ECE 232 Lab 1

Matthew Dombroski

February 15, 2016

Abstract

In this lab a basic publisher and subscriber node to interface with ROS and the turtlebot base were written in C++. In addition, GenGetOpt was used to parse command line input arguments to facilitate changing control parameters of the robot without hard-coding such parameters and requiring a recompile. The publisher node advertised basic open-loop control commands while the subscriber node collected Odometry data from the Turtlebot base. Data of the robot's position, yaw, angular and linear velocity was gathered and plotted and then compared to expected, or ideal, odometry plots.

1 Code Implementation

1.1 Parts 1 to 3

A very basic ROS publisher node was implemented to advertise type Twist messages to the `/cmd_vel_mux/input/navi` topic. The first iteration of the publisher node used hard coded values for the Twist parameters `angular.z` and `linear.x` and for the length of time the commands would be published. The second iteration of the publisher node was to incorporate command line input arguments to set the linear and angular velocities and time to run. These input parameters were defined using GenGetOpt, where a ros_publisher.ggo file defined the command line options available to the user to input. In the ros_publisher.cpp file before the publisher node is created these options are parsed by GenGetOpt and assigned to variables inside the main function. These values are then passed to the Twist message to be advertised to the Turtlebots navi topic. In addition the time input option is used to set the runtime of the main execution loop.

A basic subscriber node was also created. The subscriber node subscribed to the `/odom` topic and passed the odometry data to a callback function which printed the data to the command line.

1.2 Part 4

The publisher node from the previous parts was modified for this section of the lab to include a hard coded section to control the robot's angular and linear velocity with a sine function.

Depending on which trial was being completed one of the lines of code was commented out so that only one of the velocities was controlled via a sine curve. If both lines were commented out the robot would be able to accept velocities from the command line.

The subscriber node for this part was modified to listen to specific fields of the `/odom` data and export the data into two files in CSV format, one file for pos data and another for twist data. These two files were written for each of the different configuration of commands specified in the lab. In particular the subscriber wrote the pos x, y, z, and w fields to the pos data file and the twist linear.x and angular.z data to the twist file.

1.3 Plotting

For the creation of the required plots MATLAB was used as the processing environment. Using the CSV file import function every data file was converted into matrix variables inside MATLAB which could then be used for plots or calculations. The robot's position, linear, and angular velocity were plotted direct from the matrix data. The yaw at each point in time was computed from quaternion data using the necessary transformation equation.

$$\psi = atan2(2(q_0q_3 + q_1q_2), 1 - 2(q_2^2 + q_3^2))$$

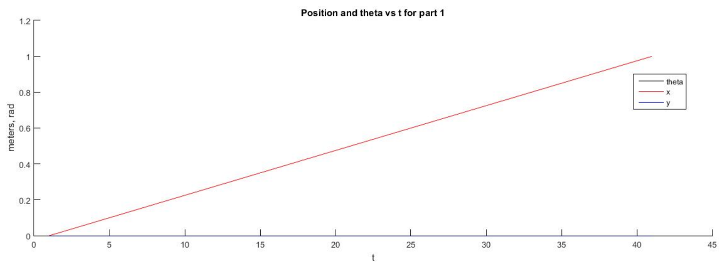
After these computations the yaw data could be plotted the same as any other robot parameter.

Note that for each plot of measured data the x-axis shows the indexes of the matrix values. For a 4 second trial there are 200 indexes of data, corresponding to a 0.02s sampling interval from the sensors. Similarly for a 10 second trial there are 500 samples, again corresponding to a 0.02s sampling interval for the sensor read rate. For the predicted data values were generated in 0.1s intervals and so the time axis on the predicted plots is in tenths of a second.

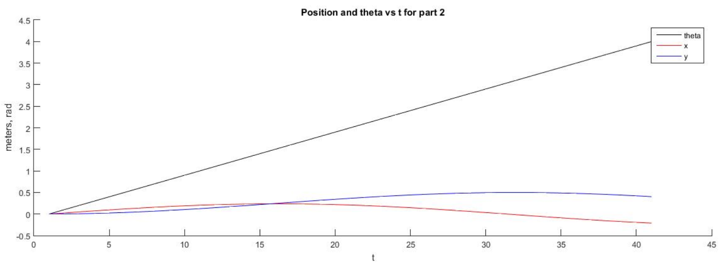
2 Data and Analysis

The most striking observation from the data was how different the physical movement of the robot was in comparison to the predicted or theoretical movement. For linear velocity changes acceleration and deceleration played a large role in shaping the output, causing large discrepancies between the commanded motion and executed motion. The best example of these differences was when the robot moved forward at 0.25 m/s. Because the robot needed some amount of time to accelerate the traveled distance was under 0.9m when the expected traveled distance was 1m.

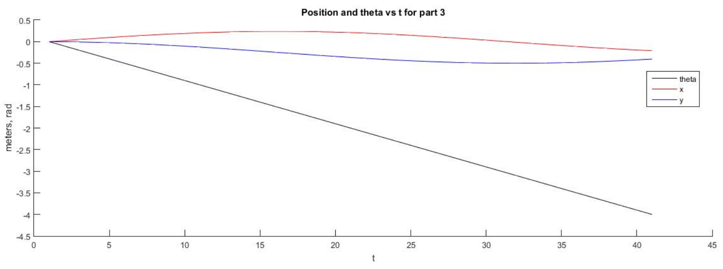
In addition the recorded data from the robot showed that the angular velocity control to be very noisy. All yaw vs time recordings were quite different than their ideal forms, deviating very quickly from the expected. path. The best example of this deviation is shown when the angular velocity is given as a sine wave. The recorded angular velocities look more similar to a square wave than a smooth sine wave.



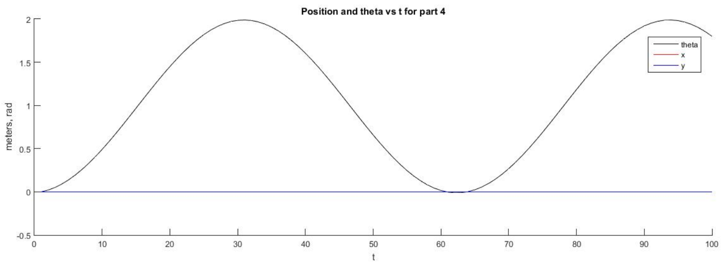
(a) Part 1



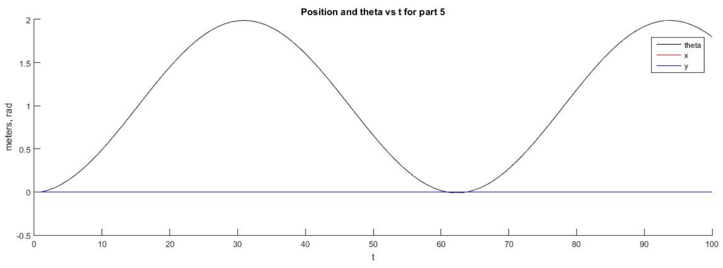
(b) Part 2



(c) Part 3

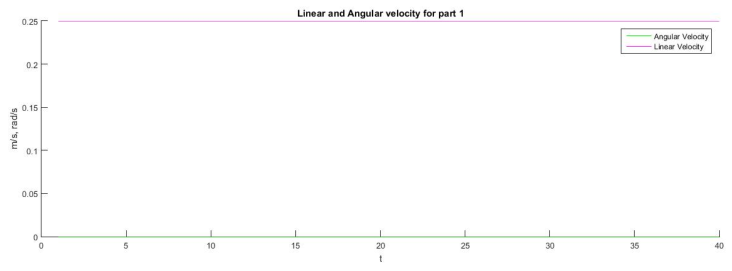


(d) Part 4

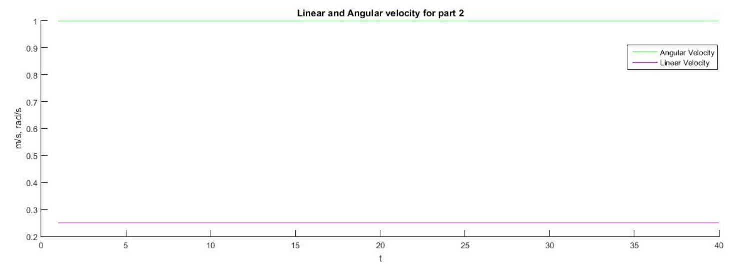


(e) Part 5

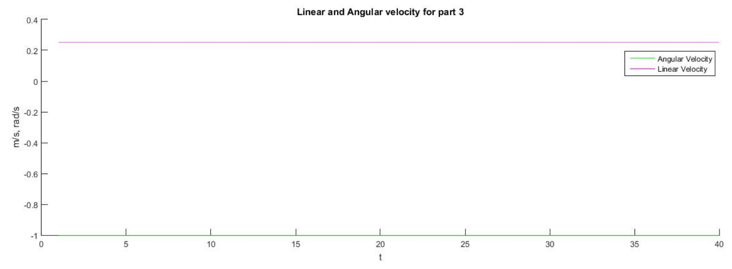
Figure 1: Predicted position and yaw vs time



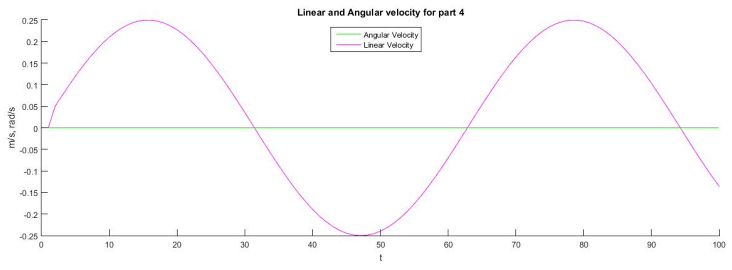
(a) Part 1



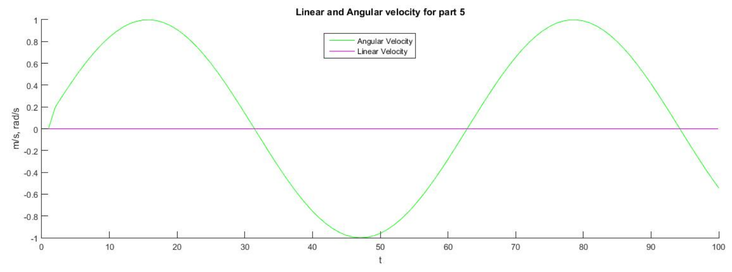
(b) Part 2



(c) Part 3

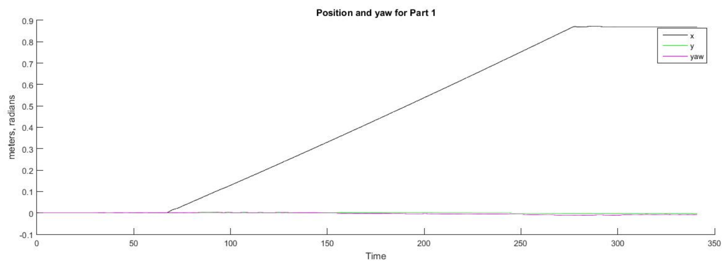


(d) Part 4

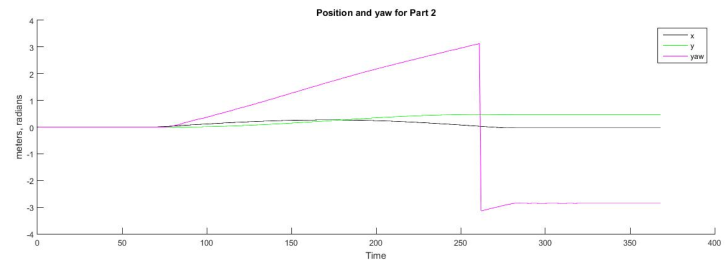


(e) Part 5

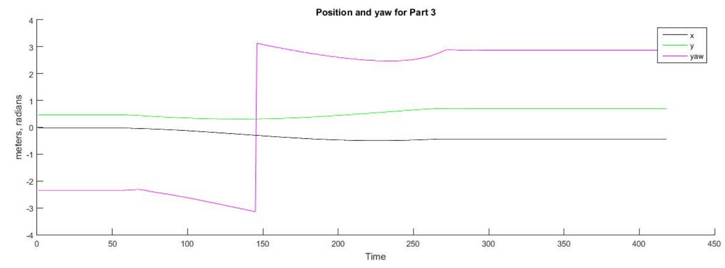
Figure 2: Predicted velocities vs time



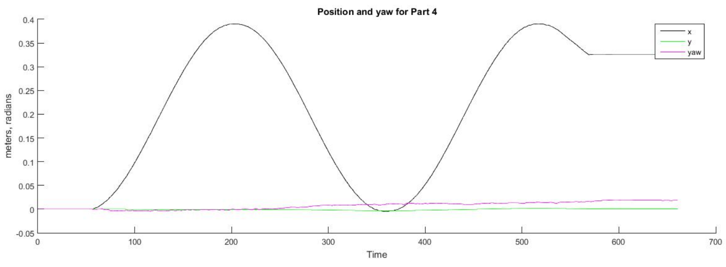
(a) Part 1



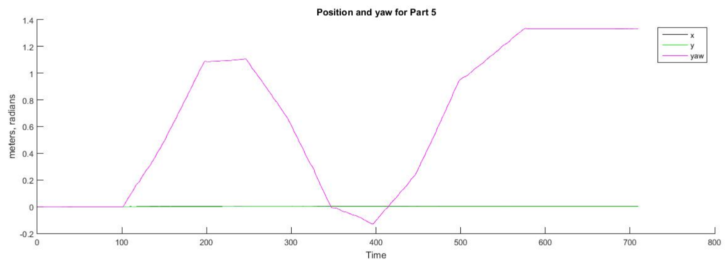
(b) Part 2



(c) Part 3

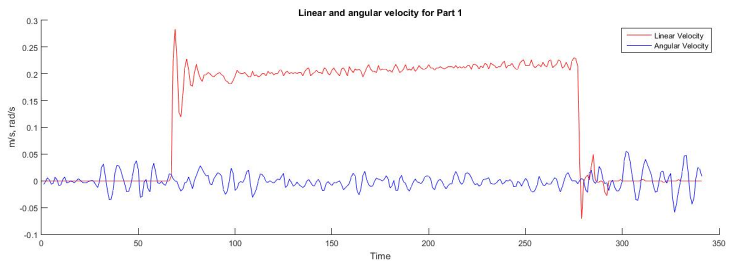


(d) Part 4

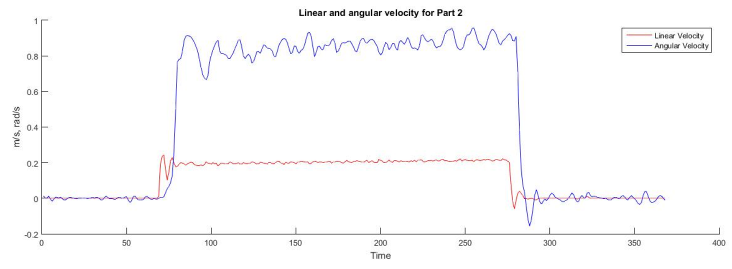


(e) Part 5

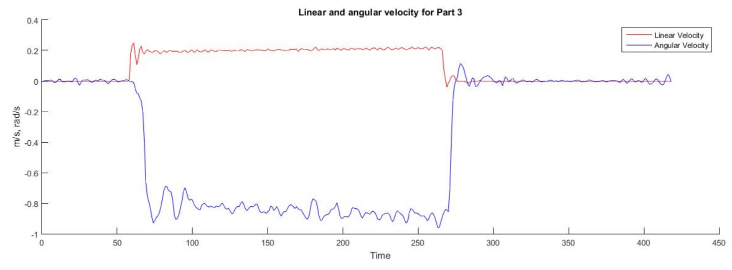
Figure 3: Measured position and yaw vs time



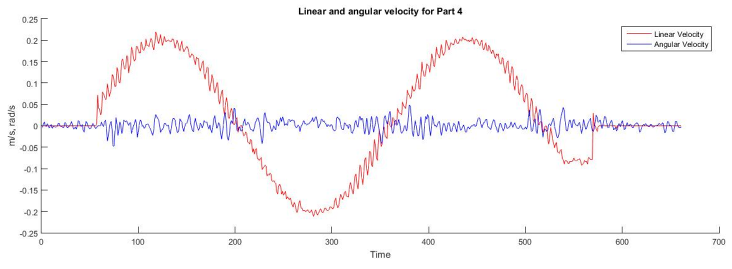
(a) Part 1



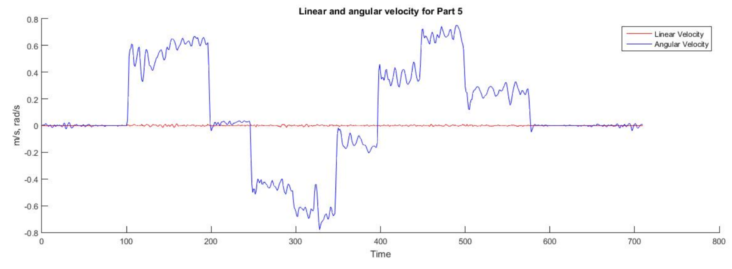
(b) Part 2



(c) Part 3



(d) Part 4



(e) Part 5

Figure 4: Measured velocities vs time

3 Laboratory Questions

3.1 Question 1

Figures 1 and 2 show the predicted robot position and motion. MATLAB was used to calculate the predicted path of movement and the predicted velocities. Similar to the subscriber node created in C++, several MATLAB functions were written to calculate the position, yaw, and velocities in 0.1s intervals, with ideal functions being used for the computations rather than recording robot data. These computations were accomplished by first initializing the position and yaw to 0. Then in 0.1s intervals the delta yaw angle was calculated based on the angular velocity and adding this angle to the current yaw angle. From this new yaw angle the next incremental x and y were computed. For the trials where the velocities were a function of sine the current velocity was calculated by inputting the current time to the sine function and then basing position calculations off this new velocity.

While somewhat overkill for what was required, computing the predicted paths in this manner is somewhat similar to the manner in which the simulator in lab 2 will compute robot motion and can be used as a template for the next lab.

3.2 Question 2

Figures 3 and 4 show the measured robot position and velocity data. While the measured data never disagrees qualitatively with the predicted plots (ie the robot turned left when commanded to go right), the magnitudes of the measured data do differ significantly from the predicted plots. In almost all of the measured data plots there was a significant amount of noise in the robot's velocity and position measurements. This noise is a result of random sensor noise, mechanical slippage in the motors and gear systems and environment variables such as an uneven floor surface. In addition factors such as acceleration further skewed recorded data, causing the robot to travel shorter distances and react more slowly than predicted.

3.3 Question 3

The `/mobile_base/events/bumper` topic contains the state data for the robot bumpers. The message type of this topic is texttt kobuki_msgs/BumperEvent. On each bumper press or release the id number of the bumper pressed/released and its state are advertised. If more than one bumper press event occurs at a time several events are successively advertised. To listen to these bumper events a subscriber node was subscribed to the `/mobile_base/events/bumper` topic and pointed to a callback function `print_msgs`. Inside the callback function the message variable `state` is checked and if the state is 1 (pressed) the name of the bumper that was pressed is printed to the terminal.