Laboratory 3

Mobile Robot Perception

CSC 232 – Autonomous Mobile Robots

I affirm that I have not given nor received any unauthorized help on this assignment, and that all work is my own.

Abstract

This lab was an exercise in implementing a sense of sight for Turtlebot. This involved using the Kinect mounted to the Turtlebot to retrieve information about the distance between the Turtlebot and the objects ahead of the Turtlebot. In accordance to the philosophy of having a standalone system for testing, the simulator for the previous lab was edited to have range finder capabilities and to gather data from hypothetical virtual objects. Along with these implementations come the need to further tailor the CMake files and, as a new task, create a custom launch file for the sensor’s messaging channels.

Code Analysis

The implementation of this lab required some critical evaluation. First, the simulator was edited to be capable of emulating a robot with range finder faced with objects around it. To do this, three steps were required.

The first was finding a simple way of characterizing certain kinds of objects which for this lab are specifically the wall and the cone. This was done by giving the wall an angle and a center point, and the cone a radius and a center point. The key to these descriptions were to use the least amount of information while still encapsulating all the characteristics of the object. For these two, a wall can be easily emulated by a line and a cone by a circle, so each object’s characteristics respect those needs.

The next was a way of characterizing a range finder. A range finder has a number of principle parts: the sweep angle and how many data points are acquired. This can then be broken down into holding the minimum and maximum viewing angles, and both an angular increment and total amount of samples. With this, sweeps can be accurately made to sample from a virtual world.

Lastly, a system for evaluating distances needed to be found. The approach that was used in this lab was scanning a view of each object separately, and then combining them to form a total view. The method of finding the distances between the wall/cone and the bot was done by evaluating geometric properties of both kinds of objects and exploiting them for simple calculations. The distance to a point on the wall could be modeled by two lines intersecting and finding the distance to that intersection. Solving that system of equations is very easy for the computer. The cone was rather tricky. Using the property that a line tangent to a circle is the maximum distance away that could every be measured, a maximum angle for intersection could easily be found with an inverse trig function. Then, any angle larger than the one found would be known to miss, which any angle in bound would be known to hit. Then, if an angle was within bounds, the law of cosines could be used to evaluate for the distance to the cone. It is important to note that the distance from the bot to the center of the cone is known as both positions are given, allowing for that reference length to be used (rather critically).

Implementing the measurement simulator was then a task of efficiently calculating the mathematics. Communication of the measurements was just an exercise in instantiating publishers correctly and storing the data for when it was needed. Specifics can be seen in the code

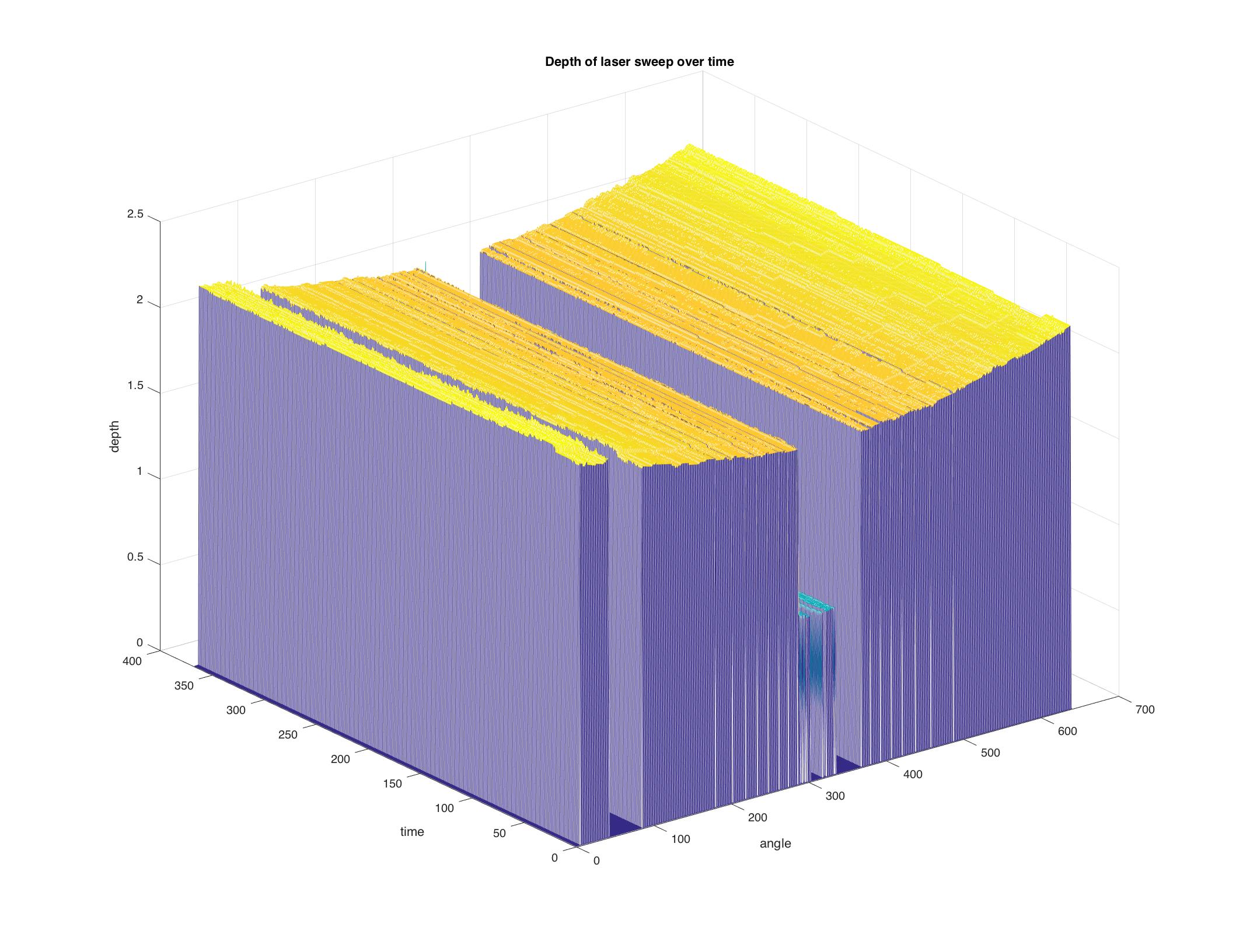
The GUI was another large portion of this lab. Most the framework was given, but the implementation of both the callback fucntions and drawing the data was left to the student. Data was stored in vectors in the gui.h file. The odometry of the bot was stored in a 2-D double vector where each row would hold a snapshot of the full odometry message. The saving was done in the callback function for Odometry. The range finder data was stored similarly. A 2-D vector was used to store the distances found from the ranges part of /scan. Then, a set of 1-D vectors were used to store the other parts of the /scan message: angle\_min, angle\_max, and angle\_increment. This was all handled in the callback functions.

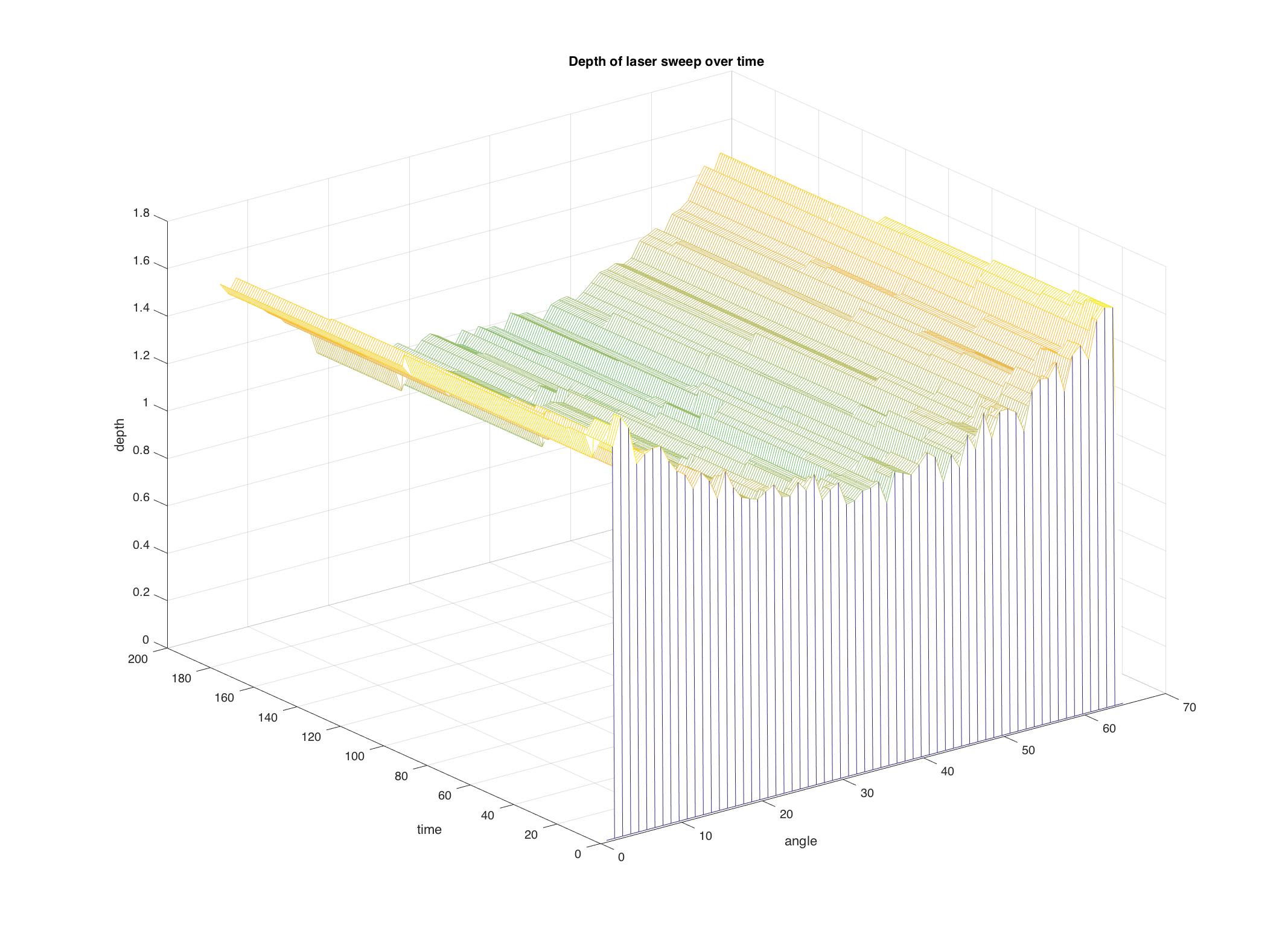
The drawing of the ranges was then done in the paintGL function, which is currently written to draw all scans. This is fine for now, but as more scans are taken, especially if the robot is moving, a version where many less are drawn at a time for speed purposes will be created. Using the basic structure for drawing lines and the geometry involved with the stored information, a picture is made showing a fan of measurements coming from the bot.

The last sections of code to be written are those that have to do with the backend of the build model. A new launch file was written to start up the range finder message channels and the CMake file was edited to include the GUI files and corresponding OpenGL packages that were needed.

Data/ Questions

Physical turtlebot data graphed in time and beam angle



Virtual bot data graphed in time and beam angle

The raw data is available in .csv files attached.

The graphs are both very similar. Each show the basic contour of the wall getting closer to the bot as the beam angle is more centered. As both the virtual bot and the physical turtlebot are not moving, the general trend of the graph does not change over time.

The three main differences can be seen in the amount of samples per scan, the actual objects detected, and the noise in the measurements. The physical scanner has 10 times more resolution than the virtual bot. The virtual bot was also not able to detect the cone, a bug that is still being addressed.

The noise factor in the virtual bot is much higher than that of the physical readings. Through time, the noise is contained and draws a mostly consistent curve, but is much more drastic than that of the physical measurements.

The code that goes into simulating the robot is also easily the most complicated and lengthy part of the implementation. Comparatively, the simulator has as many lines as all the other functional files combined. Furthermore, the simulator is written in one file, making the code very trimmed, compared to that of the GUI files that have many lines just to reference classes in other files. Basically, the densest part of the code is still the longest.

Conclusion

The readings from the Kinect were very stable and representative of the environment, while the virtual measurements were not. The GUI is computationally expensive and needs to be tweaked for future use. The simulator is easily the toughest part of to implement (and it’s not practically required!).