Borja Rojo
CSC232
Lab 2
Howard

Lab 2

Mobile Robot Simulation

I affirm that I have not given or received any unauthorized help on this assignment, and that all work is my own.

The objective of this was was to create two executables that represent the navigator for the Turtlebot and a simulator of a Turtlebot. The navigator sends corresponding commands to various channels and also receives data from the Turtlebot or simulator program about the current state of the robot. The simulator is meant to replace the Turtlebot by receiving command messages, handling them as the Turtlebot would, and sending state information for the navigator to receive.

The structure of the navigator is split into two parts. The first is an object called Commander that handles the commands to be sent to the Turtlebot. The second is an object called StateTracker which stores the data being received that represents the odometry of the Turtlebot.

Commander holds four doubles for the linear velocity, angular velocity, time, and rate as well as a node and two publishers that publish to geometry_msgs::Twist and std_msgs::Empty. The constructor is overloaded to have a default set of values for the doubles or to be able to custom set those parameters. Each instantiates the publishers as well.

There are a number of setters as well for each of the variables. A setter was also implemented to be able to set all the parameters at once. Two getters were added to get the time and rate variables.

There is an overloaded publishing function that can publish both stored and passed doubles as the linear and angular velocities in the Twist message. A caveat to note is that the custom publishing function does not reassign the stored variables to the passed parameters. I felt this was unnecessary and possibly problematic as it allows for the Commander to publish commands without affecting itself. Later, the need to implement maneuvering commands may allow for the Commander to use this feature to both execute those commands and keep it's general heading.

StateTracker holds five different vectors for the x and y linear position, the yaw, the linear velocity, and angular velocity along time and a nod and subscriber to /odom. It also has it's own filestream. Its constructor only instantiates the subscriber.

The callback function for the subscriber receives a nav_msgs::Odometry message and pushes values in the message into the corresponding vectors. This then allows for the StateTracker to store different snapshots of the Turtlebot's odometry. This function uses converts the quaternion in the message to a yaw, which is then stored. This function is also overloaded to be able to store a set of values manually passed in.

A function to convert a quaternion to a yaw was implemented to handle the form of the Odometry message. It passes a vector as a parameter and uses atan2 to compute.

The StateTracker also has two methods for accessing the stored states. One prints the states in order out onto the command line in a tabular delimited format, and the other writes the states to files in the same way.

Borja Rojo
CSC232
Lab 2
Howard

The main of this excutable consists of the basic setup for gengetopt command line parsing, ROS initialization, both object initialization and a command loop. For this lab, a set of commands was also set to be able to choose between the different trails on the command line. At the end of the main, the StateTracker writes its data to a file. All the command variables, the file name, and the trail to run can all be set in the command line.

The structure of the simulator only has one object called Odometry. This class handles everything. It has five doubles for the odometry values, a ros::Time class to interact with time passage, a double dt that stores a change in time, a vector that holds the noise coefficients for this lab, and a node, publisher for the odometry, and two subcribers for the Twist and Empty messages. The constructor sets all the odometry variables to 0, the Time to Time.now(), and initialized the communication paths. The constructor is overloaded so that the noise coefficient vector could be set at initialization if so desired. It can also be set separately.

There is a function that publishes to /odom. It creates a vector that holds a quaternion for the orientation portion of Odometry message. This quaternion is created with a yaw-to-quaternion function that was implemented. The message's parameters are correspondingly set using this quaternion and the stored variables in the object. This will then be published to /odom for any subscriber.

The callback function for the Twist updates the current odometry in accordance to the data received. First, the time difference between the last update is established. Then, it decides what kind of update to perform depending on whether or not the angular velocity is equal to 0. If it's not, then the algorithm in table 5.3 of the textbook is performed. If it is, a different set of updates is performed (as to not divide by zero).

The callback function for the Empty messages set all the stored variables to 0 and resets the Time variable to the current time.

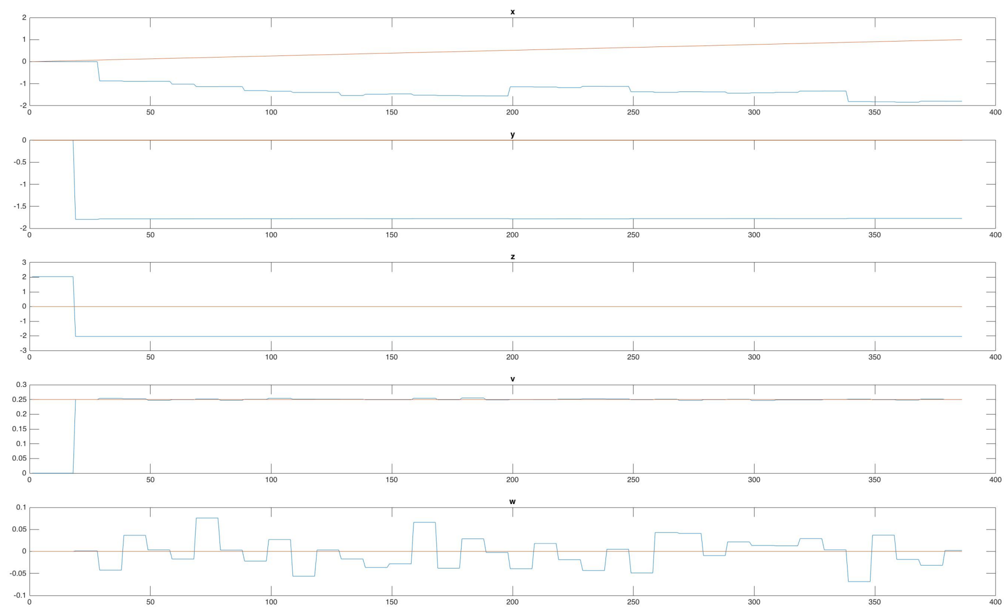
The noise sample used in the Twist callback function is created on its own using the formula from the lab. The value passed for the bound in the callback function, when this function is used, is created using the stored vector for the noise coefficients.

The function to convert yaw to a quaternion is implemented here and used in the odometry publisher. The expression used is simplified from the one given in the lab. It returns a vector.

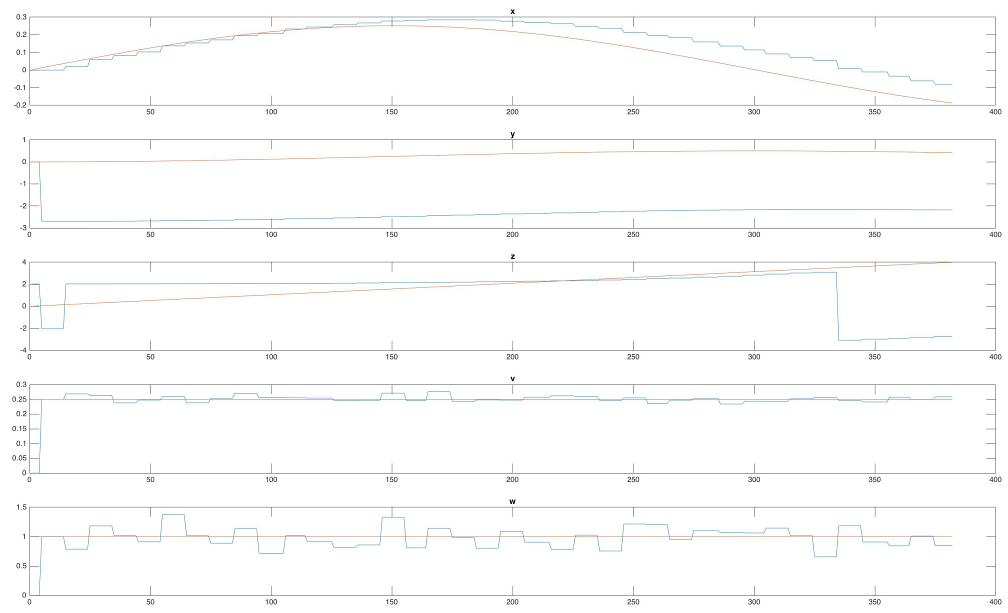
The main of this function is rather simple. First, it handles the command line parsing and ROS initialization. Then it creates two vectors that represents the noise coefficients given in the lab. Then it creates an Odometry object and takes in one of the coefficient vectors. After this set up phase, it goes into the publishing loop, which spins and also activates the subscribers. The simulator does not exit on it's own; it continues until it is stopped.

Borja Rojo
CSC232
Lab 2
Howard

Lab 2 Trail 1 Coefficients 1

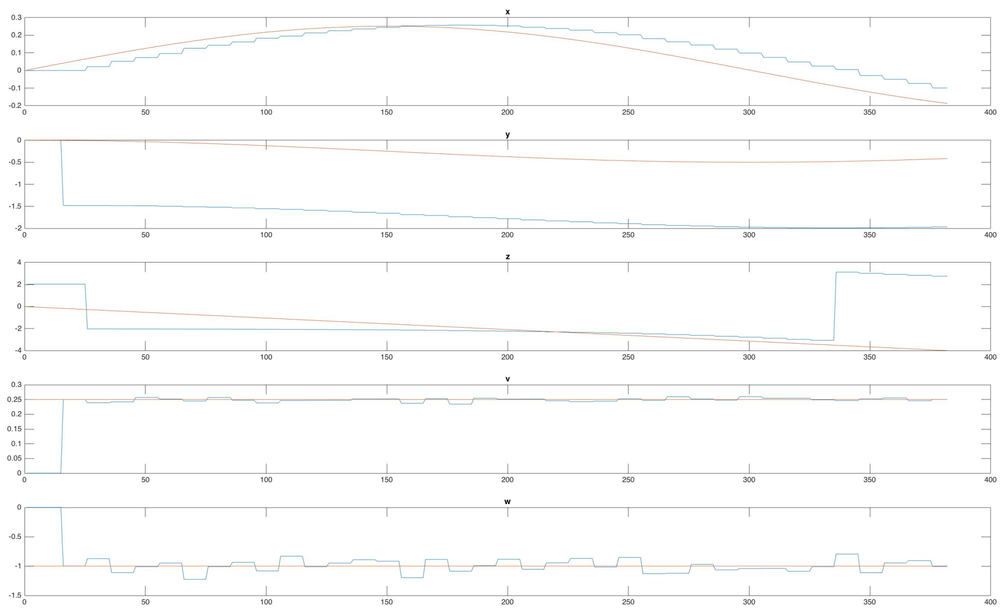


Lab 2 Trail 2 Coefficients 1

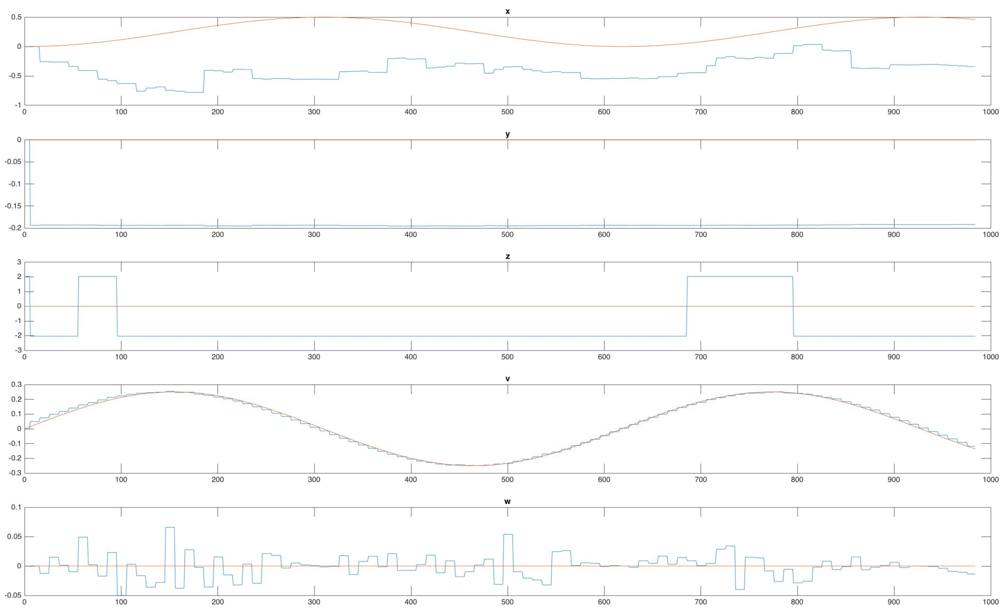


Borja Rojo
CSC232
Lab 2
Howard

Lab 2 Trail 3 Coefficients 1

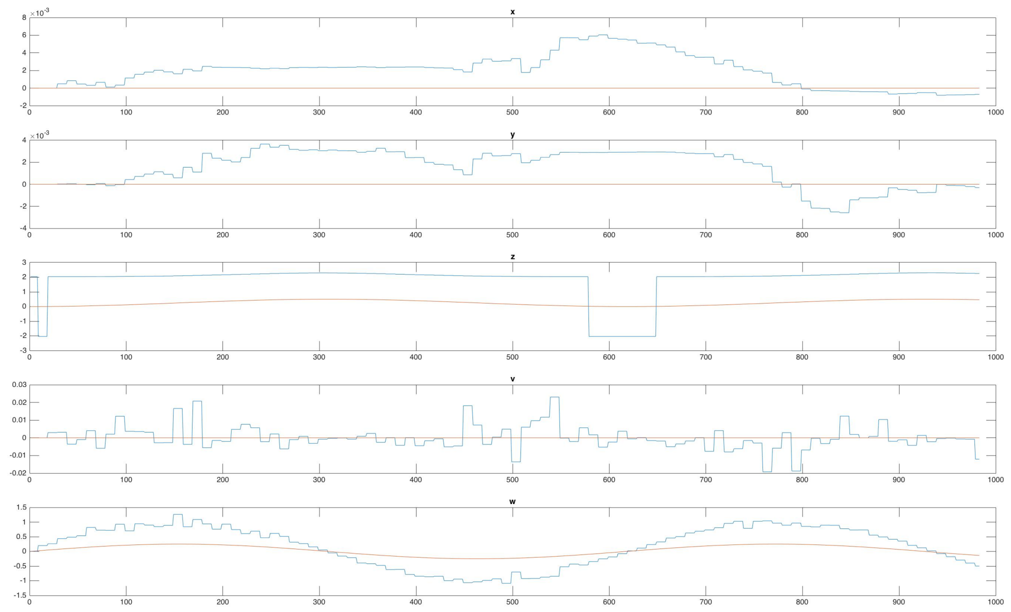


Lab 2 Trail 4 Coefficients 1



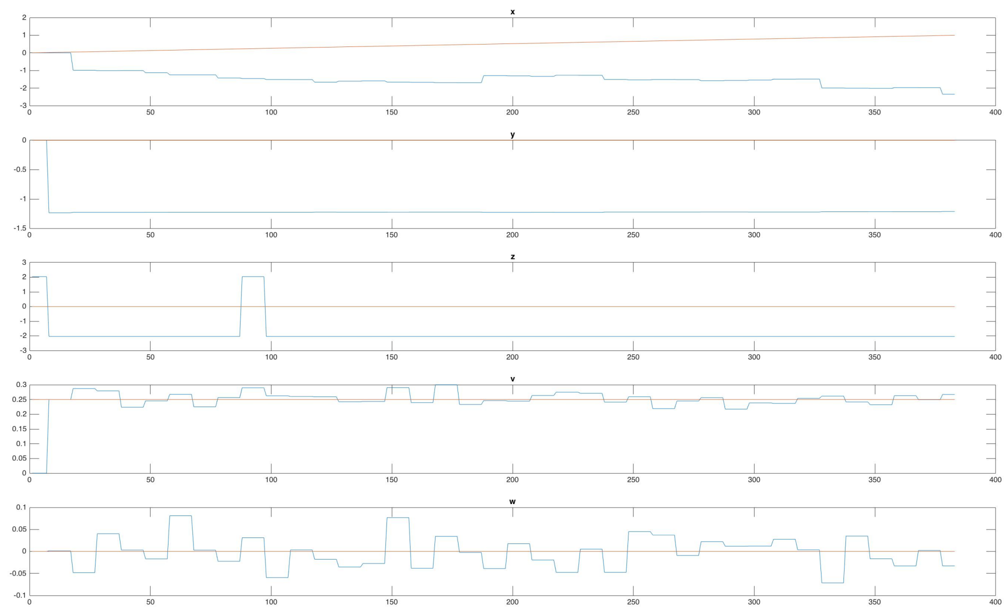
Borja Rojo
CSC232
Lab 2
Howard

Lab 2 Trail 5 Coefficients 1

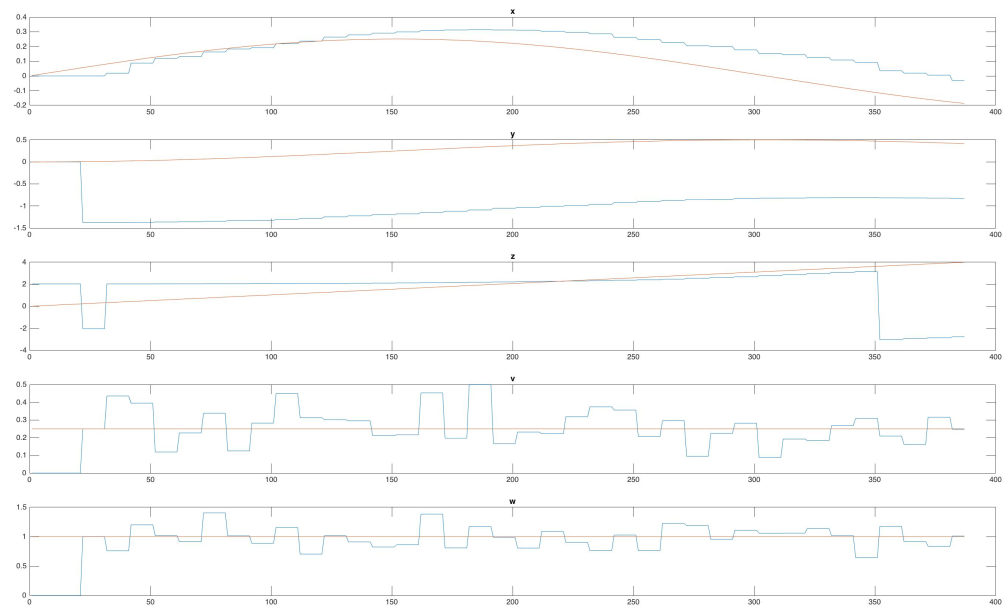


Borja Rojo
CSC232
Lab 2
Howard

Lab 2 Trail 1 Coefficients 2

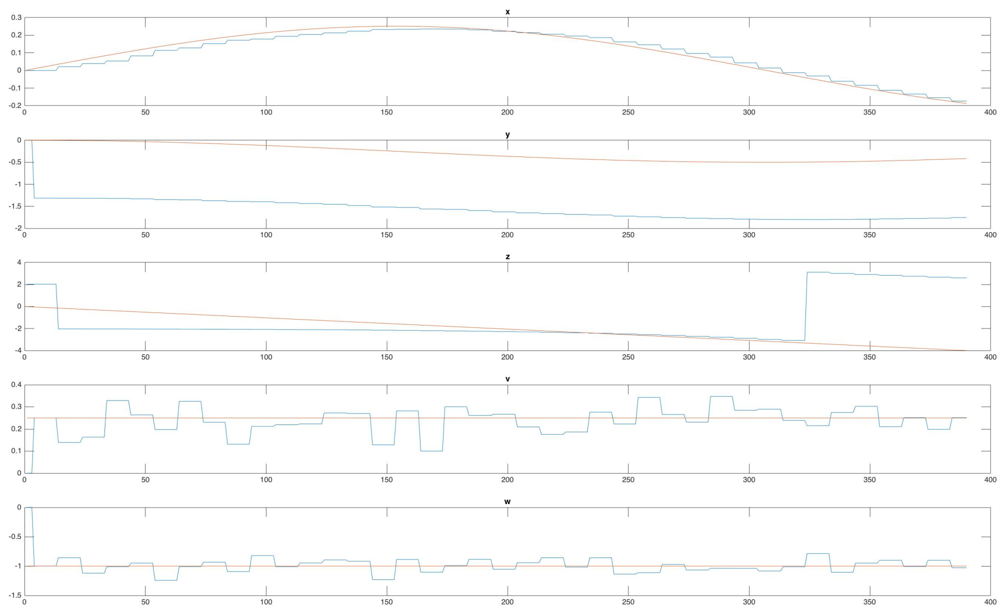


Lab 2 Trail 2 Coefficients 2

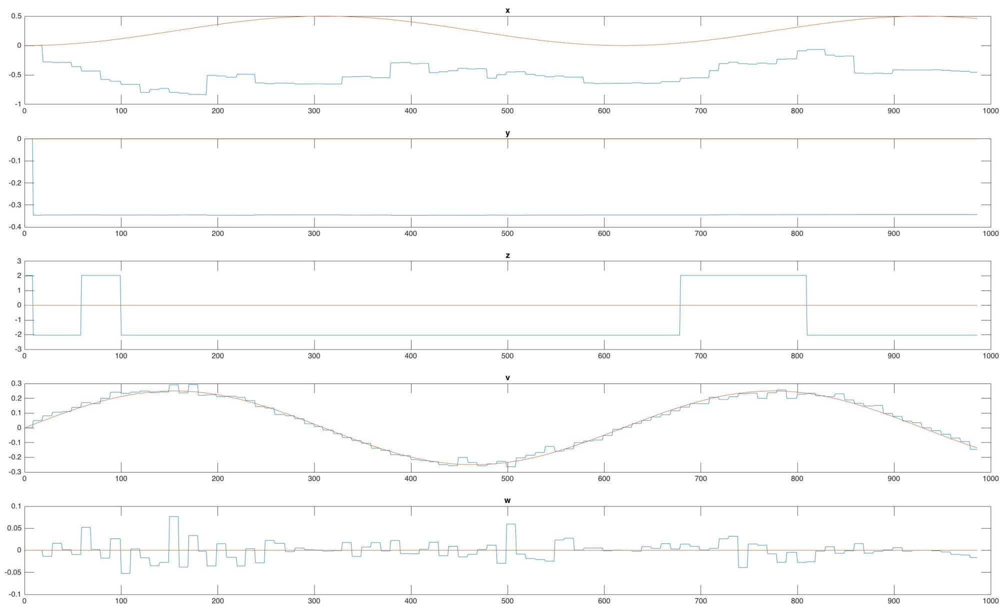


Borja Rojo
CSC232
Lab 2
Howard

Lab 2 Trail 3 Coefficients 2

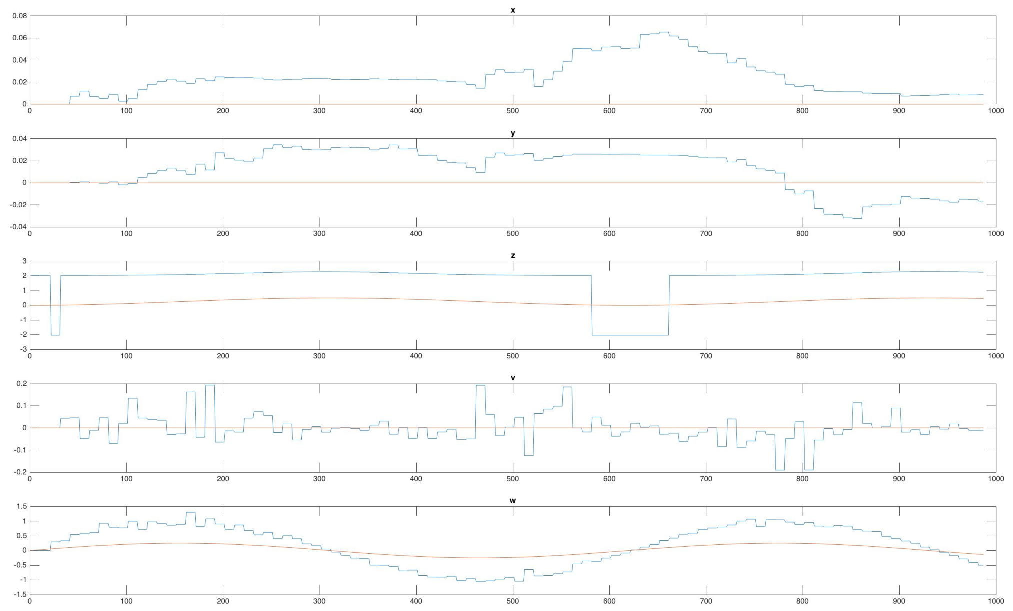


Lab 2 Trail 4 Coefficients 2



Borja Rojo
CSC232
Lab 2
Howard

Lab 2 Trail 5 Coefficients 2



The plots are accurate in some cases and not in others. There are some common trends among them all, though, which shows consistency. In general, the graphs show comparable trends.

In the first trials, the x linear position is inverted with what was expected, though it did grow linearly. In the rest of the trails, the trends matched.

The y linear position definitely had some issues. The trends of the functions matched, but the data showed a vertical shift in the trials where the y position was changing.

The yaw of the data is strange. The yaw always bounces between 2 and -2 depending on whether or not the expect value is positive or negative, respectively. It was expected that there was some sort of mathematical mistake in the code that was causing this, probably having to do with integer mathematics, but none could be found.

Both the linear and angular velocities matched the expected trends throughout. The only exception lies in the expected amplitude of the angular velocity when it is oscillating. It was expected for there to be an issue in the coding for the plots, but none was found.

Although some of the issues were not found, they are expected to be remedied soon in order to properly simulate the robot.