Borja Rojo

CSC232

Howard

Laboratory 1

University of Rochester

**Abstract**

This lab was an introduction to the core interactive features when programming for the Turtlebot. First, a subversion repository was made on the cycle1 CSUG servers. Allows projected to be developed with all the subversion features. Among those is the ability to access code from a variety of places which makes using different laptops in the lab much easier. Then, CMake was set up. CMake is a tool used to develop in C++ that allows for easy compiling and linking of multi-file projects. A CMakeLists.txt file was made that specifies what C++ Makefiles to create. CMakealso allows for the integration of gengetopt files, a file type that allows for custom command-line arguments to be specified. To use this, the gengetopt file, partner C++ code, and the CMakeLists.txt file need to be set up properly. This involves integrating gengetopt into the C++ code with proper includes and initialization in the mains as well as adding specific lines to the CMakeLists.txt file. After that, ROS’s features were explored to become more familiar with the ROS environment. Once roscore and roslaunch are running, an interface with the Turtlebot becomes active. This allows messages to be communicated between the computer and the Turtlebot through different “topics”. These topics hold different sets of information, such as linear and angular velocities, that the robot will listen to and follow as directions or will transmit as sensor data. In the code, this is accomplished through the use of subscribers and publishers. Two files were created that exemplify these exact features. The subscriber file listened to data about the wheel odometry from the robot, and the publisher published wheel commands to the robot. Then, the publisher was fitted with a set of commands for the Turtlebot to follow, and data was recorded from the subscriber about the Turtlebots movements.

**Code**

The Code was split into two files: the publisher and the subscriber.

The publisher has all the expected includes, such as ros.h, the command-line header created by the gengetopt file, and standard libraries like iostream and math.h. The publisher also has an include for geometry\_msgs/Twist.h so that it can communicate movement commands to the robot. Once the Publisher (object) and message are set up, the publisher runs through an open loop control sequence whose speed is regulated by a Rate object and a counter that limits how many times the loop is run through. Every cycle, the counter is incremented by one, the Twist message is set with a corresponding angular velocity, and the message is then published on the Twist channel. The Turtlebot then receives this information every loop and adjusts its motion accordingly.

The subscriber also has all the expected includes. The subscriber is responsible for writing the read odometrys to a file, so it has fstream, and subscribes to he nav\_msgs/Odometry.h channel, so it has that included as well. The subscriber reads information from the ROS topics with the use of a callback function. Every time a topic is updated, this function is called and run in the code. The subscriber’s only use for the information it receives to to write it to the file, so the callback function does just that. Its open loop is done using the ros::spin() function that continuously spins searching for changes in the topics.

**Questions**

Q1)

1.

2.

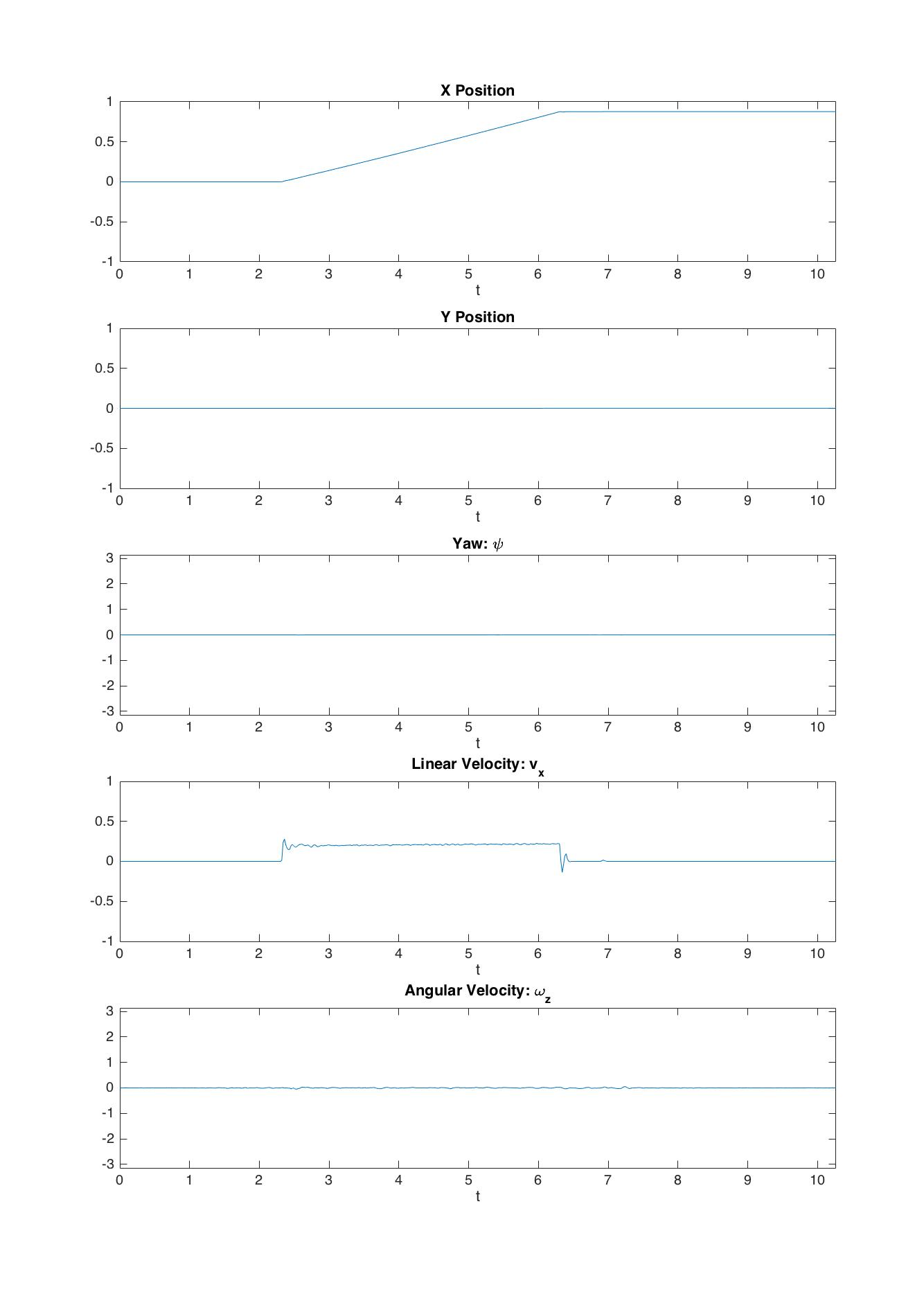
3.

4.

5.

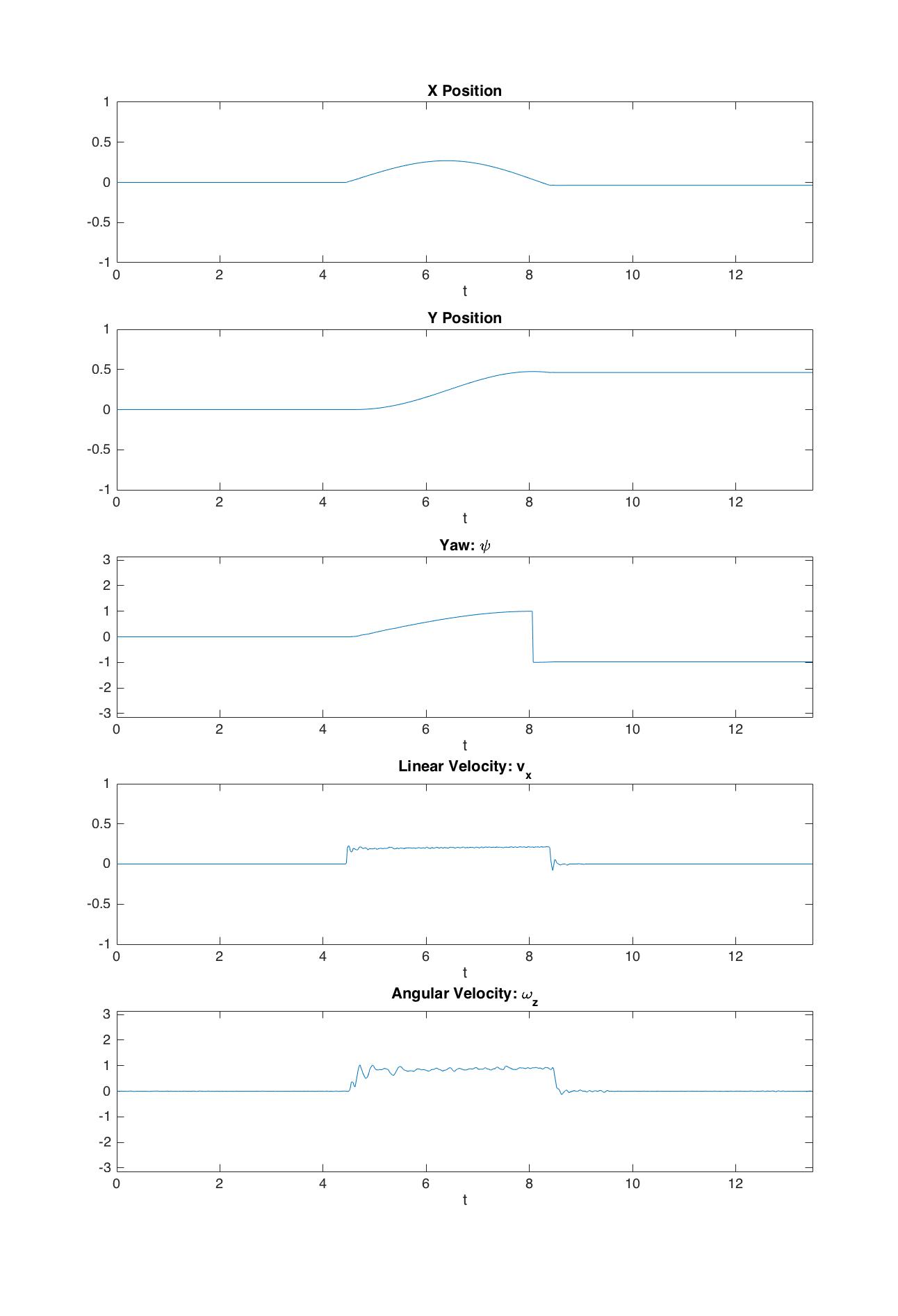
Q2)

Sequence 1



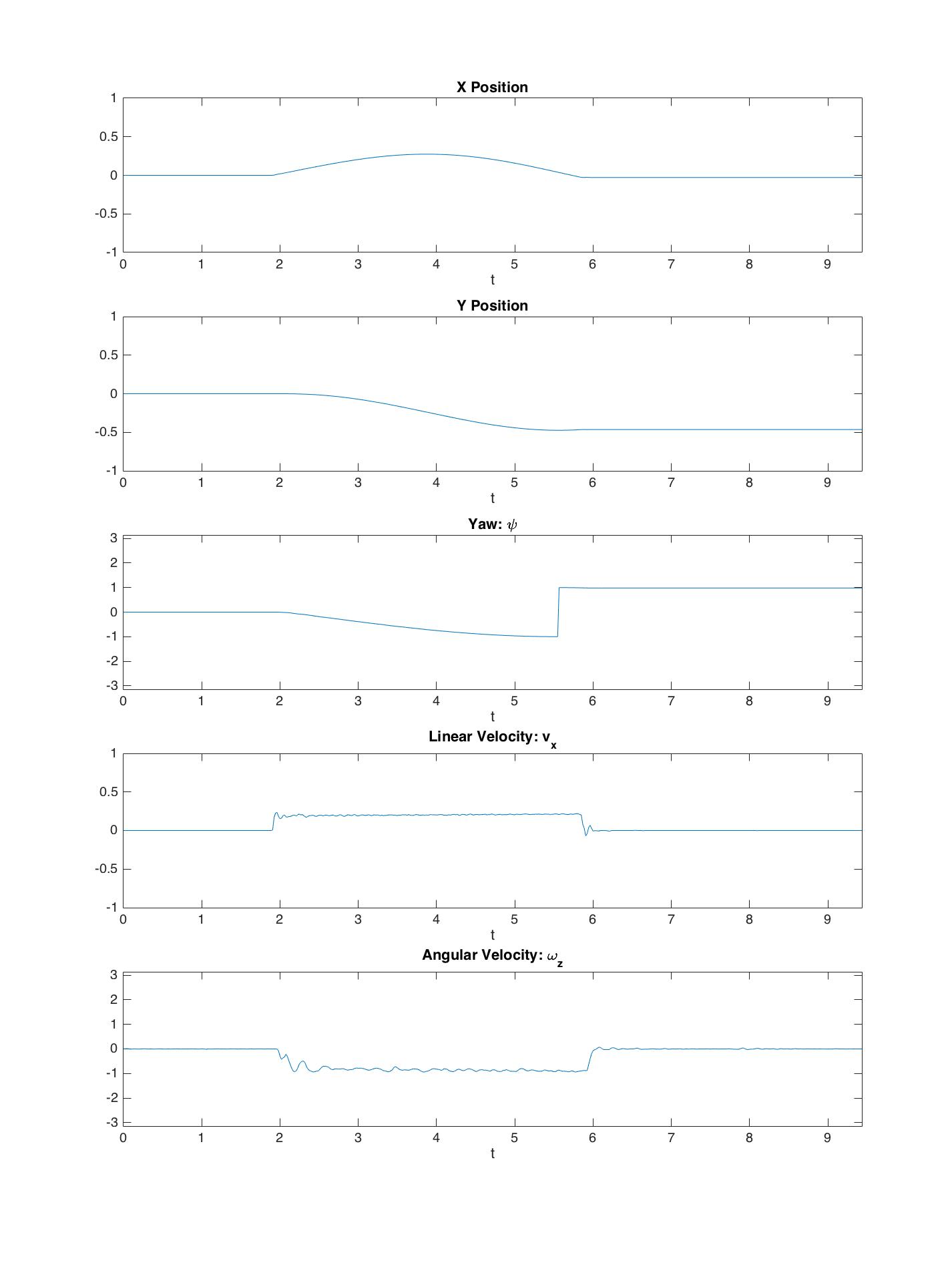
This plot is very representative of the predicted outcome. There is a constant increase in the x position, no increase in the y position, no change in the yaw, a constant linear velocity, and no angular velocity. This is predicted due to the nature of having only a linear velocity published, causing only the x position and linear velocity to change at all, and the x position to be the integral of the linear velocity, demonstrating basic kinematics.

Sequence 2



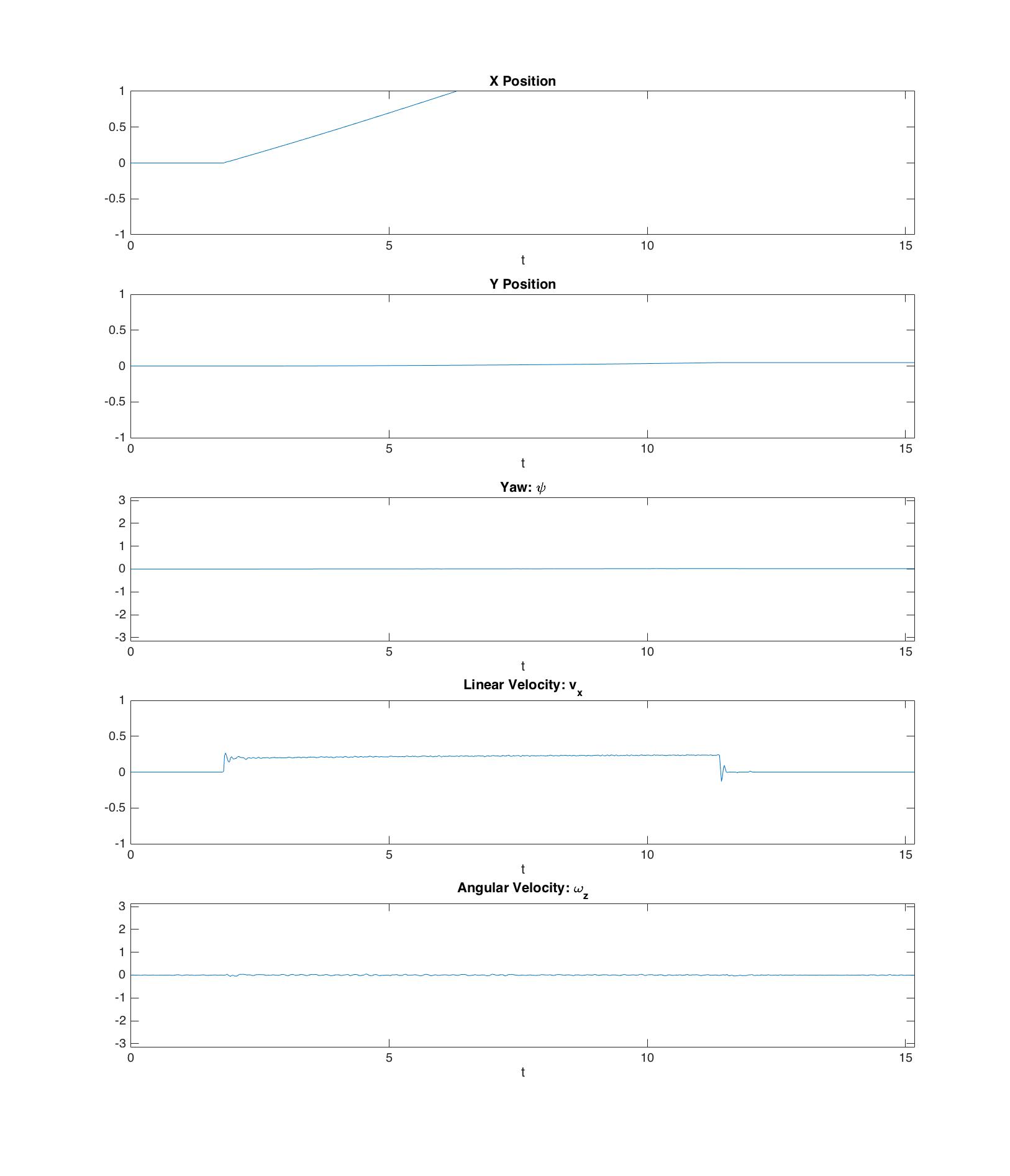
This sequence also matches the predicted results. The movement of the Turtlebot is in a circle, with a constant a linear and angular velocity. This means that the x and y component will have a sinusoidal curve, where x is represented by sin(t) and y is represented by cos(t). The yaw is continuously changing because it is spinning in a circle. The quick change in yaw is a result of the Turtlebot reading solely between pi and –pi, so when it passes a full turn around, it jumps. The linear and angular velocities match the results.

Sequence 3



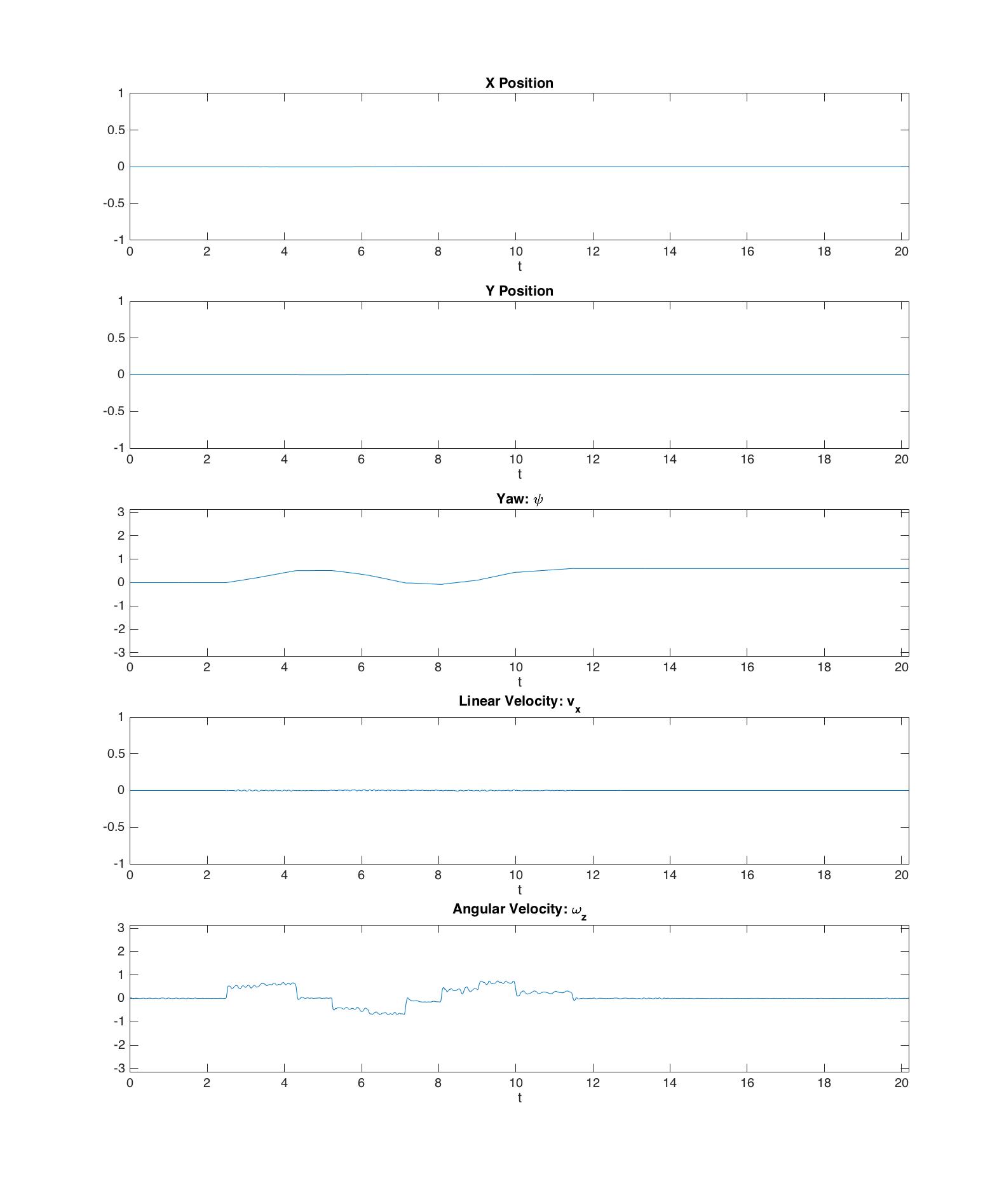
These experimental results also match the predicted results. Due to the negative value of the angular velocity, all corresponding motion was accounted for. The Turtlebot should have equal bot opposite angular components for velocity and yaw. Also, the Turtlebot turns the opposite way then the last sequence, so the y component of position is also equal, but opposite.

Sequence 4



These results do not match the predicted results. This is due to an error in the coding. The sinusoidal component was not added to the linear velocity, so therefore there was no sinusoidal data in the experimental linear velocity and the x position. There was no angular velocity published, so the y, yaw, and angular velocity components did not change, which did match my results.

Sequence 5



These results did not match the predicted results. This is due to an error code. In the sinusoidal component, a multiplicative factor was introduced in the form of integer division. This causes the internal sine argument to step, as opposed to update smoothly. The rest of the predictions match.

Q3)

Topic: mobile\_base/events/bumper

Message type: kobuki\_msgs::BumperEvent

Implementation in code