ECE 183DA – Design of Robotic Systems 1

Lab Assignments 3 & 4 – State Estimation and Motion Planning of a Mobile Two-Wheeled Robot

Date: 13th March 2018
Professor: Ankur Mehta
Team Watercress: Haoyang Ye (304456202), Shounak Roy (204482466), Rehan Shah(204406948),

1. ABSTRACT

This lab report presents a ground-up methodology of building a state estimator for a mobile 2 wheeled robot. The state estimator is then utilized for motion planning of the robot in a 2D environment. The mathematical framework that describes the input-output model of this robot plant is laid out in the report. System dynamics including relevant sensor and actuator responses are analyzed and a state estimation algorithm is designed by implementing an Extended Kalman Filter (EKF). A Rapidly Exploring Random Tree (RRT) algorithm is used to determine the trajectory that gets the robot to a desired goal state while avoiding obstacles in the environment. Simulations and physical experimentation are performed to determine the performance (in terms of accuracy and efficiency) of our state estimator and motion controller over time.

2. INTRODUCTION

2.1 Brief Overview

This lab consists of two parts - state estimation and motion planning. The plant in this case is a two wheeled car with a tail that drags along. The wheels rotate independently of each other on continuous rotation servos. The angular velocity of the wheels are controlled by a PWM signal from an ESP8266 microcontroller. A web interface is provided so that a user can control the inputs given to the robot. Two LIDAR sensors are used to detect objects. One of them points forward and the other one points to the right with respect to the robot. It is intended that the robot will be placed in a box with the walls of the box being the boundaries of its range of motion. There may also be obstacles in the box which the robot should avoid. The final goal is to have the robot reach a desired goal state without hitting any of the obstacles.

A high level block diagram of our system is shown below -

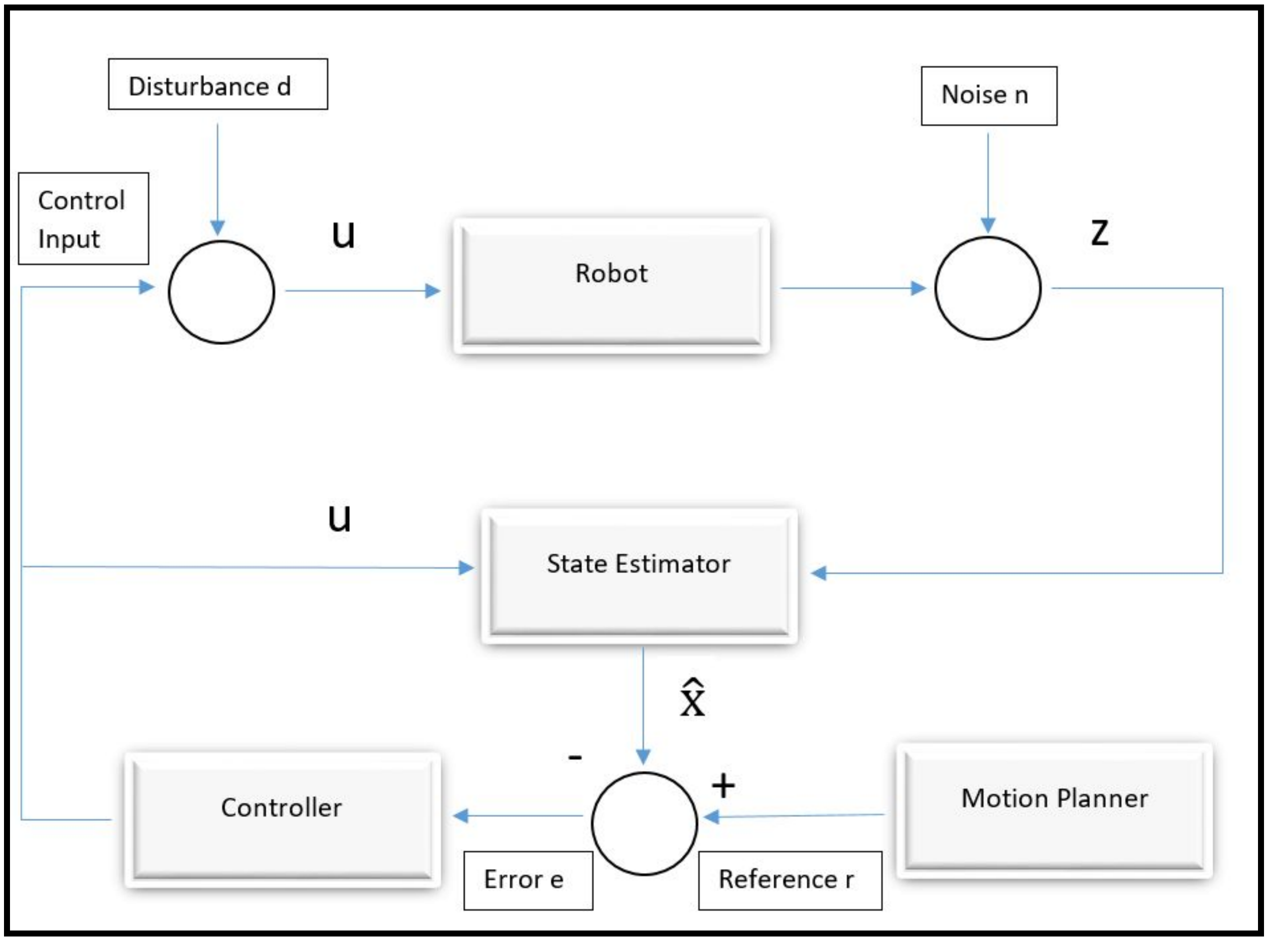
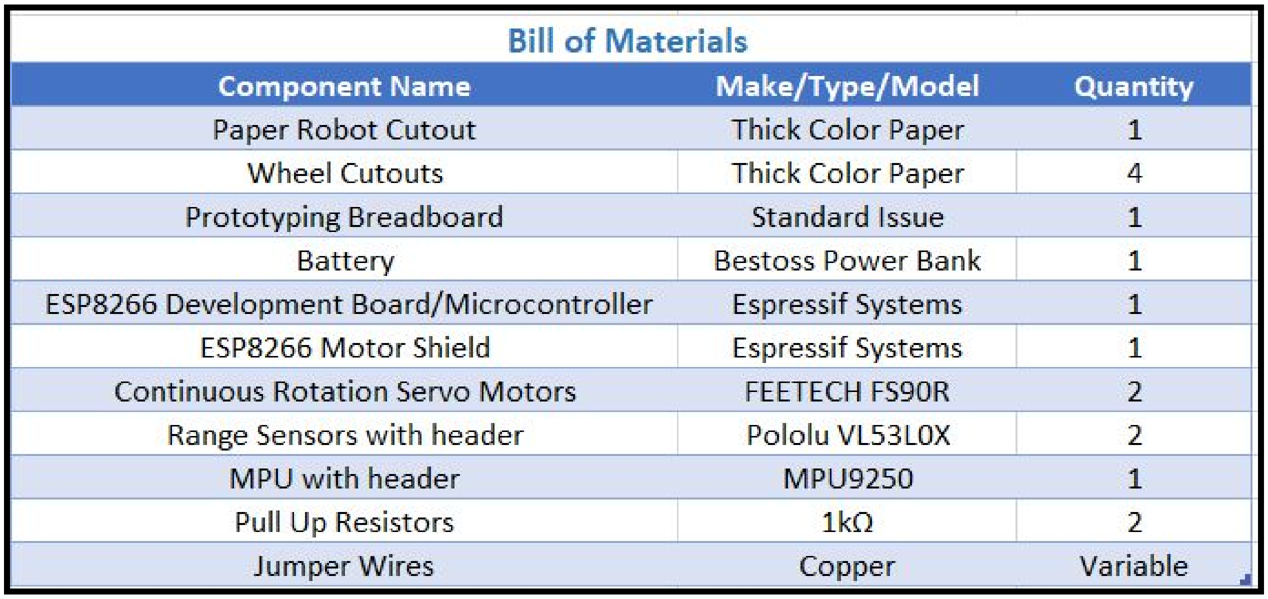


Figure 2.1: Block Diagram of the Comprehensive Robotic System

2.2 Hardware Requirements and Setup Information

To get started on the project, the following are the hardware requirements -



Bill of Materials

Component Name	Make/Type/Model	Quantity
Paper Robot Cutout	Thick Color Paper	1
Wheel Cutouts	Thick Color Paper	4
Prototyping Breadboard	Standard Issue	1
Battery	Bestoss Power Bank	1
ESP8266 Development Board/Microcontroller	Espressif Systems	1
ESP8266 Motor Shield	Espressif Systems	1
Continuous Rotation Servo Motors	FEETECH FS90R	2
Range Sensors with header	Pololu VL53L0X	2
MPU with header	MPU9250	1
Pull Up Resistors	1kΩ	2
Jumper Wires	Copper	Variable

Figure 2.2: Hardware Requirements for the Project

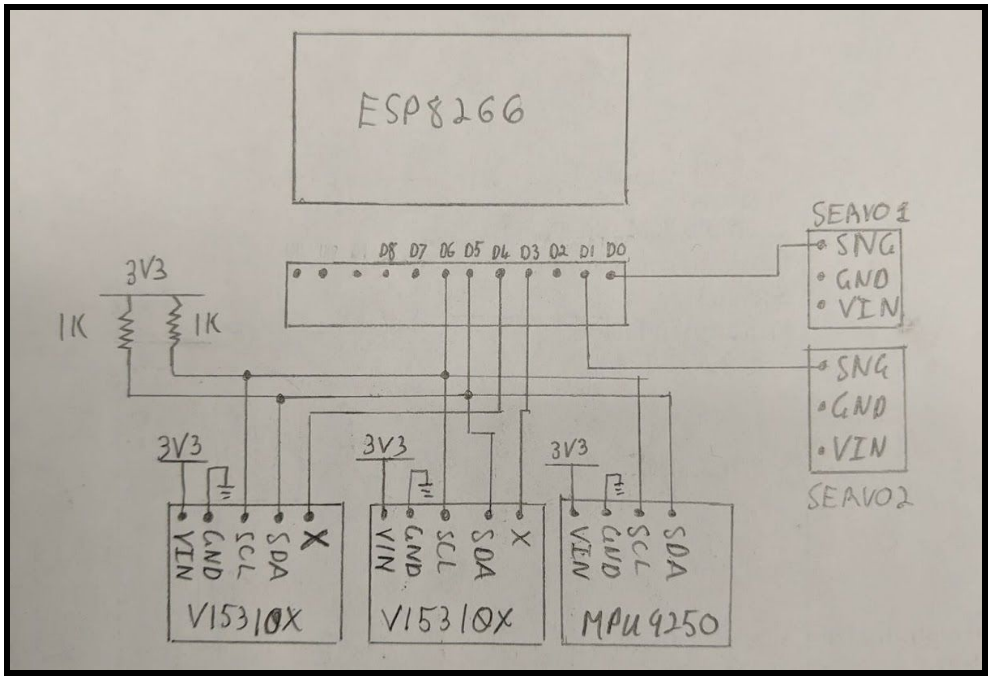


Figure 2.3: Wiring Diagram Showing the Interconnections between the Servos, Range Sensors, MPU Motion Tracking Device, and the ESP8266

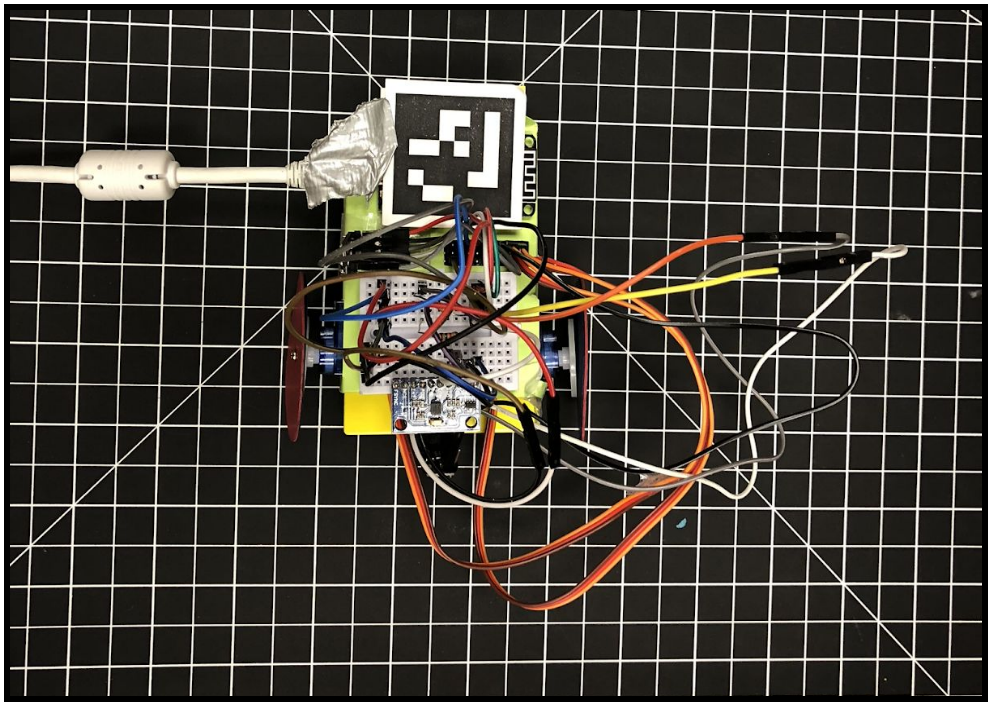


Figure 2.4: Paperbot With Sensor Modules Connected on the Grid Layout of a 21''by 26'' Box

The construction of the paperbot is a fairly straightforward sequential process and the steps are outlined in the git repository whose link can be found in the appendix. The wiring of the different sensors to the microcontroller is outlined in Figure 2.2. For software setup details and sample code to test the motion of the bot and the functioning of the sensors, refer to the github link. Additional details regarding sensor calibration and communication can also be found in the lab startup guide on the github repository.

3. STATE ESTIMATION

3.1 System Description

The state of the 2-wheeled mobile robot is essentially its 3 DOF position in the designated 2D space. We also include constant velocity terms as derivatives of the positional state. As a result our robot state has 6 variables - The 'x' and 'y' positional coordinates, the angular bearing 'θ'

and their respective derivatives. We have 2 input variables from the 2 control inputs which are the independent PWM values sent to the wheels.

The sensor outputs from the front and the right LIDARs consist of straight line distances to the walls in the front and the right of the robot respectively. The inertial measurement unit (IMU) returns an absolute bearing that indicates the angle of the robot with respect to the magnetic north. It further provides us with the angular rate measurement however we don't factor this into our setup. Thus, our system effectively has 3 output values.

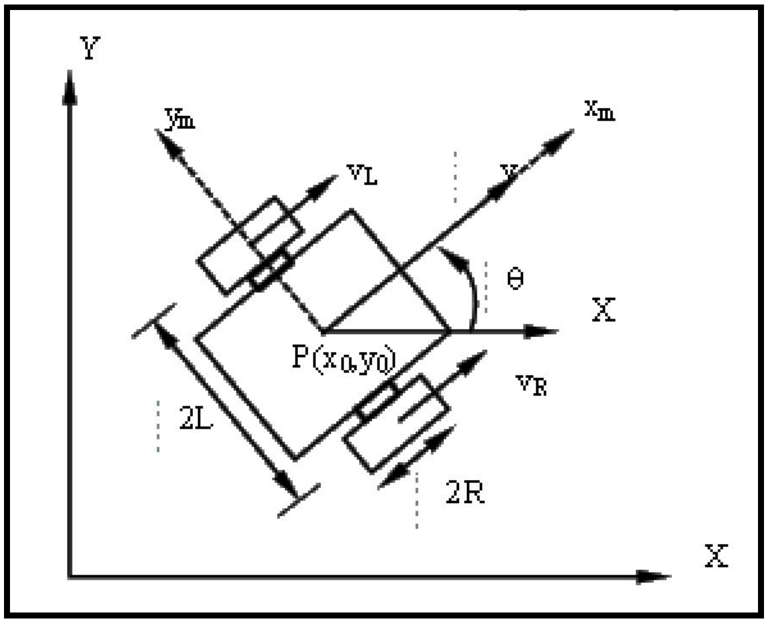


Figure 3.1: Labelled Schematic of the Two-Wheeled Mobile Robot Used to Model the System

3.2 Algorithm for State Estimation

The algorithm we utilize for state estimation of the paperbot is **Kalman Filtering** which is essentially used in any situation where we have uncertain information about some dynamic system and we can make an educated guess about what the system is going to do next. Because our system is fundamentally non-linear, we use the **Extended Kalman Filter** which is the non-linear version of the Kalman filter that linerarises about the current mean and covariance. We use this algorithm because it is considered the gold standard in the theory of non-linear state estimation.

3.3 Mathematical Setup

In the Extended Kalman filter (EKF), the state transition and observation models are not linear function of the state but instead are differentiable functions. The complete set of equations governing the extended Kalman Filter are shown below -

Model:

$$\boldsymbol{x_k = Ax_{k-1} + Bu_k}$$

$$\boldsymbol{z_k = Cx_k + v_k}$$

Predict:

$$\boldsymbol{\hat{x}_k = A\hat{x}_{k-1} + Bu_k}$$

$$\boldsymbol{P_k = AP_{k-1}A^T}$$

Update:

$$\boldsymbol{G_k = P_kC^T(CP_kC^T + R)^{-1}}$$

$$\boldsymbol{\hat{x}_k \leftarrow \hat{x}_k + G_k(z_k - C\hat{x}_k)}$$
$$\boldsymbol{P_k \leftarrow (I - G_kCP_k)}$$

In the above formulation - $\boldsymbol{x_k}$ is the current state of the system, $\boldsymbol{x_{k-1}}$ is its previous state, $\boldsymbol{A}$ is the state transition matrix of the previous state, $\boldsymbol{B}$ is the state transition matrix of the control input vector, $\boldsymbol{C}$ is the observation matrix based on the current state, $\boldsymbol{v_k}$ is Gaussian process noise, $\boldsymbol{P_k}$ is the predicted covariance estimate and $\boldsymbol{G_k}$ is the Optimal Kalman Gain used to update the current state estimate.

The State Vector in our case:

$$\hat{x}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{\theta}_k \end{bmatrix}$$

The State Transition Matrix A:

$$A = \begin{bmatrix} 1 & 0 & 0 & t & 0 & 0 \\ 0 & 1 & 0 & 0 & t & 0 \\ 0 & 0 & 1 & 0 & 0 & t \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The State Transition Matrix B:

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{2} cos\theta & \frac{1}{2} cos\theta \\ \frac{1}{2} sin\theta & \frac{1}{2} sin\theta \\ -1/2L & -1/2L \end{bmatrix}$$

The Measurement Vector:

$$z_k = \begin{bmatrix} d_1 \\ d_2 \\ \theta \end{bmatrix}$$

The measurement function is calculated analytically using geometric methods. The measurement function creates an idealized model of the robot. Given the x and y coordinates of the robot as well its orientation, the function returns the expected sensor measurements for the front and the right sensor. For each sensor we first construct a line passing through the coordinate of the sensor with a slope given by the orientation of the vehicle. Then we find the point of intersection between the line and the edge of the box. However, in some cases this may be the top and bottom edges and in other cases the left and right edges. In order to determine which edge it is, we actually take two points - one is the intersection of the line with the line y_d (Edge dimension) and the other is the intersection with the line x_d (Edge Dimension). Depending on the orientation of the robot this may also be the x axis or the y axis itself). Having found these points we compute the distance between the position of the robot and the two points and then take the minimum of the two. This gives us one sensor measurement and an identical process is repeated for the other sensor.

A graphical representation of our calculation is shown on the next page in Figure 3.2 -

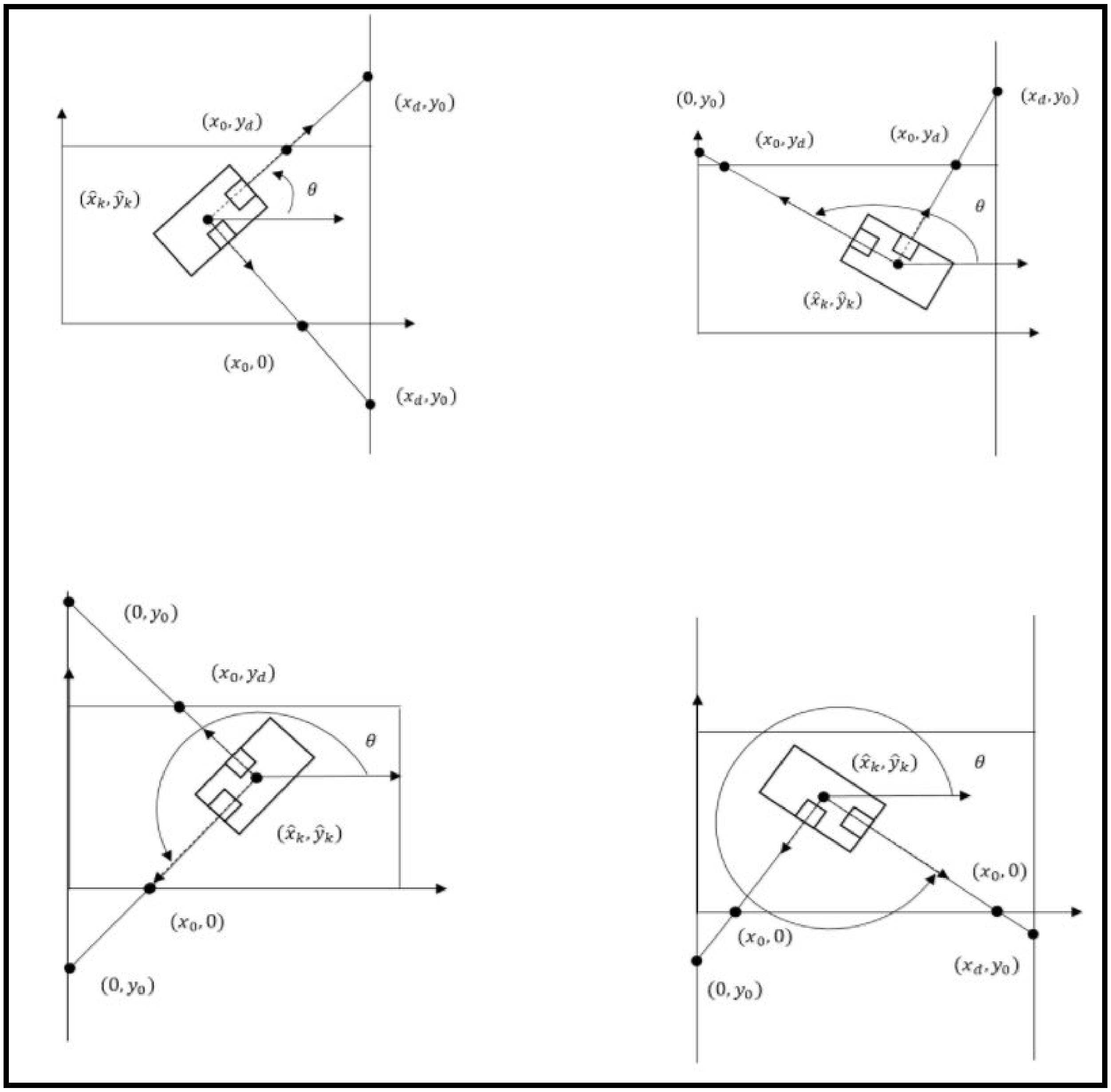


Figure 3.2: Different Geometrical Parameters Based on Bot Orientation

The calculations to determine the measurement values are shown below -

$$x_0 = \frac{y_d - \hat{y}_k}{\tan(\theta)} + \hat{x}_k, \qquad r_1 = \sqrt{(y_d - \hat{y}_k)^2 + (x_0 - \hat{x}_k)^2}$$

$$y_0 = \tan(\theta) * (x_d - \hat{x}_k) + \hat{y}_k, \qquad r_2 = \sqrt{(y_0 - \hat{y}_k)^2 + (x_d - \hat{x}_k)^2}$$

$$x_0 = \frac{-\hat{y}_k}{\tan(\theta)} + \hat{x}_k, \qquad r_3 = \sqrt{(\hat{y}_k)^2 + (x_0 - \hat{x}_k)^2}$$

$$y_0 = \tan(\theta) * (-\hat{x}_k) + \hat{y}_k, \qquad r_4 = \sqrt{(y_0 - \hat{y}_k)^2 + (\hat{x}_k)^2}$$

$$for\ \theta \in (0,90), d_1 = \min(r_1, r_2)\,, d_2 = \min(r_3, r_2)$$

$$for\ \theta \in (90,180), d_1 = \min(r_1, r_4)\,, d_2 = \min(r_1, r_2)$$

$$for\ \theta \in (180,270), d_1 = \min(r_4, r_3)\,, d_2 = \min(r_1, r_4)$$

$$for\ \theta \in (0,90), d_1 = \min(r_2, r_3)\,, d_2 = \min(r_3, r_4)$$

$$for\ \theta \in (0,360, \ldots), d_1 = L - \hat{x}_k, d_2 = \hat{y}_k$$

$$for\ \theta \in (0,90,450, \ldots), d_1 = W - \hat{y}_k, d_2 = L - \hat{x}_k$$

$$for\ \theta \in (0,180,540, \ldots), d_1 = \hat{x}_k, d_2 = W - \hat{y}_k$$

$$for\ \theta \in (0,270,630, \ldots), d_1 = \hat{y}_k, d_2 = \hat{x}_k$$

3.4 MATLAB Implementation

The EKF function in MATLAB is utilized to perform state estimation on our robot. It takes three input arguments - a state transition function, a measurement function and an initial state. The state transition function outputs the current state of the robot given its previous state and the control input. In order to do this we create the matrices highlighted in section 3.3. The function then simply multiplies the matrices to give the current state. The EKF uses the ‘state transition’ function and the ‘measurement’ function to determine the state of the robot. The EKF has a ‘correct’ function which updates the filter based on the sensor measurements and the expected sensor values from the measurement function. It also has a ‘predict’ function which predicts the location based on the previous inputs and the current state. The output of this function is the estimated state returned as a vector xhat of size M where M is the number of states of the robot system - in our case 6. The other output is the state estimation error covariance (the P matrix) which is returned as an M x M matrix where M is again 6 in our case.

3.5 Ground Truth Localization

For our lab, Aruco tags are used to localize our robot within the rectangular box. This localization tool records the ground truth position and orientation of the robot in real time. Aruco tags are essentially n x n binary grids of black and white blocks that are uniquely patterned for each tag thus making them easily identifiable. A camera is positioned at (x,y)=(0,0) with respect to the tag so that it can reproduce ground truth measurements. An Aruco tag is attached to our robot and to obstacles in the environment. An executable Aruco test script is then run and a live log of the robot position is generated. This data is compared with our state estimation model to determine the accuracy of our method.

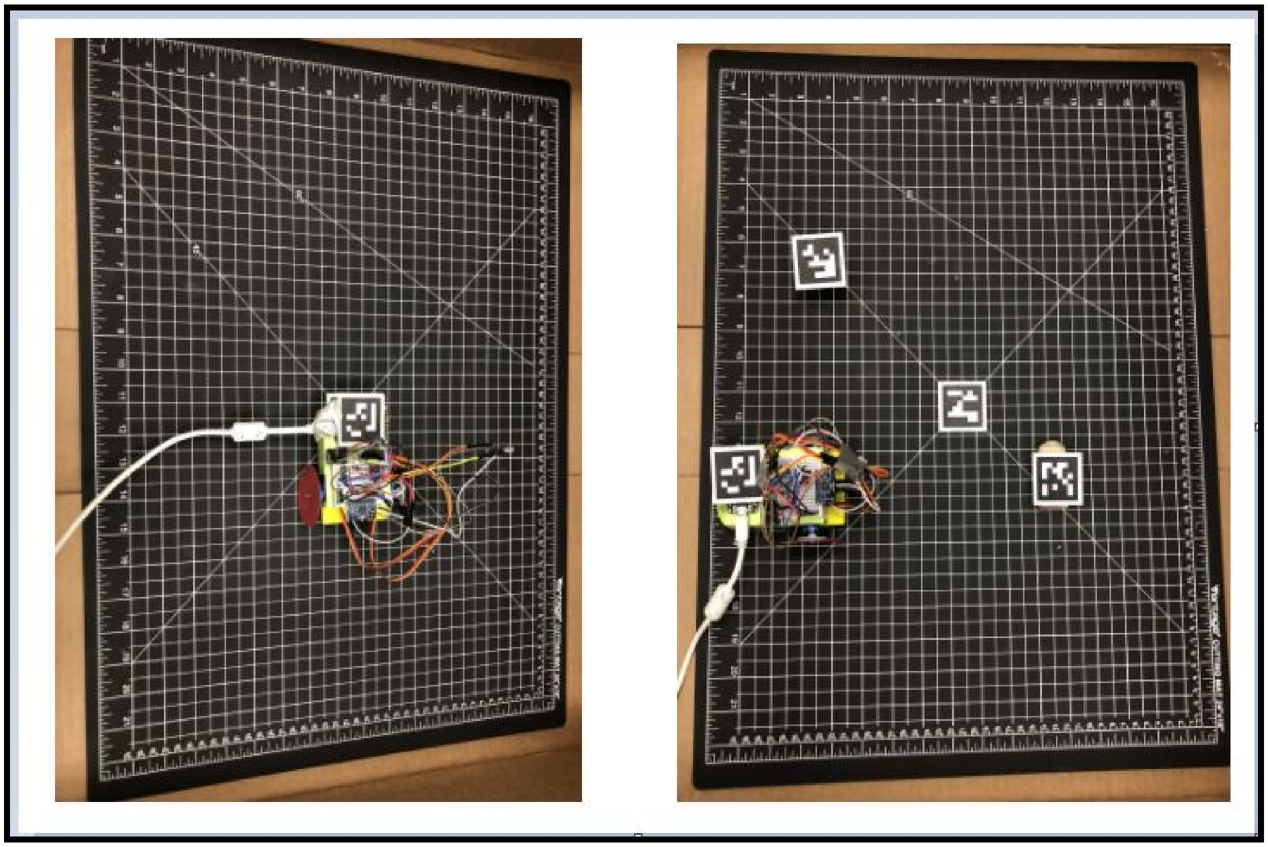


Figure 3.3: (L-R)The Paperbot with an Attached Acuro Tag; The Tagged Paperbot in an Environment with Acuro Tagged Obstacles

3.6 State Estimation Results

The two figures 3.4 and 3.5 below, show the trajectory of the robot during ground truth localization. Figure 3.4 shows the trajectory of the robot moving in open ground or without obstacles. We moved the robot forward, backward, and rightward. After comparing the results with the estimations from the state estimator, we gained mostly accurate and occasionally inaccurate results for our state estimator. Most of the inaccurate estimations occurred when the robot was going in the forward direction. We believe that the main reason for this is that our robot's paper structure was damaged, causing the robot to have a rightward bias when it was supposed to move straight forward. Other possible reasons include sensor noise and the fact that the robot was connected to a laptop through a USB cable.

Figure 3.5 shows the trajectory when the robot is trying to go past an obstacle. The robot first tried going to the right and then tried going to the left multiple times. Our estimator had a higher accuracy than the previous run. We believe that the reason was that the robot was moving in very small increments in order to go around the obstacle and such movements reduced the effect of the structural damage. However, we noticed that the wheels were a little deformed and made turning harder, which eventually impacted the accuracy of estimations.

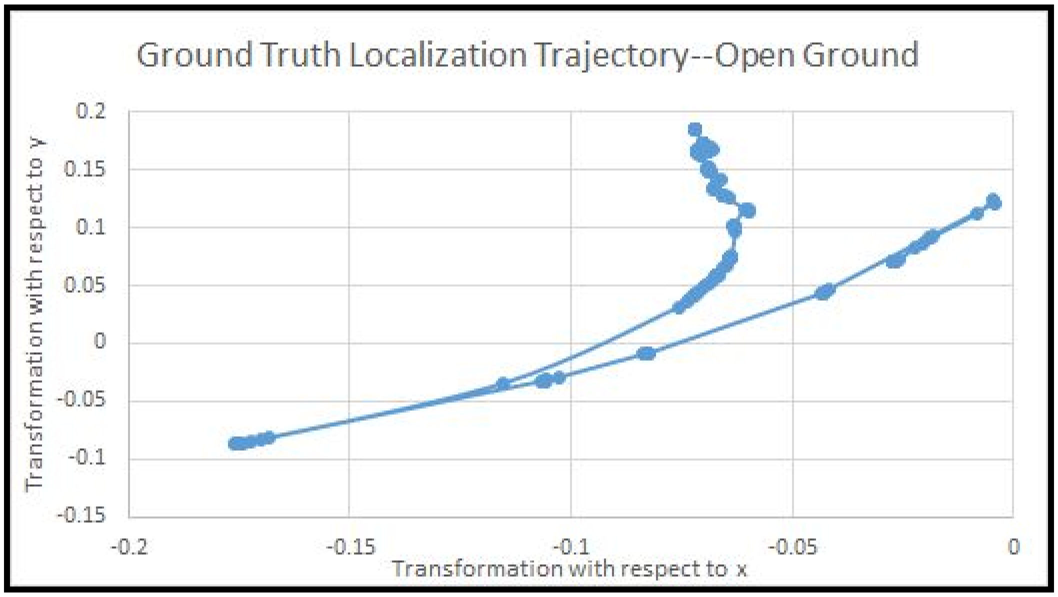


Figure 3.4: Ground Truth Localization Trajectory in Open Space

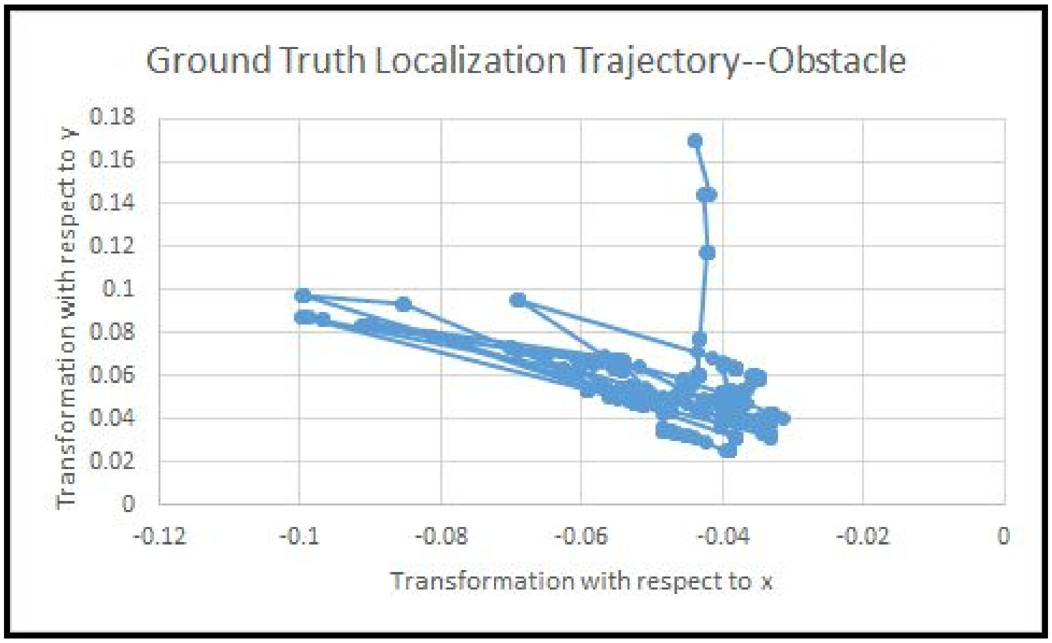
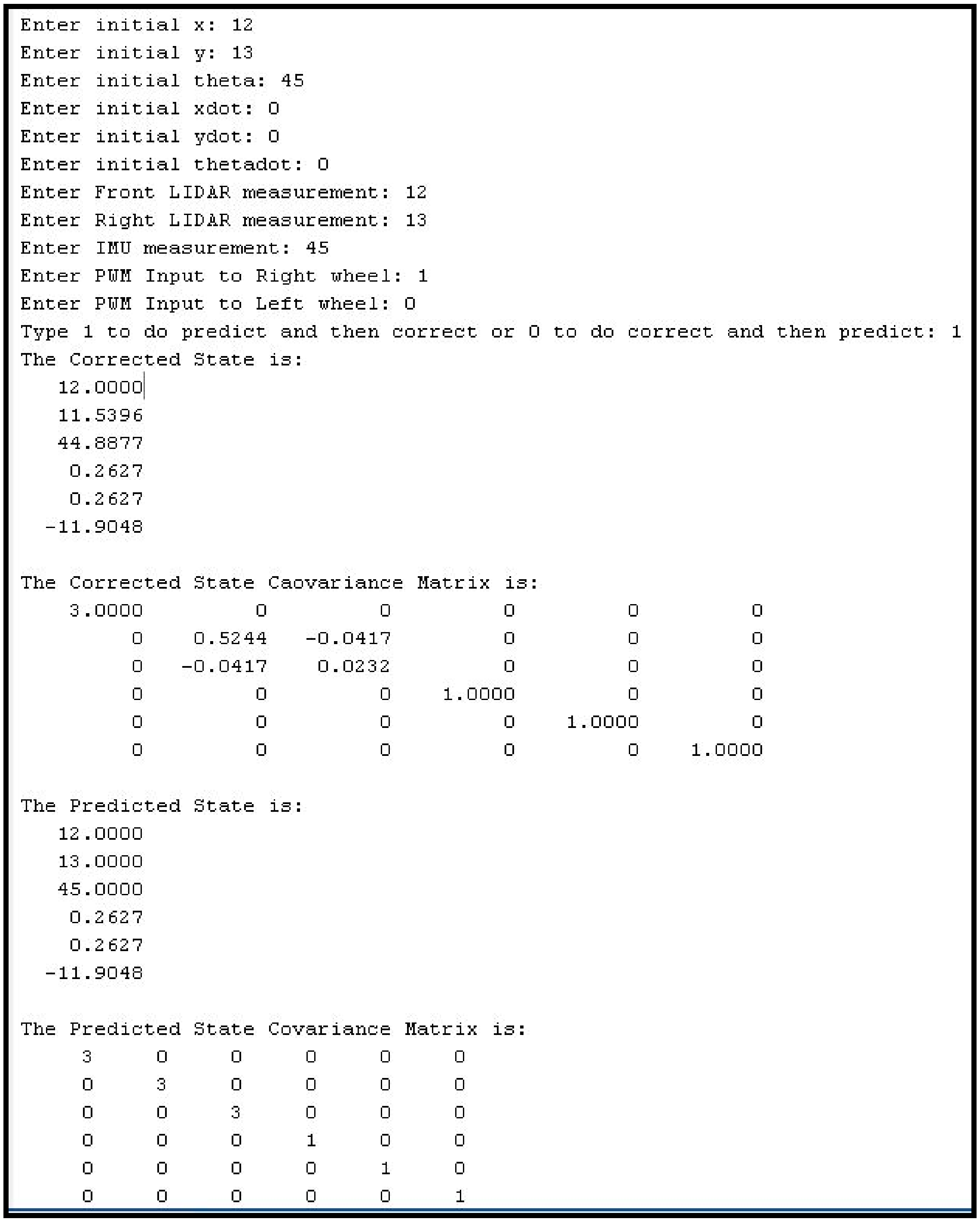


Figure 3.5: Ground Truth Localization Trajectory Involving Obstacle Avoidance

Figure 3.6 below provides a comprehensive screenshot of the result of the Extended Kalman Filter test script that was run on MATLAB. This particular screengrab is for the robot situated at (x,y) = (12,13) on the 26 x 21 rectangular grid. Its bearing is 45 degrees with respect to magnetic north. The robot is static and the bearing is constant so the other 3 state variables are set to 0. A sample measurement generated by the 2 LIDARs and the IMU unit on our bot are entered as sensor inputs. The control inputs are the two sample PWM values sent to the wheels. The corrected state vector and the corrected state covariance matrix are displayed as outputs. Similarly, the predicted state vector and the predicted state covariance matrix are also displayed as outputs.



```
Enter initial x: 12
Enter initial y: 13
Enter initial theta: 45
Enter initial xdot: 0
Enter initial ydot: 0
Enter initial thetadot: 0
Enter Front LIDAR measurement: 12
Enter Right LIDAR measurement: 13
Enter IMU measurement: 45
Enter PWM Input to Right wheel: 1
Enter PWM Input to Left wheel: 0
Type 1 to do predict and then correct or 0 to do correct and then predict: 1
The Corrected State is:
   12.0000
   11.5396
   44.8877
    0.2627
    0.2627
  -11.9048

The Corrected State Caovariance Matrix is:
    3.0000         0         0         0         0         0
         0    0.5244   -0.0417         0         0         0
         0   -0.0417    0.0232         0         0         0
         0         0         0    1.0000         0         0
         0         0         0         0    1.0000         0
         0         0         0         0         0    1.0000

The Predicted State is:
   12.0000
   13.0000
   45.0000
    0.2627
    0.2627
  -11.9048

The Predicted State Covariance Matrix is:
     3     0     0     0     0     0
     0     3     0     0     0     0
     0     0     3     0     0     0
     0     0     0     1     0     0
     0     0     0     0     1     0
     0     0     0     0     0     1
```

Figure 3.6: MATLAB EKF Function Output Based on Test Inputs

4. MOTION PLANNING

4.1 Algorithm for Trajectory Planning

We utilize the Rapidly Exploring Random Tree (RRT) algorithm (developed by Steven M. Lavalle and James J. Kuffner Jr.) for trajectory planning. It is designed to search multi-dimensional spaces with high efficiency by building a space-filling tree. In general, the tree is constructed incrementally from samples drawn randomly from the search space. It is inherently biased to grow towards large unsearched areas of the problem space. The pseudocode for the algorithm is described below -

For a general configuration space ***C*** -

```
Algorithm BuildRRT
        Input: (Initial configuration $q_{init}$, number of vertices in RRT $K$, Incremental Distance $\Delta q$)
        Output: RRT Graph $G$
        $G$.init($q_{init}$)
        for k = 1 to $K$
        $q_{rand}$ ← RAND_CONF()
        $q_{near}$ ← NEAREST_VERTEX( $q_{rand}$, $G$)
        $q_{new}$ ← NEW_CONF( $q_{near}$, $q_{rand}$, $\Delta q$)
        $G$.add vertex($q_{new}$)
        $G$.add_edge($q_{near}$, $q_{new}$)
        return $G$
```

In this algorithm, RAND_CONF grabs a random configuration q_{rand} in C. NEAREST_VERTEX is a function that runs through all vertices v in graph G, calculates the distance between q_{rand} and v using some measurement function thereby returning the nearest vertex. NEW_CONF selects a new configuration q_{new} by moving an incremental distance Δq from q_{near} in the direction of q_{rand}.

4.2 RRT Implementation Details

Motion planning aims to determine a trajectory from an initial state to a specified goal state, avoiding any obstacles on the way. Our robot has two input controls for a 3 DOF State Space making it a non-holonomic system. For this lab we assume that we know the location of the obstacles. Then we use the RRT algorithm to compute a trajectory that takes us to the desired goal state. The input to the RRT function is a picture where the obstacles are marked in black and the allowable space where the robot can move is white. The user is then prompted to click on the picture to select an initial state and then click again to select a goal state. The RRT algorithm then computes a trajectory between the two points. This is done by selecting random points and trying to navigate from these points to the closest point that has already been explored. If we hit an obstacle on the way then that is recorded otherwise the route taken is added to a tree which has all the points the robot can get to. Doing this repeatedly eventually gets us to the goal state or tells us that there is no path to the goal state. The maximum number of vertices in the tree can be set which tells the program how long we would like to find the goal state for.

4.3 Motion Planning Results

Figure 4.1 below shows the local environment within which the robot has to navigate. The rectangular walls act as the boundary and the rectangular blocks act as obstacles both of which the algorithm is intended to avoid when building a trajectory from an initial state in the rectangular box (x_i, y_i, θ_i) to a final state within the same box (x_f, y_f, θ_f). The coordinates of the box are in units of pixels. Our test box is 26 inches by 21 inches and the pixel count along each axis is a scaled version of these dimensions. The length is 26*27 = 702 pixels and the width is 21*23 = 483 pixels. The actual coordinates of the robot in the box are scaled to the pixel representation by a straightforward unitary method calculation.

As can be seen in Figure 4.2, once the user indicates a desired starting point, the program prompts him to indicate a goal point. On receiving this input, the algorithm proceeds to build a trajectory to the end point based on what is discussed in Step 3 below. Although, after multiple iterations it may find a trajectory to the end point, this path planning might not be optimal. As a consequence, it is left to the user to extract the most optimal trajectory. The RRT graph generated is shown in Figure 4.3 with the goal state indicated with a brown point.

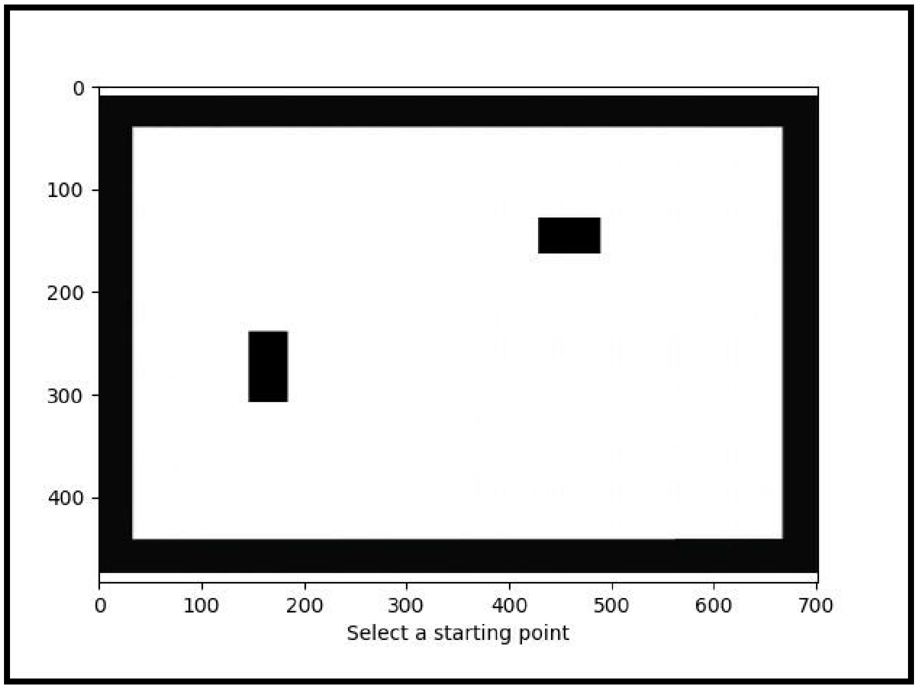


Figure 4.1: Map of Local Environment (Box) with Rectangular Obstacles

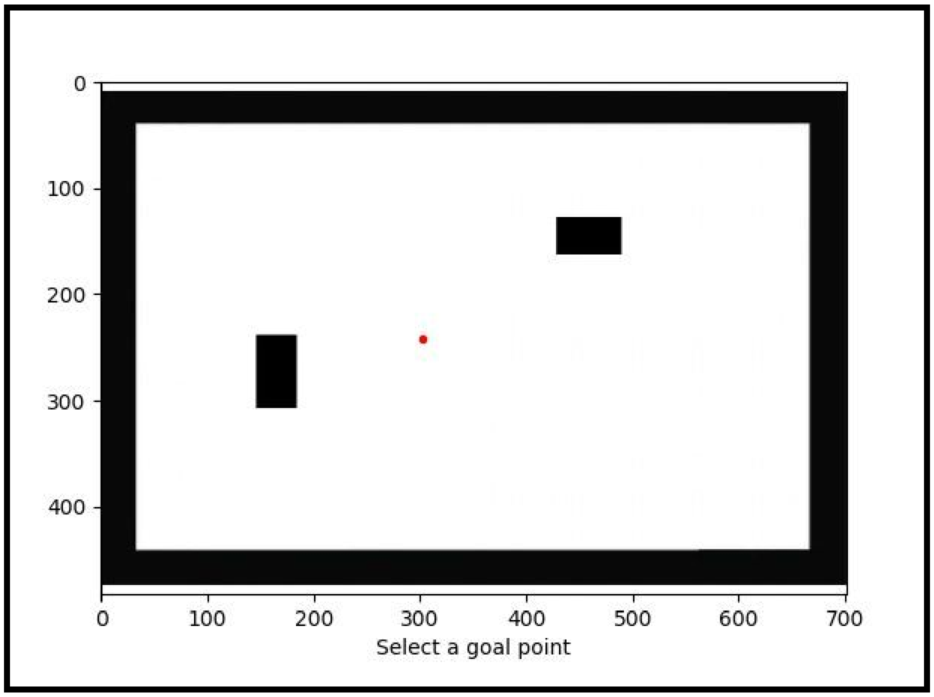


Figure 4.2: Map of local environment with Obstacles and an Indicative Starting Point

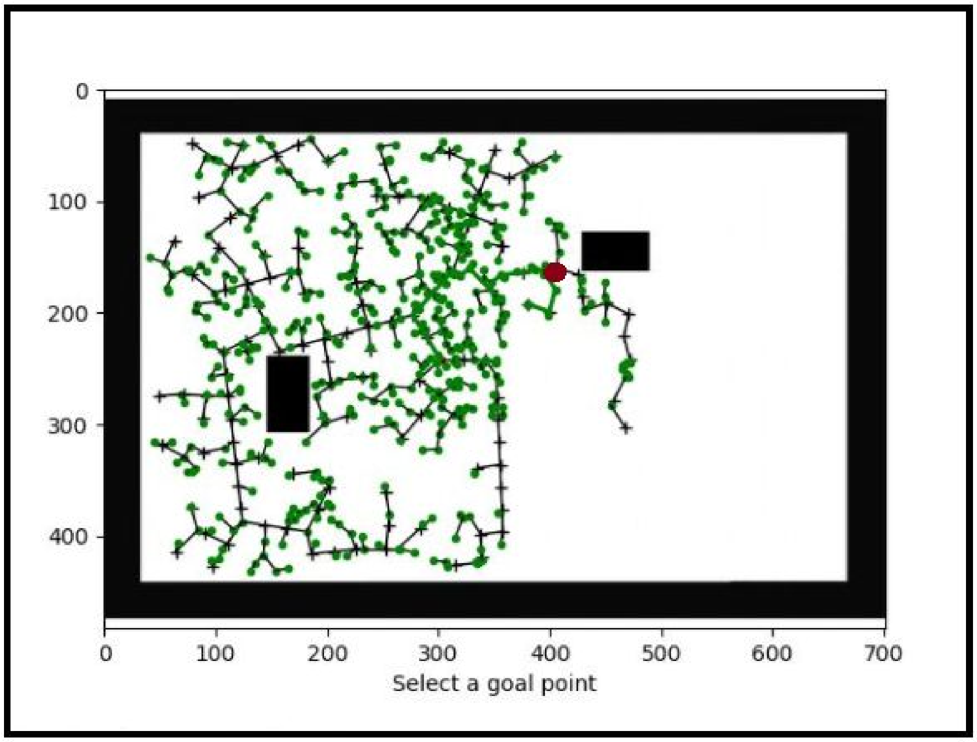


Figure 4.3: Result after running the RRT algorithm showing the tree created. The trajectory from the start to the goal (Brown Point) is then extracted from the tree

5. EXPERIMENTATION PROCEDURE

5.1 Verification of End to End Robot Plant Operation

Just to reiterate, in this lab we aim to perform state estimation and motion planning on a two-wheeled mobile robot with a dragging tail.

Step 1: Upload the files in the data folder and the paperbot.ino sketch onto the Arduino. This sketch provides a web interface to interact with the Arduino. It also reads data values from the IMU sensor and the LIDARs and prints them to the Serial. After uploading the sketch the user is required to connect to the Arduino via WiFi and open the IP address listed in the sketch on a browser. This works as the interface to pass commands to the mobile robot.

Step 2: Run the MAIN.py file in the Main Driver Code folder. This program reads data from the Serial thus reading sensor values as well as user inputs. The sensor readings and inputs are then passed to a MATLAB script implemented in the State Estimation folder. The MATLAB script maintains an Extended Kalman Filter object. This object takes a state transition function and a measurement function as inputs. The state transition function utilizes the user inputs to determine the expected position of the mobile robot. The measurement function uses a sensor model to provide the expected sensor measurements given the location and orientation of the vehicle. The Extended Kalman Filter updates the filter with actual sensor measurements on each iteration and tracks the corrected state output thus completing the state estimation segment of the lab.

Step 3: For the motion planning segment, a button called RRT is provided on the web interface. When the user clicks the RRT button, the MAIN.py script recognizes this input and then invokes the rrt.py script provided in the same folder. The rrt.py script opens an image of the box given in box.jpg. This box is treated as the environment the robot is in, where the whitespace is the place the robot is allowed to move around in and the black figures represent rectangular obstacles. The user is first prompted to select a starting point for the robot by clicking on the image and then prompted to click on another point on the image which is the goal state that the robot wants to get to. The program then takes these two points and runs an RRT algorithm to compute a trajectory from the start to the goal state (if one exists). The trajectory computed is shown on the figure as well as returned to the MAIN.py script and printed to the output console. The parameters for the RRT function can also be specified to change the amount of time one is willing to wait to check whether a route exists.

6. CONCLUSION AND FUTURE WORK

This lab successfully demonstrates methods that can be used for state estimation and motion planning. For the state estimation part, the Extended Kalman Filter implemented in MATLAB takes a state transition function and a sensor model and computes a corrected state as well as a predicted state along with the corresponding covariance matrices. The state transition matrices computed for the state transition function and the sensor model can be scaled to include more sensors and more parameters. The motion planning part successfully implements an RRT algorithm to compute a trajectory from an initial state to a goal state, and provides a visual depiction of the RRT and also prints the trajectory that the user can follow. Future work in this field can use the computed trajectory as inputs to a controller to make the robot autonomously follow that trajectory. The state estimator can used to confirm that the robot follows the computed trajectory.

7. REFERENCES

Ali, Abduladhem & Ali, Ramzy & Fortuna, Luigi & Frasca, Mattia & Xibilia, M.G.. (2012). Wireless underwater mobile robot system based on ZigBee. 117-122. 10.1109/ICFCN.2012.6206853.

Mehta, A. (2018, February 16). Mehtank / paperbot. Retrieved March 13, 2018, from https://git.uclalemur.com/mehtank/paperbot

Majidi, A. J. (2017, December 27). ArianJM/rapidly-exploring-random-trees. Retrieved March 13, 2018, from https://github.com/ArianJM/rapidly-exploring-random-trees

Steven M, LaValle & Jame J. Kuffner, Jr.
LaValle, S.M. & Kuffner, J.J., Jr. (2001). Rapidly-Exploring Random Trees: Progress and Prospects. *Algorithmic and Computational Robotics: New Directions*, pp. 293-308. http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01.pdf

Zolghadr, J. & Cai, Y. (20). Locating A Two-wheeled Robot Using Extended Kalman Filter. https://pdfs.semanticscholar.org/b632/cb585165203c09fbb9007d9ad135309e6e73.pdf

8. APPENDIX

Link to GitHub Repository: *https://github.com/rehan141196/EE183DA_Lab3*