ECE4160 Lab 11 (Real Localization)

Akshati Vaishnav

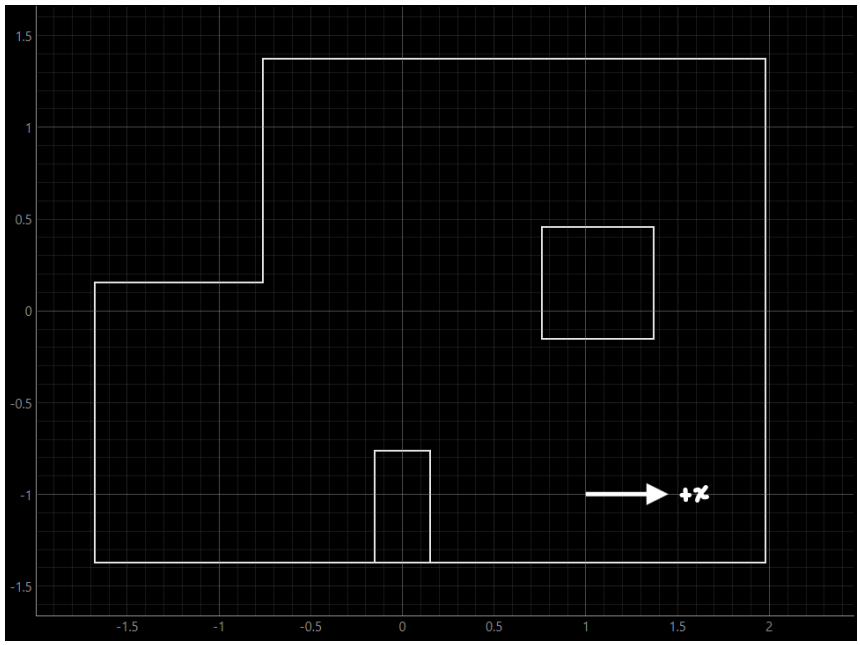
Overview

The purpose of this lab was to perform localization using the actual robot. The Bayes Filtering algorithm which we use was described in Lab 10. Here is the algorithm again for reference:

1. **Algorithm Bayes_Filter** $(bel(x_{t-1}),\ u_t,\ z_t)$:
2. **for** all x_t do — Transition probability/ action model
3. $\overline{bel}(x_t) = \Sigma_{x_{t-1}}\ p(x_t \mid u_t,\ x_{t-1})\ bel(x_{t-1})$ **(Prediction step)**
4. $bel(x_t) = \eta\ p(z_t \mid x_t)\ \overline{bel}(x_t)$ **(Update/measurement step)**
5. **end for** — Measurement probability/ sensor model
6. **return** $bel(x_t)$

Although the Bayes filter has the prediction and the update steps, we only implemented the update step in this lab. This was because there is significant noise when the robot turns, which results in the prediction step adding extra computational costs with not much benefit (low accuracy).

The essential idea is that the robot state is 3 dimensional and is given by (x, y, θ). The robot's world spans from $[-5.5, 6.5)$ ft in the x-direction, $[-4.5, 4.5)$ ft in the y-direction, and $[-180, 180)$ degrees in the θ-axis. We collect our robot's ToF data as the robot spins in a counter-clockwise direction starting with pointing to the $+x$-axis. The map is shown below:



Recap

Since we had already worked the localization code in the previous lab, we were given the localization code in this lab in the localization_extras.py file. Since the code was different from the code we wrote and tested in the previous lab, I tested the code again to ensure that the results matched up our expectations. The results from the run are shown below:

Odometry is shown in red, ground truth is shown in green, and the beliefs are shown in blue.
This graph shows how the provided code still works in simulation, which means that we can move on to do the rest of the lab.

Lab Overview

In the previous lab, the sim robot collected data by rotating around a simulation of the actual map. In this lab, we collect actual ToF data and pass it through the our bayes filtering localization algorithm. Specifically, we only run the update step of the algorithm.

Arduino Setup

On the Arduino side, I used the same code as I did for the Lab 9 (mapping), in which the robot rotated on-axis and collected ToF data. When I implemented this in Lab 9, I made my robot rotate in the clockwise direction, but I had to perform angle manipulations to ensure that the robot's motion matched the convention of rotating counter-clockwise. So, I changed my mapping code such that the robot would rotate in the counter-clockwise direction by default, eliminating the necessity of performing angular manipulations after collecting the data.

My final code is shown below:

```
case MAPPING: {
    // initialize variables and empty all arrays
    // ...

    //ccw rotation
    for(int exp_ang = 360; exp_ang > 0; exp_ang-=10) {
        while(1){
            get_dmp_data();
            if(exp_ang == 360) {
                first_angle = curr_ang;
            }

            curr_ang = yaw_dmp - first_angle;

            if(curr_ang<0) {
                curr_ang+=360;
            }
            if(exp_ang>150 && curr_ang<0) {
                curr_ang+=360;
            }

            err = (float)curr_ang - (float)exp_ang;

            if(abs(err) < 0.5 && err > -0.5) {
                run_motors(0,0,0,0,0);
                float temp_dist = 0.0;
                distanceSensor.startRanging();
```

```
                for(int k = 0; k < 5; k++){ //average 5 ToF measurements
                    while (!distanceSensor.checkForDataReady()) {
                        delay(1);
                    }
                    temp_dist += distanceSensor.getDistance();
                }
                tof_dist = temp_dist/5;
                dist[dist_idx] = tof_dist;
                dist_time[dist_idx] = millis()-start;

                ang_arr[dist_idx] = (float)curr_ang;
                expected_ang_arr[dist_idx] = (float)exp_ang;
                distanceSensor.clearInterrupt();
                distanceSensor.stopRanging();
                dist_idx+=1;
                break;

            }

            //travel the shortest route to the expected angle
            if(err>180) {
                err = err-180;
            } else if(err<-180) {
                err = err+180;
            }

            P = kp * err;
            pwm = P + I + D; //only performing P control


            //ensure that PWM is rescaled to not be too low or too high
            if(err<0) {
                //multiply PWM by 2 because CCW rotation is weaker
                pos = 1;
                pwm = (int(float(float(3*pwm)/255.0)*pwm_range)+min_pwm);
                rotate_ccw(0, abs(int(1*pwm)), abs(1*pwm), 0, 0);
            } else {
                pos = -1;
                pwm = int(float(float(pwm)/255.0)*pwm_range)-min_pwm;
                rotate_cw(abs(int(1*pwm)), 0, 0, abs(1*pwm), 0);
            }
            pwm1_arr[i] = pos*abs(pwm);
            tof_time[i] = millis() - start;
            i++;
        }
    }

    run_motors(0, 0, 0, 0, 0); //stop motors

    // send data back to laptop
    // ...
```

Although I started out by implementing my Arduino code such that my robot rotates in the CCW direction, I realized that my robot was unable to turn on-axis when I performed CCW rotation regardless of how high I set my PWM values. So, I ended up changing the code such that my robot rotates in the CW direction and reversed the order of the received data on the Python notification handler side.
The rotation of my robot in the CW direction is shown in Video 1 (outside of the PDF).

Python Setup

The primary class in which we had to develop code was in the RealRobot class, which is shown below:

```
class RealRobot():
```

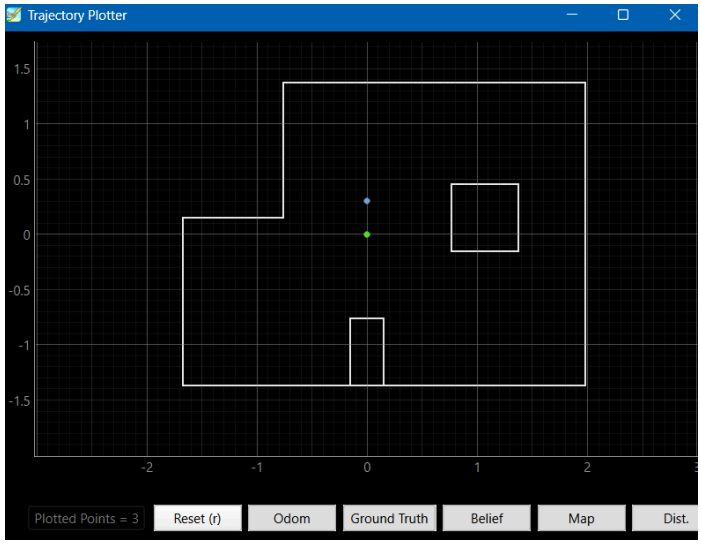
```
    # init function here

    def pwm_notification_handler(self, uuid, byte_array):

        self.message1.append(self.ble.bytearray_to_string(byte_array)[:])
        data = self.ble.bytearray_to_string(byte_array)
        arr = data.split("|")

        if(self.dist_idx1==0):
            self.dist_idx1 = int(arr[4])

        if(self.idx1 < self.dist_idx1):
            self.angle1.append(float(arr[0]))
            self.expected1.append(float(arr[1]))
            self.dist1.append(float(arr[2])/1000.)
            self.dist_time1.append(float(arr[3])/1000.)

        self.idx1+=1

    async def sleep_for_3_secs():
        await asyncio.sleep(3)

    async def perform_observation_loop(self, rot_vel=120):

        self.ble.send_command(CMD.MAPPING, data="1|0|0")

        # Wait for data to accumulate
        while len(self.dist1) < 72:
            print(f"Collected {len(self.dist1)} distances so far...")
            await self.sleep_for_3_secs()

        # reverse arrays to account for clockwise motion
        self.dist1 = self.dist1[::-1]
        self.angle1 = self.angle1[::-1]

        sensor_ranges = np.array(self.dist1)[np.newaxis].T
        sensor_bearings = np.array(self.angle1)[np.newaxis].T
        return sensor_ranges, sensor_bearings
```

The key function to implement in the python side was perform_observation_loop, which calls our Arduino command, waits for the arrays to fill up, and returns the collected distance and angle data. I also placed my notification handler into this class, as shown above. Since the perform_observation_loop command had to wait for the data to accumulate, I had to convert it into an asynchronous function and then call the sleep asynchronous function, as shown above and described by the lab handout. Furthermore, I followed lab instructions to add the `await` keyword to `loc.get_observation_data()` and the `async` keyword to `BaseLocalization.get_observation_data()`.

One important thing to note is that the distance must be in meters and the angle must be in degrees for the provided localization code to work!

Results (Part 1)

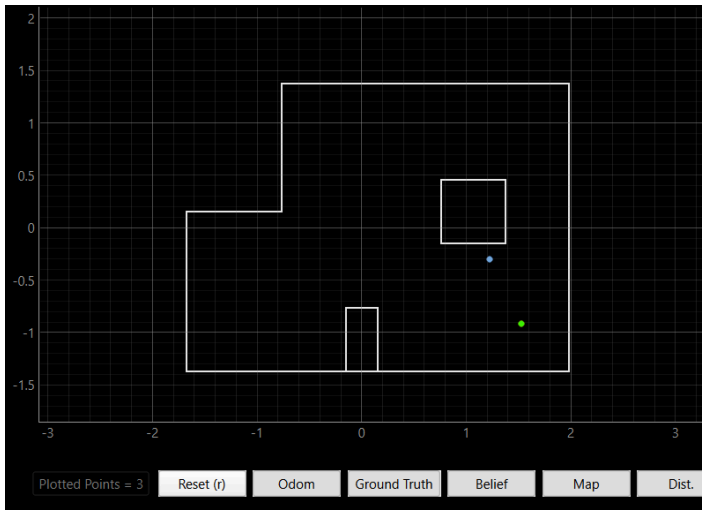
(0, 0)


Trajectory Plotter
Plotted Points = 3
Reset (r)
Odom
Ground Truth
Belief
Map
Dist.

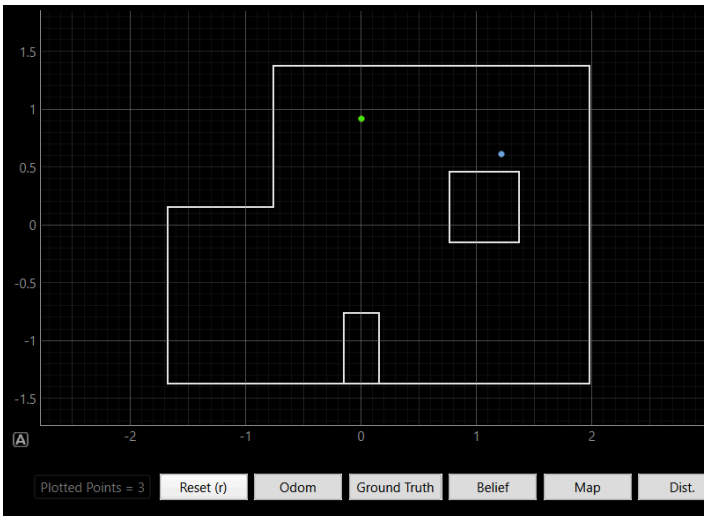
(5, 3)


Plotted Points = 3
Reset (r)
Odom
Ground Truth
Belief
Map
Dist.

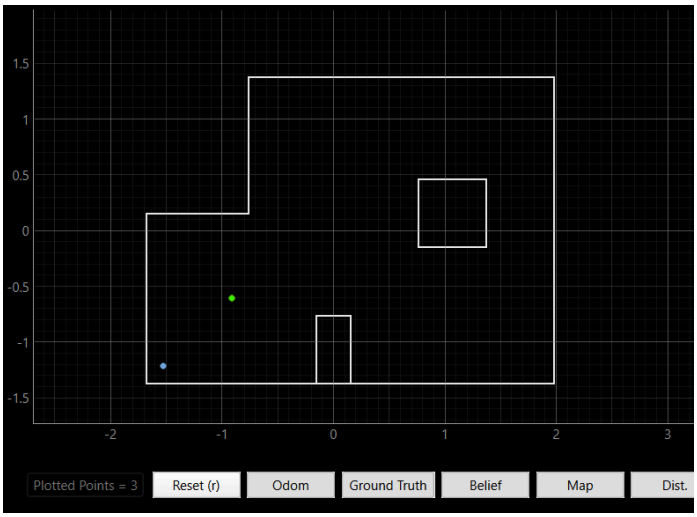
(5, -3)


Plotted Points = 3
Reset (r)
Odom
Ground Truth
Belief
Map
Dist.

(0, 3)



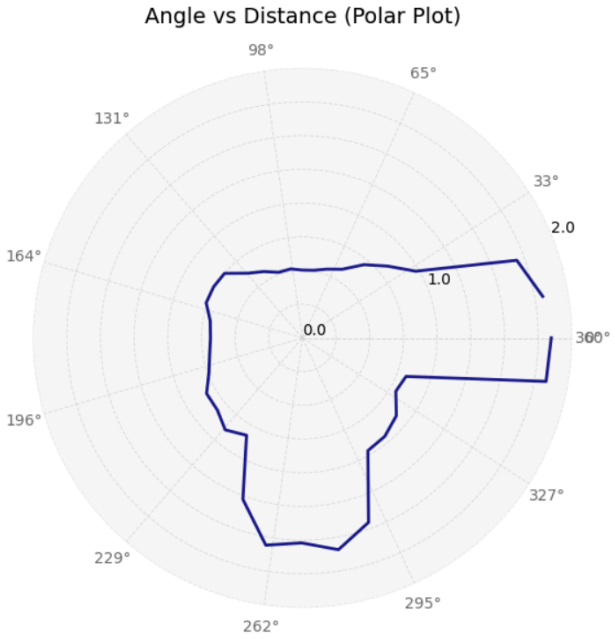
(-3, -2)



Discussion (Part 1)

My localization results were unfortunately not as good as I had expected. The results shown above were unfortunately not very repeatable and most of the time, my localization would predict the belief to be either inside the square box or around it. Point (0, 3) above is the best example of this.
I tried changing the sensor noise, as suggested by the lab handout, but that just changed the certainty of the belief.
In order to ensure that my ToF data was being recorded properly, I plotted the recorded data as a polar plot and on the world map, and the

data seemed to map quite well, even better than my results from Lab 9. The polar plot below is shown for the (0,3) point.

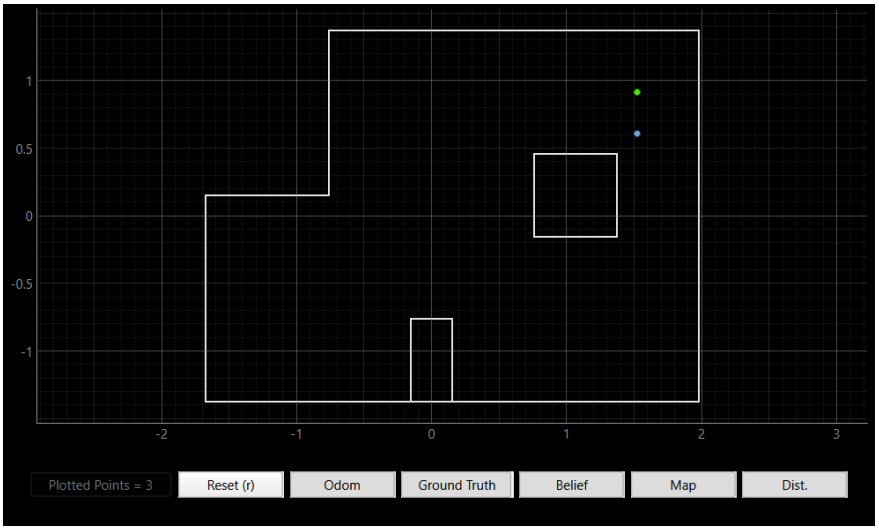


I also tried rotating my robot counter-clockwise and removing the array reversal on the Python side to adjust for CW movement, but this also resulted in bad localization results.
I asked multiple TAs and Professor Helbling (and posted on Ed), and it seems like some students are having this issue but there is no solution yet. I am trying to improve my Arduino code to see if a different approach to collecting data could result in improved results. I am also planning on running my code on another robot to see if my ToF sensor could be the issue, although I don't think that it is since my data is quite good in the plot above.
Getting this part is quite important for Lab 12, and I will continue to debug this over the next few days.

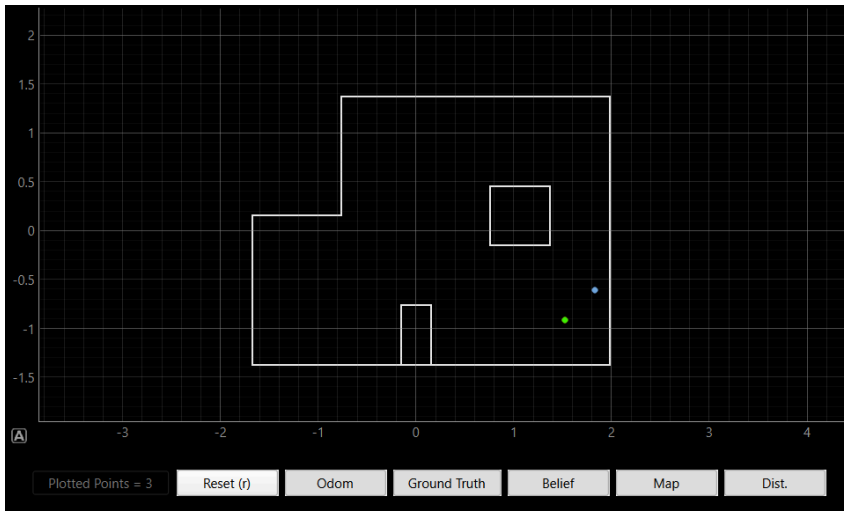
Results (Part 2)

After hours of debugging with the Professor, the TA, and Becky, I was able to figure out the issue with my lab. It turns out that both of my ToF sensors on my robot had terrible range even when they were on Long distance mode (*what are the odds?!*). This issue with the range resulting in points more than around 1 - 1.5 meters away to be capped at a lower value, which ended up messing with the Bayes filter. I ended up using Becky's robot and flashed my code onto it as her robot was working for her and voila! Here are my updated plots:

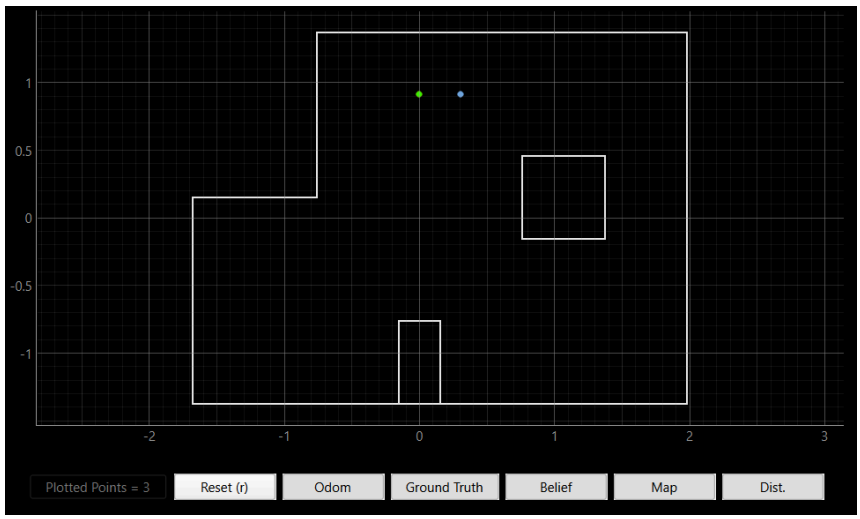
(5, 3)



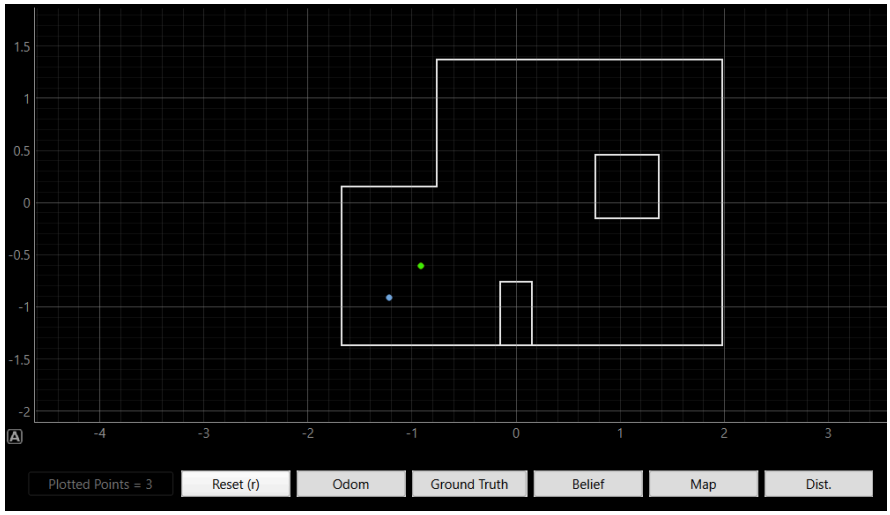
(5, -3)



(0, 3)



(-3, -2)



Although I wasn't able to record a video of the sight, Becky's robot also rotated slightly better (more on-axis) than mine.
For Becky's robot, I used her side sensor, so I had to rotate the robot by -90 degrees such that it pointed in the $-y$-direction. I also edited my code (again) such that my robot would rotate in the counter-clockwise direction so I didn't have to edit the python code to reverse my arrays.
Here is a screenshot of my update step results for the point (0, 3):

```
2025-05-01 17:03:49,909 | INFO     |: Update Step
2025-05-01 17:03:49,912 | INFO     |:      | Update Time: 0.003 secs
2025-05-01 17:03:49,912 | INFO     |: Bel index     : (np.int64(6), np.int64(7), np.int64(4)) with prob = 0.9999999
2025-05-01 17:03:49,912 | INFO     |: Bel_bar prob at index = 0.00051440329218107
2025-05-01 17:03:49,915 | INFO     |: Belief        : (0.305, 0.914, -90.000)
```

The belief probability is quite high and relatively accurate which is great!

Discussion (part 2)

Preface: Although I was ~~quite~~ a little disappointed that my hardware was the issue, this lab taught me how intricate embedded systems debugging can be (and that you really can't trust anything). The excitement of getting my Bayes Filter was truly like no other.
Overall, through this debugging process, I learned how important accurate distance measurements are for the Bayes filter to work properly. Although my belief plots are not completely accurate, the beliefs are quite close to the ground truth which demonstrates that my robot generally understands its surroundings, which is a win!

References

Shoutout to TA Cameron, Becky, and Professor Helbling for their support!!
I referred Mikayla Lahr's website and the lab handout. I also received debugging help from Annabel and Katarina, along with TA Nandita and TA Cheney.