit	3	\$12.73
Pozyx "Ready to Localize" Kit	1	\$675.82

The chassis of the robot was selected because it provided enough surface area for all of the components to be mounted with ease, had a simple design, and was fairly cheap. When controlling the robot after it was built, we noticed that the Intel Edison robots had a tendency to tip forward because of the weight of the battery on the front. To remedy this problem, washers were used to weight the backend of the Intel Edison

robots down.

The software for the robot was written in Python, which was chosen because of its ease of testing new iterations of the code. Python allowed us to save time by not compiling our code after every change. Additionally, Python can be installed on any Linux OS, so it allowed us to have one version of code that worked on all platforms. See Table 4.1 for a bill of materials which lists out the components of the prototype robot along with their quality and cost.

Chapter 5

Switching Architecture

5.1 Design

One of the main features of our architecture is the ability to seamlessly switch the controller of a robot, or a cluster, to another. Even though the switching architecture is extensive and involves code in three different layers - Matlab, jMatlab, and the user interface - the user of the system is easily able to use this functionality through the GUI. With the push of a button, the website updates the list of available controllers and robots in the system, and populates two dropdown menus. The list can be refreshed any time by performing the same actions. The user is then able to mix and match controllers and robots as they please. Once the desired configuration has been set, the user presses another button which sends the configuration over to jMatlab, through Dataturbine, and to Matlab when in manual control mode or to the robot when in waypoint navigation mode.

5.2 Implemented Controllers

In Multi-bot Easy Control Hierarchy, our team implemented two modes of control. The first one is off-board manual control with a joystick and the second one is onboard waypoint navigation via the graphical user interface. The architecture - with the user's input - is able to switch between these two modes of control using the user interface.

5.2.1 Manual Control

The joystick control was implemented in Simulink. A joystick Simulink block reads in data from a physical joystick. This logic is stored in JoystickControl_v2.mdl. Our Simulink model adjusts the gains of the input to fit our prototype robot. It also translates the joystick's raw values into motor values. The resulting output is sent to a Matlab function called MECH_send, referenced from MECHDTCConnection.m. MECHDTCConnection.m takes in a robot name, an IP address, and TCP port, which are all specified by the

user. A variable named `connect` is assigned to a controller made from the IP address and the TCP port. The controller is a Matlab function called `controller.m`. `controller.m` takes three arguments: `dataturbineip`, `dataturbineport`, `controllername`. This `connect` is registered to the function `MECHDataParser` with the function called `registerfunction` which is found in `registerfunction.m`. This Matlab function takes three parameters: `controller`, `functionname`, `channelname`. A variable called `channelidx` is assigned the return value of the function `addcommandchannel.m`. `addcommandchannel` takes two parameters: `controller`, `channelname`. It adds the command channel to the controller. The controller is started with the function called `start.m`. It takes in the controller variable and starts it.

The input is formatted into a unsigned integer vector with the source and destination ID, the control scheme, and the motor value vector. The command packet is sent to Dataturbine with the Matlab file `sendcommand.m`. The Matlab file `sendcommand.m` sends the command packet on the Dataturbine channel on the specified controller. `sendcommand.m` takes three parameters: `controller`, `channelidx`, `data`.

5.2.2 Waypoint Navigation

Waypoint navigation is implemented on the robot in a Python function called `waypointNavigation()` which is contained in the file `MotorController.py`. `waypointNavigation()` uses the the robot's heading, position data, and the waypoint location. The waypoint's theta value, the radian offset of the robot's location compared to the waypoint location, is calculated with the arc tangent of the difference between waypoint location and the robot's location. Phi, the radian offset of the robot's heading compared to the waypoint's location, is calculated with the difference between the theta and the robot's heading. The motor speeds are calculated to reduce phi to zero and then to reduce the difference of the waypoint location and the robots current location.

If the difference between the waypoint's location and the robot's location is under a threshold, the robot will register that it successfully reached a waypoint.

5.2.3 Switching Implementation

Every command packet that the robot receives has a control field for the controller algorithm that the robot uses to interpret the packet. Packets are parsed every 100ms, therefore the minimum latency to switch to a different controller is 100ms. The robot's current state for which controller it is being controlled by is controlled by this command packet. What ever value the control field of the last packet was is the current state of the robot. The robot's software only has four different controllers: waypoint navigation, velocity-heading, steering-throttle, and raw motor values. Using these four control modes, any off-board controller could be implemented by future engineers.

The user interface is the only part of our system that was made to switch between controller sources. It

does so by changing which channel on Dataturbine that the robot is listening to, not by changing the control field. An overview of what this looks like is depicted in Figure 5.1. On the channel that the robot is listening to, the controller could change the field in the command packet to alter how the packet is being interpreted, but we did not implement a controller with that functionality. We were able to test that functionality before we integrated with Dataturbine and Matlab through some scripts that interacted with the robot, but these were just for testing. This attribute could be very useful in the future because it does not require the robot to change which Dataturbine channel it is listening to for switching between different controllers. Changing channels is quite a bit slower than changing the control field in the command packet, but we were unable to test how much slower.

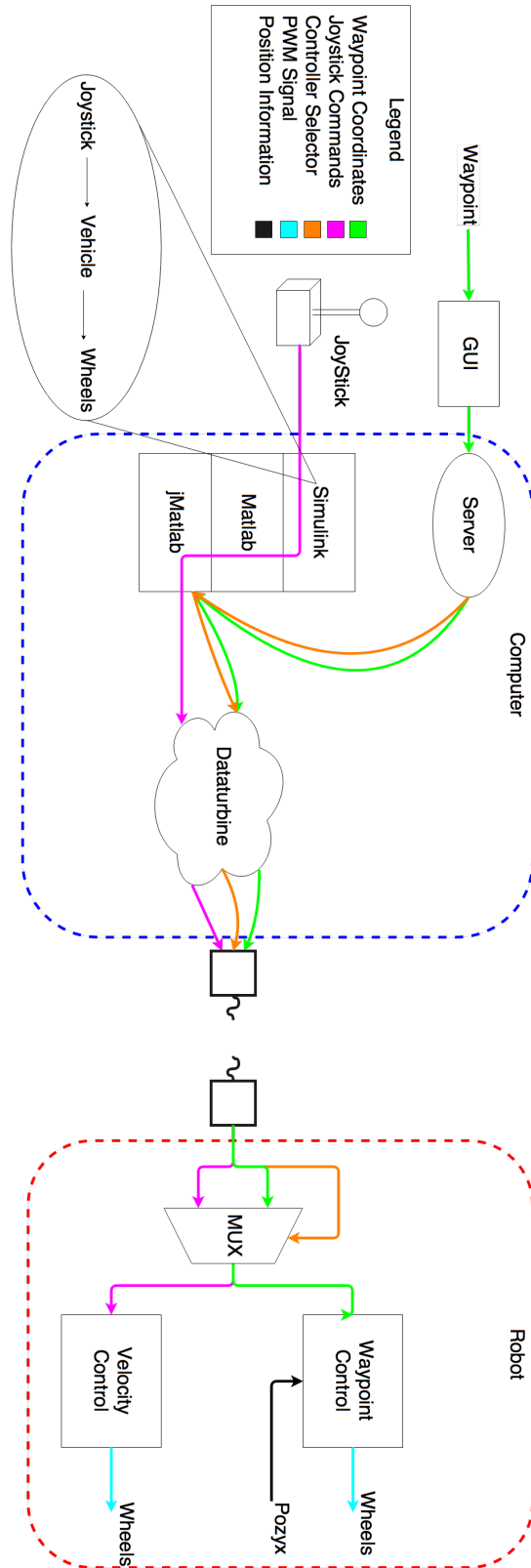


Figure 5.1: Implemented Controller Switching Overview

Chapter 6

Pozyx

Pozyx is a positioning system that uses ultra-wideband technology to wirelessly provide positioning and motion data. It also contains a 9-DoF inertial measurement unit, consisting of accurate motion sensors such as an accelerometer, gyroscope, and magnetometer. As we were performing small-scale testing indoors, GPS was unsuitable both due to its unreliability in indoor environment as well as its lack of accuracy: a GPS measurement has an average error of 6-10m, larger than our entire test area. We used the Pozyx's tutorial code and adapted it to work with our system. For our testing area, we set up four Pozyx anchors at four corners of a rectangle to create a test reference frame. Each of our robots has a Pozyx tag on it in order to get heading of each robot and compare distances relative to each anchor to obtain each robot's position within this reference frame.

6.1 Testing

Our team created several testing procedures to understand Pozyx's functionalities and their limitations. In order to test Pozyx claim of 10 cm accuracy, the steps to determine the accuracy is shown in Table 6.1. Because Pozyx is a 3D positioning system, it could be potentially affected by the heights of the Pozyx anchors. We tested this by observing the output change as a result of height, shown in Table 6.2. One of our algorithms and modes of control depends on the accuracy of Pozyx. Waypoint navigation needed to be tested with our software. The steps required to test waypoint navigation with the Pozyx are shown in Table 6.3. The robot needs to come close to the waypoints to determine the effectiveness of our algorithm and our implementation of the positioning system.

6.2 Results

During initial testing shown in Figure 6.1, we have noticed frequent outliers from the actual point ranging from 100mm to 500mm resulting in a standard deviation of x and y to be 189 and 182 respectively

Table 6.1: Position Accuracy

Position Accuracy	
1	Set up Pozyx anchors
2	Measure and mark a known point inside the grid
3	Set robot at known point
4	Run program that collects 100 positioning data
5	Calculate the average of the data
6	Calculate the standard deviation of the data

Table 6.2: Grid Offset

Grid Offset	
1	Set up Pozyx anchors
2	Measure and mark a known point inside the grid
3	Set robot at known point
4	Run program that prints position of robot infinitely
5	Increase one of the anchor's height
6	Observe outputs of program
7	Decrease the same anchor's height that is lower than the original height
8	Observe outputs of program
9	Repeat 5-8 for rest of the anchors

which is undesirable. To solve this problem, we implemented a moving median filtering algorithm to remove position outliers from robots. After filtering, the measured points from actual position are all within 100 mm resulting in a standard deviation of x and y to be 4 and 9 respectively which is reliable enough for our testing

Table 6.3: Waypoint Navigation Accuracy

Waypoint Navigation Accuracy	
1	Set up Pozyx anchors
2	Measure and mark a known point inside the grid
3	Set robot at known point
4	Observe how close the robot stops near each waypoints

purposes. The results of that test are shown in Figure 6.2.

During initial testing, the height of each anchor played an important factor to the xyz-offset of the grid. Because we have only implemented ground robots, we switched the Pozyx readings to a 2D mapping rather than a 3D space. While changing each of the anchor's height, we noticed that each of them followed a different pattern.

When heights are decreased the opposite pattern shown in Table 6.4 occurs for each anchor. The Pozyx anchor position is shown in Figure 6.3. Since these changes can affect accuracy of marked waypoints through days of repeated testing and that we are sharing these anchors with other groups who are testing with different height specifications, we decided to put all the anchors on the ground because it easier to setup because it requires zero to minimal amount of remeasurements for height.

Table 6.4: Increasing Height of Pozyx Anchor

Pozyx Anchor Position	X Offset	Y Offset
A	No change	Increase
B	Decrease	Increase
C	No change	Decrease
D	Decrease	Decrease

For moving to multiple waypoints, because of the filtering, the robots do recognize when it is within 100mm from the actual point and will move on to the next waypoint as shown in Figure 6.4.

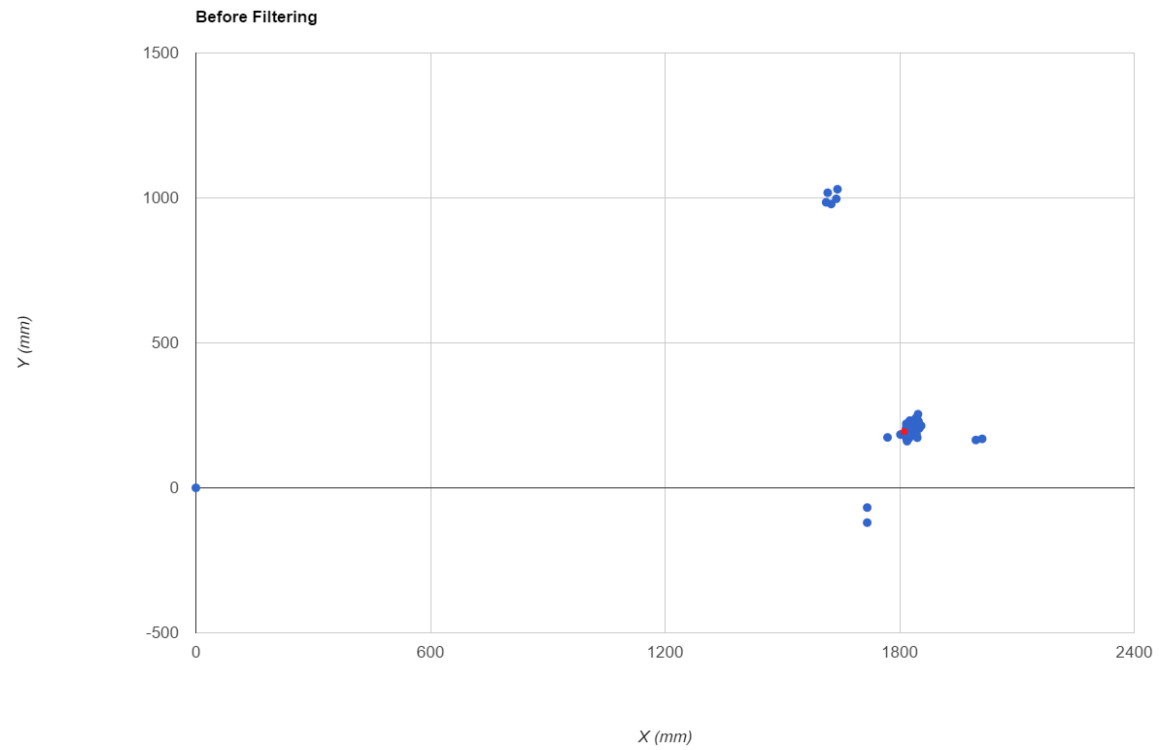


Figure 6.1: Pozyx Position Distribution Before Filtering

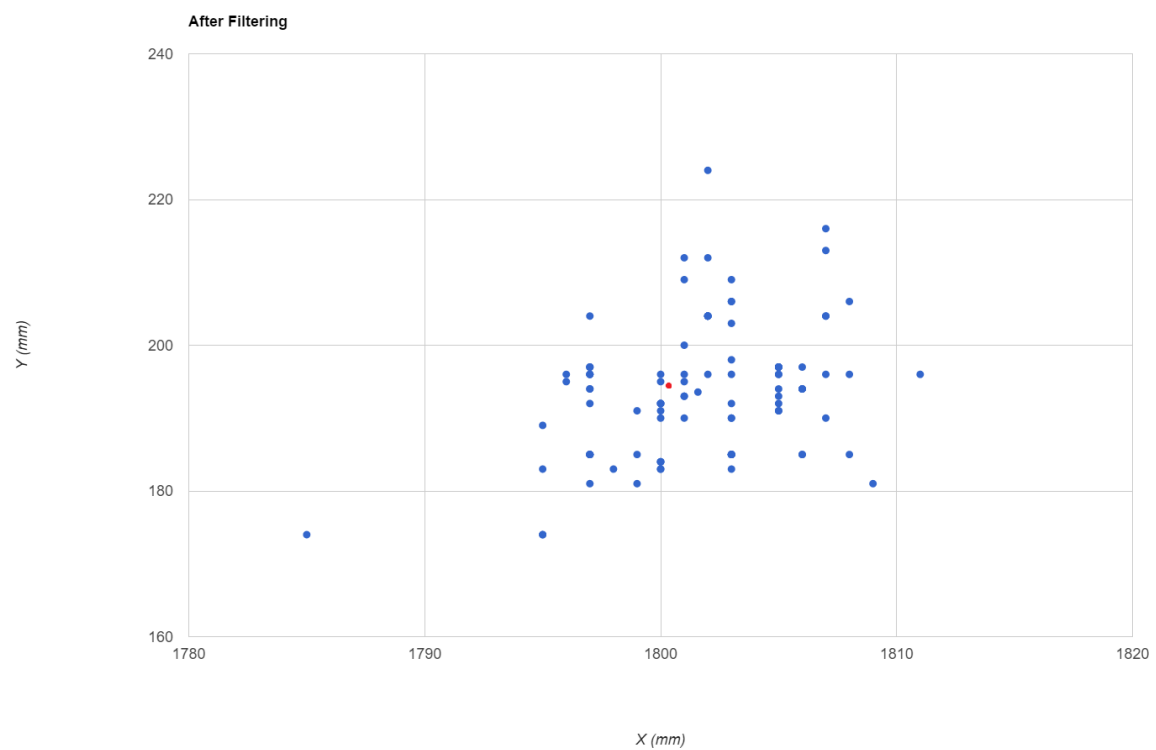


Figure 6.2: Pozyx Position Distribution After Filtering



Figure 6.3: Pozyx Anchor Layout

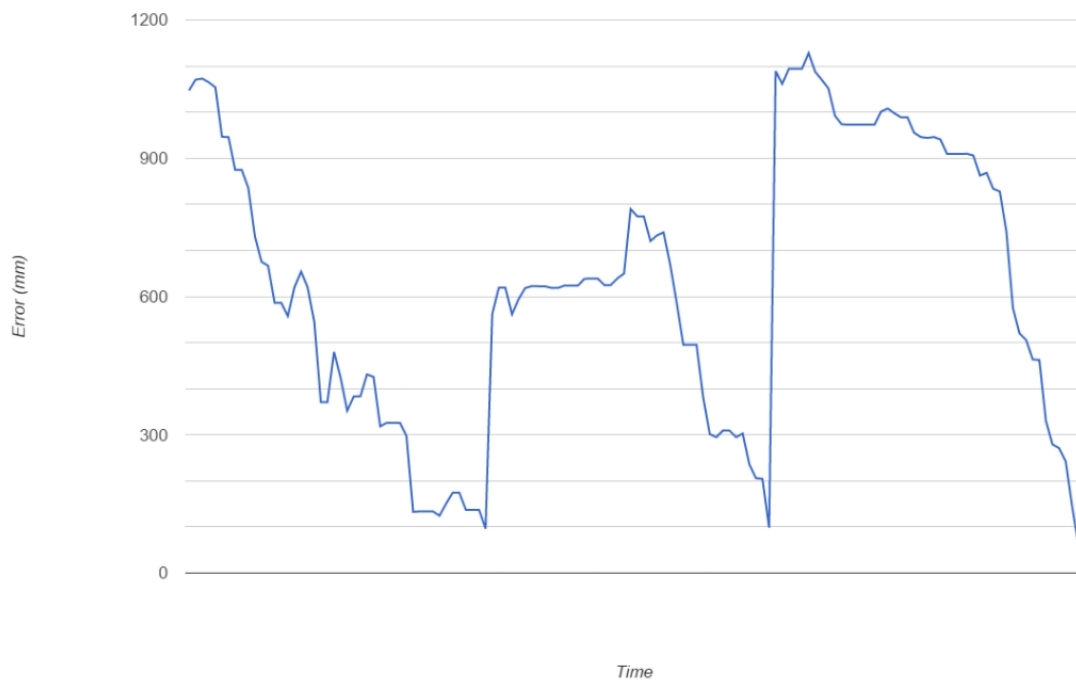


Figure 6.4: Pozyx Waypoint Error

Chapter 7

Societal Issues

Our team created a control architecture to aid engineers in the operation and creation of robots. The project spans interdisciplinary studies from computer engineering to electrical and mechanical engineering. Because of this, the ethical components of our architecture will include the components from the electrical and mechanical engineering field. We must consider the ethics behind robots and their interactions with humans, the environment, and other robots. Multi-bot Easy Control Hierarchy (MECH) aims to alleviate the problems that engineers face when using and creating a robot.

Ethical We believe that our team does not need to consider ethical ramifications of how a user will purposely use their robot with MECH as its platform. MECH is just an architecture of communication to aid operators in controlling their robot with any control scheme. It is the responsibility of the operator to use their robot in an ethical way. We do not plan to have software that ensures the robot is used in an ethical way. We do, however, hope that the students at the RSL may use our project in their robots in a way that provides great benefit.

Social The students in the Robotics Systems Lab (RSL) at Santa Clara University research the different applications of robots and how robots can benefit humans and the environment. This includes research in space, environmental monitoring, rescue rovers, and others at the RSL. We hope that the engineers creating such robots will use our architecture in their creations. Often times, the students create entirely new communication protocols for each new platform. The mechanical engineers at the RSL may not have a strong computer engineering background and make mistakes when programming their robots. MECH will make it easier for the mechanical engineers to establish a communication protocol by providing most of the code excluding robot-specific algorithms. The engineers have more time to complete their projects that could have a benefit to humanity and the world. This can result in more features, better safety, and more optimization in the robots produced.

Political Multi-bot Easy Control Hierarchy serves a purpose for a small community. The Robotic Systems Lab consists of students, staff, and professors of Santa Clara University. We hope our project can make an impact on their work and lives. But we do not expect MECH to affect a larger society outside Santa Clara University.

Economic As the advances in robotics and computers increase exponentially, countless jobs will be replaced with technology. Our project aids in the development of robots by giving engineers more time to create.

Health and Safety The software that we create can have errors and bugs. This will not be a problem in small, wheeled robots used for fun. In a quadcopter, however, there are blades that pose a danger to the surroundings because of its high rotational speeds. If the operator lost control of the quadcopter because of errors in our architecture, there could be damage to the robot, people in the vicinity, and the environment. Crashing the robot is expensive because of its parts and the long replacement time for the drone. The blades of the quadcopter can easily harm a person, resulting in hospitalization and injuries. Flying robots have a possibility to collide with aircraft such as passenger planes and helicopters which have a much larger impact in losses.

Manufacturability Since MECH is mostly software-based, there is not much of a need to manufacture our project. But the robots using our architecture must be manufactured. We have included a tutorial on how to build the robot prototype our team has used for testing. Robots ranging from simple hobby level to more professional level can be integrated with our system. The equipment used to build these robots could range from cheap to expensive depending on their respective level.

Sustainability Our system allows for additional types of robots and controllers to be implemented. It can be expanded to other platforms and the different levels of control. MECH also helps engineers use their resources wisely by taking the heavy work of establishing a switching architecture and communication protocol.

Environmental Impact Our project does not impact the environment directly but there is potential in the projects that use our architecture. As an example, we will use a previous project of the RSL. Many of the projects focus on environmental studies, and in this case, a cluster of kayaks equipped with sensors are used to monitor pollution in a lake. This potential application of our system and the robots can impact the fight against pollution and climate change.

Usability When adding new robots to our platform, users of the Multi-bot Easy Control Hierarchy must implement their own algorithms specific to their robot. We minimize user-caused errors with our well-written documentation that helps the user implement correct code. We have an ethical duty to ensure that our documentation and product is easy and safe to use for the customer, thus guaranteeing an efficient use of time.

Lifelong learning This project helped inspire us to seek new knowledge about unfamiliar fields of study. An example is that we needed an indoor positioning system to showcase our project. So after considering several options, we settled with using Pozyx. It is relatively new so we needed to learn how it operates by reading through its online documentation and performing unit testing to understand its capabilities and limitations. Because there were several other teams also testing with Pozyx, our group is able to get some previous testing data and assistance from them when we run into problems. This experience has strengthened our idea that information is always available somewhere and that peers and mentors can help guide us towards the right direction.

Compassion Building robots is a time-consuming task. Every project usually requires different specifications and functionalities for their robots so roboticists would need to rebuild them and redesign their code. Most of the time, they spend the majority of their time working on their robot rather than performing with it because they need to reconfigure or revise their code whenever they want to make adjustments or alterations. We made our project with the intention of reducing time spent on changing controller codes by allowing the user to be able to seamlessly switch between any type of controllers they wish to use for their robots. This would relieve some of the pressure from deadlines and investors by allowing the user to be able to allocate more time on accomplishing their project objectives rather than on readjusting their robot.

Chapter 8

Summary

8.1 Project Overview

Our team's objective was to make an architecture that makes it easy to seamlessly switch between controllers and various types of control signals such as actuator, velocity, and fleet levels. Extra precaution had to be taken during the reconfiguration of our system as the different controllers need to be synchronized and the communication between controllers and robots needs to be correctly routed.

Our team developed an architecture that afforded roboticists the ability to change the configuration of their robots' controller. The user can configure and monitor their robots through a web-based graphical user interface (GUI). Various changes can be made to the Matlab controllers, Python algorithms, and Simulink blocks to obtain the desired setup. The Pozyx indoors positioning system tracks the location of the robot and allows the user to employ waypoint navigation if needed. We have successfully demonstrated controlling two differential-drive robots individually and simultaneously with either waypoint navigation or joystick controller while also being able to switch between these two forms of control through the graphical user interface.

8.2 Future Work

Due to time limitations, we were unable to implement off-board waypoint calculations in Matlab. There were also plans to implement more advanced forms of cluster control, such as follow-the-leader and various pattern formations.

In the future, we hope to implement additional key features into our architecture, as well as run tests that incorporate a more extensive set of hardware in order to ensure cross-platform compatibility. Our immediate attention is focused on adding off-board calculations in Matlab and later creating more complex cluster control algorithms. A good starting point to ensure cross-platform compatibility would be to incorporate GPS and XBee technology into newly built robots utilizing our architecture.

We are currently testing the Multi-bot Easy Control Hierarchy (MECH) with land-based differential-drive robots. Our team hopes to test our system with any type of ground, aerial, or marine robot in the future.

Our vision is for MECH to support numerous different controllers, platforms, and communication protocols. We also hope to provide a library of the different control algorithms with our system.

As we have developed the architecture, we have faced a number of challenges in the implementation of the system. As our prototype robots were ran by an Intel Edison, the optical encoders on the wheels couldn't be utilized. The processing power of the Intel Edison was limited, and an Arduino Nano would've been required to process the encoder information in a timely manner. We opted The optical encoders in our robot prototype were not compatible with the Intel Edison. We wanted to add another component to handle the encoder, but opted not to in favor of developing more important features.

We made major changes to our technology after the fourth month of development. Matlab was required to be the language used for the controllers instead of Python. This caused some confusion because our team was not proficient in Matlab.

Pozyx is a newer technology, recently being used by the Robotic Systems Lab. Because of that, we needed to research and experiment with the software and the beacons.

References

- [1] C. A. Kitts and I. Mas, "Cluster space specification and control of mobile multirobot systems," *Mechatronics, IEEE/ASME Transactions on*, vol. 14, no. 2, pp. 207–218, April 2009.
- [2] Azevedo, Drew; Beltram, Sam; DelVecchio, Gregorio; and Hopner, Ben, "MARV: Marine Autonomous Research Vessel" (2016). Interdisciplinary Design Senior eses. Paper 20.
- [3] H. Ishidaa, K. Suetsugua, T. Nakamotoa, T. Moriizumia, "Plume- Tracking Robots: A New Application of Chemical Sensors", Faculty of Engineering, Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro-ku, Tokyo 152, (Apr. 2001). Available online.
- [4] T. Adamek, C. A. Kitts and I. Mas, "Gradient-Based Cluster Space Navigation for Autonomous Surface Vessels," in *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 2, pp. 506-518, April 2015.
- [5] I. Mas, S. Li, J. Acain, C. A. Kitts, "Entrapment/escorting and patrolling missions in multi-robot cluster space control", *Intelligent Robots and Systems," IEEE/RSJ International Conference on, IROS 2009*, pp. 5855–5861. St. Louis, MO, Oct. 2009.
- [6] A. Westgate, "Dynamic control migration between a base station and a remote robot", Santa Clara University, School of Engineering, 2010

Appendix A

Appendix

A.1 Annotated Bibliography

Dong, Xiwang, et al. "Time-Varying Formation Control For High-Order Swarm Systems With Switching Directed Topologies." *Information Sciences* 369.(2016): 1-13. Academic Search Complete. Web. 6 Oct. 2016

In this article, researchers put forth a formation control protocol that they use to control a swarm of robots. Using an approach that listens to the feedback of each robot in the swarm, the researchers could move the robots into certain patterns in a specified amount of time. They defined the time-varying formation reference function mathematically and proved that using a switching interaction topology has no effect on their mathematical model. In concluding their research, they set forth a way to expand upon the possible formations of that their model can support, and showed the data they collected from simulations to prove their theoretical model.

Lacroix, Philippe, et al. "Decentralized Control Of Cooperative Multi-Robot Systems." *Integrated Computer-Aided Engineering* 6.4 (1999): 259. Academic Search Complete. Web. 6 Oct. 2016.

This article details how researchers devised a way for robots to work together to complete a task that a single robot could not do. Specifically they looked at two nonholonomic robots working to move a beam to a specified location. They described a two stage control structure that uses only the position of the robot and how the task has evolved to determine what the robots should do next. The higher level of control lays out what changes need to be made to complete the task, and the lower level of control determines what the robots should do to make the changes the high level described. The researchers provided how they set up experiments and their simulated results to show that their control scheme can complete tasks and work around obstacles, even if only a single robot detects the obstacle.

Tang, Qirong, et al. "Relative Observation For Multi-Robot Collaborative Localisation Based On Multi-Source Signals." *Journal Of Experimental & Theoretical Artificial Intelligence* 26.4 (2014): 571-591. Business Source Complete. Web. 23 Oct. 2016.

Researchers used a distributed extended Kalman filter (EKF) that used robot odometry and a technology that's best described as indoor GPS to accurately track the pose (location and heading) of a robot. In addition to those localization strategies, they also used IR sensors and gyroscopes to discover the error of their EKF technique and to produce more accurate localization readings. They analyze results from experiments to predict how quickly an error in the system propagated and to determine the steady state error of their sensors. Using their analysis of their experiments, they created a collaborative localisation strategy for a multi-robot system. Data from simulations were provided to demonstrate their work.

"Control Architecture For Multi-Robot System." (2014): USPTO Patent Applications. Web. 23 Oct. 2016.

This is a patent that describes a way to control the positions of robots that alter the geography of the site they are working on in a cluster. It describes a centralized control scheme where all the robots communicate with a single server that processes the input and transmits to the robots where to move next. The patent also covers a display that shows operators the relative positions of the robots and the areas the robots are working on. The patent allows for coordination of the robots to help each other complete larger tasks by completing smaller tasks. The ground control station utilizes a database that can be accessed from multiple systems at once, which allows for multiple operators in different locations to view the work of the robots. Most of the patent is dedicated to describing how the database is kept up to date by various systems on the robot.

Qian, Dianwei, et al. "Leader-Following Formation Control of Multiple Robots with Uncertainties through Sliding Mode and Nonlinear Disturbance Observer." *ETRI Journal* 38, no. 5: 1008-1018. *Applied Science & Technology Source*, EBSCOhost (accessed October 24, 2016).

This paper describes a control scheme for controlling multiple robots in a follow the leader control scheme. The researchers used a sliding mode control method with a nonlinear disturbance observer (NDOB) technique to add uncertainties to the robot cluster. Essentially, they moved the surface that the robots were on and used the NDOB to measure the uncertainties the researchers created. First order and second order sliding mode control methods are compared to demonstrate their effectiveness under certain uncertainties. The first order was good at estimating position inaccuracies, and the second order was better at estimating velocity inaccuracies. During the research for this paper, the authors proved a condition for the Lyapunov theory to

be sufficient for a asymptotically stable formation of the robots. Unfortunately, the Lyapunov theory was not described in this paper, and further research is needed validate their control scheme on more general robot formations other than leader-follower formations.

A.2 Literature Review

A.2.1 Proposed Objectives

The main objectives of the Multi-bot Easy Control Hierarchy were defined by Dr. Christopher Kitts, our senior design project advisor. Kitts wants the purpose to be reducing workload and time needed for the robot implementation process.

A.2.2 Previous Work

Basics of Design Controllers for Industrial Robots

Although the students at the Robotics Systems Lab (RSL) mainly construct robots that are able to transport themselves, there is no reason that we can extend the Multi-bot Easy Control Hierarchy to industrial robots. Industrial robots, as described by Tihomir Latinovic and his group¹, are robots that imitate human motions to provide safety and precision in industrial applications. Latinovic and his group have written about the basics of designing a controller for these types of robots in the journal *Acta Technica Corviniensis*. They test each axis of the IRB 2000 robot from ABB. This results in the details of the robot's workspace angle and maximum speeds. They also include the components of the control panel which needs control methods such as joystick, terminal, or programmed motions.

Our project will include the control of robots that are not limited to stationary robots. Latinovic's proposed controller will be processed on a computer separate from the robot. Our project, however, will be able to calculate motion off or on board the robot.

Control Architecture for Multi-Robot System

John Posselius and his group patented an architecture² to control how the robots in a multi-robot system is positioned. Their architecture controls agricultural robots in a cluster based on their geography. Posselius' architecture is based on a centralized server architecture. The robots communicate with this server to determine the locations and positions of the robots in the system. The centralized server is able to process and send out commands that the robots in the system are able to follow. The patent also allows the operator to

¹Latinovic, Tihomir, et al. "The Basics Of Designing Controllers For Industrial Robots (Eg. Robots Abb Irb 2000)." *Acta Technica Corviniensis - Bulletin Of Engineering* 4.3 (2011): 101-104. Academic Search Complete. Web. 23 Oct. 2016

²"Control Architecture For Multi-Robot System." (2014): USPTO Patent Applications. Web. 23 Oct. 2016.

monitor the agricultural robots by sending back the speed and location of the robots. The centralized server has the capability to allow multiple operators to simultaneously view the process of the robots.

Our system will have similar functionalities as the architecture described by Posselius. However, they have described only a centralized point of control in their system. Our project will allow for calculations to be performed on-board the robot. The operator of our system will be able to switch between calculations from ground control and on the robots themselves. We would also have a graphical user interface for our operators to monitor the robots in the system. In addition to a multi-robot system, the Multi-bot Easy Control Hierarchy is able to switch between fleet, vehicle, and actuator modes. The architecture described by Posselius cannot switch between these modes.

Feedback Control Strategies for Quadrotor-type Aerial Robots

Ozbek, Önkol, and Efe³ have developed strategies that would help in the operation of unmanned aerial vehicles (UAV). They describe how these UAVs can be operated autonomously or by a person. Our architecture will not have autonomous capacities, but the team can learn from the research that these engineers have done in aerial robots. Özbek and his team describe the approach of using a proportional-integral-derivative (PID) control, sliding mode control, feedback control, and fuzzy control in the field of aerial vehicles. They have developed the math used in these control schemes which we can implement when testing our architecture with quadcopters. Much of the math involved in aerial robots is probably familiar with the engineers in the RSL. The concepts can also be used in ground or water based vehicles.

Ozbek's work only concerns the controls in aerial robots. Our system will be able to handle different types of robot which can be operated and controlled at the same time.

Decentralized Control of Cooperative Multi-Robot Systems

Philippe Lacroix and his group devised a decentralized approach for the control of cooperative multi-bot system performing a task such as navigation through obstacles to a goal⁴. Their design consisted of two robots that are four-wheeled front-steering vehicles equipped with a frontal one-degree-of-freedom lift and two modules: a task-oriented beam controller that determines the motion of the beam that would bring it closer to the goal and a robot based low-level controller that computes a desirable motion for the center of the beam based on the robot's position and orientation relative to the goal. Their robots also do not communicate with each other as both the robots are computing their own path based on what each of them see with their frontal camera. This might be useful to us in learning how to create a follow the leader function where we

³Ozbek, Necdet Sinan, et al. "Feedback Control Strategies For Quadrotor-Type Aerial Robots: A Survey." *Transactions Of The Institute Of Measurement & Control* 38.5 (2016): 529-554. Academic Search Complete. Web. 23 Oct. 2016.

⁴Lacroix, Philippe, et al. "Decentralized Control Of Cooperative Multi-Robot Systems." *Integrated Computer-Aided Engineering* 6.4 (1999): 259. Academic Search Complete. Web. 6 Oct. 2016.

only communicate to the lead robot and the other one will follow the lead.

Leader-Following Formation Control of Multiple Robots with Uncertainties through Sliding Mode and Nonlinear Disturbance Observer

Dianwei Quan, Shiwen Tong, and Chengdong Li created a control scheme for the leader-following function of multiple robots that combines the sliding mode control method with the nonlinear disturbance observer technique. They broke down the multi-bot system into several leader-follower pairs where all followers share one leader. The leader follows a predetermined path while the followers continuously maintains desired positions relative to the leader. While their project used a predetermined path, our group plans to allow the leader to be either controlled with a controller or follow a waypoint-based path. They also created several algorithms⁵ for their control scheme on calculating the distance and angles between followers and leader which may be useful for our own follow-the-leader function and fleet control.

While they used sliding mode control method and nonlinear disturbance observer technique as their implementation, they have also listed several more alternatives such as robust control, predictive control, adaptive control, decentralized control, and feedback linearization. We plan to research further into which control scheme is most suited for our project.

⁵Qian, Dianwei, et al. "Leader-Following Formation Control of Multiple Robots with Uncertainties through Sliding Mode and Nonlinear Disturbance Observer." ETRI Journal 38, no. 5: 1008-1018. Applied Science & Technology Source, EBSCOhost (accessed October 24, 2016).

A.3 Source Code

A.3.1 Robot

```
#####
MotorController.py
#####

#!/usr/bin/env python

# This file was created by Ryan Cooper in 2016 for a Raspberry Pi
# This class controls the motors for the robot which are configured as
# a differential drive, this code is written for a raspberry pi,
# TODO but should be reworked to load a driver that drives the motors
import time
import sys
import math

from multiprocessing import Process
from multiprocessing import Queue
from multiprocessing import Pipe

import util

#WARNING: calling print too frequently will cause high latency from control input to
        reaction

class MotorController(Process):
    # possible states
    STEERING_THROTTLE_OFFBOARD = 1
    STEERING_THROTTLE_ONBOARD = 2
    TANK = 3
    VELOCITY_HEADING = 4
    WAYPOINT = 5
    ENCODER_TEST = 6
    state = STEERING_THROTTLE_OFFBOARD

    # possible velocity heading states
    TURNING = 0
    DRIVING = 1
    # velocity heading state
    vhState = TURNING

    LEFT = 0
    RIGHT = 1

    mPowers = [0, 0]
    direction = [0, 0] # forward or backward
    # set by time.time(), used to stop bot when dced
    lastQueue = 0
    go = True

    # for vel/heading mode
    desiredHeading = 0
    # should be in mm/sec
    desiredVel = 0
    currentHeading = 0
    requiredCounts = 0
```



```

motorOffValue = 1024
motorHighValue = 2048
motorLowValue = 0

waypointTravelSpeed = 75 # out of 100
waypointThresh = 5 # centimeters

def __init__(self, motorDriver):
    super(MotorController, self).__init__()
    self.driver = motorDriver()
    self.driver.setDC([0,0],[0,0])

# vel in m/s
def setDCByVel(self, vel):
    if vel > 0:
        self.direction = [0, 0]
    else:
        self.direction = [1, 1]
    for i in range(0, 2):
        if abs(vel) > util.maxVel:
            self.mPowers[i] = self.driver.maxDC
        elif abs(vel) < util.minVel:
            self.mPowers[i] = 0
        else:
            # experimenal, play with minDC, and minVel because maxVel was observerd at
            # maxDC
            self.mPowers[i] = util.transform(vel, util.minVel, util.maxVel,
                                             self.driver.minDC, self.driver.maxDC)
    self.driver.setDC(self.mPowers,self.direction)

def exitGracefully(self):
    self.mPowers = [0, 0]
    self.driver.setDC(self.mPowers, self.direction)
    go = False

def steeringThrottle(self, data):
    steering = util.transform(data[1], self.motorLowValue, self.motorHighValue, -1, 1)
    throttle = util.transform(data[2], self.motorLowValue, self.motorHighValue, -1, 1)
    # used in steering to change motor velocities
    maxSp = 35
    maxSm = 220
    # max possible speed when moving forward
    maxMove = 220
    minMove = 0
    # sp is what will get added (plus) to t (which is the throttle value)
    sp = util.transform(abs(steering), 0, 1, 0, maxSp)
    # sm is what will get subtracted (minus) to t (which is the throttle value)
    sm = util.transform(abs(steering), 0, 1, 0, maxSm)
    t = util.transform(abs(throttle), 0, 1, minMove, maxMove)
    L = t
    R = t
    end = self.motorOffValue
    if throttle < 0:
        if steering < 0:
            # right motor should slow down, left motor should speed up
            L += sp
            R -= sm
        else:
            # left motor should slow down, right motor should speed up

```

```

        L -= sm
        R += sp
    end = self.motorHighValue
else:
    if steering < 0:
        # left motor should slow down, right motor should speed up
        L -= sm
        R += sp
    else:
        # right motor should slow down, left motor should speed up
        L += sp
        R -= sm
    end = self.motorLowValue
mL = util.transform(util.clampToRange(L, 0, 255), 0, 255, self.motorOffValue, end)
mR = util.transform(util.clampToRange(R, 0, 255), 0, 255, self.motorOffValue, end)
    #sys.stdout.write(str(mL) + " " + str(mR) + "\n")
self.changeMotorVals(mL, mR)

# this function will consume the controllerQueue, which was filled by DDMCServer
# and will change the motors powers and directions according to what was in the queue
# it also will monitor that the bot is still receiving commands, and if it isn't, it
# will stop the bot
def handleControllerQueue(self):
    #print self.state
    # if there hasn't been anything in the queue in half a second
    if self.state != self.VELOCITY_HEADING and time.time()-self.lastQueue > .5 and
        util.controllerQueue.empty():
        # stop the bot
        self.direction = [0, 0]
        self.mPowers = [0, 0]
        self.lastQueue = time.time()
    else:
        while not util.controllerQueue.empty(): # this is a while so that the most
            recent thing in the queue is the resultant command that is done
            good = True
            try:
                # nowait because this process was called from the main loop which
                # controls the self.motors
                # so we don't want this function to block.
                data = util.controllerQueue.get_nowait()
            except Queue.Empty as msg:
                # realistically this should never happen because we check to see that the
                # queue is not empty
                # but it is shared memory, and who knows?
                good = False
            if good:
                mL = self.motorOffValue
                mR = self.motorOffValue
                if data[0] == self.STEERING_THROTTLE_OFFBOARD or data[0] == self.TANK: #
                    recieved motor level commands
                    self.state = data[0]
                    mL = data[1]
                    mR = data[2]
                    self.changeMotorVals(mL, mR)
                elif data[0] == self.STEERING_THROTTLE_ONBOARD: # recieved joystick
                    information (throttle, steering)
                    print data
                    self.state = data[0]
                    self.steeringThrottle(data)# this calls changeMotorVals()

```

```

elif data[0] == self.VELOCITY_HEADING:
    print "velHeading entered"
    self.state = data[0]
    self.vhState = self.TURNING
    self.desiredVel = util.transform(data[1], self.motorLowValue,
                                     self.motorHighValue, -util.maxVel, util.maxVel)
    if abs(self.desiredVel) >= .1:
        self.mPowers = [0, 0]
        self.driver.setDC(self.mPowers, self.direction)
    self.desiredHeading = data[2]
    self.goToHeading(self.desiredHeading)
elif data[0] == self.WAYPOINT:
    self.waypointNavigation(data[1], data[2])
if data[0] != self.WAYPOINT:
    # consume the queue so that way when the robot is switched to
    # waypoint, it gets fresh data
    while not util.positionQueue.empty():
        util.positionQueue.get_nowait()
self.lastQueue = time.time()

# this sets up the values used to drive the motors
# it does not drive the motor because this function is tied to the queue
# and only gets executed when something is in the queue
# yet we want the motors to be constantly receiving control information
# motorLowValue <= mL, mR <= motorHighValue, motorOffValue means the wheels won't turn
def changeMotorVals(self, mL, mR):
    if mL > self.motorOffValue:
        self.direction[self.LEFT] = 1
        self.mPowers[self.LEFT] = util.clampToRange(util.transform(mL,
                                                                    self.motorOffValue, self.motorHighValue, 0, 100), self.driver.minDC-1,
                                                                    self.driver.maxDC)
    else:
        self.direction[self.LEFT] = 0
        self.mPowers[self.LEFT] = util.clampToRange(util.transform(mL,
                                                                    self.motorOffValue, self.motorLowValue, 0, 100), self.driver.minDC-1,
                                                                    self.driver.maxDC)
    if self.mPowers[self.LEFT] < self.driver.minDC:
        self.mPowers[self.LEFT] = 0

    if mR > self.motorOffValue:
        self.direction[self.RIGHT] = 1
        self.mPowers[self.RIGHT] = util.clampToRange(util.transform(mR,
                                                                    self.motorOffValue, self.motorHighValue, 0, 100), self.driver.minDC-1,
                                                                    self.driver.maxDC)
    else:
        self.direction[self.RIGHT] = 0
        self.mPowers[self.RIGHT] = util.clampToRange(util.transform(mR,
                                                                    self.motorOffValue, self.motorLowValue, 0, 100), self.driver.minDC-1,
                                                                    self.driver.maxDC)
    if self.mPowers[self.RIGHT] < self.driver.minDC:
        self.mPowers[self.RIGHT] = 0
    #print self.mPowers

    if self.state != self.VELOCITY_HEADING:
        self.driver.setDC(self.mPowers, self.direction)
    # VELOCITY_HEADING mode calls setDC from either goToHeading, or setDCbyVel

def goToHeading(self, h):
    if abs(self.currentHeading-h) > .01:

```

```

        if h > 2*math.pi:
            # make h < 2pi
            h = h-(2*math.pi*math.floor(h/(2*math.pi)))
        elif h < -2*math.pi:
            # make h > -2pi
            h = h+(2*math.pi*math.floor(h/(2*math.pi)))
        angDiff = h-self.currentHeading
        self.changeHeadingByRadians(angDiff)

def changeHeadingByRadians(self, h):
    if abs(h) > .01:
        self.resetEncoders()
        if h > 0:
            self.direction = [1, 0]
        else:
            self.direction = [0, 1]
        # angle*radius=arclen
        sys.stdout.write("Moving by radians: ")
        sys.stdout.write(str(h))
        sys.stdout.write("\n")
        dist = h*util.botWidth/2.0
        sys.stdout.write("dist=")
        print dist
        self.requiredCounts = int(abs(dist/util.distPerBlip))
        sys.stdout.write(" requiredCounts ")
        print self.requiredCounts
        self.mPowers = [50, 50]
        self.driver.setDC(self.mPowers,self.direction)

def resetEncoders(self):
    self.gpioQueue.put(['resetEncoders'])

# PID part of the wheel controller loop
def controlPowers(self, vel, pin): #TODO possible use mm/sec instead of m/s because it
    will be more accurate because floating point is bad
    if vel != -1:
        if self.desiredVel != 0:
            p = self.desiredVel-vel
            sys.stdout.write("Vel difference: ")
            sys.stdout.write(str(p))
            pPWM = 0
            if abs(p) >= util.minVel:
                if p > 0:
                    pPWM = util.transform(p, util.minVel, util.maxVel, self.driver.minDC,
                        self.driver.maxDC)
                else:
                    pPWM = -util.transform(-p, util.minVel, util.maxVel,
                        self.driver.minDC, self.driver.maxDC)
            sys.stdout.write("PWM effort: ")
            print pPWM
            if(pin == util.leftEncPin):
                self.mPowers[self.LEFT] = util.clampToRange(self.mPowers[self.LEFT]+pPWM,
                    0, 100)
            elif(pin == util.rightEncPin):
                self.mPowers[self.RIGHT] =
                    util.clampToRange(self.mPowers[self.RIGHT]+pPWM, 0, 100)
            else:
                print "Encoder is reading data to an unexpected pin"
        else:

```

```

        self.mPowers = [0, 0]
        self.driver.setDC(self.mPowers, self.direction)

def handleEncoderQueue(self):
    while not util.encQueue.empty():
        good = True
        try:
            # nowait because this process was called from the main loop which controls
            # the motors
            # so we don't want this function to block.
            data = util.encQueue.get_nowait()
        except Queue.Empty as msg:
            # realistically this should never happen because we check to see that the
            # queue is not empty
            # but it is shared memory, and who knows?
            good = False
        if good:
            if self.state == self.VELOCITY_HEADING:
                if self.vhState == self.TURNING:
                    # note, does not check which motor moved the desired amount, possible
                    # change this
                    print data
                    if data[1] >= self.requiredCounts:
                        self.vhState = self.DRIVING
                        self.currentHeading = self.desiredHeading
                        self.setDCByVel(self.desiredVel)
                    else:
                        # data[2] = seconds/blip
                        # convert to rotations per second
                        # then multiply by distance wheel travels in one rotation
                        # result is mm/second
                        vel = -1
                        if data[2] > -1:
                            vel = util.stateChangesPerRevolution/data[2]
                        sys.stdout.write("vel=")
                        print vel
                        sys.stdout.write("desiredVel=")
                        print self.desiredVel
                        # calls setDC()
                        # pid part of the loop
                        self.controlPowers(vel, data[0])
            elif self.state == self.ENCODER_TEST:
                print data
                if data[0] >= self.requiredCounts:
                    print "wtf"
                    self.mPowers = [0, 0]
                    self.driver.setDC(self.mPowers, self.direction)
                    time.sleep(1)
                    self.mPowers = [35, 35]
                    self.requiredCounts = util.stateChangesPerRevolution
                    self.driver.setDC(self.mPowers, self.direction)

def waypointNavigation(self, wx, wy):
    mPos = None
    while not mPos:
        # consume queue until we get newest data
        while not util.positionQueue.empty():
            mPos = util.positionQueue.get_nowait()
        # mPos[0] = mPos_x, mPos[1] = mPos_y, mPos[2] = mPos_heading

```

```

x = wx-mPos[0]
y = wy-mPos[1]
if x > self.waypointThresh or y > self.waypointThresh:
    d = math.sqrt(x*x+y*y)
    theta = math.atan2(y,x)
    phi = theta-(mPos[2]-math.pi)
    if phi < 0 :
        rm = self.waypointTravelSpeed
        lm = self.waypointTravelSpeed*math.cos(phi)
    elif phi > 0:
        rm = self.waypointTravelSpeed*math.cos(phi)
        lm = self.waypointTravelSpeed
    else:
        lm = self.waypointTravelSpeed
        rm = self.waypointTravelSpeed
    self.mPowers = [math.fabs(rm), math.fabs(lm)]
    self.direction = [0 if rm > 0 else 1, 0 if lm > 0 else 1]
else:
    self.mPowers = [0, 0]
    self.direction = [0, 0]
self.driver.setDC(self.mPowers, self.direction)

def run(self):
    self.go = True
    try:
        while self.go:
            # print self.mPowers
            # print self.direction
            self.handleControllerQueue()
            self.handleEncoderQueue()
    except KeyboardInterrupt as msg:
        print "KeyboardInterrupt detected. MotorContoller is terminating"
        self.go = False
    except Exception as msg:
        print "Motor controller"
        print msg
        self.mPowers = [0, 0]
        self.driver.setDC(self.mPowers, [0, 0])
    finally:
        self.exitGracefully()

#####
Encoder.py
#####

import sys, time

import util

# the fastest I have seen it move is 0.511192102610497 m/s

class Encoder:
    count = 0
    pSize = 10
    periods = [-1.0]*pSize
    periodIndex = 0
    timeout = .1
    lastEdge = 0

```

```

def __init__(self):
    pass

def edgeDetected(self, pin):
    self.count += 1
    ctime = time.time()
    elapsedTime = ctime-self.lastEdge
    if elapsedTime <= self.timeout:
        self.periods[self.periodIndex] = elapsedTime
        # increment self.periodIndex and keep it within range of self.pSize =
        len(self.periods)
        self.periodIndex = (self.periodIndex+1)%self.pSize;
    self.lastEdge = ctime
    util.encQueue.put([self.count])

def resetPeriod(self):
    self.periods = [-1]*self.pSize
    self.periodIndex = 0

# returns seconds/blip
def getAveragePeriodBetweenBlips(self):
    ave = 0.0
    i = 0
    for i in range(0, self.pSize):
        if self.periods[i] == -1: # invalid period, therefore return what is got
            break
        else:
            ave += self.periods[i]
    # return average of valid periods, i+1 because i will never equal self.pSize
    if i < 10:
        return -1
    else:
        return ave/(i+1)

# if level = None a stall occurred
def waitForEdgeResponse(self, level, elapsedTime):
    if elapsedTime >= self.timeout: #Stall occurred
        #TODO handle stall
        print "OH NO! A STALL"
        self.count = 0
        self.resetPeriod()
    else:
        self.count += 1
        self.periods[self.periodIndex] = elapsedTime
    # increment self.periodIndex and keep it within range of self.pSize = len(self.periods)
    self.periodIndex = (self.periodIndex+1)%self.pSize;
    util.encQueue.put([self.pin ,self.count, self.getAveragePeriodBetweenBlips()])

#####
util.py
#####

# utility functions

diameterOfWheel = 65.087 # mm
radiusOfWheel = 32.5435 # mm
circumferenceOfWheel = 204.476841044199 # mm diameterOfWheel*pi

```

```

stateChangesPerRevolution = 40.0 # there are 20 slots, but 40 state changes
cirDivChanges = circumferenceOfWheel/stateChangesPerRevolution
distPerBlip = 5.11192102610497 # mm circumferenceOfWheel/stateChangesPerRevolution
maxVel = 511.92102620497 # mm/s distPerBlip/getAverageBlip() when pwm = 100
minVel = .1 # mm/s just a guess //TODO make not a guess
botWidth = 139.7 #mm distance from middle of tire to the other wheel's middle of its tire

# this is different for both RPi and Edison
# TODO, needs to be in encoder file or config.txt
leftEncPin = 11
rightEncPin = 12

microcontroller = None

# used to change pin or pwm values, or to request an input or analog read
controllerQueue = None
gpioQueue = None
encQueue = None
#gcsDataQueue = None
positionQueue = None # TODO
positionTelemQueue = None

# returns a unique identifier for a process
def getIdentfier(process):
    return str(process).split('(')[1].split(',')[0]

def clampToRange(x, lower, upper):
    if x < lower:
        return lower
    elif x > upper:
        return upper
    else:
        return x

# maps x which is in the range of in_min to in_max to x's corresponding
# value between out_min and out_max
def transform(x, in_min, in_max, out_min, out_max):
    in_max = float(in_max)
    return (x - in_min) * (out_max - out_min)/(in_max - in_min) + out_min

#####
config.txt
#####

# This file is used to tell the code what the system configuration is
# The microcontroller, and the motor driver should be specified
# see DDStarter.determineDriver() for how to use comments and what keywords are used
# remember to comment out the unused configuration lines

# config for Raspberry pi using L298 and a USB wifi dongle
#microcontroller = RPi
#driver = L298
#commDriver = Wifi

# config for Intel Edison using L298
microcontroller = Edison
driver = L298

```



```

commDriver = UnixSocket
#commDriver = Xbee
positionDriver = Pozyx

#####
DDStarter.py
#####

#!/usr/bin/env python

# This file was created by Ryan Cooper in 2016
# to control a raspberry pi that is hooked up to motor controls
# that control motors that create a differential drive
# it can be controlled by keyboard or by an commProcess controller (possibly any HCI
  controller)
from multiprocessing import Pipe
from multiprocessing import Queue
from multiprocessing import Manager
import time, sys, argparse, signal

from MotorController import MotorController
import util

class DDStarter:
    # encQueue in makeClasses() is filled by both Lencoder and Rencoder, and is consumed by
    # motorController
    # the array in encQueue is of the form [pin, count, timeSinceLast]
    # controllerQueue in makeClasses is filled by DDMCServer, and is consumed by
    # motorController
    # commands in the queue come from a DDMCClient
    # a manager creates Queues that are safe to share between processes
    motorController = None
    # Differential Drive Motor Controller (DDMC)
    commProcess = None
    gpioProcess = None
    positionerProcess = None
    # pipes are used to terminate processes
    ePipeLeft = None
    ePipeRight = None
    # var that holds microcontroller info from config.txt
    microcontroller = ""

    def __init__(self):
        parser = argparse.ArgumentParser(description="Arguments are for debuggin only.")
        parser.add_argument('--encoders', dest='testEncoder', action="store_true",
            help="Starts encoder system test")
        args = parser.parse_args()
        self.setupIPC()
        if args.testEncoder:
            self.runEncoderTest()
        else:
            self.runNormally()
        # Catch SIGINT from ctrl-c when run interactively.
        signal.signal(signal.SIGINT, self.signal_handler)
        # Catch SIGTERM from kill when running as a daemon.
        signal.signal(signal.SIGTERM, self.signal_handler)
        # This thread of execution will sit here until a signal is caught
        signal.pause()

```

```

def signal_handler(self, signal, frame):
    self.exitGracefully()

def setupIPC(self):
    # used to create multi-process safe queues
    manager = Manager()
    # queues for interprocess communication
    util.encQueue = manager.Queue()
    util.controllerQueue = manager.Queue()
    util.gpioQueue = manager.Queue()
    util.positionQueue = manager.Queue()
    util.positionTelemQueue = manager.Queue()

# TODO
def runEncoderTest(self):
    (motorDriver, commDriver, encoderDriver, gpioDriver) = self.determineDrivers()
    # passing arguments to processes
    self.gpioProcess = gpioDriver([util.leftEncPin, util.rightEncPin])
    self.motorController = MotorController(motorDriver)
    self.motorController.state = self.motorController.ENCODER_TEST
    self.motorController.requiredCounts = 20
    # have to setup pins afterward because gpioProcess needs to be setup first
    self.gpioProcess.start()
    self.motorController.start()
    self.motorController.driver.setDC([30,30], [0,0])

def runNormally(self):
    (motorDriver, commDriver, encoderDriver, gpioDriver, positioner) =
        self.determineDrivers()
    # passing arguments to processes
    self.gpioProcess = gpioDriver([util.leftEncPin, util.rightEncPin])
    self.motorController = MotorController(motorDriver)
    self.commProcess = commDriver()
    self.positionerProcess = positioner()
    # have to setup pins afterward because gpioProcess needs to be setup first
    self.gpioProcess.start()
    self.motorController.start()
    self.commProcess.start()
    self.positionerProcess.start()

def determineDrivers(self):
    sys.path.append('drivers/')
    sys.path.append('GPIO/')
    sys.path.append('Comm/')
    sys.path.append('Positioner/')
    # used to pull configuration from file
    util.microcontroller = ""
    driver = ""
    commDriver = ""
    positionDriver = ""
    conf = open('config.txt', 'r')
    line = conf.readline()
    while line != "":
        # if the first character is '#', this line is a comment
        if line[0] != '#':
            words = line.split()
            if len(words) > 0:
                # in all cases words[1] == '='

```

```

        if words[0] == 'microcontroller':
            util.microcontroller = words[2]
        elif words[0] == 'driver':
            driver = words[2]
        elif words[0] == 'commDriver':
            commDriver = words[2]
        elif words[0] == 'positionDriver':
            positionDriver = words[2]
    line = conf.readline()
conf.close()
motorDriver = None
comm = None
enc = None
gpio = None
postioner = None

if util.microcontroller == 'RPi':
    try:
        import RPiGPIODriver
    except ImportError as err:
        print err
        print "Could not import RPiGPIODriver"
        sys.exit(1)
    gpio = RPiGPIODriver.RPiGPIODriver
elif util.microcontroller == 'Edison':
    try:
        import EdisonGPIODriver
    except ImportError as err:
        print err
        print "Could not import EdisonGPIODriver"
        sys.exit(1)
    gpio = EdisonGPIODriver.EdisonGPIODriver

if driver == 'L298':
    try:
        import L298Driver
    except ImportError as err:
        print "Could not import L298Driver"
        sys.exit(1)
    motorDriver = L298Driver.L298Driver
if commDriver == 'Wifi':
    try:
        import WifiComm
    except ImportError as err:
        print "Could not import Comm/WifiComm"
        sys.exit(1)
    comm = WifiComm.WifiComm
elif commDriver == 'Xbee':
    try:
        import XbeeComm
    except ImportError as err:
        print "Could not import Comm/XbeeComm"
        sys.exit(1)
    comm = XbeeComm.XbeeComm
elif commDriver == 'UnixSocket':
    try:
        import UnixSocketComm
    except ImportError as err:
        print "Could not import Comm/UnixSocketComm"

```

```

        sys.exit(1)
    comm = UnixSocketComm.UnixSocketComm
    if positionDriver == 'Pozyx':
        try:
            import PozyxPositioner
        except ImportError as err:
            print "Could not import Positioner/Pozyx"
            sys.exit(1)
        positioner = PozyxPositioner.PozyxPositioner
    return (motorDriver, comm, enc, gpio, positioner)

def exitGracefully(self):
    try:
        print "Program was asked to terminate."
        print "Waiting for processes to exit..."
        if self.commProcess:
            self.commProcess.join()
        if self.motorController:
            self.motorController.join()
        if self.gpioProcess:
            self.gpioProcess.join()
        if self.positionerProcess:
            self.positionerProcess.join()
        print "Done"
        sys.exit(0)
    except Exception as msg:
        print "An exception occured while trying to terminate"
        print msg
        sys.exit(1)

if '__main__' == __name__:
    r = DDStarter()

#####
Positioner/Positioner.py
#####

#!/usr/bin/env python

from multiprocessing import Process
from multiprocessing import Queue

import signal, time, sys, os
sys.path.append(os.path.abspath('../'))
import util

class PositionerBaseClass(Process):

    def __init__(self):
        super(PositionerBaseClass, self).__init__()

    def getPosition(self):
        raise NotImplementedError("Override getPosition in class that inherits PositionerBaseClass")

    def getHeading(self):

```

```

        raise NotImplementedError("Override getHeading in class that inherits
                                   PositionerBaseClass")

    def sendError(self):
        raise NotImplementedError("Override sendError in class that inherits
                                   PositionerBaseClass")

    def run(self):
        try:
            while self.go:
                pos = self.getPosition()
                mag = self.getHeading()
                if pos:
                    util.positionQueue.put((pos, mag))
                    util.positionTelemQueue.put((pos, mag))
                else:
                    print "pos undef"
                    time.sleep(.01)
        except KeyboardInterrupt as msg:
            print "KeyboardInterrupt detected. CommProcess is terminating"
            self.go = False
        finally:
            self.exitGracefully()

#####
Positioner/PozyxPositioner.py
#####

#!/usr/bin/env python
"""
Heavily based upon the Pozyx ready to localize tutorial (c) Pozyx Labs
Documentation on the code used here can be found at:
https://www.pozyx.io/Documentation/Tutorials/ready\_to\_localize/Python
"""

from time import sleep
import sys, os

from pypozyx import *
import __future__
from Positioner import PositionerBaseClass

sys.path.append(os.path.abspath('../'))
import util

class PozyxPositioner(PositionerBaseClass):
    go = True

    def __init__(self):
        super(PozyxPositioner, self).__init__()
        # shortcut to not have to find out the port yourself
        serial_port = None
        try:
            serial_port = get_serial_ports()[0].device
        except IndexError as msg:
            print("Could not find serial connection to pozyx. Positioner will be turned off
                  for this run")
        if serial_port is not None:
            remote_id = 0x6069          # remote device network ID

```

```

        remote = False                # whether to use a remote device
        if not remote:
            remote_id = None

    # necessary data for calibration, change the IDs and coordinates yourself
    anchors = [DeviceCoordinates(0x6019, 1, Coordinates(0, 0, 196)),
                DeviceCoordinates(0x6049, 1, Coordinates(3874, 0, 232)),
                DeviceCoordinates(0x6044, 1, Coordinates(0, 2451, 174)),
                DeviceCoordinates(0x607F, 1, Coordinates(3874, 2775, 155))]
    algorithm = POZYX_POS_ALG_UWB_ONLY # positioning algorithm to use
    dimension = POZYX_3D #POZYX_3D      # positioning dimension
    height = 1000                # height of device, required in 2.5D positioning
    pozyx = PozyxSerial(serial_port)
    self.initializePozyx(pozyx, anchors, algorithm, dimension, height, remote_id)
else:
    self.go = False

def initializePozyx(self, pozyx, anchors, algorithm=POZYX_POS_ALG_UWB_ONLY,
                    dimension=POZYX_3D, height=1000, remote_id=None):
    self.pozyx = pozyx
    self.anchors = anchors
    self.algorithm = algorithm
    self.dimension = dimension
    self.height = height
    self.remote_id = remote_id
    self.pozyx.clearDevices(self.remote_id)
    self.setAnchorsManual()

def getPosition(self):
    if not self.go:
        return None
    position = Coordinates()
    status = self.pozyx.doPositioning(position, self.dimension, self.height,
                                       self.algorithm, remote_id=self.remote_id)
    if status == POZYX_SUCCESS:
        return str(position.x) + ", " + str(position.y) + ", " + str(position.z)
    else:
        return None

def getHeading(self):
    if not self.go:
        return "0.0"
    orientation = EulerAngles()
    status = self.pozyx.getEulerAngles_deg(orientation)
    return str(orientation.heading)

def publishPosition(self, position):
    network_id = self.remote_id
    if network_id is None:
        network_id = 0
    position.x = position.x / 25.4
    position.y = position.y / 25.4
    position.z = position.z / 25.4
    print "POS ID {}, x(mm): {pos.x} y(mm): {pos.y} z(mm): {pos.z}".format(
        "0x%0.4x" % network_id, pos=position)
    util.positionQueue.put(position)

def sendError(self, operation):
    error_code = SingleRegister()

```

```

network_id = self.remote_id
if network_id is None:
    self.pozyx.getErrorCode(error_code)
    print "ERROR %s, local error code %s" % (operation, str(error_code))
    util.positionQueue.put(str(error_code));
    return
status = self.pozyx.getErrorCode(error_code, self.remote_id)
if status == POZYX_SUCCESS:
    print "ERROR %s on ID %s, error code %s" % (operation, "0x%0.4x" % network_id,
        str(error_code))
    util.positionQueue.put(str(error_code));
    return
else:
    self.pozyx.getErrorCode(error_code)
    print "ERROR %s, couldn't retrieve remote error code, local error code %s" %
        (operation, str(error_code))
    util.positionQueue.put(str(error_code));
    return
    # should only happen when not being able to communicate with a remote Pozyx.

def setAnchorsManual(self):
    """Adds the manually measured anchors to the Pozyx's device list one for one."""
    status = self.pozyx.clearDevices(self.remote_id)
    for anchor in self.anchors:
        status &= self.pozyx.addDevice(anchor, self.remote_id)
    if len(self.anchors) > 4:
        status &= self.pozyx.setSelectionOfAnchors(POZYX_ANCHOR_SEL_AUTO,
            len(self.anchors))
    return status

def publishAnchorConfiguration(self):
    for anchor in self.anchors:
        print "ANCHOR,0x%0.4x,%s" % (anchor.network_id, str(anchor.coordinates))

def exitGracefully(self):
    pass

if __name__ == "__main__":
    p = PozyxPositioner()
    while True:
        pos = p.getPosition()
        head = p.getHeading()
        if pos:
            print(pos)
            print(head)
        sleep(.5)

#####
GPIO/GPIOBaseClass.py
#####

#!/usr/bin/env python

from multiprocessing import Process
from multiprocessing import Queue
import time, os, sys

sys.path.append(os.path.abspath('..'))

```

```

import util
from Encoder import Encoder

class GPIOBaseClass(Process):

    OUTPUT = "Override in inherited class"
    INPUT = "Override in inherited class"
    PWM = "Override in inherited class"
    ANALOG_INPUT = "Override in inherited class"

    encDict = {}

    # child's init should call the super constructor
    # and things like
    #   GPIO.setmode() if raspberry pi
    #   or GPIO(debug=False) if edison
    def __init__(self, encoderPins):
        super(GPIOBaseClass, self).__init__()
        for p in encoderPins:
            e = Encoder()
            self.encDict[p] = e
            a = Process(target=self.setupWaitForEdgeISR, args=(self.edgeDetected, p))
            a.start()

    # pins should be a tuple of which pins to setup
    # modes should be the corresponding mode for each pin in pins
    # should do something like
    # if len(pins) > 1:
    #     for i in range(0, len(pins)):
    #         GPIO.setupPin(pins[i], modes[i]) # or respective code for platform
    # elif len(pins) == 1:
    #     GPIO.setupPin(pins, modes)
    def setup(self, pins, modes):
        raise NotImplementedError("Override setup in class that inherits GPIOBaseClass")

    # pins should be a tuple of which pins to use for pwm
    # frequencies should be the corresponding PWM wave frequencies in herts for each pin
    # should be very similar to setupPins
    def setupPWM(self, pins, frequencies):
        raise NotImplementedError("Override setupPWM in class that inherits GPIOBaseClass")

    # args should be tuples, lists or a single int
    # changes the frequencies of the pwm signals on pins
    def changeFrequency(self, pins, frequencies):
        raise NotImplementedError("Override changeFrequency in class that inherits
GPIOBaseClass")

    # args should be tuples, lists or a single int
    # sets the duty cycle of the pwm signal on the pwm pins based on powers
    # value should be between 0-100, but some implementations may have an actual resolution
    # of 255
    def setDC(self, pins, values):
        raise NotImplementedError("Override setDC in class that inherits GPIOBaseClass")

    # args should be tuples, lists or a single int
    # writes the values in levels to the corresponding pin in pins
    def write(self, pins, levels):
        raise NotImplementedError("Override writeToPin in class that inherits
GPIOBaseClass")

```



```

# pin is not a list, or tuple! It is a single pin
# _read should return the digital value of the pin, do a digitalRead()
def _read(self, pin):
    raise NotImplementedError("Override _read in class that inherits GPIOBaseClass")

# pin is not a list, or tuple! It is a single pin
# _analogRead should return the analogReading of the pin, doesn't make sense with RPi,
# need adc
def _analogRead(self, pin):
    raise NotImplementedError("Override _analogRead in class that inherits
GPIOBaseClass")

# should still override this function to cleanup gpio and pwm stuff
# but also make sure to call this function from the super class
def exitGracefully(self):
    # this is done to break out of the while loop in run so process terminates
    util.gpioQueue = None
    #for p in self.waitingProcs:
    #    p.join()

def consumeQueue(self):
    while not util.gpioQueue.empty():
        a = util.gpioQueue.get_nowait()
        if a:
            if a[0] == 'setup':
                # a[1] is pins
                # a[2] is modes
                self.setup(a[1], a[2])
            elif a[0] == 'setupPWM':
                # a[1] is pins
                # a[2] is frequencies
                self.setupPWM(a[1], a[2])
            elif a[0] == 'changeFrequency':
                # a[1] is pins
                # a[2] is frequencies
                self.changeFrequency(a[1], a[2])
            elif a[0] == 'setDC':
                # a[1] is pins
                # a[2] is values
                self.setDC(a[1], a[2])
            elif a[0] == 'write':
                # a[1] is pins
                # a[2] is levels
                self.write(a[1], a[2])
            elif a[0] == 'exitGracefully':
                self.exitGracefully()
            elif a[0] == 'analogRead':
                # a[1] = uniqueProcessIdentifier
                # a[2] = pin to read
                raise NotImplementedError("analogRead is unsupported")
            elif a[0] == 'resetEncoders':
                for key in encDict:
                    encDict[key].resetPeriod()

# it is assumed that the pin is already setup
def setupWaitForEdgeISR(self, callback, pin):
    raise NotImplementedError("Override _setupWaitForEdgeISR in class that inherits
GPIOBaseClass")

```

```

def edgeDetected(self, pin):
    print "hello"
    self.encDict[pin].count += 1
    ctime = time.time()
    elapsedTime = ctime-self.encDict[pin].lastEdge
    if elapsedTime <= self.encDict[pin].timeout:
        self.encDict[pin].periods[self.encDict[pin].periodIndex] = elapsedTime
        # increment self.encDict[pin].periodIndex and keep it within range of
        self.encDict[pin].pSize = len(self.encDict[pin].periods)
        self.encDict[pin].periodIndex =
            (self.encDict[pin].periodIndex+1)%self.encDict[pin].pSize;
    self.encDict[pin].lastEdge = ctime
    util.enqueue.put([self.encDict[pin].count])

def resetPeriod(self):
    self.encDict[pin].periods = [-1]*self.encDict[pin].pSize
    self.encDict[pin].periodIndex = 0

# returns seconds/blip
def getAveragePeriodBetweenBlips(self):
    ave = 0.0
    i = 0
    for i in range(0, self.encDict[pin].pSize):
        if self.encDict[pin].periods[i] == -1: # invalid period, therefore return what
            is got
            break
        else:
            ave += self.encDict[pin].periods[i]
    # return average of valid periods, i+1 because i will never equal self.pSize
    if i < 10:
        return -1
    else:
        return ave/(i+1)

def run(self):
    try:
        a = None
        while util.gpioQueue:
            self.consumeQueue()
    except KeyboardInterrupt as msg:
        print "KeyboardInterrupt detected. GPIOProcess is terminating"
    finally:
        self.exitGracefully()

#####
GPIO/EdisonGPIDriver.py
#####

#!/usr/bin/env python

from GPIOBaseClass import GPIOBaseClass
import mraa
import sys, os

class EdisonGPIDriver(GPIOBaseClass):
    OUTPUT = mraa.DIR_OUT
    INPUT = mraa.DIR_IN

```

```

PWM = ""
ANALOG_INPUT = ""
gpioDict = {}
pwmDict = {}

def __init__(self, encoderPins):
    super(EdisonGPIDriver, self).__init__(encoderPins)

# args should be tuples, list, or a single int
def setup(self, pins, modes):
    if type(pins) is list or type(pins) is tuple:
        for i in range(0, len(pins)):
            if modes[i] == 'INPUT':
                self.gpioDict[pins[i]] = mraa.Gpio(pins[i])
                self.gpioDict[pins[i]].dir(self.INPUT)
            elif modes[i] == 'OUTPUT':
                self.gpioDict[pins[i]] = mraa.Gpio(pins[i])
                self.gpioDict[pins[i]].dir(self.OUTPUT)
            elif modes[i] == 'ANALOG_INPUT':
                pass
            elif modes[i] == 'PWM':
                self.setupPWM(pins[i], 60)
    else:
        if modes == 'INPUT':
            self.gpioDict[pins] = mraa.Gpio(pins)
            self.gpioDict[pins].dir(self.INPUT)
        elif modes == 'OUTPUT':
            self.gpioDict[pins] = mraa.Gpio(pins)
            self.gpioDict[pins].dir(self.OUTPUT)
        elif modes == 'ANALOG_INPUT':
            pass
        elif modes == 'PWM':
            self.setupPWM(pins, 60)

# args should be tuples, lists, or a single int
def setupPWM(self, pins, frequencies):
    if type(pins) is list or type(pins) is tuple:
        for i in range(0, len(pins)):
            self.pwmDict[pins[i]] = [mraa.Pwm(pins[i]), False]
            self.pwmDict[pins[i]][0].period(1.0/frequencies[i])
    else: # must be an int
        self.pwmDict[pins] = [mraa.Pwm(pins), False]
        self.pwmDict[pins][0].period(1.0/frequencies)

# args should be tuples, lists, or a single int
# it is assumed that the entries of pins are already setup as PWM outputs
def changeFrequency(self, pins, frequencies):
    if type(pins) is list or type(pins) is tuple:
        for i in range(0, len(pins)):
            self.pwmDict[pins[i]][0].period(1.0/frequencies[i])
    else: # must be an int
        self.pwmDict[pins][0].period(1.0/frequencies)

# args should be tuples, lists, or a single int
# it is assumed that the entries of pins are already setup as PWM outputs
def setDC(self, pins, values):
    if type(pins) is list or type(pins) is tuple:
        for i in range(0, len(pins)):
            if values[i] >= 0 and values[i] <= 100:

```

```

        if values[i] != 0:
            if not self.pwmDict[pins[i]][1]:
                self.pwmDict[pins[i]][0].enable(True)
                self.pwmDict[pins[i]][1] = True
                self.pwmDict[pins[i]][0].write(values[i]/100.0)
            else:
                self.pwmDict[pins[i]][0].write(0.0)
                #self.pwmDict[pins[i]][0].enable(False)
                self.pwmDict[pins[i]][1] = False
        else:
            print "Incorrect duty cycle value was provided"
    else: # must be an int
        if values >= 0 and values <= 100:
            if values != 0:
                if not self.pwmDict[pins][1]:
                    self.pwmDict[pins][0].enable(True)
                    self.pwmDict[pins][1] = True
                    self.pwmDict[pins][0].write(values/100.0)
                else:
                    self.pwmDict[pins][0].write(0.0)
                    #self.pwmDict[pins][0].enable(False)
                    self.pwmDict[pins][1] = False

# args should be tuples, lists, or a single int
# it is assumed that the pins are already setup to be outputs
def write(self, pins, levels):
    if type(pins) is list or type(pins) is tuple:
        for i in range(0, len(pins)):
            self.gpioDict[pins[i]].write(levels[i])
    else: # must be an int
        self.gpioDict[pins].write(levels)

def setupWaitForEdgeISR(self, callback, pin):
    g = mraa.Gpio(pin)
    g.dir(self.INPUT)
    g.isr(mraa.EDGE_BOTH, self.edgeDetected, self.edgeDetected)

# ait is assumed that the pin was setup to be a self._gpio.INPUT before this is called
def _read(self, pin):
    return self.gpioDict[pin].read()

# ait is assumed that the pin was setup to be a self._gpio.ANALOG_INPUT before this is
# called
def _analogRead(self, pin):
    pass

def exitGracefully(self):
    super(EdisonGPIDriver, self).exitGracefully()

if __name__ == '__main__':
    from multiprocessing import Manager
    from multiprocessing import Queue
    import time, sys, os
    sys.path.append(os.path.abspath('.'))
    import util
    m = Manager()
    util.gpioQueue = m.Queue()
    e = EdisonGPIDriver([])
    e.start()

```

```

util.gpioQueue.put(["setup", [5, 6, 7, 8], ["OUTPUT", "OUTPUT", "OUTPUT", "OUTPUT"]])
util.gpioQueue.put(["setup", [3, 9], ["PWM", "PWM"]])
util.gpioQueue.put(["write", [5, 6, 7, 8], [0, 1, 0, 1]])
util.gpioQueue.put(["setDC", [3, 9], [25,50]])
time.sleep(1)
util.gpioQueue.put(["setDC", [3, 9], [0,0]])
time.sleep(1)
util.gpioQueue = None

#####
GPIO/RPiGPIDriver.py
#####

#!/usr/bin/env python

from GPIOBaseClass import GPIOBaseClass
import RPi.GPIO as GPIO

# remember this is a Process
class RPiGPIDriver(GPIOBaseClass):
    OUTPUT = GPIO.OUT
    INPUT = GPIO.IN
    PWM = GPIO.OUT
    ANALOG_INPUT = "Raspberry pi doesn't have analog in"

    # dictionary of pins and pwmObjs
    # _pwmObjs is of the form {pin: (pwmObj, pwmStarted), ...}
    _pwmObjs = {}

    def __init__(self, encoderPins):
        GPIO.setmode(GPIO.BOARD)
        GPIO.setwarnings(False)
        super(RPiGPIDriver, self).__init__(encoderPins)

    # args should be tuples, lists, or a single ints
    def setup(self, pins, modes):
        if type(pins) is list or type(pins) is tuple:
            for i in range(0, len(pins)):
                if modes[i] == 'INPUT':
                    GPIO.setup(pins[i], self.INPUT, pull_up_down=GPIO.PUD_DOWN)
                elif modes[i] == 'OUTPUT' or modes[i] == 'PWM':
                    GPIO.setup(pins[i], self.OUTPUT)
                elif modes[i] == 'ANALOG_INPUT':
                    GPIO.setup(pins[i], self.ANALOG_INPUT)
            else: # must be ints
                if modes == 'INPUT':
                    GPIO.setup(pins, self.INPUT, pull_up_down=GPIO.PUD_DOWN)
                elif modes == 'OUTPUT' or modes == 'PWM':
                    GPIO.setup(pins, self.OUTPUT)
                elif modes == 'ANALOG_INPUT':
                    GPIO.setup(pins, self.ANALOG_INPUT)

    # args should be tuples, lists, or a single ints
    def setupPWM(self, pins, frequencies):
        if type(pins) is list or type(pins) is tuple:
            # create a pwmObj for every pin in pins that does not have one associated with it
            for i in range(0, len(pins)):
                if not pins[i] in self._pwmObjs:

```

```

        self._pwmObjs[pins[i]] = [GPIO.PWM(pins[i], frequencies[i]), False]
    else: # must be ints
        if not pins in self._pwmObjs:
            self._pwmObjs[pins] = [GPIO.PWM(pins, frequencies), False]

# args should be tuples, lists, or a single ints
def changeFrequency(self, pins, frequencies):
    if type(pins) is list or type(pins) is tuple:
        for i in range(0, len(pins)):
            # highly frowned upon if trying to change frequency of a pwm obj that hasn't
            # been made yet
            # yet still acceptable, but is redundant and slow
            if not pins[i] in self._pwmObjs:
                self.setupPWM(pins[i], frequencies[i])
            else:
                self._pwmObjs[pins[i]][0].ChangeFrequency(frequencies[i])
    else: # must be ints
        if not pins in self._pwmObjs:
            self.setupPWM(pins, frequencies)
        else:
            self._pwmObjs[pins][0].ChangeFrequency(frequencies)

# args should be tuples, lists, or a single ints
def setDC(self, pins, values):
    if type(pins) is list or type(pins) is tuple:
        for i in range(0, len(pins)):
            if pins[i] in self._pwmObjs:
                if values[i] == 0:
                    self._pwmObjs[pins[i]][0].stop()
                    self._pwmObjs[pins[i]][1] = False
                # if values[i] is valid
                elif values[i] > 0 and values[i] <= 100:
                    # if pwm is active/not stopped/ started
                    if self._pwmObjs[pins[i]][1]:
                        self._pwmObjs[pins[i]][0].ChangeDutyCycle(values[i])
                    else:
                        self._pwmObjs[pins[i]][0].start(values[i])
                        self._pwmObjs[pins[i]][1] = True
                else:
                    print "Incorrect duty cycle value was provided"
            else:
                print "Process tried to set pin that was not setup as a PWM pin to have a
                    duty cycle"
    else: # must be ints
        if pins in self._pwmObjs:
            if values == 0:
                self._pwmObjs[pins][0].stop()
                self._pwmObjs[pins][1] = False
            # if values is valid
            elif values > 0 and values <= 100:
                # if pwm is active/not stopped/ started
                if self._pwmObjs[pins][1]:
                    self._pwmObjs[pins][0].ChangeDutyCycle(values)
                else:
                    self._pwmObjs[pins][0].start(values)
                    self._pwmObjs[pins][1] = True
            else:
                print "Incorrect duty cycle value was provided"
        else:

```

```

        print "Process tried to set pin that was not setup as a PWM pin to have a
            duty cycle"

# args should be tuples, lists, or a single ints
# it is assumed that the pins are already setup to be outputs
def write(self, pins, levels):
    if type(pins) is list or type(pins) is tuple:
        for i in range(0, len(pins)):
            GPIO.output(pins[i], levels[i])
    else: # must be ints
        GPIO.output(pins, levels)

def setupWaitForEdgeISR(self, callback, pin):
    GPIO.setup(pin, self.INPUT, pull_up_down=GPIO.PUD_DOWN)
    GPIO.add_event_detect(pin, GPIO.BOTH, callback=callback)

# it is assumed that pin is setup to be GPIO.INPUT
def _read(self, pin):
    return GPIO.input(pin)

def _analogRead(self, pin):
    print ANALOG_INPUT
    return -1

def exitGracefully(self):
    super(RPiGPIODriver, self).exitGracefully()
    GPIO.cleanup()

#####
drivers/L298Driver.py
#####

#!/usr/bin/env python

import sys, os
sys.path.append(os.path.abspath('..'))
import util

class L298Driver:
    pwmPins = []
    dirPins= []
    maxDC = 100
    minDC = 20

    def __init__(self):
        # TODO replace this with a read from a file
        if util.microcontroller == 'Edison':
            self.pwmPins = [3, 9]
            self.dirPins = [[5, 6], [7, 8]]
        elif util.microcontroller == 'RPi':
            self.pwmPins = [38, 37]
            self.dirPins = [[31,32], [33,35]]
        util.gpioQueue.put(['setup', self.pwmPins, ['PWM', 'PWM']])
        util.gpioQueue.put(['setupPWM', self.pwmPins, [60, 60]])
        util.gpioQueue.put(['setup', self.dirPins[0], ['OUTPUT', 'OUTPUT']])
        util.gpioQueue.put(['setup', self.dirPins[1], ['OUTPUT', 'OUTPUT']])

    def setDirection(self, direction):

```

```

        for i in range(0, 2):
            if direction[i]:
                util.gpioQueue.put(['write', self.dirPins[i][0], 1])
                util.gpioQueue.put(['write', self.dirPins[i][1], 0])
            else:
                util.gpioQueue.put(['write', self.dirPins[i][0], 0])
                util.gpioQueue.put(['write', self.dirPins[i][1], 1])

# set PWM duty cycle
# powers should be an array with 2 indexes [mLeftPower, mRightPower]
# direction should be an array with 2 indexes [mLeftDir, mRightDir]
def setDC(self, powers, direction):
    self.setDirection(direction)
    for i in range(0, len(powers)):
        util.gpioQueue.put(['setDC', self.pwmPins[i], powers[i]])

#####
Comm/CommBaseClass.py
#####

#!/usr/bin/env python

from multiprocessing import Process
from multiprocessing import Queue
# import signal so that main threads try except statement will catch keyboard interrupt
import thread, signal, struct, time, sys, os
sys.path.append(os.path.abspath('../'))
import util

class CommBaseClass(Process):
    # This queue is filled with the data that is sent from ground control to the robot
    # it is consumed by the pilot (whenever that gets written)
    recvQueue = None
    # This is the queue that gets filled by the position process.
    # it is consumed in this process and the data is sent to what is sending commands
    # to the robot
    sendQueue = None
    # This is to terminate the threads that are started
    go = True
    # This is to know how many bytes to read in recv(). recv() is implemented in child class
    # can also be set in handleIncomingData()
    #bytesToRead = 12
    bytesToRead = 6

    def __init__(self):
        super(CommBaseClass, self).__init__()
        self.recvQueue = util.controllerQueue
        self.sendQueue = util.positionTelemQueue

    def waitForConnection(self):
        raise NotImplementedError("Override waitForConnection in class that inherits
CommBaseClass")

# override this function in inherited class, but make sure to call this function in
# the beginning. For example:
# def resetClient(self, waitForReconnect = True):
#     super(ChildClass, self).resetClient(waitForReconnect)
#     somecodes()

```



```

def resetClient(self):
    print "Controller disconnected!"
    # Stop the bot
    self.recvQueue.put([3,1500,1500])
    # remember to inherit, override, and call super class function!!!

# should be blocking
# returns a list of the bytes recieved from a connection in the child class
def recv(self):
    raise NotImplementedError("Override recv in class that inherits CommBaseClass")

# don't override this unless necessary
# This is called from run()
def handleIncomingData(self):
    self.waitForConnection()
    while self.go:
        data = self.recv()
        if len(data) == 0:
            self.resetClient()
            self.waitForConnection()
        elif len(data) == self.bytesToRead: # fill recvQueue for pilot to consume
            srcdstids = struct.unpack('<B', data[0])[0]
            controlScheme = struct.unpack('<B', data[1])[0]
            if controlScheme != 4: # defined as VELOCITY_HEADING in MotorController.py
                lm = struct.unpack('<h', data[2:4])[0] # left motor,
                    steering, or velocity
                rm = struct.unpack('<h', data[4:6])[0] # right motor, throttle, or
                    heading
            self.recvQueue.put([
                controlScheme, # control scheme
                lm,
                rm])
        elif False:
            vel = struct.unpack('<i', data[4:8])[0]
            heading = struct.unpack('<f', data[8:12])[0]
            self.recvQueue.put([
                controlScheme,
                vel,
                heading])

# sends data across the connection in the child class
def send(self, data):
    raise NotImplementedError("Override send in class that inherits CommBaseClass")

def handleOutgoingData(self):
    while not self.sendQueue.empty():
        d = self.sendQueue.get_nowait()
        a = ''
        for i, p in enumerate(d) :
            if p is not None:
                a += p
                if i != len(d)-1:
                    a += ", "
        a += ";"
        self.send(a)

def exitGracefully(self):
    raise NotImplementedError("Override exitGracefully in class that inherits
        CommBaseClass")

```

```

def run(self):
    try:
        t = thread.start_new_thread(self.handleIncomingData, ())
        while self.go:
            self.handleOutgoingData()
            time.sleep(.01)
    except KeyboardInterrupt as msg:
        print "KeyboardInterrupt detected. CommProcess is terminating"
        self.go = False
    finally:
        self.exitGracefully()

#####
Comm/UnixSocketComm.py
#####

#!/usr/bin/env python

#DDMC=Differential Drive Motor Control

# This file was created by Ryan Cooper in 2016
# This class starts a server to listen to an xbox controller which is plugged into a
# client computer that can connect to the robot
import socket, time, sys

from multiprocessing import Process
from multiprocessing import Queue
from multiprocessing import Pipe

from CommBaseClass import CommBaseClass

class UnixSocketComm(CommBaseClass):
    sock = None
    connected = False

    def __init__(self):
        super(UnixSocketComm, self).__init__()
        self.sock = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
        #self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # TODO figure out
        # if this is necessary
        # listen for only one connection

    def waitForConnection(self):
        while not self.connected:
            try:
                self.sock.connect('uds_socket')
                self.connected = True
            except socket.error, msg:
                print "couldn't connect to unix domain socket. Is the java server running?"
                print >>sys.stderr, msg

    def resetClient(self):
        super(UnixSocketComm, self).resetClient()
        self.sock.close()
        self.sock = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)

    def recv(self):

```

```

        if self.connected:
            return self.sock.recv(self.bytesToRead)
        else:
            return 0

    def handleIncomingData(self):
        try:
            super(UnixSocketComm, self).handleIncomingData()
        except socket.error as msg:
            self.resetClient()
            self.waitForConnection()

    def send(self, data):
        if self.connected:
            self.sock.sendall(data)

    def exitGracefully(self):
        if self.sock:
            self.sock.close()

#####
Comm/WifiComm.py
#####

#DDMC=Differential Drive Motor Control

# This file was created by Ryan Cooper in 2016
# This class starts a server to listen to an xbox controller which is plugged into a
# client computer that can connect to the robot
import socket, time, sys
from socket import error as socket_error

from multiprocessing import Process
from multiprocessing import Queue
from multiprocessing import Pipe

from CommBaseClass import CommBaseClass

class WifiComm(CommBaseClass):
    serversocket = None
    clientsocket = None

    def __init__(self):
        super(WifiComm, self).__init__()
        self.serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.serversocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.serversocket.bind('', 12345)
        # listen for only one connection
        self.serversocket.listen(1)
        print "Wifi server started"

    def waitForConnection(self):
        if self.clientsocket == None:
            connected = False
            while not connected:
                try:
                    sys.stdout.write('Waiting for DDMCClient connection... ')

```

```

        sys.stdout.flush()
        (self.clientsocket, address) = self.serversocket.accept()
        connected = True
        print 'DDMC controller connected'
    except Exception as msg:
        print 'Client connection failed with message:'
        print msg
        print 'I will retry connecting in one second.'
        time.sleep(1)

    def resetClient(self):
        super(WifiComm, self).resetClient()
        if self.clientsocket:
            self.clientsocket.close()
        self.clientsocket = None

    def recv(self):
        if self.clientsocket:
            return self.clientsocket.recv(self.bytesToRead)
        else:
            return 0

    def handleIncomingData(self):
        try:
            super(WifiComm, self).handleIncomingData()
        except socket_error as msg:
            self.resetClient()
            self.waitForConnection()

    def send(self, data):
        data = data + '\n';
        if self.clientsocket:
            self.clientsocket.send(data)

    def exitGracefully(self):
        if self.clientsocket:
            self.clientsocket.close()
        if self.serversocket:
            self.serversocket.close()

#####
Comm/XbeeComm.py
#####

#!/usr/bin/env python

# to install xbee:
# wget
# https://pypi.python.org/packages/53/13/c3f53a63b5a9d8890e7911580c466b25a90b732a500d437a1205fef47a68/XBee-2.
# tar -xvf Xbee-2.2.3.tar
# cd Xbee-2.2.3
# python setup.py install
from xbee import XBee
# install with:
# pip install pyserial
import serial

from CommBaseClass import CommBaseClass

```

```

import threading

class XbeeComm(CommBaseClass):

    ser = None
    xbee = None
    # variable used to make sure there is no race condition on recvData
    dataCondition = None
    recvData = None
    # Hopefully this is the coordinator
    dest_addr = 0

    def __init__(self):
        super(XbeeComm, self).__init__()
        self.ser = serial.Serial('/dev/ttyAMA0', 9600)
        self.dataCondition = threading.Condition()
        self.xbee = XBee(self.ser, callback=self.fillRecv)

    def waitForConnection(self):
        pass

    def resetClient(self):
        pass

    # I decided to make use of an asynchronous callback to recieve data instead
    # of the xbee.wait_read_frame() because I wanted this process's thread to wait
    # not the Xbees thread to wait
    def fillRecv(self, data):
        self.dataCondition.acquire()
        self.recvData = data
        self.dataCondition.release()

    def recv(self):
        # possible use this function for synchronous mode. Will it work in threads?
        # return self.xbee.wait_read_frame()

        # wait until self.fillRecv(data) fills recvData
        self.dataCondition.wait()
        self.dataCondition.acquire()
        d = self.recvData
        self.recvData = None
        self.dataCondition.release()
        return d

''' Put here for reference, taken from python module xbee.ieee.Xbee
#Format:
#       {name of command:
#         [{name:field name, len:field length, default: default value sent}
#         ...
#         ]
#       ...
#       }
api_commands = {"at":
                [{ 'name':'id',      'len':1,      'default':b'\x08'},
                  { 'name':'frame_id', 'len':1,      'default':b'\x00'},
                  { 'name':'command',  'len':2,      'default':None},
                  { 'name':'parameter', 'len':None, 'default':None}],
                "queued_at":
                [{ 'name':'id',      'len':1,      'default':b'\x09'},

```

```

        {'name':'frame_id', 'len':1, 'default':b'\x00'},
        {'name':'command', 'len':2, 'default':None},
        {'name':'parameter', 'len':None, 'default':None}],
    "remote_at":
        [{ 'name':'id', 'len':1, 'default':b'\x17'},
          { 'name':'frame_id', 'len':1, 'default':b'\x00'},
          # dest_addr_long is 8 bytes (64 bits), so use an unsigned long long
          { 'name':'dest_addr_long', 'len':8, 'default':struct.pack('>Q', 0)},
          { 'name':'dest_addr', 'len':2, 'default':b'\xFF\xFE'},
          { 'name':'options', 'len':1, 'default':b'\x02'},
          { 'name':'command', 'len':2, 'default':None},
          { 'name':'parameter', 'len':None, 'default':None}],
    "tx_long_addr":
        [{ 'name':'id', 'len':1, 'default':b'\x00'},
          { 'name':'frame_id', 'len':1, 'default':b'\x00'},
          { 'name':'dest_addr', 'len':8, 'default':None},
          { 'name':'options', 'len':1, 'default':b'\x00'},
          { 'name':'data', 'len':None, 'default':None}],
    "tx":
        [{ 'name':'id', 'len':1, 'default':b'\x01'},
          { 'name':'frame_id', 'len':1, 'default':b'\x00'},
          { 'name':'dest_addr', 'len':2, 'default':None},
          { 'name':'options', 'len':1, 'default':b'\x00'},
          { 'name':'data', 'len':None, 'default':None}]
    }
'''

def send(self, d):
    self.xbee.send("tx", data=d, dest_addr=self.dest_addr)

def exitGracefully(self):
    # need to halt to stop process that is listening for data
    self.xbee.halt()
    self.ser.close()

def testControllerQueue():
    while not util.controllerQueue.empty(): # this is a while so that the most recent thing
        in the queue is the resultant command that is done
        good = True
        try:
            # nowait because this process was called from the main loop which controls the
            # motors
            # so we don't want this function to block.
            data = util.controllerQueue.get_nowait()
        except Queue.Empty as msg:
            # realistically this should never happen because we check to see that the queue
            # is not empty
            # but it is shared memory, and who knows?
            good = False
        if good:
            print data

if __name__ == '__main__':
    from multiprocessing import Manager
    manager = Manager()
    util.controllerQueue = manager.Queue()
    util.gcsDataQueue = manager.Queue()
    xb = XbeeComm()
    xb.start()
    thread.start_new_thread(testControllerQueue, ())

```

```

        time.sleep(5)
    for i in range(3):
        util.gcsDataQueue.put(['derp'])
        time.sleep(10)

```

```

#####
DataturbineExporter.java
#####

// Written by Scot Tomer
package edu.scu.engr.rs1.deca;

import java.io.File;

import com.rbnb.sapi.SAPIException;

import com.etsy.net.JUDS;
import com.etsy.net.UnixDomainSocketServer;

public class DataturbineExporter {
    /**
     * Sets up the send and receive threads based on arguments passed in. Expected
     * to be run from the command line.
     * @param arg
     *      An array of three Strings, containing the full path to the unix
     *      domain socket to listen on, the dataturbine server host and port,
     *      and the name of the robot (for setting up the dataturbine source and
     *      sink).
     */
    public static void main(String[] arg) throws Exception {
        try {
            File danglingSock = new File( arg[1] );
            danglingSock.delete( );
            // If this fails due to permissions error, the socket constructor will
            // raise an exception, so just gobble all exceptions here.
        } catch ( Exception error ) { }

        UnixDomainSocketServer server = new UnixDomainSocketServer( arg[1],
            JUDS.SOCK_STREAM );
        Runtime.getRuntime( ).addShutdownHook( new SocketCleanup( server ) );

        Sender sender;
        Receiver receiver;
        try {
            sender = new Sender( arg[0], arg[2], server.getInputStream( ) );
            receiver = new Receiver( arg[0], arg[2], server.getOutputStream( ) );
        } catch ( SAPIException error ) {
            System.out.println( "Could not connect to dataturbine." );
            return;
        }

        sender.start( );
        receiver.start( );

        sender.join( );
        receiver.join( );
    }
}

```

```

#####
Receiver.java
#####

package edu.scu.engr.rs1.deca;

import java.io.IOException;
import java.io.OutputStream;

import com.rbnb.sapi.Sink;
import com.rbnb.sapi.ChannelMap;
import com.rbnb.sapi.SAPIException;

public class Receiver extends Thread {
    private Sink internalSink;
    private OutputStream client;

    public Receiver( String dataTurbine, String botName , OutputStream out ) throws
        Exception {
        internalSink = new Sink( );
        internalSink.OpenRBNBConnection( dataTurbine, botName + "-sink" );
        ChannelMap watchList = new ChannelMap( );
        watchList.Add( "controller/" + botName );
        internalSink.Subscribe( watchList );
        client = out;
    }

    @Override
    public void run ( ) {
        while ( true ) {
            // The argument to Fetch is read timeout in ms. If there is no data,
            // it will eventually time out and return an empty channel map. I
            // actually have no idea what circumstances it will throw an
            // SAPIException under, because killing rbnb causes it to throw
            // java.lang.IllegalStateException.
            ChannelMap getmap;
            try {
                getmap = internalSink.Fetch( 1000 );
            } catch ( SAPIException error ) {
                System.out.println( "SAPIException: " + error.getMessage( ) );
                break;
            } catch ( IllegalStateException error ) {
                // this occurs if the rbnb server shuts down while we are waiting for
                // a client read.
                System.out.println( "Dataturbine has vanished." );
                break;
            }
            if ( getmap.NumberOfChannels( ) > 0 ) {
                byte[] message = getmap.GetDataAsByteArray( 0 )[0];
                try {
                    client.write( message );
                } catch ( IOException error ) {
                    System.out.println( "Client write error: " + error.getMessage( ) );
                    break;
                }
            }
        }
        System.exit( 1 );
    }
}

```



```

    }
}

#####
Sender.java
#####

package edu.scu.engr.rsl.deca;

import java.util.HashMap;
import java.io.IOException;
import java.io.InputStream;

import com.rbnb.sapi.Source;
import com.rbnb.sapi.ChannelMap;
import com.rbnb.sapi.SAPIException;

public class Sender extends Thread {
    // private int channel;
    private HashMap<String, Integer> chanMap;
    private Source internalSource;
    private ChannelMap outputChannels;
    private InputStream client;

    public Sender( String dataTurbine, String botName, InputStream in ) throws Exception {
        internalSource = new Source( );
        outputChannels = new ChannelMap( );
        chanMap = new HashMap<String, Integer>( );
        chanMap.put( "states", outputChannels.Add( "states" ) );
        internalSource.OpenRBNBConnection( dataTurbine, botName );
        client = in;
    }

    @Override
    public void run ( ) {
        while ( true ) {
            byte[] message = new byte[512];
            try {
                int bytesRead;
                try {
                    bytesRead = client.read( message );
                } catch ( IllegalStateException error ) {
                    // this occurs if the rbnb server shuts down while we are waiting for
                    // a client read.
                    System.out.println( "Dataturbine has vanished." );
                    break;
                }
            }
            if ( bytesRead < 0 ) {
                System.out.println( "client disconnected." );
                break;
            } else if ( bytesRead > 0 ) {
                String packet = new String( message, 0, bytesRead );
                try {
                    outputChannels.PutDataAsString( chanMap.get( "states" ), packet );
                    internalSource.Flush( outputChannels, true );
                } catch ( SAPIException error ) {
                    System.out.println( "barf: " + error.getMessage( ) );
                    break;
                }
            }
        }
    }
}

```

```

        }
    }
    } catch ( IOException error ) {
        System.out.println( "dang" + error.getMessage( ) );
        break;
    }
}
System.exit( 1 );
}
}

```

```

#####
SocketCleanup.java
#####

```

```

package edu.scu.engr.rsl.deca;

import com.etsy.net.UnixDomainSocketServer;

class SocketCleanup extends Thread {
    UnixDomainSocketServer server;

    public SocketCleanup( UnixDomainSocketServer server ) {
        this.server = server;
    }

    @Override
    public void run( ) {
        if (server != null) {
            System.out.println( "Cleaning up socket..." );
            server.unlink( );
        }
    }
}

```

```

=====
InstallPrerequisites.sh
=====

```

```
#!/bin/bash
```

```
cd ../
```

```
pip install pyserial
```

```
git clone https://github.com/pozyxLabs/Pozyx-Python-library.git && cd Pozyx-Python-library
python setup.py install
cd ../
```

```
git clone Pillager225@gitlab.com:DecaBot/JUDS.git && cd JUDS
./autoconf.sh && JAVA_HOME=/usr/lib/jvm/java-8-openjdk ./configure
make -j2 && mv juds-*.jar juds.jar
cd ../
```

```
git clone https://Pillager225@gitlab.com/DecaBot/DataturbineExporter.git
cp JUDS/juds.jar DataturbineExporter
cd DataturbineExporter
make -j2 JAR=/usr/lib/jvm/java-8-openjdk/bin/jar
```

```

cd ../

cp DataturbineExporter/build/DataturbineExporter.jar DifferentialDrive/
cp JUDES/juds.jar DifferentialDrive/

#####
StartRobot.sh
#####

#!/bin/bash
java -jar DataturbineExporter.jar 192.168.43.38:3333 uds_socket robot_1&
./DDStarter.py

```

A.3.2 WebUI

```

#####
ui/index.html
#####

<!DOCTYPE html>
<html>
  <head>
    <title>MECH</title>
    <link rel="stylesheet" type="text/css" href="main.css">
  </head>
  <body>
    <div id="title-div" class="unselectable">Multi-bot Easy Control Hierarchy</div>
    <div id="page_table-div">
      <table id="page-table" class="page-table">
        <tr id="page_table_row_1">
          <td id="page_table_col_11" rowspan="2">
            <div id="robot-container">
              <div id="robot-container-title" class="generic-title-with-button
                unselectable">Robots
              <button type="button" onclick="removeDisconnected()" id="add-robot-button"
                class="generic-button unselectable">Remove Disconnected</button>
            </div>
            <div id="robot-item-container"></div>
          </div>
          <td id="page_table_col_12">
            <div id="graphics-container">
              <div id="graphics-container-title" class="generic-title
                unselectable">Graphics</div>
              <div id="canvas-container">
                <canvas id="canvas" class="unselectable"> </canvas>
              </div>
            </div>
          </td>
        </tr>
        <tr id="page_table_row_2">
          <td id="page_table_col_21">
            <div id="configuration-container">
              <div id="configuration-container-title" class="generic-title
                unselectable">Configuration</div>
              <div id="configuration-dropdown-div" class="dropdown unselectable">

```

```

        <button id="configuration-button"
            onclick="toggleDropdown('configuration-dropdown')" class="dropbtn
            unselectable">Configuration</button>
        <div id="configuration-dropdown" class="dropdown-content unselectable">
            <div onclick="setDropdown('configuration-button','Single')">Single</div>
            <div onclick="setDropdown('configuration-button','Cluster')">Cluster</div>
        </div>
    </div>
</div>
<div id="waypoint-container">
    <div id="waypoint-container-title" class="generic-title unselectable">Waypoints
        <button type="button" onclick="sendWaypoints()" id="waypoints-button"
            class="generic-button unselectable">Send</button>
    </div>
    <div id="coordinates-list" class="flx">
        <div id="waypoint-holder-div">
            X: <textarea id="textarea-x" rows="1" cols="4"
                onkeypress="submitTextWaypoint()"></textarea>
            Y: <textarea id="textarea-y" rows="1" cols="4"
                onkeypress="submitTextWaypoint()"></textarea>
        </div>
        <div id="coordinates-list-actual" class="flx">
        </div>
    </div>
</div>
<div id="settings-container">
    <div id="settings-container-title" class="generic-title-with-button
        unselectable">Controllers
        <button type="button" onclick="requestDT()" id="request-dt-button"
            class="generic-button unselectable">Request</button>
    </div>
    <div id="settings-container-content" class="unselectable">
        <div class="dropdown unselectable">
            <button id="controllers-button"
                onclick="toggleDropdown('controllers-dropdown')" class="dropbtn
                unselectable">Controllers</button>
            <div id="controllers-dropdown" class="dropdown-content unselectable">
                <div
                    onclick="setDropdown('controllers-button','Waypoints')">Waypoints</div>
            </div>
        </div>
        <div class="dropdown unselectable">
            <button id="robots-button" onclick="toggleDropdown('robots-dropdown')"
                class="dropbtn unselectable">Robots</button>
            <div id="robots-dropdown" class="dropdown-content unselectable">
            </div>
        </div>
        <div id="send-button-div" class="dropdown unselectable">
            <button id="send-button" onclick="reconDT()" class="dropbtn
                unselectable">Send</button>
        </div>
    </div>
</div>
</td>
</tr>
</table>
</div>
<script src="main.js"></script>
</body>

```

</html>

#####

ui/main.css

#####

```
.dropbtn {
  background-color: #222222;
  color: #c9c9c9;
  padding: 10px;
  font-size: 13px;
  border: none;
  cursor: pointer;
  outline: none;
}
.dropdown {
  position: relative;
  display: inline-block;
}
.dropdown-content {
  color: #c0c0c0;
  min-width: 150px;
  display: none;
  position: absolute;
  background-color: #333333;
  z-index: 1;
}
#configuration-button{
  display:none;
}
#configuration-dropdown-div {
  margin:0 auto;
  position:relative;
  top:10px;
  left:172px;
}
textarea {
  display:inline-block;
  resize:none;
  margin:0 auto;
  width:50px;
  border:1px solid #111111;
  outline:none;
  background-color:#222222;
  color:#F9F9F9;
  font-size:14px;
  height:19px;
  padding:2px;
  position:relative;
  top:7px;
}
#waypoint-holder-div {
  margin:5px auto;
  margin-top:-10px;
  height:25px;
}
.dropdown-content div {
  font-size:14px;
  color: #c0c0c0;
```

```

padding: 10px 10px;
text-decoration: none;
display: block;
margin:0 auto;
}
.dropdown-content div:hover {
background-color: #111111;
}
.show {
display:block;
}
.flx {
display:flex;
width:100%;
overflow:auto;
height:210px;
flex-direction:column;
flex-wrap:nowrap;
justify-content:flex-start;
margin:0;
padding:5px;
}
.generic-button {
background-color:#222222;
color:#C0C0C0;
float:right;
position:relative;
left:-10px;
border:none;
outline:none;
cursor:pointer;
}
#waypoints-button {
display:none;
}
.generic-button:hover {
color:#00FF00;
}
.coordinates {
color:white;
min-height:20px;
/*width:90%;*/
display:flex;
overflow:auto;
margin:4px;
margin-right:23px;
margin-bottom:-1px;
padding:5px;
background-color:#333333;
}
.page-table {
width:100%;
height:100 %;
margin:0 auto;
border-collapse: separate;
border-spacing: 10px;
}
.display-field {
height:18px;

```

```

        text-align:left;
        margin:none;
        float:left;
        padding-left:10px;
        display:block;
    }
    .display-value {
        height:18px;
        text-align:right;
        /*margin:0 auto;*/
        display:block;
        float:right;
        padding-right:10px;
    }
    .field-flex-container {
        display:flex;
        float:left;
        flex-direction:column;
    }
    #canvas-container {
        height:95%;
        width:100%;
        border:0px solid black;
        min-width:635px;
    }
    .value-flex-container {
        display:flex;
        float:right;
        flex-direction:column;
    }
    #title-div {
        margin: 0px auto;
        height: 7vh;
        min-height: 7vh;
        text-align: center;
        font-size: 30px;
    }
    #robot-container-helper {
        height: 18px;
    }

    #robot-item-container {
        display:flex;
        flex-direction:column;
        overflow: auto;
        width: 100%;
        min-width:200px;
        background-color: #222222;
    }
    .generic-title {
        padding:5px;
        width: 100%;
        text-align:center;
        height: 18px;
    }
    .generic-title-with-button {
        padding:5px;
        width: 100%;
        text-align:center;

```

```

        text-indent:50px;
        height: 18px;
    }
    .unselectable {
        -webkit-user-select:none;
    }
    .status-disconnected {
        color:red !important;
    }
    .status-connected {
        color:#00FF00 !important;
    }
    .robot-item-div {
        width:90%;
        margin:0 auto;
        border:1px solid #111111;
        outline:none;
        margin-top:10px;
        min-width:175px
    }
    .dropdown {
        margin:0px auto;
        margin-left:50px;
        margin-right:50px;
        margin-top:10px;
        margin-bottom:10px;
        background-color:#222222;
        color:rgb(192, 192, 192);
        outline:none;
        border:1px solid #111111;
    }
    .robot-item-caption {
        height:18px;
        text-align:center;
        border:none;
        /*border-bottom:1px solid black;*/
        width:100%;
    }
    .remove-robot-button {
        cursor:pointer;
        position:relative;
        top:-18px;
        height:18px;
        width:18px;
        text-align:center;
        vertical-align:middle;
        padding:0;
        border:none;
        /*border-bottom:1px solid black;*/
        outline:none;
        background-color:#222222;
        float:right;
        color:rgb(192, 192, 192);
        z-index:1;
    }
    #page_table-div {
        position:absolute;
        top:5vh;
        right:0;

```



```

    left:0;
    bottom:10vh;
    height:90vh;
    width:100%;
}
#page_table_col_11 {
    width:15.23vw;
    height:100%;
    border:1px solid #111111;
    background-color:#111111;
}
#page_table_col_12 {
    /*width:75%;*/
    height:70%;
    border:1px solid #111111;
    background-color:#111111;
}
#page_table_col_21 {
    width:75%;
    height:30%;
    border:1px solid #111111;
    background-color:#111111;
}
#heading {
    width:708px;
    text-align:center;
    margin-top:-5px;
    margin-left:-5px;
    margin-bottom:10px;
    border-bottom:1px solid black;
}
#send-button-div {
    position:relative;
    left:120px;
    top:100px;
}
#robot-container {
    width:100%;
    height:100%;
    background-color:#222222;
}
#configuration-container {
    float:left;
    width:32%;
    height:100%;
    background-color:#222222;
}
#waypoint-container {
    float:right;
    width:32%;
    height:100%;
    background-color:#222222;
    margin-left:10px;
}
#settings-container {
    margin:0 auto;
    width:32%;
    height:100%;
    background-color:#222222;
}

```

```

}
#graphics-container {
  width:100%;
  height:100%;
  background-color:#222222;
}
body {
  /*height: 100%;*/
  width: 100%;
  -webkit-overflow-scrolling: touch;
  overflow: auto;
  height:96vh;
  background-color:#111111;
  color:rgb(192, 192, 192);
}
html {
  height: 100%;
  width: 100%;
  overflow: auto;
}

```

```

#####
ui/main.js
#####

```

```

var number_of_robots = 0;
var number_of_fields = 9;
var field_line_height = 18;
var single_robot_string = false;
var ws;
var current_robot_list = [];
var selected_robots = [];
var data_timeout = 1000;
var removal_timeout = 1000000000000;
var pozyx_x_max = 3946;
var pozyx_y_max = 2854;
var canvas_flag_active = false;
var canvas_flag_position = {x:undefined,y:undefined};
var current_coordinate_list = [];
var number_of_coordinates = 0;
var current_clusters = [];
var current_sinks = [];
var current_channels = [];
var current_sources = [];
var manual_control = false;

function submitTextWaypoint() {
  var key = window.event.keyCode;
  var canvas = document.querySelector('#canvas');
  if(key == 13) {
    number_of_coordinates++;
    var x_actual = document.querySelector('#textarea-x').value;
    var y_actual = document.querySelector('#textarea-y').value;
    document.querySelector('#textarea-x').value = "";
    document.querySelector('#textarea-y').value = ""
    document.querySelector('#textarea-x').blur();
    document.querySelector('#textarea-y').blur();
    var x = Math.round((x_actual * canvas.width) / pozyx_x_max);
    var y = Math.round(((pozyx_y_max - y_actual) * canvas.height) / pozyx_y_max);

```

```

    current_coordinate_list.push({x:x,x_actual:x_actual,y:y,y_actual:y_actual});
    var coordinates = document.createElement('div');
    coordinates.innerHTML = 'Waypoint ${number_of_coordinates}: [ ${x_actual} , ${y_actual}
        ] [ ${x} , ${y} ]';
    coordinates.classList.add('coordinates');
    var parent = document.getElementById('coordinates-list-actual');
    parent.insertBefore(coordinates,parent.firstChild);
    if(document.querySelector('#waypoints-button').style.display != 'inline' &&
        selected_robots.length > 0)
        document.querySelector('#waypoints-button').style.display = 'inline';
    drawRobots();
}
}
/* set the canvas' size once the page has loaded (js gets executed afterward) */
setCanvasDimensions();
/* adjust canvas size if window is resized */
window.onresize = setCanvasDimensions;
/* toggles the shown content of a div dropdown */
function toggleDropdown(dropdown) {
    document.getElementById(dropdown).classList.toggle("show");
}
/* removes all currently timed out robot cards */
function removeDisconnected() {
    current_robot_list.forEach((robot) => {
        if(robot.status === 'Disconnected') {
            removeRobot(robot);
        }
    });
    drawRobots();
}
/* if a robot has been set to be in a cluster, it removes it */
function removeRobotFromCluster(robot) {
    for(var i = 0; i < current_clusters[robot.cluster].length; i++) {
        if( current_clusters[robot.cluster].robotNumber === robot.robotNumber) {
            current_clusters[robot.cluster].splice(i,1);
        }
    }
}
/* adjusts the cluster dropdown accordingly, pushes all robots in a cluster to an array */
function setDropdown(dropdown,option) {
    document.getElementById(dropdown).innerHTML = option;
    var clstr = [];
    selected_robots.forEach((robot) => {
        /* we have specified that the robot is not in a cluster with the dropdown */
        if(robot.configuration === 'Cluster') {
            if(option != 'Cluster') {
                removeRobotFromCluster(robot);
            }
        }
        /* set the new configuration option */
        robot.configuration = option;
        /* add to list of clusters if the configuration is a cluster */
        if(option === 'Cluster') {
            robot.cluster = current_clusters.length;
            clstr.push(robot);
        }
    })
    if(option === 'Cluster')
        current_clusters.push(clstr);
}

```

```

}
/* if we click on the dropdown */
window.onclick = function(event) {
  /* if we clicked on the button, show all the content */
  if (!event.target.matches('.dropbtn')) {
    document.querySelectorAll('.dropdown-content').forEach((dropdown) => {
      if(dropdown.classList.contains('show'))
        dropdown.classList.remove('show')
    });
  }
}
/* function to recognize key events */
document.addEventListener('keydown', function(e) {
  /* if escape is pressed, we deactivate the waypoint flag, and unselect any selected
  robots */
  if(e.key === 'Escape' || e.keyCode === 27) {
    if(canvas_flag_active) {
      escapeFlag();
    } else if(!canvas_flag_active && selected_robots.length > 0) {
      unselectAll();
      drawRobots();
    } else {
    }

    /* we remove the last active waypoint */
  } else if(e.key === 'Backspace' || e.keyCode == 8) {
    if(document.activeElement.id != "textarea-x" && document.activeElement.id !=
      "textarea-y") {
      removeLastWaypoint();
    }
  } else if(e.key === 'f' || e.keyCode === 70) {
    var canvas = document.getElementById('canvas');
    canvas.addEventListener('mousemove', showFlag);
    canvas_flag_active = true;
  } else if(e.key === 'm') {
    if(manual_control)
      manual_control = false;
    else
      manual_control = true;
  } else if(manual_control && e.key === 'ArrowUp') {
    console.log('yep');
    controlRobots('u');
  } else if(manual_control && e.key === 'ArrowDown') {
    controlRobots('d');
  } else if(manual_control && e.key === 'ArrowLeft') {
    controlRobots('l');
  } else if(manual_control && e.key === 'ArrowRight') {
    controlRobots('r');
  } else {
    console.log(e.key, e.keyCode);
  }
});
function controlRobots(key) {
  selected_robots.forEach((robot) => {
    if(key === 'u') {
      var vars = [];
      vars.push(`${robot.robotNumber}`);
      vars.push(' dd');
      vars.push(' 0');
    }
  });
}

```

```

vars.push(' ${robot.x}');
var y = parseInt(robot.y) + 20;
if(y > pozyx_y_max)
    y = 0;
else if(y < 0)
    y = pozyx_y_max - 1;
vars.push(' ${y}');
vars.push(' 0');
vars.push(' 0');
vars.push(' ${robot.heading}');
updateRobot(vars);
} else if(key === 'd') {
var vars = [];
vars.push(' ${robot.robotNumber}');
vars.push(' dd');
vars.push(' 0');
vars.push(' ${robot.x}');
var y = parseInt(robot.y) - 20;
if(y > pozyx_y_max)
    y = 0;
else if(y < 0)
    y = pozyx_y_max - 1;
vars.push(' ${y}');
vars.push(' 0');
vars.push(' 0');
vars.push(' ${robot.heading}');
updateRobot(vars);
} else if(key === 'l') {
var vars = [];
vars.push(' ${robot.robotNumber}');
vars.push(' dd');
vars.push(' 0');
var x = parseInt(robot.x) - 20;
if(x > pozyx_x_max)
    x = 0;
else if(x < 0)
    x = pozyx_x_max - 1;
vars.push(' ${x}');
vars.push(' ${robot.y}');
vars.push(' 0');
vars.push(' 0');
vars.push(' ${robot.heading}');
updateRobot(vars);
} else if(key === 'r') {
var vars = [];
vars.push(' ${robot.robotNumber}');
vars.push(' dd');
vars.push(' 0');
var x = parseInt(robot.x) + 20;
if(x > pozyx_x_max)
    x = 0;
else if(x < 0)
    x = pozyx_x_max - 1;
vars.push(' ${x}');
vars.push(' ${robot.y}');
vars.push(' 0');
vars.push(' 0');
vars.push(' ${robot.heading}');
updateRobot(vars);

```

```

    }
  });
}
/* reset the background on the robot cards, reset outline of robots on canvas, empty
   selected robots array */
function unselectAll() {
  resetAllStrokes();
  resetAllBackgrounds();
  selected_robots = [];
}
/* send waypoints over websocket */
function sendWaypoints() {
  for(var i = 0; i < current_coordinate_list.length; i++) {
    selected_robots.forEach((robot) => {
      var send_string = 'robot_${robot.robotNumber} w
        ${current_coordinate_list[i].x_actual} ${current_coordinate_list[i].y_actual}';
      ws.send(send_string);
      send_string = 'robot_${robot.robotNumber} s ${i+1}
        ${current_coordinate_list[i].x_actual} ${current_coordinate_list[i].y_actual}';
      ws.send(send_string);
      console.log(send_string);
    });
  }
}
/* removes the little flag that accompanies the cursor when it's active */
function escapeFlag() {
  canvas.removeEventListener('mousemove', showFlag);
  canvas_flag_active = false;
  canvas_flag_position = {x:undefined,y:undefined};
}
/* removes the last active waypoint from both the canvas, and the waypoints container */
function removeLastWaypoint() {
  if(number_of_coordinates > 0) {
    number_of_coordinates--;
    if(number_of_coordinates == 0) {
      document.querySelector('#waypoints-button').style.display = 'none';
    }
    current_coordinate_list = current_coordinate_list.slice(0,-1);
    document.querySelector('#coordinates-list-actual').removeChild(document.querySelector('#coordinates-list-actual
      > div'));
  }
}
/* removes all waypoints, both from the container and canvas, currently unused */
function removeAllWaypoints() {
  if(!canvas_flag_active) {
    number_of_coordinates = 0;
    current_coordinate_list = [];
    canvas_flag_position = {x:undefined,y:undefined};
    var coordinates = document.querySelectorAll('#coordinates-list > div');
    coordinates.forEach((coordinate) => {
      document.querySelector('#coordinates-list').removeChild(coordinate);
    });
  }
}
/* original flag style, currently unused */
function actualFlag(x,y) {
  var ctx = canvas.getContext('2d');
  ctx.lineTo(x+21,y+22);
  ctx.lineTo(x+28,y+22);

```

```

    ctx.lineTo(x+28,y+25);
    ctx.lineTo(x+12,y+25);
    ctx.lineTo(x+12,y+22);
    ctx.lineTo(x+19,y+22);
    ctx.lineTo(x+19,y+8);
    ctx.lineTo(x+12,y+4);
    ctx.lineTo(x+21,y+0);
}
/* alternate flag style */
function alternateFlag(x,y) {
    var ctx = canvas.getContext('2d');
    ctx.lineTo(x+21,y+25);
    ctx.lineTo(x+19,y+25);
    ctx.lineTo(x+19,y+12);
    ctx.lineTo(x+8,y+8);
    ctx.lineTo(x+21,y+0);
}
/* draws the flag on the canvas at x,y and includes a counter i */
function drawFlag(x,y,i) {
    var canvas = document.getElementById('canvas');
    if (canvas.getContext) {
        var ctx = canvas.getContext('2d');
        ctx.fillStyle = 'FFFFFF';
        ctx.strokeStyle = 'FFFFFF'
        ctx.font = "10px Arial"
        if(i)
            ctx.fillText(i.toString(),x+27,y+27);
        ctx.lineWidth = 2;
        ctx.beginPath();
        ctx.fillStyle = 'FF0000';
        ctx.strokeStyle = 'FFFFFF'
        ctx.moveTo(x+21,y+0);
        alternateFlag(x,y);
        ctx.closePath()
        ctx.stroke();
        ctx.fill();
    }
}
/* draw all currently active flags at their respective locations on the canvas */
function drawFlags() {
    var canvas = document.getElementById('canvas');
    var context = canvas.getContext('2d');
    for(var i = 0; i < number_of_coordinates; i++) {
        if(i > 0) {
            context.setLineDash([5]);
            context.beginPath();
            context.lineWidth = 2;
            context.strokeStyle = 'FFFFFF';
            context.moveTo(current_coordinate_list[i].x,current_coordinate_list[i].y);
            context.lineTo(current_coordinate_list[i-1].x,current_coordinate_list[i-1].y);
            context.stroke();
            context.setLineDash([]);
        }
    }
    for(var i = 0; i < number_of_coordinates; i++) {
        drawFlag(current_coordinate_list[i].x-20,current_coordinate_list[i].y-35,i+1);
    }
}
/* connects selected robots together with a line, also shows central point */

```

```

function connectSelected() {
    var canvas = document.getElementById('canvas');
    var context = canvas.getContext('2d');
    var width = canvas.width;
    var height = canvas.height;
    var mid_x = undefined;
    var mid_y = undefined;
    context.setLineDash([5]);
    if(selected_robots.length > 1) {
        mid_x = 0;
        mid_y = 0;
        for(var i = 1; i < selected_robots.length; i++) {
            var current_x = Math.round((selected_robots[i].x / pozyx_x_max) * width);
            var current_y = Math.floor(height - ((selected_robots[i].y / pozyx_y_max) * height));
            var prev_x = Math.round((selected_robots[i-1].x / pozyx_x_max) * width);
            var prev_y = Math.floor(height - ((selected_robots[i-1].y / pozyx_y_max) * height));
            mid_x += prev_x;
            mid_y += prev_y;
            if(i === (selected_robots.length - 1)) {
                mid_x += current_x;
                mid_y += current_y;
            }
            context.beginPath();
            context.lineWidth = 3;
            context.strokeStyle = '#0000FF';
            context.moveTo(current_x, current_y);
            context.lineTo(prev_x, prev_y);
            context.stroke();
            if((i === (selected_robots.length - 1)) && selected_robots.length != 2) {
                var prev_x = Math.round((selected_robots[0].x / pozyx_x_max) * width);
                var prev_y = Math.floor(height - ((selected_robots[0].y / pozyx_y_max) * height));
                context.beginPath();
                context.lineWidth = 3;
                context.strokeStyle = '#0000FF';
                context.moveTo(current_x, current_y);
                context.lineTo(prev_x, prev_y);
                context.stroke();
            }
        }
        mid_x /= i;
        mid_y /= i;
        context.moveTo(mid_x, mid_y);
        context.setLineDash([]);
        context.fillStyle = '#FFFF00';
        context.arc(mid_x, mid_y, 5, 0, 2 * Math.PI);
        context.fill();
    }
    context.setLineDash([]);
}

/* sends a request message to dataturbine for sources, sinks, and channels */
function requestDT() {
    if(ws.readyState == 1) {
        ws.send('requestDTconfig()');
        console.log('Requesting channels and sinks from DataTurbine...');
    } else {
        window.alert('Not connected to server.');
```

```

    }
}

function reconDT() {
```



```

    controller_name = document.querySelector('#controllers-button').innerHTML;
    robot_name = document.querySelector('#robots-button').innerHTML;
    reconfigureDT(robot_name,controller_name);
}
/* sends reconfigured settings to dataturbine */
function reconfigureDT(sinkName,chanName) {
    if(ws.readyState == 1) {
        ws.send('reconfSink2Chan(' + sinkName + ', ' + chanName + ')');
        console.log('Sending updated configuration to DataTurbine.');
```

```

    } else {
        window.alert('Not connected to server.');
```

```

    }
}
/* parses info received from datatubine for sources, sinks, channels, sets dropdown
    accordingly */
```

```

function setDT(message) {
    message = message.substring(10);
    var m = message.split(',');
    current_channels = [];
    current_sinks = [];
    current_sources = [];
    document.querySelectorAll('#controllers-dropdown>div').forEach((element) => {
        if(element.innerHTML != 'Waypoints')
            document.querySelector('#controllers-dropdown').removeChild(element);
    });
    document.querySelectorAll('#robots-dropdown>div').forEach((element) => {
        document.querySelector('#robots-dropdown').removeChild(element);
    });
    m.forEach((msg) => {
        if(msg.includes('Channel')) {
            var msg = msg.split(' ')[0];
            current_channels.push(msg);
            if(!msg.includes('robot')) { /* controller */
                var parent = document.querySelector('#controllers-dropdown');
                var child = document.createElement('div');
                child.addEventListener('click', function() {
                    setDropdown('controllers-button', msg);
                });
                child.innerHTML = msg;
                parent.appendChild(child);
                console.log('Controller: ${msg}');
```

```

            } else { /* robot */
                msg = msg.split('-')[0];
                var non_duplicate = true;
                document.querySelectorAll('#robots-dropdown>div').forEach((element) => {
                    if(element.innerHTML === msg) {
                        non_duplicate = false;
                    }
                });
                if(non_duplicate) {
                    var parent = document.querySelector('#robots-dropdown');
                    var child = document.createElement('div');
                    child.addEventListener('click', function() {
                        setDropdown('robots-button', msg);
                    });
                    child.innerHTML = msg;
                    parent.appendChild(child);
                }
            }
        }
    });
}
```

```

    } else if(msg.includes('Sink')) {
        var msg = msg.split(' ')[0];
        current_sinks.push(msg);
    } else if(msg.includes('Source')) {
        var msg = msg.split(' ')[0];
        current_sources.push(msg);
    } else {}
    });
}
/* sets the robot card fields to 0 */
function resetValues(r) {
    r.values.forEach((value,i) => {
        if(r.fields[i]!='Robot ID')
            value.innerHTML = '0';
    });
}
function drawCornerFlag() {
    var canvas = document.querySelector('canvas');
    var context = canvas.getContext('2d');
    var canvas_container = document.getElementById('canvas-container');
    var width = canvas_container.offsetWidth;
    var height = canvas_container.offsetHeight;
    context.beginPath();
    context.lineWidth=2;
    context.rect(width-50,10,40,40);
    context.strokeStyle = '#FFFFFF';
    context.stroke();
}
function drawGrid() {

}
/* draws all active robots on the canvas */
function drawRobots() {
    var bound_x = pozyx_x_max;
    var bound_y = pozyx_y_max;
    var canvas = document.getElementById('canvas');
    var context = canvas.getContext('2d');
    var canvas_container = document.getElementById('canvas-container');
    var width = canvas_container.offsetWidth;
    var height = canvas_container.offsetHeight;
    context.clearRect(0, 0, width, height);
    drawFlag(width-48,18);
    if(canvas_flag_active) {
        drawFlag(canvas_flag_position.x,canvas_flag_position.y);
    }
    drawFlags();
    connectSelected();
    current_robot_list.forEach((robot) => {
        context.beginPath();
        var current_x = Math.round((robot.x / bound_x) * width);
        var current_y = Math.floor(height - ((robot.y / bound_y) * height));
        context.lineWidth = 3;
        context.arc(current_x, current_y, robot.radius, 0, 2 * Math.PI);
        context.moveTo(current_x, current_y);
        context.lineTo(current_x + ((robot.radius) * Math.cos(2*Math.PI -
            robot.heading)),current_y + ((robot.radius) * Math.sin(2*Math.PI - robot.heading)));
        context.strokeStyle = robot.color_stroke;
        context.fillStyle = robot.color_fill;
        context.fill();
    });
}

```

```

        context.stroke();
        drawCornerFlag();
    });
}
window.setInterval(drawRobots, 50);
/* sets the canvas dimensions */
function setCanvasDimensions() {
    var canvas = document.getElementById('canvas');
    var width = window.innerWidth * 0.73408;
    var height = window.innerHeight * 0.58378;
    canvas.width = width;
    canvas.height = height;
}
/* generates a random number number between low and high, used in color generation for
    robots */
function generateRandom(low,high) {
    return Math.random() * (high - low) + low;
}
/* creates a robot object with the array of variables supplied in the argument */
function createRobot(vars) {
    var r_f = Math.round(generateRandom(0,255));
    var g_f = Math.round(generateRandom(0,255));
    var b_f = Math.round(generateRandom(0,255));
    var r_s = Math.round(generateRandom(0,255));
    var g_s = Math.round(generateRandom(0,255));
    var b_s = Math.round(generateRandom(0,255));
    var fill_color = 'rgb(${r_f}, ${g_f}, ${b_f})';
    var stroke_color = 'rgb(${r_s}, ${g_s}, ${b_s})';
    var robot = {
        fields: [],
        values: [],
        robotNumber: undefined,
        timeRemainingDisconnect: 0,
        timeRemainingRemoval: 0,
        status: 'Connected',
        x: undefined,
        y: undefined,
        heading: 0, /* radians */
        radius: 15,
        color_fill: fill_color,
        color_stroke: '#FFFFFF',
        configuration: 'Single',
        controller: 'Waypoint',
        cluster: undefined
    };
}
for(var i = 0; i < number_of_fields; i++) {
    var current_field = document.createElement('div');
    current_field.classList.add('display-field');
    current_field.classList.add('robot_item-' + vars[0]);
    current_field.classList.add('unselectable');
    var val_id;
    /* sets the robot card fields */
    switch(i) {
        case 0:
            current_field.innerHTML = 'Robot ID'; val_id = 'robot_id'; break;
        case 1:
            current_field.innerHTML = 'Type'; val_id = 'type'; break;
        case 2:
            current_field.innerHTML = 'Color'; val_id = 'color'; break;
    }
}

```

```

    case 3:
        current_field.innerHTML = 'Position X'; val_id = 'position_x'; break;
    case 4:
        current_field.innerHTML = 'Position Y'; val_id = 'position_y'; break;
    case 5:
        current_field.innerHTML = 'Position Z'; val_id = 'position_z'; break;
    case 6:
        current_field.innerHTML = 'Velocity'; val_id = 'velocity'; break;
    case 7:
        current_field.innerHTML = 'Heading'; val_id = 'heading'; break;
    case 8:
        current_field.innerHTML = 'Status'; val_id = 'status'; break;
    default:
        current_field.innerHTML = 'N/A'; break;
}
var current_value = document.createElement('div');
current_value.classList.add('display-value');
current_value.classList.add('robot_item-' + vars[0]);
current_value.classList.add('unselectable');
if(val_id === 'status') {
    current_value.classList.add('status-connected');
    current_value.innerHTML = robot.status;
} else if(val_id === 'color') {
    current_value.style.backgroundColor = fill_color;
} else {
    current_value.innerHTML = vars[i];
}
robot.fields.push(current_field);
robot.values.push(current_value);
}
robot.robotNumber = parseInt(robot.values[0].innerHTML);
return robot;
}
/* handles robot timeout */
function timer(r) {
    clearTimeout(r.timeRemainingDisconnect);
    clearTimeout(r.timeRemainingRemoval);
    r.timeRemainingDisconnect = setTimeout(() => {
        r.status = 'Disconnected';
        r.values[number_of_fields-1].innerHTML = r.status;
        r.values[number_of_fields-1].classList.remove('status-connected');
        r.values[number_of_fields-1].classList.add('status-disconnected');
    }, data_timeout);
    r.timeRemainingRemoval = setTimeout(() => {
        removeRobot(r);
    }, removal_timeout);
}
/* updates existing robots with provided variables, otherwise creates them */
function updateRobot(vars) {
    var add_robot_flag = true;
    if(vars.length < 8) { // probably no velocity, so we pad
        vars[7] = vars[6];
        vars[6] = 0;
    }
    if(current_robot_list.length) {
        for(var i = 0; i < current_robot_list.length; i++) {
            if(current_robot_list[i].values[0].innerHTML == vars[0]) {
                add_robot_flag = false;
                for(var j = 0; j < number_of_fields; j++) {

```

```

        if(current_robot_list[i].fields[j].innerHTML === 'Status') {
            current_robot_list[i].status = 'Connected';
            current_robot_list[i].values[j].innerHTML = current_robot_list[i].status;
            current_robot_list[i].values[j].classList.remove('status-disconnected');
            current_robot_list[i].values[j].classList.add('status-connected');
        } else if(current_robot_list[i].fields[j].innerHTML === 'Position X') {
            current_robot_list[i].x = vars[j];
            current_robot_list[i].values[j].innerHTML = vars[j];
        } else if(current_robot_list[i].fields[j].innerHTML === 'Position Y') {
            current_robot_list[i].y = vars[j];
            current_robot_list[i].values[j].innerHTML = vars[j];
        } else if(current_robot_list[i].fields[j].innerHTML === 'Heading') {
            current_robot_list[i].heading = parseFloat(vars[j]);
            current_robot_list[i].values[j].innerHTML = Math.round(((parseFloat(vars[j])) *
                180 / Math.PI) * 100)/100;
        } else if(current_robot_list[i].fields[j].innerHTML === 'Color') {

        } else {
            current_robot_list[i].values[j].innerHTML = vars[j];
        }
    }
    timer(current_robot_list[i]);
}
}
drawRobots();
if(add_robot_flag) {
    addRobot(createRobot(vars));
}
} else {
    addRobot(createRobot(vars));
}
}
}
/* parses message received from websocket */
function checkMessage(m) {
    if(m.includes('DTConfig: ')) {
        console.log('Received channels and sinks from DataTurbine.');
```

```

        setDT(m);
    } else if(m.includes('Source') || m.includes('Sink') || m.includes('Channel')) {
        setDT(m);
    } else if(manual_control == false && m.includes('robot_')) {
        // var robots_ = m.split(';');
        // robots_.forEach((message) => {
        //     if(message.length > 0) {
        //         var robot_vars = message.split(',');
        //         robot_vars[0] = message.split(',')[0].slice(6);
        //         updateRobot(robot_vars);
        //     }
        // });
        var robot_id = m.split(',')[0].slice(6).split('-')[0];
        var robot_vars = m.split(';')[0].split(',');
        robot_vars[0] = robot_id;
        updateRobot(robot_vars);
    } else {
        // console.log(m);
    }
}
}
/* adds a new robot card */
function addRobot(r) {
    number_of_robots += 1;

```

```

var container = document.getElementById('robot-item-container');
var rc = document.createElement('div');
rc.style.height = (((number_of_fields + 1) * field_line_height) + 2) + 'px';
rc.id = 'rc-' + r.robotNumber;
rc.classList.add('robot-item-div');
rc.classList.add('robot-item-' + r.robotNumber);
rc.addEventListener('mousedown', displaySettings);
var cap = document.createElement('div');
cap.id = 'cap-' + r.robotNumber;
cap.classList.add('robot-item-caption');
cap.classList.add('robot-item-' + r.robotNumber);
cap.classList.add('unselectable');
cap.innerHTML = 'Robot ' + r.values[0].innerHTML;
var field_flex_container = document.createElement('div');
field_flex_container.id = 'robot_item_field_flex_container-' + r.robotNumber;
field_flex_container.classList.add('field-flex-container');
field_flex_container.classList.add('robot-item-' + r.robotNumber);
var value_flex_container = document.createElement('div');
value_flex_container.id = 'robot_item_value_flex_container-' + r.robotNumber;
value_flex_container.classList.add('value-flex-container');
value_flex_container.classList.add('robot-item-' + r.robotNumber);
rc.appendChild(cap);
rc.appendChild(field_flex_container);
rc.appendChild(value_flex_container);
for(var i = 0; i < number_of_fields; i++) {
    field_flex_container.appendChild(r.fields[i]);
    value_flex_container.appendChild(r.values[i]);
}
current_robot_list.push(r);
container.appendChild(rc);
timer(r);
}
/* removes a robot */
function removeRobot(r) {
    if(r.configuration === 'Cluster') {
        removeRobotFromCluster(r);
    }
    number_of_robots--;
    for(var i = 0; i < current_robot_list.length; i++) {
        if(r.robotNumber === current_robot_list[i].robotNumber) {
            current_robot_list.splice(i,1);
        }
    }
    r.fields[0].parentElement.parentElement.parentElement.removeChild(document.getElementById('rc-'
        + r.robotNumber));
}
/* sets robot card background color, canvas robot outline color, brings up cluster
    information */
function displaySettings(event, id, graphics) {
    if(current_coordinate_list.length > 0)
        document.getElementById('waypoints-button').style.display = 'block';

    var rnumber = 0;
    var container;
    var selection_flag = false;
    if(id) {
        rnumber = id
    }
}

```

```

current_robot_list.forEach((robot) => {
  if(!graphics)
    rnumber = this.id.slice(this.id.indexOf('-')+1)
  if(robot.robotNumber == rnumber) {
    if(robot.color_stroke != '#00FF00') {
      if(!event.shiftKey)
        resetAllStrokes();
      robot.color_stroke = '#00FF00';
    } else {
      resetAllStrokes();
      robot.color_stroke = '#00FF00';
    }
  }
  if(robot.configuration === 'Cluster') {
    if(!event.shiftKey) {
      selected_robots = [];
      resetAllStrokes();
      resetAllBackgrounds();
    }
    current_clusters[robot.cluster].forEach((robot) => {
      selected_robots.push(robot);
      robot.color_stroke = '#0000FF';
      document.getElementById('rc-'+robot.robotNumber).style.backgroundColor = 'rgb(24,
        24, 24)';
      selection_flag = true;
    });
  } else {
    selected_robots.push(robot);
  }
  document.getElementById('configuration-button').style.display = 'block';
  document.getElementById('configuration-button').innerHTML = robot.configuration;
  drawRobots();
}
});
if(!selection_flag) {
  if(!graphics)
    container = this;
  else
    container = document.getElementById('rc-'+id);
  if(container.style.backgroundColor != 'rgb(24, 24, 24)') {
    if(!event.shiftKey)
      resetAllBackgrounds();
    container.style.backgroundColor = 'rgb(24, 24, 24)';
  } else {
    resetAllBackgrounds()
    container.style.backgroundColor = 'rgb(24, 24, 24)';
  }
}
if(current_coordinate_list.length > 0)
  document.querySelector('#waypoints-button').style.display = 'inline';
}
/* resets the background color of the robot cards */
function resetAllBackgrounds() {
  document.querySelectorAll('.robot-item-div').forEach((robot_item) => {
    robot_item.style.backgroundColor = 'rgb(34, 34, 34)';
  });
}
/* gets the mouse position relative to the canvas */
function getMousePos(canvas, evt) {
  var rect = canvas.getBoundingClientRect();

```

```

    return {
        x: evt.clientX - rect.left,
        y: evt.clientY - rect.top
    };
}
/* sets the flag to follow the cursor */
function showFlag(evt) {
    var pos = getMousePos(canvas, evt);
    canvas_flag_position.x = pos.x - 20;
    canvas_flag_position.y = pos.y - 27;
    drawFlag(pos.x - 20, pos.y - 27);
    canvas_flag_active = true;
}
/* resets the outline of the robots on the canvas */
function resetAllStrokes() {
    document.getElementById('waypoints-button').style.display = 'none';
    selected_robots = [];
    document.getElementById('configuration-button').style.display = 'none';
    current_robot_list.forEach((robot) => {
        robot.color_stroke = '#FFFFFF';
    });
}
/* creates a new websocket */
ws = new WebSocket("ws://127.0.0.1:9002/");
/* listen to clicks on canvas, determine if robot was clicked */
document.getElementById('canvas').addEventListener('mousedown', function canvasClick(e) {
    var canvas = document.getElementById('canvas');
    var canvas_coord = canvas.getBoundingClientRect();
    var x = parseInt(e.clientX) - parseInt(canvas_coord.left);
    var y = (canvas_coord.bottom - canvas_coord.top) - (parseInt(e.clientY) -
        parseInt(canvas_coord.top));
    x_actual = Math.round((x / canvas.width) * pozyx_x_max);
    y_actual = Math.round((y / canvas.height) * pozyx_y_max);
    var canvas_container = document.getElementById('canvas-container');
    var width = canvas_container.offsetWidth;
    var height = canvas_container.offsetHeight;
    var shiftFlag = false;
    var onRobot = false;
    /* flag box */
    if((x > (width - 50) && x < (width - 10)) && (y > (height - 50) && (y < height-10))) {
        if(!canvas_flag_active) {
            var canvas = document.getElementById('canvas');
            canvas.addEventListener('mousemove', showFlag);
            canvas_flag_active = true;
        } else if(canvas_flag_active) {
            canvas.removeEventListener('mousemove', showFlag);
            canvas_flag_active = false;
        }
    } else if(canvas_flag_active) {
        current_coordinate_list.push({x:x,y:height-y,x_actual:x_actual,y_actual:y_actual});
        var coordinates = document.createElement('div');
        coordinates.innerHTML = 'Waypoint ${++number_of_coordinates}: [ ${x_actual} ,
            ${y_actual} ] [ ${x} , ${y} ]';
        coordinates.classList.add('coordinates');
        var parent = document.getElementById('coordinates-list-actual');
        parent.insertBefore(coordinates, parent.firstChild);
        if(document.querySelector('#waypoints-button').style.display != 'inline' &&
            selected_robots.length > 0)
            document.querySelector('#waypoints-button').style.display = 'inline';
    }
});

```



```

    } else {
      current_robot_list.forEach((robot) => {
        if(Math.abs(robot.x - x_actual) < 80 && Math.abs(robot.y - y_actual) < 80) {
          displaySettings(e, robot.robotNumber, true);
          onRobot = true;
        } else {

        }
      });
      if(!onRobot && !e.shiftKey) {
        resetAllStrokes();
        resetAllBackgrounds();
      }
    }
  });
  /* websocket connection is active */
  ws.onopen = function() {
    console.log("Connection established!");
  }
  /* websocket connection is closed */
  ws.onclose = function() {
    console.log("Connection closed!");
  }
  /* whenever there's a message received over the websocket */
  ws.onmessage = function(m) {
    /* handle message */
    if(typeof m.data === 'string') {
      checkMessage(m.data);
    } else if(m.data instanceof Blob) {
      var reader = new FileReader();
      reader.onload = function() {
        checkMessage(reader.result);
      }
      reader.readAsText(m.data);
    } else if (m.data instanceof ArrayBuffer) {
      console.log(m.data instanceof ArrayBuffer);
    }
  }
  /* clear the console of all text */
  function clearConsole() {
    console.API;
    if (typeof console._commandLineAPI !== 'undefined') {
      console.API = console._commandLineAPI;
    } else if (typeof console._inspectorCommandLineAPI !== 'undefined') {
      console.API = console._inspectorCommandLineAPI;
    } else if (typeof console.clear !== 'undefined') {
      console.API = console;
    }
    console.API.clear();
  }
}

```

```

#####

```

```

server/server.js

```

```

#####

```

```

chalk = require('chalk');
ws = require('ws');
net = require('net');
logger = require('./logger.js');

```

```

const info = 'info';
const debug = 'debug';
const warning = 'warning';

logger.log.info('Server started');

color_error = chalk.bold.red;
color_packet = chalk.cyan;
color_connect = chalk.green;
color_disconnect = chalk.red;
color_websocket = chalk.yellow;
color_tcp = chalk.magenta;

websocketClient = undefined;
wsPort = 9002;

tcpClient = undefined;
tcpPort = 9001;

const wss = new ws.Server({
  perMessageDeflate: false,
  port: wsPort
});

/* websocket server */
wss.on('connection', (ws) => {
  /* message received */
  ws.on('message', (m) => {
    console.log('Received on ${color_websocket('Websocket')}: ${color_packet(m)}');
    if(tcpClient) {
      tcpClient.write(m);
      console.log('Sent on ${color_tcp('TCP')}: ${color_packet(m)}');
    }
  });
  /* websocket closed */
  ws.on('close', () => {
    console.log(`${color_websocket('Websocket client')} has
      ${color_disconnect('disconnected')}.`);
    logger.log.info('Websocket client has disconnected');
    websocketClient = undefined;
  })
  /* websocket error */
  ws.on('error', (e) => {
    console.log(`An ${color_error('error')} has occurred: ${color_error(e.message)}`);
    logger.log.error(e.message);
    websocketClient = undefined;
  });
  console.log(`${color_websocket('Websocket client')} has ${color_connect('connected')}.`);
  logger.log.info('Websocket client has connected');
  websocketClient = ws;
})
/* tcp server */
const nss = net.createServer((c) => {
  tcpClient = c;
  console.log(`${color_tcp('TCP client')} has ${color_connect('connected')}.`);
  logger.log.info('TCP client has connected')
  /* tcp closed */
  c.on('end', () => {

```

```

    console.log(`${color_tcp('TCP client')} has ${color_disconnect('disconnected')}.`);
    logger.log.info('TCP client has disconnected');
    tcpClient = undefined;
  });
  /* tcp message */
  c.on('data', (m) => {
    console.log('Received on ${color_tcp('TCP')}: ${color_packet(m)}');
    if(websocketClient) {
      if (websocketClient.readyState === 1) {
        websocketClient.send(m);
        console.log('Sent on ${color_websocket('Websocket')}: ${color_packet(m)}');
      }
    }
  });
  /* tcp error */
  c.on('error', (e) => {
    console.log('An ${color_error('error')} has occurred: ${color_error(e.message)}');
    logger.log.error(e.message);
    tcpClient = undefined;
  });
});
/* tcp server listening for a connection */
nss.listen(tcpPort, () => {
  console.log(`${color_tcp('TCP server')} listening on port ${color_tcp(tcpPort)}.`);
});

/* exit on SIGINT */
process.on('SIGINT', () => { process.exit() });

/* log server shutdown */
process.on('exit', () => {
  process.stdout.write("\033[2K\033[200D");
  logger.logSync.info('Server shut down');
  console.log('Shutting down server.');
```

#####

server/robot-test.js

#####

```

var net = require('net');
var timers = require('timers')

var nss = new net.Socket();
var number_of_robots = 1;
var robots = [];
var current_waypoints = [];
var stop = false;

var tcpClient;

nss.connect(9001, '127.0.0.1', function() {
  tcpClient = nss;
  console.log('Connected to TCP server.');
```

});

```

nss.on('data', function(data) {
  if(data == 'requestDTconfig()') {
```

```

try{
  tcpClient.write('DTConfig: matlab/commands Channel,robot_1 Sink,robot_1-source/states
    Channel');
} catch(e) {}
} else if(data.includes('s')) {
  var msg = data.toString().split('robot_');
  msg.forEach((message) => {
    if(message.includes('s')) {
      var robot_id = parseInt(message.split(' ')[0]);
      var waypoint_number = parseInt(message.split(' ')[2]);
      var waypoint_x = parseInt(message.split(' ')[3]);
      var waypoint_y = parseInt(message.split(' ')[4]);
      while(current_waypoints.length < robot_id + 1) {
        current_waypoints.push({id:undefined,num:undefined,waypoints:[]});
      }
      var max_num = 0;
      if(current_waypoints[robot_id].num != undefined) {
        if(waypoint_number < current_waypoints[robot_id].num) {
          max_num = current_waypoints[robot_id].num;
        } else {
          max_num = waypoint_number;
        }
      }
      if(waypoint_number === 1) {
        beginRoute(robot_id);
      }
      current_waypoints[robot_id].id = robot_id;
      current_waypoints[robot_id].num = max_num;
      if(current_waypoints[robot_id].waypoints.length > waypoint_number - 1) {
        current_waypoints[robot_id].waypoints[waypoint_number-1] =
          {num:waypoint_number,x:waypoint_x,y:waypoint_y}
      } else if(current_waypoints[robot_id].waypoints.length === max_num + 1) {
        current_waypoints[robot_id].waypoints[waypoint_number-1] =
          {num:waypoint_number,x:waypoint_x,y:waypoint_y};
      } else {
        current_waypoints[robot_id].waypoints.push({num:waypoint_number,x:waypoint_x,y:waypoint_y})
      }
      console.log(current_waypoints);
    }
  });
}
});
function beginRoute(id) {
  robots.forEach((robot) => {
    if(robot.number === id) {
      robot.waypoint = true;
      robot.waypoint_number = 0;
      robot.stop = false;
    }
  });
}
function endRoute(id) {
  robots.forEach((robot) => {
    if(robot.number === id) {
      robot.waypoint = false;
      robot.waypoint_number = undefined;
    }
  });
}
}

```

```

function loopRoute(robot) {
  robot.waypoint = true;
  robot.waypoint_number = 0;
}
function stopRoute(robot) {
  robot.stop = true;
}
nss.on('close', function() {
  console.log('TCP connection closed.');
```

```

});
nss.on('error', (error) => {
  console.log('An error has occurred: ${error.message}');
  process.exit(1);
})
function generateRandom(low,high) {
  return Math.round(Math.random() * (high - low) + low);
}
function advanceRobot(robot) {
  robot.x_present = robot.x_future;
  robot.y_present = robot.y_future;
  robot.z_present = robot.z_future;
}
function advanceWaypoint(robot) {
  if(withinThreshold(robot)) {
    if(robot.waypoint_number++ < current_waypoints[robot.number].num) {
      try {
        robot.x_end = current_waypoints[robot.number].waypoints[robot.waypoint_number].x;
        robot.y_end = current_waypoints[robot.number].waypoints[robot.waypoint_number].y;
        robot.z_end = current_waypoints[robot.number].waypoints[robot.waypoint_number].z;
      } catch(e) {
        stopRoute(robot);
      }
    } else {
      stopRoute(robot);
    }
  }
}
function withinThreshold(robot) {
  var threshold = 10;
  var x_squared = Math.pow((robot.x_end - robot.x_present),2);
  var y_squared = Math.pow((robot.y_end - robot.y_present),2);
  var distance = Math.sqrt(x_squared + y_squared);
  if(distance < threshold) {
    console.log('Finished Waypoint ${robot.waypoint_number}');
    return true;
  }
  return false;
}
function setFuture(robot) {
  if(!robot.waypoint) {
    var robot_heading_delta = generateRandom(-180,180);
    var heading_offset = 0.04;

    if(robot_heading_delta < 0)
      robot.heading_degrees -= (heading_offset * Math.abs(robot_heading_delta));
    if(robot_heading_delta > 0)
      robot.heading_degrees += (heading_offset * Math.abs(robot_heading_delta));
    if(robot.heading_degrees < -180)
      robot.heading_degrees += 360;
  }
}

```

```

    if(robot.heading_degrees > 180)
        robot.heading_degrees -= 360;
    robot.heading = robot.heading_degrees * Math.PI / 180;
    var speed_offset = 10;

    robot.y_future = Math.round(robot.y_present + (speed_offset * Math.sin(robot.heading)));
    robot.x_future = Math.round(robot.x_present + (speed_offset * Math.cos(robot.heading)));

    if(robot.x_future > 3946)
        robot.x_future = 0;
    else if(robot.x_future < 0)
        robot.x_future = 3946;
    if(robot.y_future > 2854)
        robot.y_future = 0;
    else if(robot.y_future < 0)
        robot.y_future = 2854;
} else {
    try {
        robot.x_end = current_waypoints[robot.number].waypoints[robot.waypoint_number].x;
        robot.y_end = current_waypoints[robot.number].waypoints[robot.waypoint_number].y;
        robot.z_end = current_waypoints[robot.number].waypoints[robot.waypoint_number].z;
    } catch(e) {}
    advanceWaypoint(robot);
    calculateFuture(robot);
}
}

function calculateFuture(robot) {
    var heading = calculateHeading(robot.x_present, robot.y_present, robot.x_end, robot.y_end);
    robot.heading_delta = heading - robot.heading_degrees;
    if(robot.heading_delta < -180)
        robot.heading_delta += 360;
    if(robot.heading_delta > 180)
        robot.heading_delta -= 360;

    var heading_offset = 0.1;

    if(robot.heading_delta < 0)
        robot.heading_degrees -= (heading_offset * Math.abs(robot.heading_delta));
    if(robot.heading_delta > 0)
        robot.heading_degrees += (heading_offset * Math.abs(robot.heading_delta));
    if(robot.heading_degrees < -180)
        robot.heading_degrees += 360;
    if(robot.heading_degrees > 180)
        robot.heading_degrees -= 360;
    robot.heading = robot.heading_degrees * Math.PI / 180;

    var speed_offset = 10;

    robot.y_future = Math.round(robot.y_present + (speed_offset * Math.sin(robot.heading)));
    robot.x_future = Math.round(robot.x_present + (speed_offset * Math.cos(robot.heading)));

    if(robot.x_future > 3946)
        robot.x_future = 0;
    else if(robot.x_future < 0)
        robot.x_future = 3946;
    if(robot.y_future > 2854)
        robot.y_future = 0;
    else if(robot.y_future < 0)
        robot.y_future = 2854;
}

```

```

}
function generateRobots() {
  for(var i = 0; i < number_of_robots; i++) {
    robots.push({
      name:'robot_${i}',
      number:i,
      type:'dd',
      x_present:generateRandom(100,2900),
      y_present:generateRandom(100,2800),
      z_present:0,
      x_future:0,
      y_future:0,
      z_future:0,
      x_end:0,
      y_end:0,
      z_end:0,
      color:0,
      velocity:0,
      heading:Math.PI/2,
      heading_degrees:90,
      heading_delta:0,
      waypoint:false,
      waypoint_number:undefined,
      timer:undefined,
      stop:false
    });
  }
}
function sendRobot(robot) {
  var send_string = '${robot.name}, ${robot.type}, ${robot.color}, ${robot.x_present},
    ${robot.y_present}, ${robot.z_present}, ${robot.velocity}, ${robot.heading}';
  try {
    tcpClient.write(send_string);
  } catch(e) {
    console.error('An error has occurred: ${e.message}');
    process.exit(1);
  }
}
function setUpdates() {
  robots.forEach((robot) => {
    var t = timers.setInterval(function() {
      if(!robot.stop) {
        setFuture(robot);
        advanceRobot(robot);
      }
      sendRobot(robot);
    },25);
  });
}
generateRobots();
setUpdates();
function calculateHeading(current_x,current_y,waypoint_x,waypoint_y) {
  var theta_radians = Math.atan2((waypoint_y - current_y),(waypoint_x - current_x));
  var theta_degrees = theta_radians * 180 / Math.PI;
  return theta_degrees;
}

#####
server/logger.js

```

```
#####

const fs = require('fs');

/* async logger */
exports.log = {
  /* message level */
  error(message) {
    this.writeFile(message, 'ERROR');
  },
  warning(message) {
    this.writeFile(message, 'WARNING');
  },
  debug(message) {
    this.writeFile(message, 'DEBUG');
  },
  info(message) {
    this.writeFile(message, 'INFO');
  },
  /* write to server.log */
  writeFile(message, level) {
    if(process.platform === 'darwin') {
      var _path = __dirname + '/server.log';
      var _time = Date().split(' ').slice(0,-2).join(' ');
      var _lineending = '\n';
    } else if(process.platform === 'win32') {
      var _path = __dirname + '\\server.log';
      var _time = Date().split(' ').slice(0,-4).join(' ');
      var _lineending = '\r\n';
    }
    fs.open(_path, 'a', function(error, fd) {
      if (error)
        return console.error('Couldn't open log file: ${path}');
      var log_message = `${_time} - ${level}: ${message}.${_lineending}`;
      var buffer = new Buffer(log_message);
      fs.write(fd, buffer, 0, buffer.length, null, function(error) {
        if (error)
          return console.error('Couldn't write to log file: ${path}');
        fs.close(fd, () => { buffer = null; });
      });
    });
  }
}

/* synchronous logging */
exports.logSync = {
  /* message level */
  error(message) {
    this.writeFile(message, 'ERROR');
  },
  warning(message) {
    this.writeFile(message, 'WARNING');
  },
  debug(message) {
    this.writeFile(message, 'DEBUG');
  },
  info(message) {
    this.writeFile(message, 'INFO');
  },
  /* write to server.log */

```



```

writeFile(message,level) {
  if(process.platform === 'darwin') {
    var _path = __dirname + '/server.log';
    var _time = Date().split(' ').slice(0,-2).join(' ');
    var _lineending = '\n';
  } else if(process.platform === 'win32') {
    var _path = __dirname + '\\server.log';
    var _time = Date().split(' ').slice(0,-4).join(' ');
    var _lineending = '\r\n';
  }
  var log_message = `${_time} - ${level}: ${message}.${_lineending}`;
  var buffer = new Buffer(log_message);
  try {
    var fd = fs.openSync(_path, 'a');
  } catch(e) {
    return console.error('Couldn't open log file: ${path}');
  }
  try {
    fs.writeSync(fd, buffer, 0, buffer.length, null);
  } catch(e) {
    return console.error('Couldn't write to log file: ${path}');
  }
}
}

#####
server/server-test.js
#####

var net = require('net');
var timers = require('timers')

var nss = new net.Socket();

var tcpClient;

nss.connect(9001, '127.0.0.1', function() {
  tcpClient = nss;
  console.log('Connected to TCP server.');
```

```

});

nss.on('data', function(data) {
  console.log('Received on TCP: ' + data);
});

nss.on('close', function() {
  console.log('TCP connection closed.');
```

```

});

nss.on('error', (error) => {
  console.log('An error has occurred: ${error.message}');
  process.exit(1);
})
```

```

function generateRandom(low,high) {
  return Math.random() * (high - low) + low
}
```

```

var t = timers.setInterval(() => {
```

```

var val_0 = Math.round(generateRandom(0,4));;
var val_1 = Math.round(generateRandom(0,100));
var val_2 = Math.round(generateRandom(0,500));
var val_3 = Math.round(generateRandom(0,500));
var val_4 = Math.round(generateRandom(0,100));
var val_5 = Math.round(generateRandom(0,100));
var val_6 = Math.round(generateRandom(0,100));
var val_7 = Math.round(generateRandom(0,100));
try {
    tcpClient.write('robot_' + val_0 + ',' + val_1 + ',' + val_2 + ',' + val_3 + ',' +
        val_4 + ',' + val_5 + ',' + val_6 + ',' + val_7 + ',');
} catch(e) {
    console.error('An error has occurred: ${e.message}');
    process.exit(1);
}
}, 100);

```

```
#####
```

```
server/package.json
```

```
#####
```

```

{
  "name": "mech_server",
  "version": "1.0.0",
  "description": "Senior Design Project",
  "main": "server.js",
  "dependencies": {
    "chalk": "^1.1.3",
    "ws": "^2.2.0"
  },
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/KittsSeniorDesign/WebUI.git"
  },
  "author": "Marton Demeter",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/KittsSeniorDesign/WebUI/issues"
  },
  "homepage": "https://github.com/KittsSeniorDesign/WebUI#readme",
  "devDependencies": {}
}

```

A.3.3 Ground Control

```
#####
```

```
DTtoTCP.java
```

```
#####
```

```
package edu.scu.engr.rsl.util;
```

```

import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.Socket;

```

```

import java.net.UnknownHostException;
import java.util.Iterator;
import java.util.Observable;
import java.util.Observer;
import java.util.Vector;

import com.rbnb.sapi.ChannelMap;
import com.rbnb.sapi.ChannelTree;
import com.rbnb.sapi.SAPIException;
import com.rbnb.sapi.Sink;
import com.rbnb.sapi.Source;

import edu.scu.engr.rsl.dataturbine.DataClient;

public class DTtoTCP implements Observer {
    protected DataClient dataClient;
    protected Thread DTThread;
    protected Vector<String> channelList = new Vector<String>();
    protected Socket socket;
    protected PrintWriter sockout;
    protected SockInThread sockinThread;
    protected String tcpIP = "localhost", DTIP = "localhost", sinkName = "_UI_Sink",
        sinkChannelName = "*/states", sourceName = "_UI_Source", sourceChannelName = "_UI";
    protected int tcpPort = 9001, DTPort = 3333;
    protected ChannelMap sourceMap;
    protected Source source;

    public static final String requestDTconfig = "requestDTconfig()", reconfSink2Chan =
        "reconfSink2Chan()";

    class SockInThread extends Thread {

        protected BufferedInputStream sockin;
        protected String catchpattern = "[0-9]+[a-zA-Z]";
        SockInThread(BufferedInputStream sockin) {
            this.sockin = sockin;
        }

        // public so that this can be tested
        public void handleReconfRequest(String str) {
            String[] splitStrings = str.substring(reconfSink2Chan.length()).split(" ");
            // do this to remove the , at the end of the sink argument
            String sinkName = splitStrings[0].substring(0, splitStrings[0].length()-1);
            // removes trailing )
            String sourceChannelName = splitStrings[1].substring(0,
                splitStrings[1].length()-1);
            String outData = sinkName + " r " + sourceChannelName;
            try {
                sourceMap.PutDataAsByteArray(0, outData.getBytes());
                source.Flush(sourceMap);
                System.out.println("Supposedly asked to reconfigure " + outData);
            } catch (SAPIException e) {
                e.printStackTrace();
            }
        }

        @Override
        public void run() {
            int avail = 0;

```

```

while(!isInterrupted()) {
    try {
        avail = sockin.available();
        if(avail > 0) {
            byte[] data = new byte[avail];
            sockin.read(data, 0, avail);
            String str = "";
            for(int i = 0; i < data.length; i++) {
                str += (char)data[i];
            }
            if(str.equals(requestDTconfig)) {
                sockout.println("DTConfig: "+getChansAndSinks());
            } else if(str.length() > reconfSink2Chan.length() &&
                str.substring(0,
                    reconfSink2Chan.length()).equals(reconfSink2Chan)) {
                handleReconfRequest(str);
            } else {
                System.out.print("sending to DT as bytes:");
                System.out.println(str);
                String[] strs = str.split(" ");
                String outputStr = "";
                for(int i = 0; i < strs.length; i++) {
                    if(i > 0) {
                        outputStr += " ";
                    }
                    if(strs[i].matches("[0-9]+[a-zA-Z]+_[0-9]+")) {
                        String pos = strs[i].split("[a-zA-Z]+_[0-9]+")[0];
                        String roboNum = strs[i].split("[a-zA-Z]+_")[1];
                        String roboName = strs[i].split("[0-9]+")[1];
                        outputStr += pos + ";" + roboName + roboNum;
                    } else {
                        outputStr += strs[i];
                    }
                }
                String[] dataStr = outputStr.split(";");
                for(String s : dataStr) {
                    System.out.println(s);
                    sourceMap.PutDataAsByteArray(0, s.getBytes());
                    source.Flush(sourceMap);
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    } catch (SAPIException e) {
        e.printStackTrace();
    }
}

DTtoTCP() {
    initialize();
}

DTtoTCP(String tcpIP, int tcpPort) {
    this.tcpIP = tcpIP;
    this.tcpPort = tcpPort;
}

```

```

        initialize();
    }

    DTtoTCP(String DTIP, int DTPort, String sinkName, String sinkChannelName, String
        sourceName, String sourceChannelName, String tcpIP, int tcpPort) {
        this.DTIP = DTIP;
        this.DTPort = DTPort;
        this.sinkName = sinkName;
        this.sinkChannelName = sinkChannelName;
        this.sourceName = sourceName;
        this.sourceChannelName = sourceChannelName;
        this.tcpIP = tcpIP;
        this.tcpPort = tcpPort;
        initialize();
    }

    protected void initialize() {
        setupDataTurbineSource();
        setupSocket();
        startDataClient();
    }

    protected void setupSocket() {
        try {
            socket = new Socket(tcpIP, tcpPort);
            socket.setTcpNoDelay(true);
            sockout = new PrintWriter(socket.getOutputStream(), true);
            sockinThread = new SockInThread(new
                BufferedInputStream(socket.getInputStream()));
            sockinThread.start();
            System.out.println("Web Talker started.");
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host " + tcpIP);
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection to " + tcpIP);
            System.exit(1);
        }
    }

    protected void startDataClient() {
        dataClient = new DataClient(DTIP, Integer.toString(DTPort), sinkName);
        dataClient.addObserver(this);
        dataClient.addSinkChannel(sinkChannelName);
        channelList.add(sinkChannelName);
        dataClient.connect();
        dataClient.subscribe();
        DTThread = new Thread(dataClient);
        DTThread.start();
    }

    protected void setupDataTurbineSource() {
        sourceMap = new ChannelMap();
        sourceMap.PutTimeAuto("timeofday");
        try {
            sourceMap.Add(sourceChannelName);
        } catch (SAPIException e) {
            e.printStackTrace();
        }
    }

```

```

        source = new Source();
        try {
            source.OpenRBNBConnection(DTIP + ":" + DTPort, sourceName);
            source.Register(sourceMap);
        } catch (SAPIException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void update(Observable o, Object arg) { // MatlabController update in Scot's
        stuff
        if(o instanceof DataClient) {
            if(channelList.size() > 0) {
                //Get data out of the ChannelMap
                ChannelMap v_map = (ChannelMap) arg;
                for(int i=0; i<channelList.size(); i++) {
                    byte[] v_data = v_map.GetData(i);
                    System.out.println(v_map.GetName(i));
                    if(v_data != null && v_data.length > 0) {
                        String data = "";
                        for(int j = 0; j < v_data.length; j++) {
                            data += (char)v_data[j];
                        }
                        // prepends the data being sent to the UI with the name of the source
                        // it came from
                        data = v_map.GetName(i).split("/")[0] + ", 0, 0, " + data;
                        socketout.println(data);
                        System.out.println("sending to UI:" + data);
                    }
                }
            }
        }
    }

    protected String getChansAndSinks() {
        ChannelMap cm = new ChannelMap();
        try {
            cm.Add("*/");
            Sink sink = new Sink();
            sink.OpenRBNBConnection((DTIP + ":" + DTPort), "_DTConfigViewer");
            sink.RequestRegistration(cm);
            ChannelMap resultcm = sink.Fetch(-1, cm);
            ChannelTree ct = ChannelTree.createFromChannelMap(resultcm);
            @SuppressWarnings("unchecked")
            Iterator<ChannelTree.Node> it = ct.iterator();
            String dtconf = "";
            while(it.hasNext()) {
                ChannelTree.Node n = it.next();
                System.out.print(n.getFullName() + " " + n.getType() + ", ");
                if(!n.getType().toString().equals("Source") &&
                    !n.getType().toString().equals("Folder") && n.getFullName().charAt(0) !=
                    '_') {
                    dtconf += n.getFullName() + " " + n.getType();
                    dtconf += ",";
                }
            }
            if(dtconf.length() > 0) {
                // substring removes the trailing comma

```

```

        return dtconf.substring(0, dtconf.length()-1);
    } else {
        return dtconf;
    }
} catch (SAPIException e) {
    e.printStackTrace();
}
return "";
}

// sampleUIData should be of the form "reconfSink2Chan(sinkName, channelName)" without
// the quotes
public void testSinkReconfiguration(String sampleUIData) {
    sockinThread.handleReconfRequest(sampleUIData);
}

// args should be = [v_hostIP, v_hostPort, v_sourceName] referencing the class
// constructor
public static void main(String[] args) {
    DTtoTCP d = new DTtoTCP("localhost", 9001);
    //System.out.println("\nDTConfig: "+d.getChansAndSinks());
    try {
        System.in.read();
    } catch (IOException e) {
        e.printStackTrace();
    }
    d.testSinkReconfiguration("reconfSink2Chan(robot_2, matlab/commands)");
}
}

#####
MECHDTConnection.m
#####

function self=MECHDTConnection(robotName,ipAddress,tcpPort)

self.MECH_send = @MECH_send;
connect = controller(ipAddress,tcpPort,'matlab');
funct = registerfunction(connect,'MECHDataParser','*/MECHdata');
channelidx = addcommandchannel(connect,'commands');
start(connect);
scrdstid = 1;
controlschem = 2;
function funct = datamap(inp, in_min, in_max, out_min, out_max)
    funct = (inp - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
end
position = [0 0 0 0];
sendCounter = 0;

function MECH_send( inputVector )

    sendCounter = sendCounter+1;
    disp(sendCounter);
    inputVector = uint16([datamap(inputVector(1), -1.0, 1.0, 0.0, 2048.0)
        datamap(inputVector(2), -1.0, 1.0, 0.0, 2048.0)]);

    topleft = uint8(bitshift(inputVector(1), -8));
    botleft = uint8(bitshift(bitshift(inputVector(1), 8), -8));

```

```

    topright = uint8(bitshift(inputVector(2), -8));
    botright = uint8(bitshift(bitshift(inputVector(2), 8), -8));

    %disp(inputVector);
    A = uint8([screendstid controlschem botright topright botleft topleft]);
    disp(position);
    sendcommand(connect, channelidx, A);
end

end

#####
MECHDataParser.m
#####

% This code was written in order to test DataTurbines upload and download
% capability on XBees, Arduinos, etc. It is called by testScript.m.
%
% Written by: Jasmine Cashbaugh
% Written on: August 19, 2011
%
% *****

function dataOut = MECHDataParser(dataIn)
    byte_array = dataIn.getBytes;
    dataOut = char(java.lang.String(byte_array)); % Converts the byte
        % array to a string.

    % The following two lines are not needed to use dataParser. I kept them
    % in for debugging purposes as they display the data to the screen.
    %     disp('This is called and the data is: ');
    %     disp(dataOut);

    % Run the command.
    %     global connect % In the code for debugging. Will be removed in final
    %                     % version.
    %     global connect1
    %     global connect2
    %     global connect connect1 connect2 connect3

    assignin('base','position', dataOut);
end

```
