Lab 3

University of California, Los Angeles
EE209AS

James (Zach) Harris (UID: 105221897)
Hamza Khan (UID: 205223240)
Joshua Hannan (UID: 805221851)

jzharris@g.ucla.edu
hamzakhan@ucla.edu
jhannan@g.ucla.edu

0 Preliminaries

0(a) Links

Public Github repo (https://github.com/jzharris/ECE209AS_Project_Submissions/tree/master/Lab3). This repo contains our code in a Jupyter Notebook, evaluation metrics, visualizations, plots, and gifs.

0(b) Collaborators

Our team was composed of three people: James (Zach) Harris, Hamza Khan, and Joshua Hannan.

0(c) Other Resources

We used the datasheets for the MPU-9250 IMU [1] and the VL53L0X laser range senor [2] to determine the characteristics of these sensors. We consulted [3] in the process of determining our robot's kinematics. We referenced [4] when we determined our Extended Kalman Filter characteristics. We drew inspiration from some of the techniques described in [5] in the process of evaluating our state estimator. We consulted [6] when evaluating the convergence of our covariance matrix. We utilized some of the observations in [7] while comparing different state estimators. Otherwise, we largely consulted our notes from EE209AS Computational Robotics taught at UCLA by Ankur Mehta [8].

0(d) Aggregate Contributions

The aggregate contributions percent of each member was basically equal, 33%. During the project, we were always all three working at the same time, alternating between one person coding and the other two looking through notes and using the information to help the current designated coder. We determined our algorithms together and debugged together.

1 Robot Model

2 Mathematical Setup

We defined a few classes to represent the entire problem statement. First, we created an `Environment` class that contains details about the environment we can traverse, which is simply a rectangular box in free space with L=750mm and W=500mm. Figure 1 shows a graphical depiction of the robot in this particular environment.

2(a) Complete System Model

To define the complete system model, we created a class called `SystemModel`. This class takes parameters specific to the robot, such as its wheel radius and distance between the wheels, as well as information about time difference between sensors (Δ_t). It also has information about each sensor, including its noise standard deviation.

We created a class to model each sensor and actuator, including `MagnetometerSensor`, `RangeSensor`, `GyroscopeSensor`, and `ServoActuator`. Each sensor has functions to give

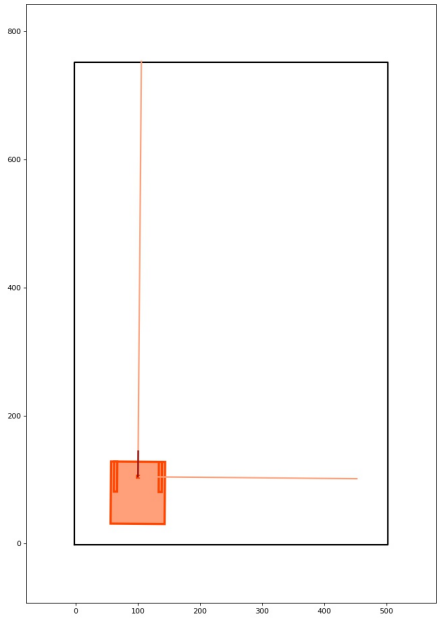

Fig. 1. Robot evironment with bounding boxes

the sensor reading both with and without noise. The actuator class has a function to add noise to its control inputs as well, based on the servo noise standard deviation.

We created a `State` object similar to the one from the previous lab, with inputs of (x, y, θ, ω) representing our current state. x and y are corresponding displacements from the origin (defined as the bottom left corner in our model), θ is the orientation of the robot (counter clockwise rotation from 12 o'clock), and ω is the angular velocity of the robot.

Mathematical Setup: Magnetometer The magnetometer was the simplest sensor to model, since its angle is expected to directly correspond to the θ value of our robot's state. At any point in time, the magnetometer sensor reading is simply the actual θ value with some added noise term σ_{mag}. The equation is

$$\theta_{mag} = \theta_{robot} + \mathcal{N}(0,\, \sigma_{mag}^2) \tag{1}$$

Mathematical Setup: Gyroscope The gyroscope sensor measures angular velocity. Given two control inputs, RPM_L and RPM_R, the wheel radius R, and the distance between the wheels l, the sensor measurements were as follows:

$$V_L = \frac{RPM_L(2\pi R)}{60} \tag{2}$$

$$V_R = \frac{RPM_R(2\pi R)}{60} \tag{3}$$

$$\omega_{gyr} = \frac{V_L + V_R}{l} + \mathcal{N}(0,\, \sigma_{gyr}^2) \tag{4}$$

Mathematical Setup: Range Sensor Calculating the range sensor output on each side was a little more complicated than for the other sensors. We will write our setup for the upwards-facing sensor, but the math is similar for the right-facing sensor (though our θ becomes $\theta - \frac{\pi}{2}$).

We first calculated the distance expected from the sensor to each wall, given our (x, y, θ) state. Figure 2 shows the angles needed to calculate the distances D. Using these angles, we obtained the following equations:

$$D_{right} = \frac{W - x}{\cos(\theta)} \tag{5}$$

$$D_{left} = \frac{x}{\cos(\pi - \theta)} \tag{6}$$

$$D_{top} = \frac{L - y}{\cos(\theta - \frac{\pi}{2})} \tag{7}$$

$$D_{bottom} = \frac{y}{\cos(\frac{3\pi}{2} - \theta)} \tag{8}$$

As we would expect, these equations would lead to many edge cases to think about. Most importantly, if we have our sensor perpendicular to a wall, then our distance would

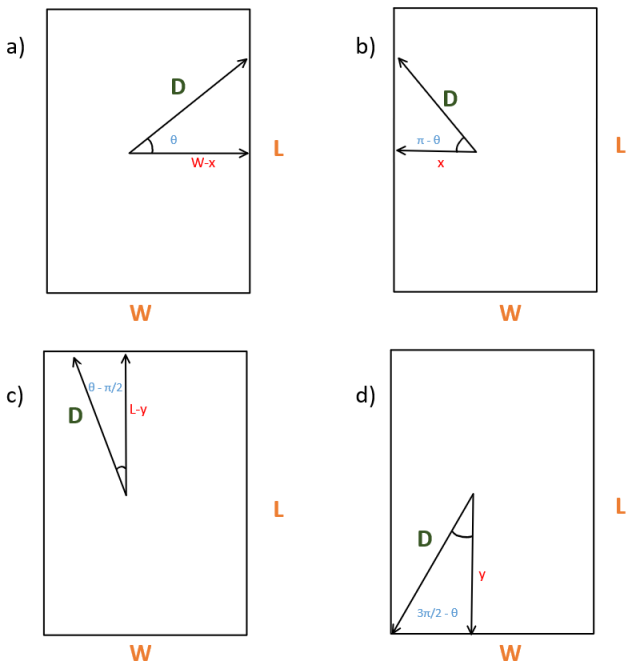

Fig. 2. Angles used to calculate the range sensor values

include a division by zero term (so we just ignore if we encounter that error as seen in our class `RangeSensor`).

Secondly, we expect to typically have two positive distance outputs from the above equations and two negatives (since the distance to the left wall would be negative when facing right). To handle this, we chose the closest wall to be the one with the smallest positive distance. We then used its distance to be the outputted range sensor distance.

$$D_{range} = \min([D_{top}, D_{bottom}, D_{left}, D_{right}] \; s.t. \; D_i > 0) + \mathcal{N}(0, \sigma^2_{range}) \tag{9}$$

Mathematical Setup: Servo Actuator For the servo actuator, our outputted range value is simply the input control values with added noise (clamped to stay within possible range of the servo inputs) needed to follow a reference trajectory.

$$RPM_{actuator} = min(max(RPM_{input} + \mathcal{N}(0, \sigma^2_{actuator}), -60), 60) \tag{10}$$

2(b) Noise Terms

We integrated noise terms into each equation using the σ^2 terms found from the project specifications and the sensor datasheets [2] and [1]. The noise terms we used were derived as shown in the following subsections.

Magnetometer Noise For the magnetometer, we saw from the datasheet that the precision was accurate to 14 bits out of 16 bits. Therefore, we had our standard deviation to be 2^{-16}.

$$\sigma_{mag} = 6.103515625e - 05 \tag{11}$$

Gyroscope Noise For the gyroscope, we saw from the datasheet that the RMS noise was 0.1 degree / s-rms.

$$\sigma_{gyr} = 0.1 \tag{12}$$

Range Sensor Noise For the range sensor, we saw that at 120cm, we had a 3% error standard deviation, giving us 3.6 cm (36mm) standard deviation.

$$\sigma_{range} = 36 \tag{13}$$

Actuator Noise For the actuator, we were given that the standard deviation is 5% of the max motor speed (60 RPM).

$$\sigma_{act} = 3 \tag{14}$$

2(c) Extended Kalman Filter

We implemented an Extended Kalman Filter using the following time update:

$$\overline{x}_{t+1} = f(\overline{x}_t, u_t, 0) \tag{15}$$

$$\Sigma_{t+1} = F_t \Sigma_t F_t^T + W_t Q W_t^T \tag{16}$$

We used the observation update as defined below. We used Joseph's Form given in equation 19 for the Sigma update to account for numerical rounding problems that lead to a non positive semi definite matrix:

$$\overline{x}_{t^+} = \overline{x}_{t^-} + \overline{\Sigma}_{t^-} H_t^T (H_t \Sigma_{t^-} H_t^T + R)^{-1} (y_t - h(\overline{x}_{t^-})) \tag{17}$$

$$K_t = \Sigma_{t^-} H^T (H_t \Sigma_{t^-} H_t^T + R)^{-1} \tag{18}$$

$$\Sigma_{t^+} = [I - K_t H] \Sigma_{t^-} [I - K_t H]^T + K_t R K_t^T \tag{19}$$

Nonlinear Kinematic Equations Nonlinear kinematic equations must be defined in order to simulate the robot moving in its environment. We approach this problem by first assuming that the desired wheel speeds will not change value between time steps. In other words, the robot will move in an arc between state x_t and x_{t+1}. This assumption enables us to define a robot facing angle θ to move around center point ICC like in Figure 3 [3].

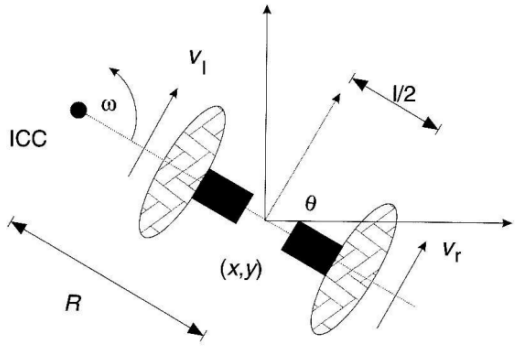

Fig. 3. Representation of robot kinematics [3]

We first defined the following useful variables:

$$\omega = \frac{V_r - V_l}{l} \tag{20}$$

$$v = \frac{V_l + V_r}{2} \tag{21}$$

$$\theta_{avg} = \theta + \frac{\omega \Delta_t}{2} \tag{22}$$

where ω is the angular velocity, v is the tangential velocity, and θ_{avg} is the difference between the robot's initial θ and final θ.

We transition the robot from state x_t to x_{t+1} by performing the operations defined by the following:

$$\mathbb{R} = \frac{l * (V_l + V_r)}{2 * (V_r - V_l)} \tag{23}$$

$$ICC = [p_{t,x} - \mathbb{R} * sin(\theta), p_{t,y} + \mathbb{R} * cos(\theta)] \tag{24}$$

$$\begin{aligned} \bar{R} &= RotationMatrix(\omega \Delta_t) \\ p_{t+1} &= (p_t - ICC) \times \bar{R} - ICC \\ \theta_{t+1} &= \theta_t + \omega \Delta_t \\ \omega_{t+1} &= \omega \end{aligned} \tag{25}$$

$\mathbb{R}$ is the radius of curvature of the path the robot will take. ICC is the center point that the robot rotates around. $\bar{R}$ is a rotation matrix designed to rotate a point by $\omega\Delta_t$. The position, angle, and angular velocity are adjusted via equation 25 and are taken to be the new state x_{t+1}.

Linear Kinematic Equations Simulated data can be acquired using the nonlinear kinematic equations, however we require a linear approximation in order to perform the Kalman Filter. We define the linear kinematic equations as follows:

$$x_{t+1} = x_t + v * \cos(\theta_{avg})\Delta_t \tag{26}$$

$$y_{t+1} = y_t + v * \sin(\theta_{avg})\Delta_t \tag{27}$$

$$\theta_{t+1} = \theta_t + \omega\Delta_t \tag{28}$$

$$\omega_{t+1} = \omega_t + \alpha_t\Delta_t \tag{29}$$

where α_t is the angular acceleration

Calculating the F Matrix To calculate the F matrix, we performed linearization by taking derivatives for each kinematic equation, giving us the following matrix:

$$F = \begin{bmatrix} 1 & 0 & -v\sin(\theta_{avg})\Delta_t & 0 \\ 0 & 1 & v\cos(\theta_{avg})\Delta_t & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{30}$$

Calculating the W Matrix To calculate the W matrix, we performed linearization in terms of our control inputs and obtained the following:

$$W = \begin{bmatrix} \frac{1}{2}\cos(\theta_{avg}) - \frac{1}{2l}v\Delta_t\sin(\theta_{avg}) & \frac{1}{2}\cos(\theta_{avg}) + \frac{1}{2l}v\Delta_t\sin(\theta_{avg}) \\ \frac{1}{2}\sin(\theta_{avg}) + \frac{1}{2l}v\Delta_t\cos(\theta_{avg}) & \frac{1}{2}\sin(\theta_{avg}) - \frac{1}{2l}v\Delta_t\cos(\theta_{avg}) \\ \frac{1}{l} & -\frac{1}{l} \\ \frac{1}{l*\Delta_t} & -\frac{1}{l*\Delta_t} \end{bmatrix} \tag{31}$$

Calculating the Q Matrix We set Q to use the standard deviations for each actuator and obtained the following matrix:

$$Q = \begin{bmatrix} (0.05\ V_l)^2 & 0 \\ 0 & (0.05\ V_r)^2 \end{bmatrix} \tag{32}$$

Calculating the R Matrix We set R to use the standard deviations for each of the sensors and obtained the following matrix:

$$R = \begin{bmatrix} 36^2 & 0 & 0 & 0 \\ 0 & 36^2 & 0 & 0 \\ 0 & 0 & (6.103515625e-5)^2 & 0 \\ 0 & 0 & 0 & 0.1^2 \end{bmatrix} \tag{33}$$

Calculating the H Matrix To calculate the H matrix, we linearized our sensor readings for all four given sensors with respect to the state variables (x, y, θ). Calculating the derivatives for the gyroscope and magnetometer was trivial (since the gyroscope has no dependence whatsoever on state, and the magnetometer only depends on the θ value).

For both range sensors, we have to conditionally calculate the derivatives with respect to the wall that the sensor is pointing to. The equations are the same for both, except that the θ for the right sensor becomes $\theta - \frac{\pi}{2}$. The derivatives depend on Equations 5 to 8, and are defined in the below equations. Note that we will use the notation of $h_{(side)}$ to refer to the row of H corresponding to the linearization when the sensor is pointing in the direction (eg: $[\frac{d\,straight}{dx}, \frac{d\,straight}{dy}, \frac{d\,straight}{d\theta}]$).

$$h_{right} = [-\frac{1}{\cos(\theta)},\ 0,\ \frac{x - W}{\cos^2(\theta)}\sin(\theta)] \tag{34}$$

$$h_{left} = [\frac{1}{\cos(\theta + \pi)},\ 0,\ -\frac{x}{\cos^2(\theta + \pi)}\sin(\theta + \pi)] \tag{35}$$

$$h_{top} = [0,\ -\frac{1}{\cos(\theta - \frac{\pi}{2})},\ \frac{y - L}{\cos^2(\theta - \frac{\pi}{2})}\sin(\theta - \frac{\pi}{2})] \tag{36}$$

$$h_{bottom} = [0,\ \frac{1}{\cos(\theta + \frac{\pi}{2})},\ -\frac{y}{\cos^2(\theta + \frac{\pi}{2})}\sin(\theta + \frac{\pi}{2})] \tag{37}$$

Putting it all together into one matrix, we get:

$$H = \begin{bmatrix} \frac{d\,straight}{dx} & \frac{d\,straight}{dy} & \frac{d\,straight}{d\theta} & 0 \\ \frac{d\,right}{dx} & \frac{d\,right}{dy} & \frac{d\,right}{d\theta} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{38}$$

where the first two rows of derivatives are obtained conditionally based on the wall which the sensor is facing (using equations 34 to 37).

3 Evaluation

3(a) Reference Trajectories

Reference trajectories are used to generate a series of inputs that transition the robot from the current state to the next state. In other words, a list of inputs U is defined such that for the given trajectory T, equation 39 holds true.

$$\begin{aligned} t_0 &= x_0 \\ t_{i+1} &= f(x_i, u_i) \forall i \in \{0, ..., |T| - 1\} \end{aligned} \tag{39}$$

Our reference trajectories are produced by first simulating a series of events we wish the robot to perform. Each trajectory is then inverted to determine the inputs needed to transition the robot to the next state in the trace. On the surface this may appear to be a circular way of hardcoding trajectories, however the first step can be replaced by an arbitrary trajectory T given it satisfies equation 39. The trajectories are inverted to get inputs in RPM by performing the following operations for $t \in \{0, ..., |T| - 1\}$:

$$\begin{aligned} \bar{R}^{-1} &= RotationMatrix(-\omega \Delta_t) \\ ICC &= (p_t - (p_{t+1} \times \bar{R}^{-1})) \times (I - \bar{R}^{-1})^{-1} \\ \mathbb{R} &= ICC_y - p_{t,y})/cos(\theta_t) \end{aligned} \tag{40}$$

$$\begin{aligned} V_l &= \frac{\omega}{\Delta_t} * (R - l/2) \\ V_r &= \frac{\omega}{\Delta_t} * (R + l/2) \end{aligned} \tag{41}$$

$$\begin{aligned} RPM_r &= V_r * \frac{60}{2\pi r} \\ RPM_l &= V_l * \frac{60}{2\pi r} \end{aligned} \tag{42}$$

The following trajectories were selected to capture the abilities and limitations of the Extended Kalman Filter implementation.

Complex Trajectory In order to test the Kalman filter's ability to handle remaining stationary and spinning in place, we derive a trajectory that moves in a more complicated pattern. In order to test the impact noise has on the system, we also simulate the same trajectory with three times the amount of noise per sensor. The inputs used to derive the trajectory are found in our source code. A gif showing the robot's movements can be found on our git repository.

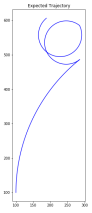

Fig. 4. Clean complex traj.

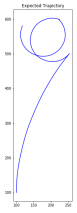

Fig. 5. Noisy complex traj.

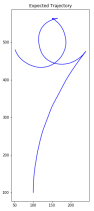

Fig. 6. 3x Noise complex traj.

Circular Trajectory In order to test the Kalman filter's ability to track a constant motion for a long period of time, we derive a trajectory that moves the robot in a circle. One of the wheels is also given a small sinusoidal variation to the sensitivity to degree change. The inputs used to derive the trajectory are found in our source code. A gif showing the robot's movements can be found on our git repository.

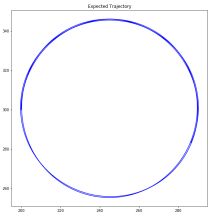

Fig. 7. Clean circular trajectory

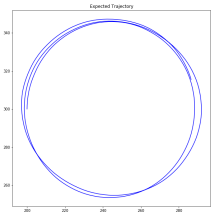

Fig. 8. Noisy circular trajectory

Pointed Trajectory In order to test the Kalman filter's ability to handle straight movements, abrupt pivots on either wheel, and changing directions, we derive a trajectory that moves the robot in a pointed pattern. The inputs used to derive the trajectory are found in our source code. A gif showing the robot's movements can be found on our git repository.

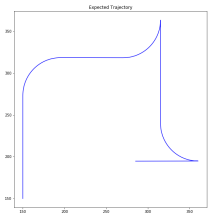

Fig. 9. Clean pointed trajectory

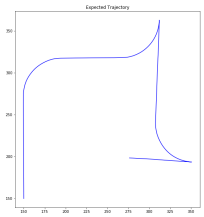

Fig. 10. Noisy pointed trajectory

Spiral Trajectory In order to test the Kalman filter's ability to handle angular accelerations and tight arcs, we derive a trajectory that moves the robot in a spiral pattern. The inputs used to derive the trajectory are found in our source code. A gif showing the robot's movements can be found on our git repository.

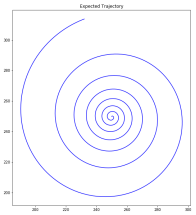

Fig. 11. Clean spiral trajectory

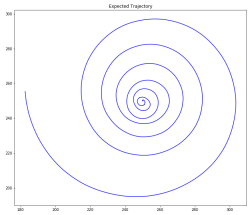

Fig. 12. Noisy spiral trajectory

3(b) Simulation

Each trajectory was simulated using the nonlinear kinematics and system model defined above. The sensor traces for each simulation are elaborated below.

Complex Trajectory Sensor Traces The complex trajectory introduces abrupt changes in sensor readings for each sensor. Although the noisy traces are very similar to their clean counterparts, it is evident from Figure 14 that the high frequency components are lost.

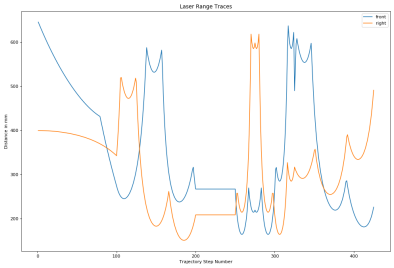

Fig. 13. Clean laser traces

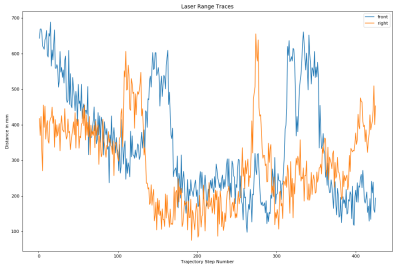

Fig. 14. Noisy laser traces

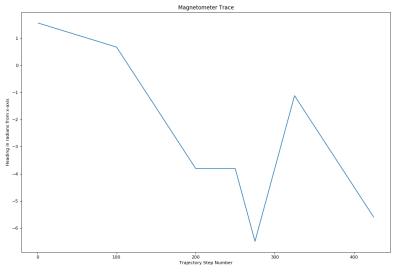

Fig. 15. Clean magnetometer trace

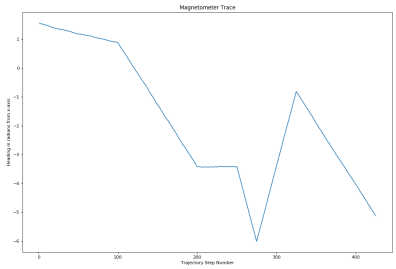

Fig. 16. Noisy magnetometer trace

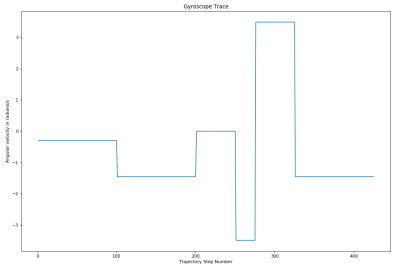

Fig. 17. Clean gyro trace

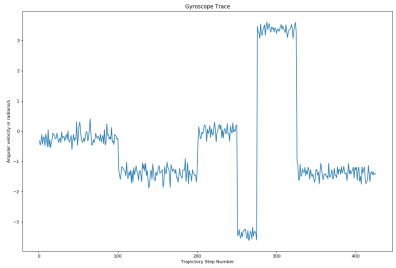

Fig. 18. Noisy gyro trace

Circular Trajectory Sensor Traces The following six figures show the sensor traces for the circular trajectory. From Figure 19 it can be seen that the laser range sensors are outputting a oscillatory pattern, which is to be expected. The readings in Figure 20 suggest that the noise in the laser sensors makes it impossible to pick out the high frequency changes in the sensor, however the low frequency components remain intact. The magnetometer remains minimally affected. The noise in the gyroscope make it impossible to measure the small change in angular velocity that was encoded in the trajectory as the robot moved around the circle.

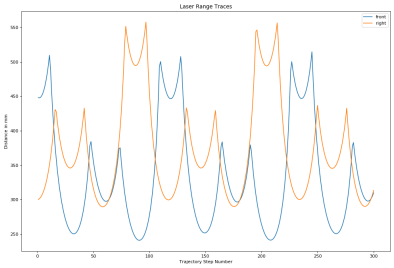

Fig. 19. Clean laser traces

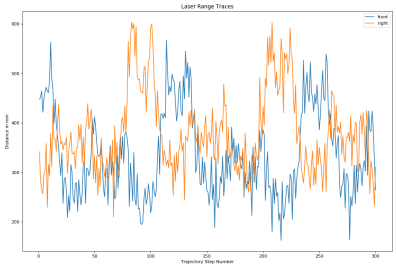

Fig. 20. Noisy laser traces

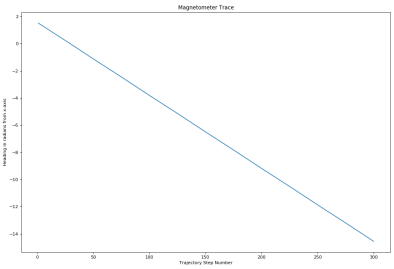

Fig. 21. Clean magnetometer trace

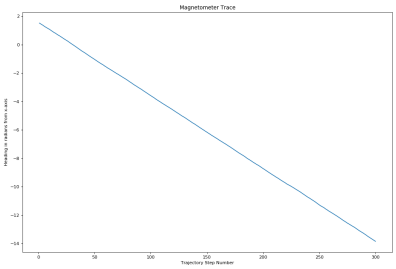

Fig. 22. Noisy magnetometer trace

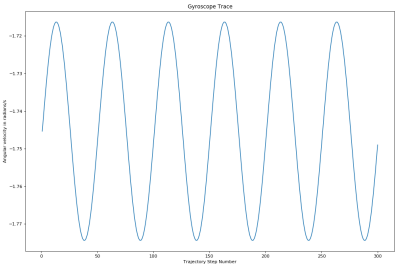

Fig. 23. Clean gyro trace

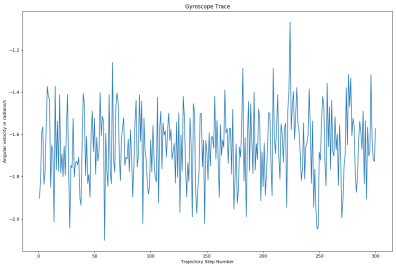

Fig. 24. Noisy gyro trace

Pointed Trajectory Sensor Traces The pointed trajectory consists of moving straight and quick turns. This pattern reduces the high frequency component of the laser sensors, which makes their noisy counterpart closer to ground truth. The rapid turning can be handled by the gyroscope sensor as shown in Figures 29 and 30.

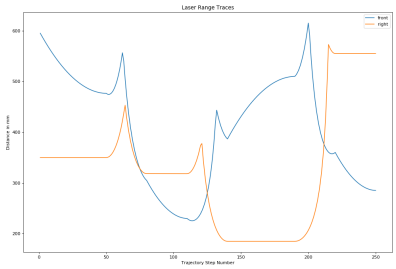

Fig. 25. Clean laser traces

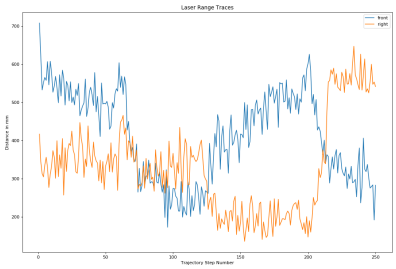

Fig. 26. Noisy laser traces

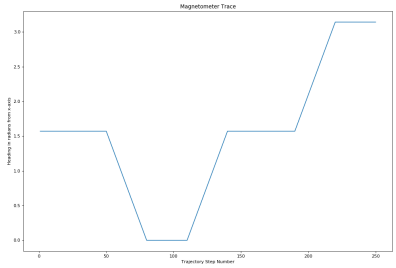

Fig. 27. Clean magnetometer trace

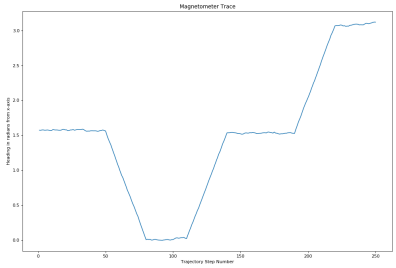

Fig. 28. Noisy magnetometer trace

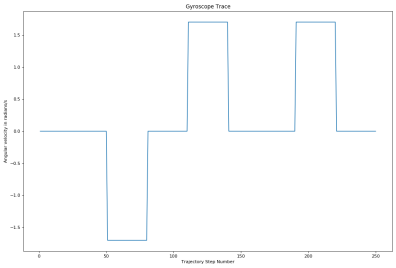

Fig. 29. Clean gyro trace

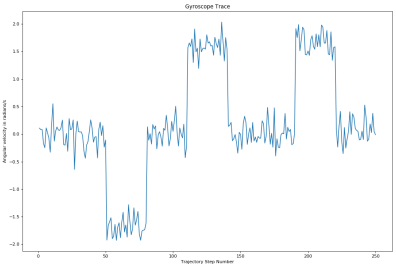

Fig. 30. Noisy gyro trace

Spiral Trajectory Sensor Traces The spiral trajectory contains a lot of high-frequency elements in the laser readings. These are lost due to noise. However, the gyroscope trace is very clean, and can theoretically be utilized by the Kalman filter to perform a better estimate than the laser sensor.

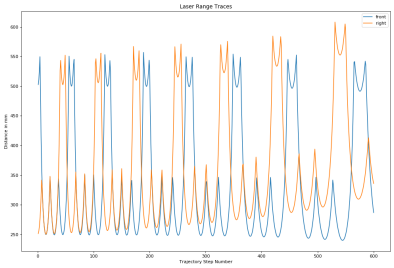

Fig. 31. Clean laser traces

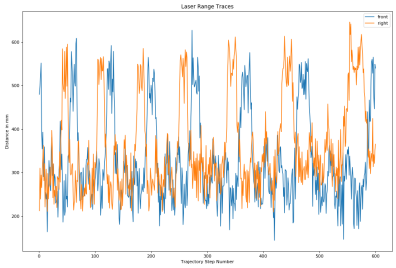

Fig. 32. Noisy laser traces

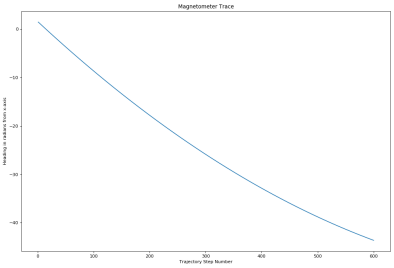

Fig. 33. Clean magnetometer trace

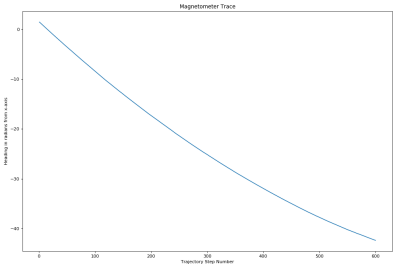

Fig. 34. Noisy magnetometer trace

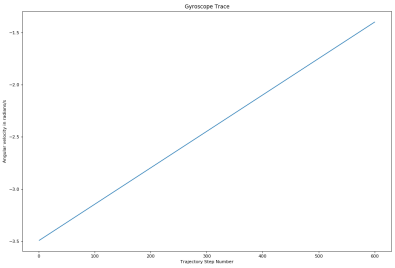

Fig. 35. Clean gyro trace

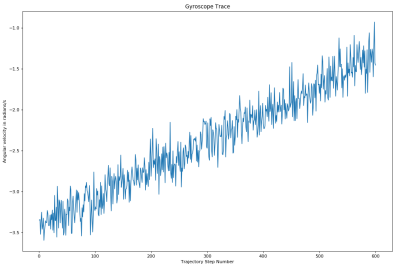

Fig. 36. Noisy gyro trace

3(c) Implementing the KF

We implemented our EKF state estimator on the reference trajectories described above. We included examples with perfect knowledge of the initial state, no knowledge of the initial state, and increased noise. We used several data metrics and graphical visualizations to characterize the performance of our state estimator both in terms of accuracy and efficiency. We provide an overview of the evaluation techniques that we used and then describe the results of our experiments. We include all of the characterizations and plots for the complex trajectory as a "baseline" model. For the sake of brevity and readability, we omit some of the plots. Please refer to the repo for our full results.

Euclidean Distance We measure the Euclidean distance between the expected (x_e, y_e) coordinate and the obtained (x_o, y_o) coordinate using the formula

$$dist = \sqrt{(x_e - x_o)^2 + (y_e - y_o)^2} \tag{43}$$

where (x_e, y_e) is the expected coordinate of the robot and (x_o, y_o) is the obtained coordinate of the robot using the EKF. The distance is calculated for each step of the trajectory and plotted accordingly. For easier visualization purposes, we plot the "smoothed" distance by plotting the average of every 10 euclidean distances. The non-smoothed plots are included in the Notebook and the repo. We calculate the mean distance between the expected and obtained over all steps of the trajectory.

θ Estimation and RMSE We measure and plot the absolute distance between the expected and obtained θ at each step in the trajectory $|\theta_e - \theta_o|$. For visualization purposes, we also plot the "smoothed" difference by plotting the average of every 10 differences in θ. We calculate the Root Mean Square Error (RMSE) of the difference between the expected and obtained θ over all trajectory steps as a metric to determine the performance of the EKF. RMSE is a common performance metric for state estimators, according to [5], since it gives particular weight to large errors. We calculate RMSE in the following manner

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(obs_i - exp_i)^2} \tag{44}$$

where n is the number of steps taken, exp_i is the expected value at the i^{th} step, and obs_i is the observed value at the i^{th} step.

ω Estimation and RMSE We measure and plot the absolute distance between the expected and obtained ω at each step in the trajectory $|\omega_e - \omega_o|$. For visualization purposes, we also plot the "smoothed" difference by plotting the average of every 10 differences in ω. We calculate the Root Mean Square Error (RMSE) of the difference between the expected and obtained ω over all trajectory steps as a metric to determine the performance of the EKF.

Covariance "Size" Estimation The previous three metrics largely measure the accuracy of the EFK's state estimation. However, the efficiency of the EKF is also important to measure. We measure the efficiency of our EKF implementation evaluating the "size" of the covariance matrix. As the size of the covariance matrix decreases, the algorithm converges. Thus, an algorithm is more efficient when it converges more quickly. Several methods exist to quantify the size of a covariance matrix, including calculating the trace norm or evaluating the determinant of the covariance matrix [6]. We chose to use the determinant of the covariance matrix as our metric to determine convergence, since it measures the volume covered by the covariance matrix. We calculated the determinant using the product of the eigenvalue of the covariance matrix Σ as follows

$$\Sigma_{size} = \prod \sqrt{\lambda_i} \tag{45}$$

where λ_i is the i^{th} eigenvalue of Σ. We plot Σ_{size} at each trajectory step to visualize the convergence of the EKF. We also plot ellipses that represent the covariance matrix in the (x, y) dimensions every 10 steps.

Complex Trajectory We compare the results of state estimator on the complex trajectory. Particularly, we demonstrate the performance on a noiseless and noisy (with regular and 3x noise) complex trajectory when the initial state is and is not known.

Figures 37, 38, and 39 show the smoothed euclidean distance for the complex trajectory with a known initial state with no noise, regular system noise, and 3x system noise.

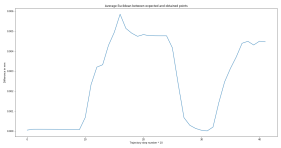

Fig. 37. Clean complex traj. smooth distance

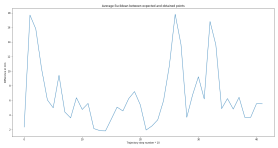

Fig. 38. Complex traj smooth distance

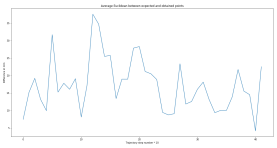

Fig. 39. 3x Noise complex traj. smooth distance

Figures 40 and 41 show the smoothed euclidean distance for the complex trajectory with an unknown initial state with no noise and regular system noise.

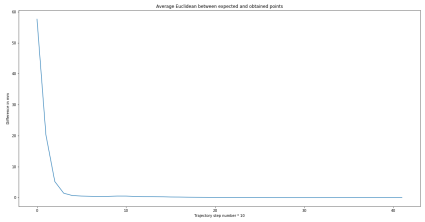

Fig. 40. Clean complex traj. smooth distance, unknown initial state

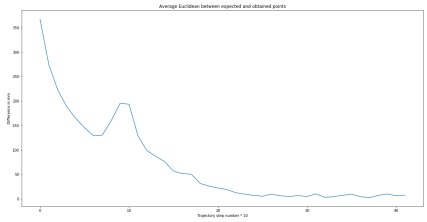

Fig. 41. Noisy complex traj. smooth distance, unknown initial state

Figures 42, 43, and 44 show the difference between theta for the complex trajectory with a known initial state with no noise, regular system noise, and 3x system noise.

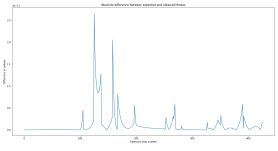

Fig. 42. Clean complex trajectory theta diff

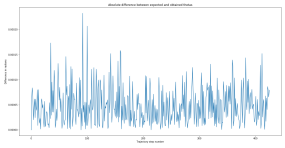

Fig. 43. Complex trajectory theta diff

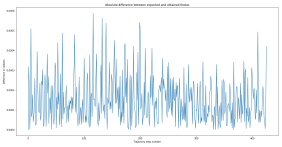

Fig. 44. 3x Noise complex trajectory theta diff

Figures 45 and 46 show the difference in theta for the complex trajectory with an unknown initial state with no noise and regular system noise.

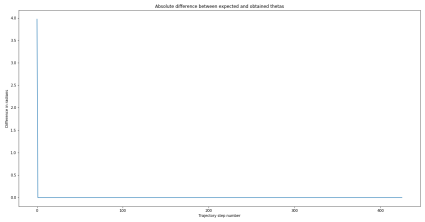

Fig. 45. Clean complex trajectory theta difference, unknown initial state

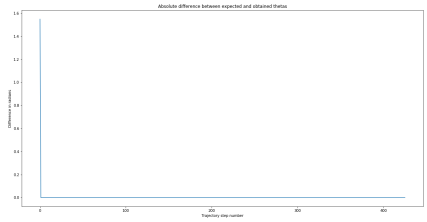

Fig. 46. Noisy complex trajectory theta difference, unknown initial state

Figures 47, 48, and 49 show the difference between omega for the complex trajectory with a known initial state with no noise, regular system noise, and 3x system noise.

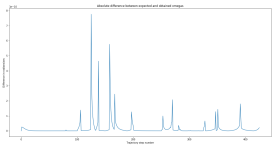

Fig. 47. Clean complex traj. omega diff

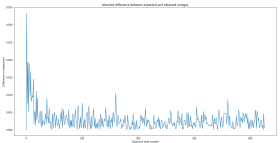

Fig. 48. Complex traj omega diff

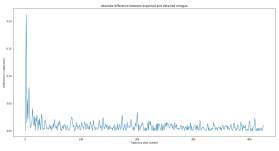

Fig. 49. 3x Noise complex traj. omega diff

Figures 45 and 46 show the difference in omega for the complex trajectory with an unknown initial state with no noise and regular system noise.

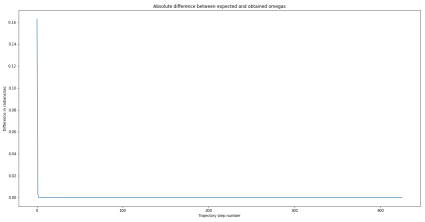

Fig. 50. Clean complex traj. omega difference, unknown initial state

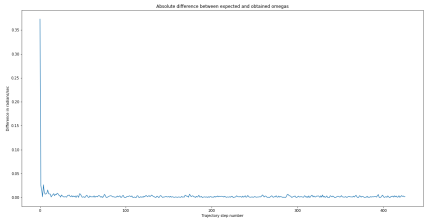

Fig. 51. Noisy complex traj. omega difference, unknown initial state

Figures 52, 53, and 54 show the covariance sizes for the complex trajectory with a known initial state with no noise, regular system noise, and 3x system noise.

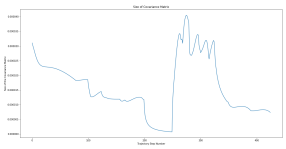

Fig. 52. Clean complex trajectory covariance size

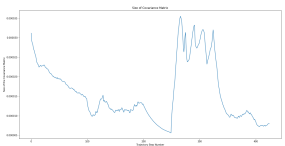

Fig. 53. Complex trajectory covariance size

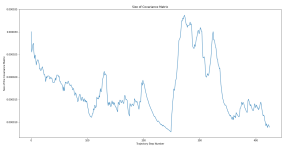

Fig. 54. 3x Noise complex trajectory covariance size

Figures 55 and 56 show the covariance sizes for the complex trajectory with an unknown initial state with no noise and regular system noise.

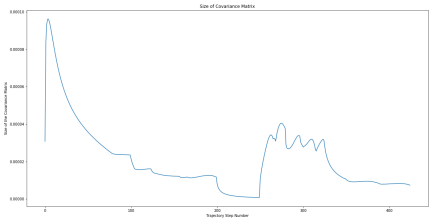

Fig. 55. Clean complex trajectory covariance size, unknown initial state

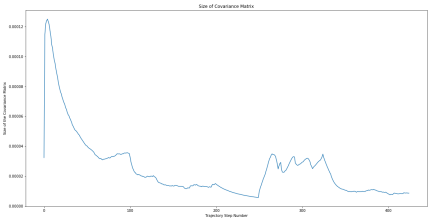

Fig. 56. Noisy complex trajectory covariance size, unknown initial state

Figures 57, 58, and 59 show the covariance ellipses for the complex trajectory with a known initial state with no noise, regular system noise, and 3x system noise.

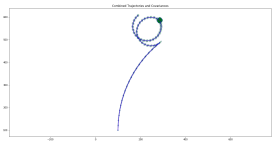

Fig. 57. Clean complex trajectory covariance ellipse

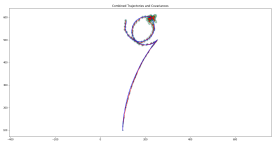

Fig. 58. Complex trajectory covariance ellipse

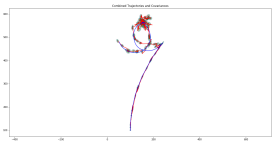

Fig. 59. 3x Noise complex trajectory covariance ellipse

Figures 60 and 61 show the covariance ellipses for the complex trajectory with an unknown initial state with no noise and regular system noise.

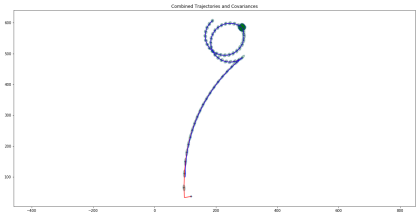

Fig. 60. Clean complex trajectory covariance ellipse, unknown initial state

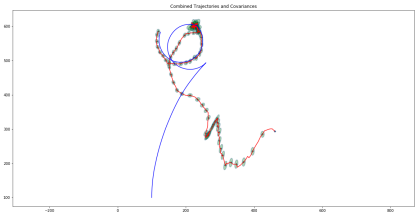

Fig. 61. Noisy complex trajectory covariance ellipses, unknown initial state

Table 1 contains the mean distance between points, RMSE of θ, and RMSE of ω for simulations of the complex trajectory with and without noise and a known and unknown initial state.

Trajectory	Mean Dist (mm)	θ RMSE (rad)	ω RMSE (rad/s)
No noise, Init pos. known	0.0025	2.902e-12	6.212e-11
Reg noise, Init pos. known	6.7034	6.056e-05	0.0041
3x noise, Init pos. known	17.136	0.00019	0.0164
No noise, Init pos. unknown	2.110	0.1925	0.00789
Reg noise, Init pos. unknown	94.064	0.0316	0.0552

Table 1. Mean Distance, θ RMSE, and ω RMSE for the complex trajectory with known and unknown initial states

We observe that the EKF performs the best on the complex trajectory without noise and a known initial state, particularly in terms of the accuracy of its state estimation. As the noise increases, the error of the EKF increases, but it is still reasonably accurate. In particular, the θ and ω errors decrease quickly and stay low throughout the remainder of the trajectories. However, the (x, y) estimate suffers when the robot spins in place. This indicates that our robot model is not fully utilizing measurements from the gyro, which indicate the amount that it's spinning (and consequently not changing (x, y) position). This error can be remedied by more effectively incorporating the gyroscope readings into our EKF and is a good source of future work.

The accuracy of the robot with an unknown initial state significantly improves over the course of iterations, even when it spins in place. The error is initially very high, but as the robot moves, it gets more information and consequently approaches the accuracy of when the initial state is known.

The size of the covariance matrix initially decreases in all trajectories as the robot moves. However, as before, the covariance increases again when the robot spins, indicating that the EKF isn't perfect. In general, however, the covariance decreases over iterations, indicates the accuracy and efficiency of our EKF.

Circular Trajectory Figures 62 and 63 show the smoothed euclidean distance for the circular trajectory with a known initial state with no noise and regular system noise.

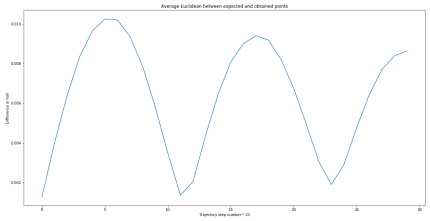

Fig. 62. Clean circular trajectory smooth distance, known initial state

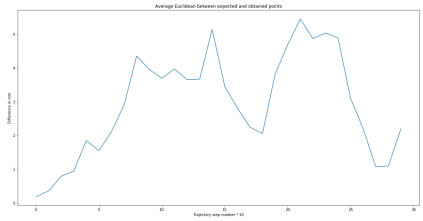

Fig. 63. Noisy circular trajectory smooth distance, known initial state

Figures 64 and 65 show the difference in theta for the circular trajectory with a known initial state with no noise and regular system noise.

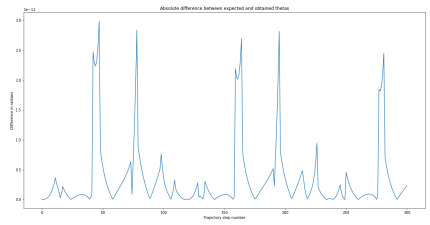

Fig. 64. Clean circular trajectory theta difference, known initial state

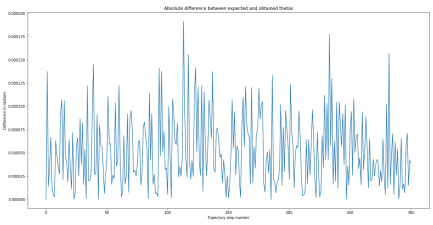

Fig. 65. Noisy circular trajectory theta difference, known initial state

Figures 64 and 65 show the difference in omega for the circular trajectory with a known initial state with no noise and regular system noise.

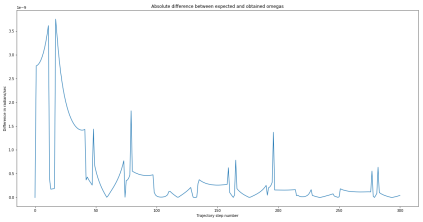

Fig. 66. Clean circular trajectory omega difference, known initial state

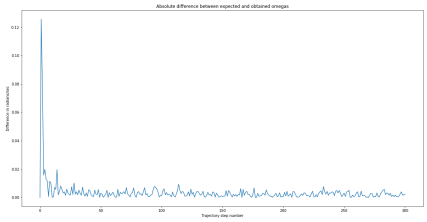

Fig. 67. Noisy circular trajectory omega difference, known initial state

Figures 68 and 69 show the covariance sizes for the circular trajectory with a known initial state with no noise and regular system noise.

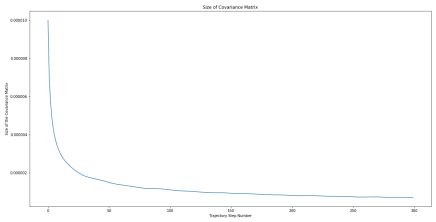

Fig. 68. Clean circular trajectory covariance size, known initial state

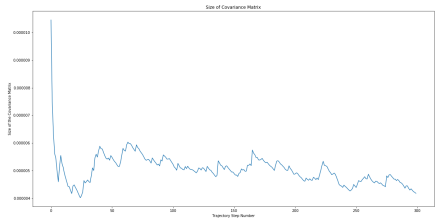

Fig. 69. Noisy circular trajectory covariance size, known initial state

Figures 70 and 71 show the circular ellipses for the complex trajectory with a known initial state with no noise and regular system noise.

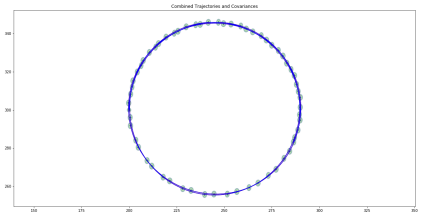

Fig. 70. Clean circular trajectory covariance ellipse, known initial state

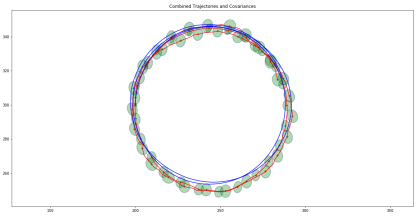

Fig. 71. Noisy circular trajectory covariance ellipses, known initial state

Table 2 contains the mean distance between points, RMSE of θ, and RMSE of ω for simulations of the circular trajectory with and without noise and a known initial state.

Trajectory	Mean Dist (mm)	θ RMSE (rad)	ω RMSE (rad/s)
No noise, Init pos. known	0.00634	6.6982e-12	8.8775e-10
Reg noise, Init pos. known	2.5297	6.1419e-05	0.0125

Table 2. Mean Distance, θ RMSE, and ω RMSE for the circular trajectory with known initial state

The accuracy of the estimation of (x, y) as well as θ remains relatively constant in trajectories with and without noise. However, the ω estimation gets significantly better in both simulations. This makes sense since the robot has a constant ω throughout the course of the trajectory. In both simulations, the size of the covariance matrix converges extremely quickly, indicating that our EKF does well on trajectories that have a constant ω. This matches our previous conclusion that the gyro measurement could be better utilized in future work to improve the (x, y) and θ estimation. As is expected, the EKF has a lower error on the noiseless system.

Pointed Trajectory Figures 72 and 73 show the smoothed euclidean distance for the pointed trajectory with a known initial state with no noise and regular system noise.

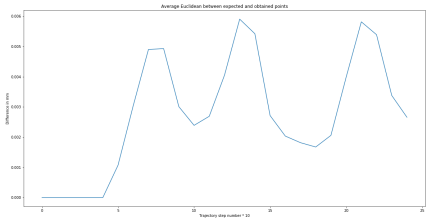

Fig. 72. Clean pointed trajectory smooth distance, known initial state

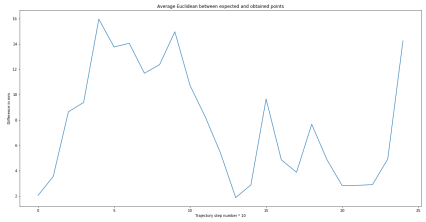

Fig. 73. Noisy pointed trajectory smooth distance, known initial state

Figures 74 and 75 show the difference in theta for the pointed trajectory with a known initial state with no noise and regular system noise.

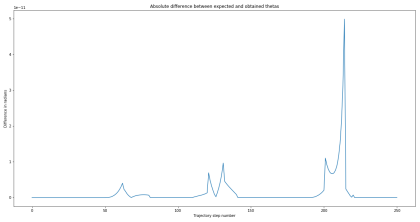

Fig. 74. Clean pointed trajectory theta difference, known initial state

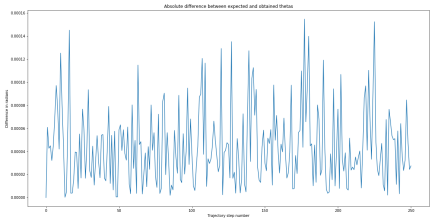

Fig. 75. Noisy pointed trajectory theta difference, known initial state

Figures 74 and 75 show the difference in omega for the pointed trajectory with a known initial state with no noise and regular system noise.

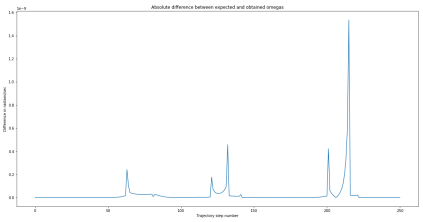

Fig. 76. Clean pointed trajectory omega difference, known initial state

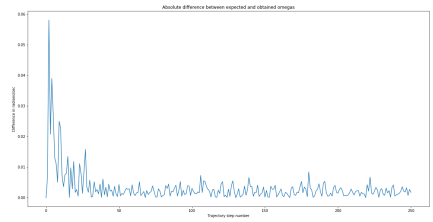

Fig. 77. Noisy pointed trajectory omega difference, known initial state

Figures 78 and 79 show the covariance sizes for the pointed trajectory with a known initial state with no noise and regular system noise.

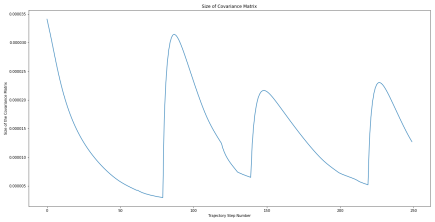

Fig. 78. Clean pointed trajectory covariance size, known initial state

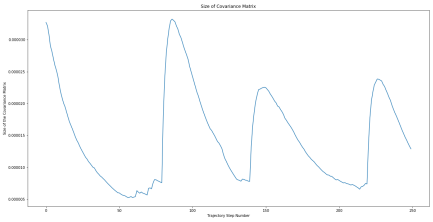

Fig. 79. Noisy pointed trajectory covariance size, known initial state

Figures 80 and 81 show the pointed ellipses for the complex trajectory with a known initial state with no noise and regular system noise.

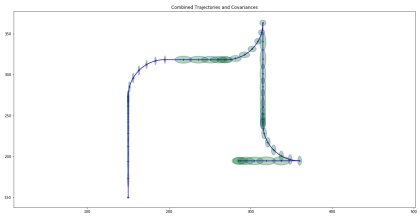

Fig. 80. Clean pointed trajectory covariance ellipse, known initial state

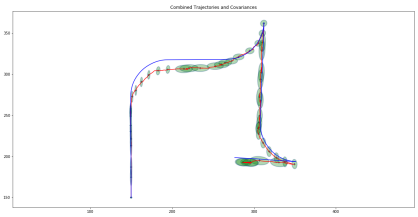

Fig. 81. Noisy pointed trajectory covariance ellipses, known initial state

Table 3 contains the mean distance between points, RMSE of θ, and RMSE of ω for simulations of the pointed trajectory with and without noise and a known initial state.

Trajectory	Mean Dist (mm)	θ RMSE (rad)	ω RMSE (rad/s)
No noise, Init pos. known	0.0027	4.5933e-12	1.1708e-10
Reg noise, Init pos. known	6.3837	6.1803e-05	0.0090

Table 3. Mean Distance, θ RMSE, and ω RMSE for the pointed trajectory with known initial state

The accuracy of the noiseless and noisy system is largely determined by whether the robot is spinning in place (when it turns the corner). When the robot has a straight trajectory, all errors decrease and the covariance converges. However, each time the robot turns, the errors spike and the size of the covariance matrix dramatically increases. This re-affirms that our robot struggles to fully incorporate knowledge that it's turning into its state estimation. As is expected, the EKF has a lower error on the noiseless system.

Spiral Trajectory Figures 82 and 83 show the smoothed euclidean distance for the spiral trajectory with a known initial state with no noise and regular system noise.

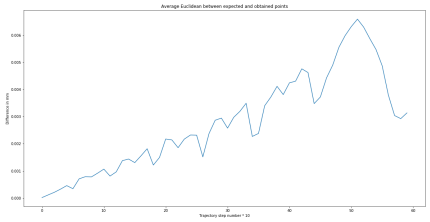

Fig. 82. Clean spiral trajectory smooth distance, known initial state

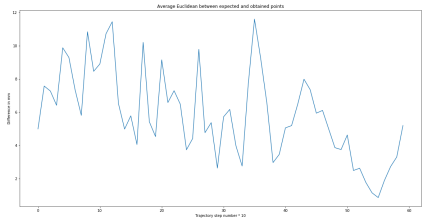

Fig. 83. Noisy spiral trajectory smooth distance, known initial state

Figures 84 and 85 show the difference in theta for the spiral trajectory with a known initial state with no noise and regular system noise.

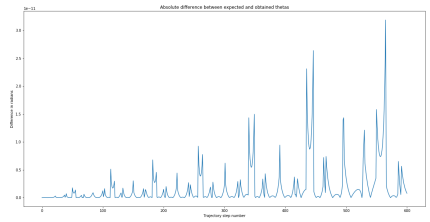

Fig. 84. Clean spiral trajectory theta difference, known initial state

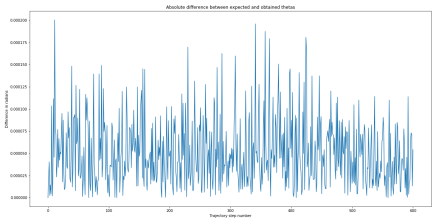

Fig. 85. Noisy spiral trajectory theta difference, known initial state

Figures 84 and 85 show the difference in omega for the spiral trajectory with a known initial state with no noise and regular system noise.

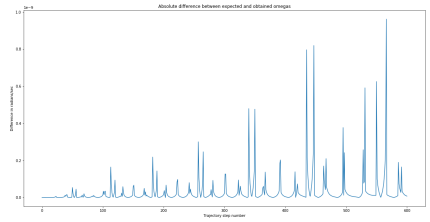

Fig. 86. Clean spiral trajectory omega difference, known initial state

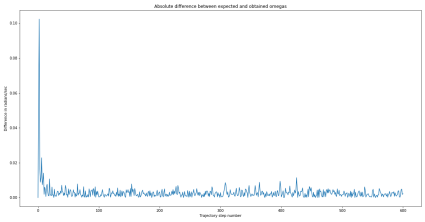

Fig. 87. Noisy spiral trajectory omega difference, known initial state

Figures 88 and 89 show the covariance sizes for the spiral trajectory with a known initial state with no noise and regular system noise.

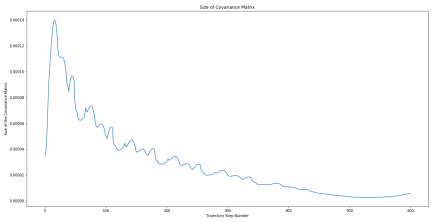

Fig. 88. Clean spiral trajectory covariance size, known initial state

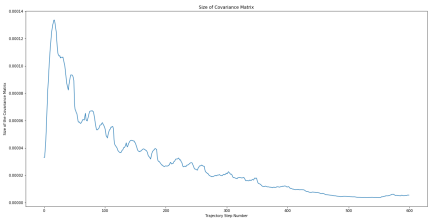

Fig. 89. Noisy spiral trajectory covariance size, known initial state

Figures 90 and 91 show the spiral ellipses for the complex trajectory with a known initial state with no noise and regular system noise.

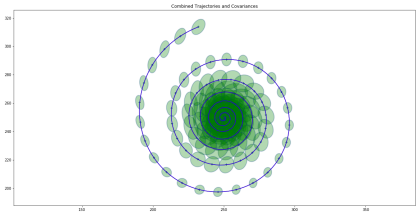

Fig. 90. Clean spiral trajectory covariance ellipse, known initial state

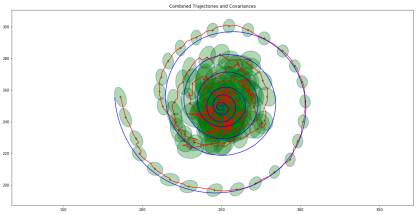

Fig. 91. Noisy spiral trajectory covariance ellipses, known initial state

Table 4 contains the mean distance between points, RMSE of θ, and RMSE of ω for simulations of the spiral trajectory with and without noise and a known initial state.

Trajectory	Mean Dist (mm)	θ RMSE (rad)	ω RMSE (rad/s)
No noise, Init pos. known	0.0028	3.931e-12	8.9924e-11
Reg noise, Init pos. known	6.3429	6.190e-05	0.0078

Table 4. Mean Distance, θ RMSE, and ω RMSE for the spiral trajectory with known initial state

The accuracy of the state estimation gets worse over time for the system without noise, but remains constant/slightly decreases for the trajectory with noise. The spiral trajectory is

more complex than the other trajectories, since (x, y), θ, and ω are changing every step. This ordinarily poses a challenge to the EKF since it uses linearization. However, the presence of noise in the second trajectory effectively "masks" out the error due to linearization. In both cases the covariance converges, indicating that the EKF does a good job estimating the state. As is expected, the EKF has a lower error on the noiseless system.

3(d) Improving System Models

More realistic system models can be incorporated by tracking an estimate of the new model aspects in the Kalman Filter. For example, sensor bias can be accommodated by estimating an offset term for each sensor with bias. Realistic sensor models would also factor in random sensor walk. This is a particular type of noise which causes the sensor to drift away from its initialization reference over time. This drift can be estimated by the Kalman Filter by including it in the state of the system. Modeling actuator uncertainty given a particular speed could be account for by adapting the W matrix to assert this relationship.

The sensor input can be further improved by adding filtered versions of the sensor data as extra observations to the Kalman Filter. A low frequency filter tuned for a noisy sensor can, for example, give the Kalman Filter nonlinear sensor fitting information that it would not have been able to perform previously.

A more interesting example is modeling wheel skid. Such a model would require a much more realistic system model that takes into account the friction between the wheels and surface, mass of the wheels, centroid of the robot, and instantaneous angular acceleration. This model is useful because it would enable the robot to interact in a variety of different environments, for example when the robot is placed onto a slipper surface. The Kalman filter would estimate the friction coefficients and instantaneous angular acceleration.

3(e) Conclusions

The task of state estimation defines the problem of determining the best estimate of a current state given a time evolution sequence and an observation (measurement) model in the presence of process and measurement noise. State estimation is a crucial part of many applications, including autonomous vehicle navigation, SLAM, and computer vision. Several methods exist to formulate and solve the problem of state estimation, including Kalman Filtering, Extended Kalman Filtering, and Unscented Kalman Filtering. In the following sections, we will discuss the performance of the state estimator that we implemented and compare it to other state estimation methods.

Effectiveness of Our EKF In this lab, we primarily focused on the implementation of an Extended Kalman Filter (EKF) to localize our robot. The EKF, unlike the regular KF, can handle non-linear stochastic difference equations for a system model, which greatly increases it's applicability to real-world systems. The general non-linear system model is given as

$$x_{t+1} = f(x_t, u_t, w_t) \tag{46}$$

$$y_t = h(x_t, v_t). \tag{47}$$

The EKF relies on the linearization around a point, which involves calculating the Jacobian at each step. Although calculating the Jacobian was feasible in our implementation, it can be quite complicated (or impossible) for complex system dynamics. Additionally,

linearization starts to breakdown for nonlinear processes since it assumes a first order approximation, which leads to inaccuracies. The EKF can also suffer from an improperly chosen initial state–in this case, the system can diverge due to the linearization. Kalman Filters make the assumption that the data points are from a Gaussian distribution. In general, this is an appropriate approximation, since sufficiently large samples approach a normal distribution according to the Central Limit Theorem. However, linearization can also detrimentally impact the EKF's estimation of the mean and covariance, leading to errors.

As discussed in section 3(c), our EKF generally does a good job performing state estimation for our various trajectories. The accuracy is always lower when the noise increases, but the error typically remains relatively constant. Furthermore, our EKF can successfully localize the robot when the initial state is unknown. Our EKF and system model is the most accurate and most efficient when the robot is a following a straight trajectory. As the angular velocity increases, however, the state estimation accuracy decreases. This is largely due to a sub-optimal incorporation of the gyroscope measurements into our system dynamics and EKF. Future work can work to more precisely integrate the gyroscope measurements.

Other State Estimators In many situations, the EKF is not the best state estimator. For systems with simple dynamics, such as linear systems with white noise and known covariance, the Kalman Filter works well. However, the Kalman filter does not handle nonlinear systems.

For systems with highly nonlinear dynamics, the Unscented Kalman Filter (UKF) is frequently a better state estimator. Instead of employing linearization, the UKF uses sampling (called the unscented transformation) to pick a set of points around the mean. This set of points is then propagated through the system dynamic functions to create new mean and covariance. This process allows the UKF to more effectively approximate the mean and covariance of the system and improve the accuracy of the state estimation.

Future work As we mentioned while analyzing our results, the gyroscope is presently being under-utilized. We have been able to develop a sensor fusion model capable of providing better results than what we presented, however, we were unable to fully characterize the new system due to time restrictions.

The new system updates the robot's position knowledge based on the gyroscope sensor. Because of this, the robot can now rely on both the laser range sensors and gyroscope to get an accurate estimate of its position. In order to do this, the H matrix is updated to account for the new dependencies.

$$H = \begin{bmatrix} \frac{d\,straight}{dx} & \frac{d\,straight}{dy} & \frac{d\,straight}{d\theta} & 0 \\ \frac{d\,right}{dx} & \frac{d\,right}{dy} & \frac{d\,right}{d\theta} & 0 \\ 0 & 0 & 1 & 0 \\ \frac{dx}{d\omega} & \frac{dy}{d\omega} & 0 & 1 \end{bmatrix} \tag{48}$$

$$\frac{dx}{d\omega} = \frac{l}{2}(\frac{1}{2}\cos(\theta_{avg}) + \frac{1}{2l}v\Delta_t \sin(\theta_{avg})) \tag{49}$$

$$\frac{dy}{d\omega} = \frac{l}{2}(\frac{1}{2}\sin(\theta_{avg}) - \frac{1}{2l}v\Delta_t \cos(\theta_{avg})) \tag{50}$$

We estimate v by measuring the velocity between the robot's current and predicted future state. By doing so, a very noisy trajectory now becomes the trajectory shown in

Figure 92. But this new filter trusts the gyro sensor too much, and fails when faced with an unknown starting state. This is evident by Figure 93.

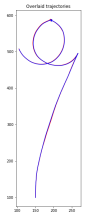

Fig. 92. New Kalman filter trajectory, known start

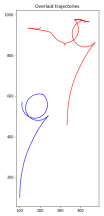

Fig. 93. New Kalman filter trajectory, unknown start

Regardless, the covariance matrix decreases much faster and smoother as shown in the Figure below. Because of this, we believe this is a beneficial direction to proceed in the future.

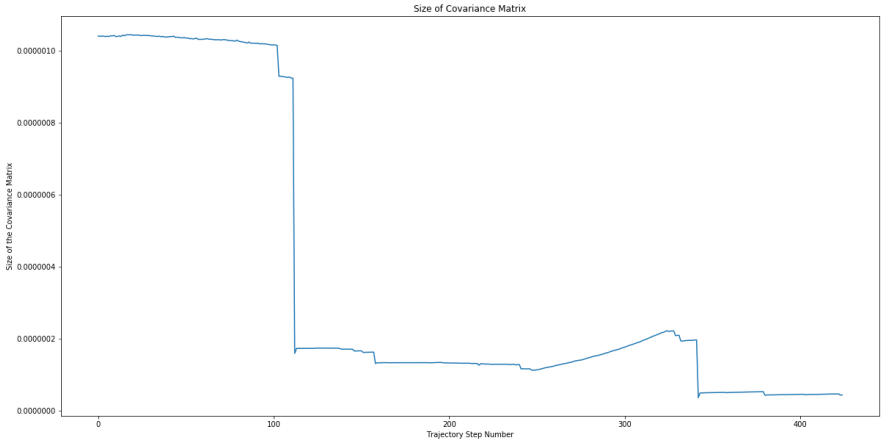

Fig. 94. Size of covariance matrix, known start

Bibliography

[1] InvenSense Inc., *Nine-Axis (Gyro + Accelerometer + Compass) MEMS MotionTracking Device*, 6 2016. Rev. 1.1.

[2] STMicroelectronics, *World's smallest Time-of-Flight ranging and gesture detection sensor*, 4 2018. Rev. 2.

[3] Dudek and Jenkin, "Cs w4733 notes - differential drive robots," 2013.

[4] G. Li, D. Qin, and H. Ju, "Localization of wheeled mobile robot based on extended kalman filtering," in *MATEC Web of Conferences*, vol. 22, p. 01061, EDP Sciences, 2015.

[5] M. Giraldo, F. Valencia, J. Espinosa, and J. D. López, "Evaluation and comparison of kalman-filter-based distributed state estimators for large-scale systems," *IFAC Proceedings Volumes*, vol. 46, no. 13, pp. 188–193, 2013.

[6] A. J. Laub, *Matrix analysis for scientists and engineers*, vol. 91. Siam, 2005.

[7] Wikipedia contributors, "Kalman filter — Wikipedia, the free encyclopedia," 2019. [Online; accessed 20-November-2019].

[8] A. Metha, "Ee209as computational robotics," 2019.

[9] R. I. Greenberg and J. M. Karp, "Motion planning for simple two-wheeled robots," in *Proceedings of the 2017 Global Conference on Educational Robotics (GCER)*, 2017.