**Lab 4**

**Abstract**

This lab was an implementation of EKF localization using the information of the gotten from the lasers scans and the internal global map. This includes implementing an EKF, a measurement model for the beams received, and communicating the predicted location to the GUI and an output file.

The measurement model was done using a geometric method that could be applied to any circle found at any orientation. This was done to allow for the most flexibility in the future. It used beams to find a circle and then compare it to what cone it might be looking at and used the best result found across the beams as the read measurement.

The GUI read information from the laser scan message and displays them as a set of projecting lines from the center of the robot. It takes the distances, minimum and maximum angles, and the increment to project lines corresponding to each measurement to accomplish this, setting the point of reference as the orientation of the robot. It also reads information from from the simulator and the localizer for the estimated pose and actual pose, which needed in the laser scans calculations.

The EKF algorithm uses the measured locations of the cones against the predicted locations of the cones and computes where it might be and how sure it is about that. This EKF algorithm uses the model that incorporates all known landmarks.

**Code**

The measurement model that was created used a circle-fitting algorithm and geometry to fit a cone to where the beams ended. The circle-fitting was done by first calculating the geometric location of the endpoints of the beam to be used, then using the intersection of perpendicular bisectors of two sides of a triangle that those points created. It’s a geometric construction of fitting a circle to three points that was found online.

This process is repeated throughout all of the beams in the laser scan message, for every cone in the global map. When a circle is found, it is compared against the predicted location of the cone currently being checked against. The closest cone found in the whole scan is saved as a measured cone. There is a margin accounted for a set of scans that produce no good results that would return some “closest” cone found in a completely inaccurate place.

These measurements are done before every run through the EFK to account for changes in location and new laser scan readings.

The GUI reads the laser scans and saves the last laser scan. Then, when it Is running through it’s draw loop, it takes the scans and the information that comes with it, such as the minimum and maximum angles as well as the increment, and draws lines projecting from the center of the robot. This is done by drawing a line between the center of the robot and the end point of the laser scan.

The end point of the laser scan is calculated using basic right-triangle geometry. Using the minimum and maximum angles as well as the increment, it is easy to find the bearing of the laser scan in relation to which laser scan is being looked at using the index being accessed (the distances are stored in a vector). Then, that bearing is added to the orientation of the robot to put it into the robot’s perspective. Then, using cosine and sine, the x and y coordinates are found. Those coordinates are then also translated to be center on the robot. This results in a point which is as far away from the robot as the laser scan, in the direction of the laser scan from the center of the robot. Drawing a line between that point and the robot’s center then becomes an accurate representation of what a laser scan would look like. This is done for every laser scan saved.

The GUI also reads in Pose messages for the estimated pose and the actual pose. Very simply, a circle drawing method was made that takes a center and a radius and draws a circle of a certain resolution around that center and at that radius. By reading in the the two poses, drawing the predicted and estimated robot was easy enough.

Once a set of estimated cones corresponding to predicted cones is found, then the EFK algorithm is run.

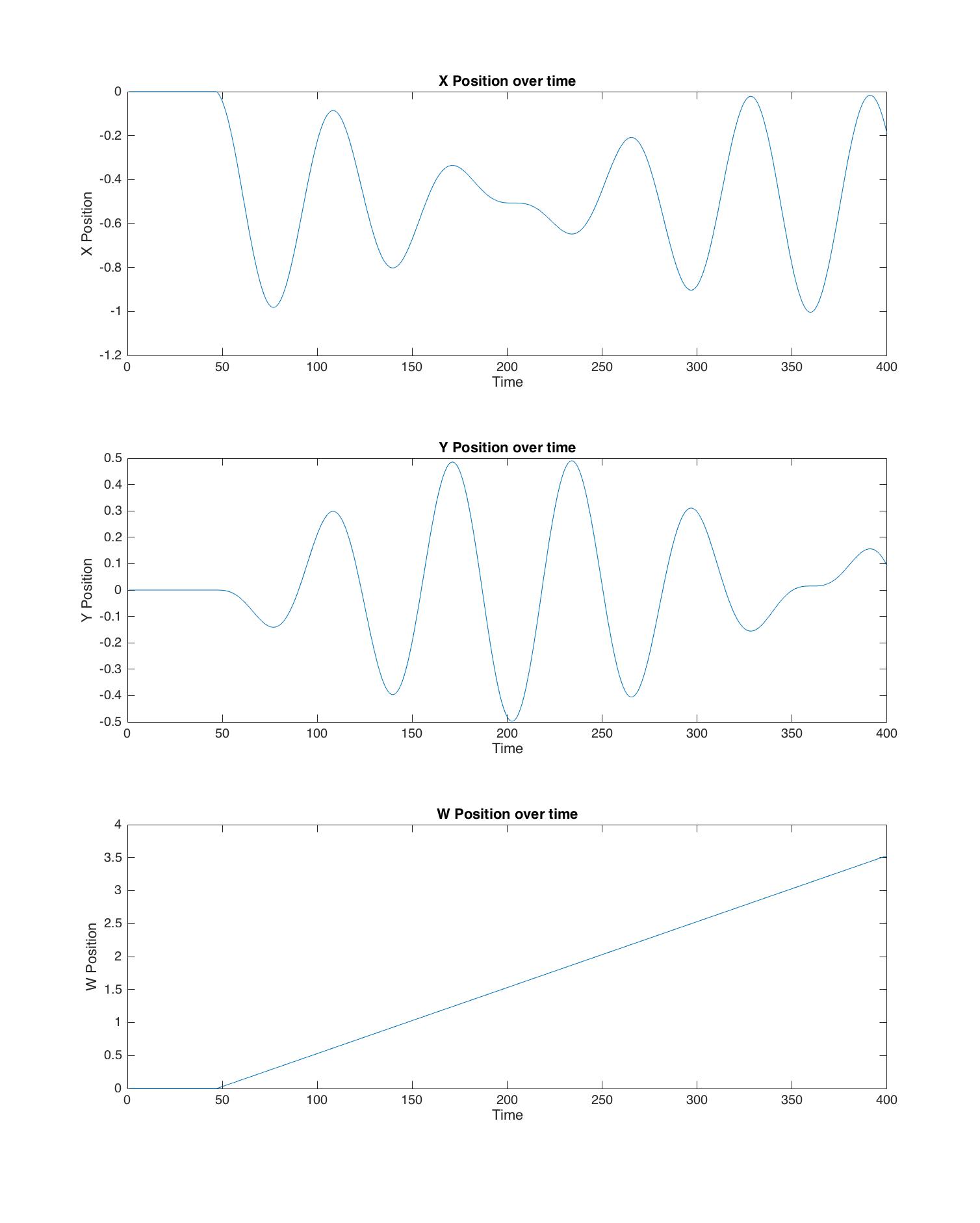
First, the Gt and Vt matrices are set and the projected mu vector is set corresponding to whether or not wt = 0. This only really matters the first couple times when the navigator is not running yet and the default angular velocity is still 0. Then the Mt matrix is set to account for error and the projected sigma matrix is set. Then, it gets to section where comparisons are made for the estimated cone locations against the predicted cone locations.

The first step is to save the new projected mu and sigma is there are no estimated cones found. If there are, then for every measured cone, each is compared against all the predicted cones to find which it most likely corresponds to. In this process, Ht and St are set for every predicted cone. Once the estimated cone has been matched with the predicted cone, the Kalman gain matrix is calculated using the Ht and St that corresponds to that predicted cone, which is then used to adjust the projected mu and sigma vector and matrix.

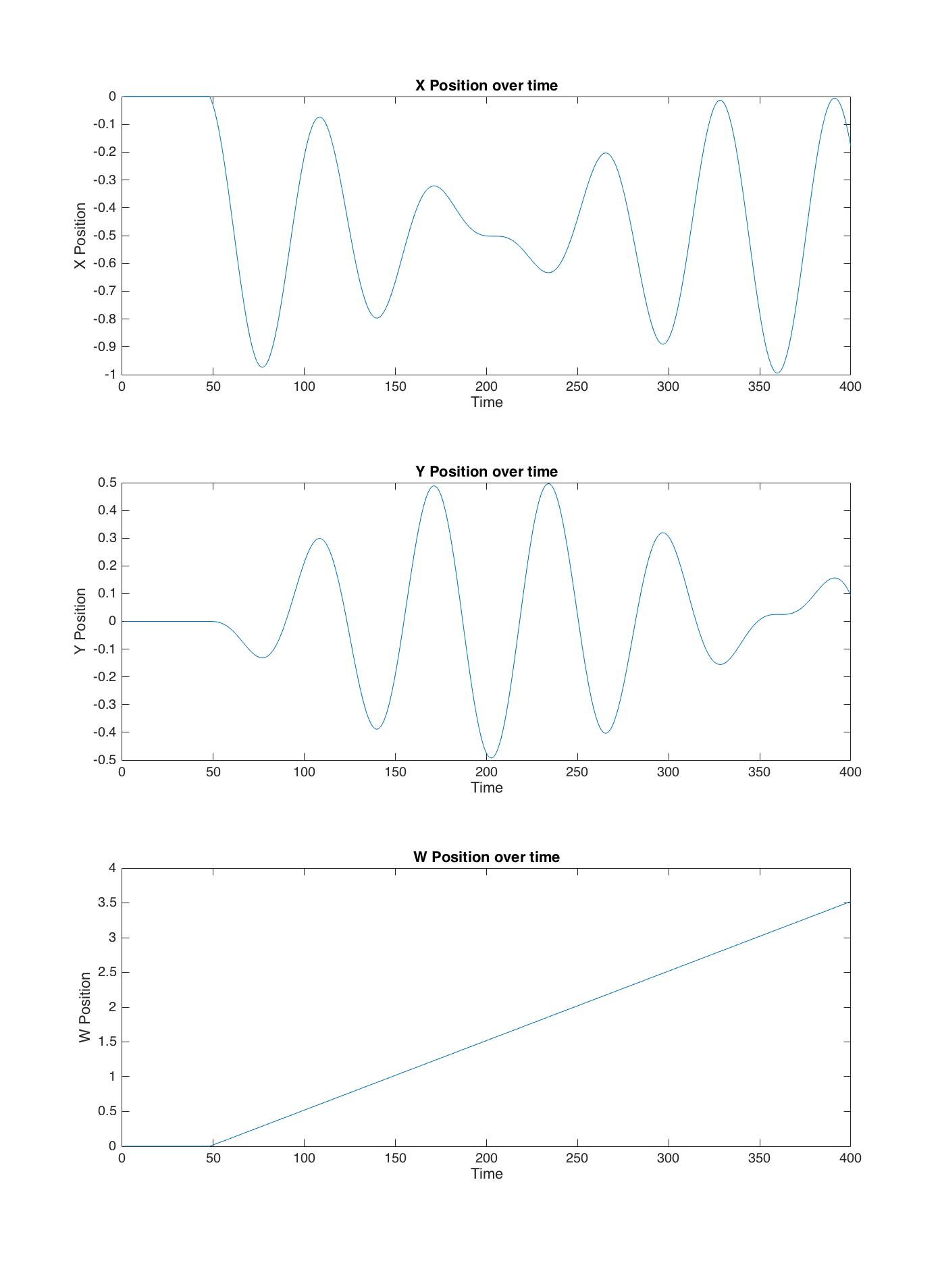
Once that is done for every estimated cone, mu and sigma are set to the current projected mu and sigma. This process repeats every cycle through the open loop.

In preparation for all the calculations used in the EFK algorithm, a Matrix class was created to be able to perform all the operations that were expected. Multiplication, addition, inversion, and transposing were the four main operations that were needed to make the EFK work. I created many methods that work for 2D vectors operating against on other. For addition, first the dimensions are checked, and then each element is added element wise by index. For multiplication by a scaler, every element is multiplied by the scaler. For two matrices being multiplied together, the dimensions are checked (first’s columns match second’s rows) and then the multiplication and addition needed to calculate the element of each row is performed. For inversion, a couple of steps are taken. Firstly, it is checked that the matrix can be inverted by checking the the determinant is not zero. Then, it uses the adjoint matrix multiplied by the determinant to get the inverted matrix. The transpose matrix is straight forward. A new matrix with opposing rows and columns is created and each is copied into the new matrix accordingly.

**Data/ Questions**



The simulated data above matches the expected result. Firstly, there is a linear angular position over time, which shows the angular velocity was constant. This matches the input. The robot starts facing forward originally, so as it moves back and forth, the x position is affected more than the y. Then as the robot turns, the x position is affected less and the y position is affect more. When the robot is facing perpendicular to it’s original position, then the y is most effected. At this point, the robot is beginning to face backwards, which the data matches. As the robot affects its y position the most, it is also affecting the x position the least.



The experimental data closely matches the simulated data. It behaves in exactly the same ways and is also accurate to the simulated data. There are some small difference, such as the simulated data having passed the -1 meter mark on the x position. Otherwise, the data is very tight.