Robotic Science and Systems - Exercise 4

Kevin Robb

I completed E0-E4 in the previous assignment, ex3. I have included the same part of the written section for these for ease of grading, but if they were already covered in the grading of ex3, click here to skip to E5, the first new problem for this assignment.

E0 - EKF Using Robotics Toolkit

Implementing the three EKFs in this section using the code in the textbook (slightly modified) yields the figures as expected. The independent localization and mapping EKFs each perform very well, but the SLAM case fails horribly when using our passed parameters instead of the defaults. Several figures were produced when running E0, and two are shown in the figure below. The failed SLAM will be shown in comparison to my implementation in part E3, in Figure 4. I'm fairly confident that my implementation is not causing it to fail unfairly, since the exact same code that produces a scrambled mess for a range of 4 is perfectly capable of performing with a range of 8, as we see in part E4.

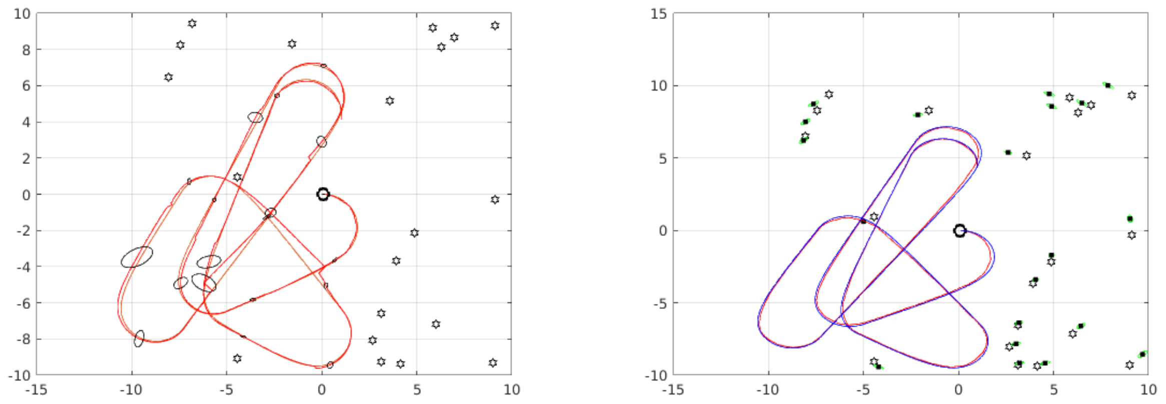


Figure 1: Some plots resulting from running E0.

E1 - EKF for Localization

We assume odometry is our control commands for distance and heading. Measurements involve a range and bearing to a certain landmark (known ID) with known global position since we have the map. We use the EKF to track the vehicle position and heading as the state.

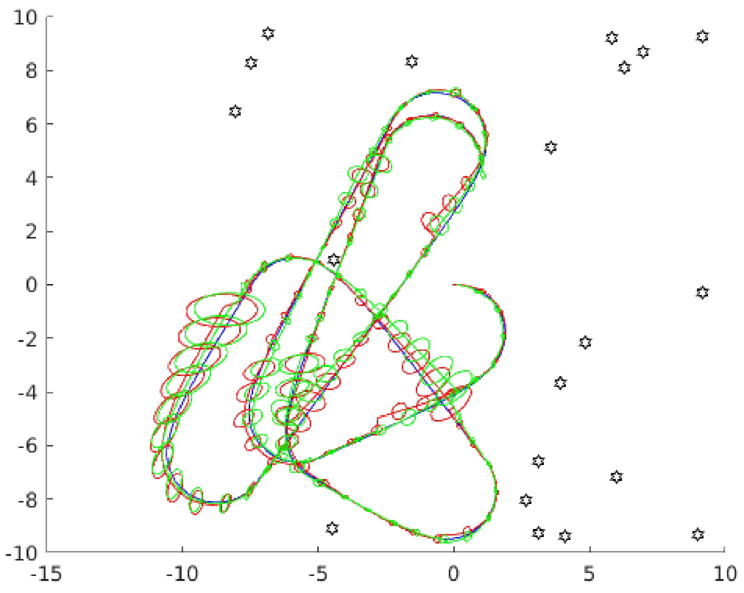


Figure 2: My EKF (red) using landmark detections for localization. Matches the true state (blue) and the Robotics Toolbox EKF implementation (green) very well.

E2 - EKF for Mapping

We assume odometry is exactly correct, so at all timesteps we have our true vehicle position. The map is not known, so using measurements of range and bearing to a landmark with known ID, our state is a list of the x,y positions of all seen landmarks.

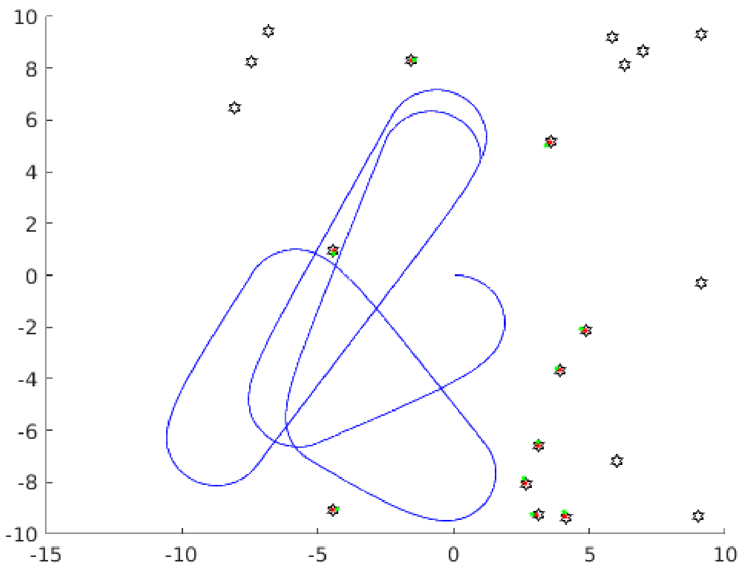


Figure 3: My EKF (red) using landmark detections for mapping. Matches the true map (black stars) and the Robotics Toolbox EKF implementation (green) very well.

E3 - EKF for SLAM

Now we don't have the true robot position or the true map, so we must track both the robot state (x,y,yaw) and the (x,y) position of all landmarks in our state. Here we will derive all Jacobians needed to perform EKF SLAM.

Our state will look like $\hat{x} = (x_v, y_v, \theta_v, x_1, y_1, ..., x_M, y_M)^T \in \mathbb{R}^{(2M+3)x1}$ for $M \in \mathbb{N}$ landmarks. Our covariance will look like

$$\hat{P} = \begin{bmatrix} \hat{P}_{vv} & \hat{P}_{vm} \\ \hat{P}_{vm}^T & \hat{P}_{mm} \end{bmatrix} \in \mathbb{R}^{(2M+3)\times(2M+3)}$$

where $\hat{P}_{vv}$ is the vehicle covariance, $\hat{P}_{mm}$ is the map covariance, and $\hat{P}_{vm}$ is the vehicle/map covariance.

First, the transition function. We assume landmark positions and covariance remain constant, while the robot position changes as it moves via the odometry commands $(\delta_d, \delta_\theta)^T$.

$$\hat{x}_{t+1} = f(\hat{x}_t, u_t) = \left[\begin{array}{c} x_{v,t} + (\delta_d + v_d)\cos\theta_{v,t} \\ y_{v,t} + (\delta_d + v_d)\sin\theta_{v,t} \\ \theta_{v,t} + \delta_\theta + v_\theta \\ \hline x_{1,t} \\ y_{1,t} \\ ... \\ x_{M,t} \\ y_{M,t} \end{array}\right]$$

Then the transition function Jacobian is:

$$F_x = \begin{bmatrix} F_{x,v} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & I_{2\times2} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & ... & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & I_{2\times2} \end{bmatrix},$$

where the submatrix for the vehicle position is

$$F_{x,v} = \begin{bmatrix} 1 & 0 & -\delta_d \sin\theta_v \\ 0 & 1 & \delta_d \cos\theta_v \\ 0 & 0 & 1 \end{bmatrix},$$

and the number of identity matrices on the diagonal equals M, the number of landmarks in the state. All zero matrices shown are the dimensions needed to fill out F_x. The transition noise Jacobian is

$$F_v = \begin{bmatrix} F_{v,v} \\ \mathbf{0}_{2\times2} \\ ... \\ \mathbf{0}_{2\times2} \end{bmatrix},$$

where the submatrix is

$$F_{v,v} = \begin{bmatrix} \cos\theta_v & 0 \\ \sin\theta_v & 0 \\ 0 & 1 \end{bmatrix},$$

and the number of zero matrices in F_v equals M, the number of landmarks in the state. We know that for the covariance prediction equation to work out, the noise matrix V must be 2×2.

$$\hat{P}^+_{t+1} = F_x \hat{P}_t F_x^T + F_v V F_v^T$$

Next, our observation function is

$$\hat{z} = h(x, p_i) = \begin{bmatrix} r + w_r \\ \arctan \frac{y_i - y_v}{x_i - x_v} - \theta_v + w_\beta \end{bmatrix},$$

where $r = \sqrt{(y_i - y_v)^2 + (x_i - x_v)^2}$. This only depends on the position of the newly measured landmark and the vehicle position. It is used for calculation of the innovation,

$$\nu = z^\# - \hat{z}$$

The observation Jacobians are then:

$$H_x = \begin{bmatrix} H_{x,v} & \mathbf{0} & \ldots & H_{x,p_i} & \ldots & \mathbf{0} \end{bmatrix}$$

where the submatrices for the vehicle and relevant landmark positions are

$$H_{x,v} = \begin{bmatrix} -\frac{x_i - x_v}{r} & -\frac{y_i - y_v}{r} & 0 \\ \frac{y_i - y_v}{r^2} & -\frac{x_i - x_v}{r^2} & -1 \end{bmatrix}$$
$$H_{x,p_i} = \begin{bmatrix} \frac{x_i - x_v}{r} & \frac{y_i - y_v}{r} \\ -\frac{y_i - y_v}{r^2} & \frac{x_i - x_v}{r^2} \end{bmatrix}$$

and the observation noise Jacobian is

$$H_w = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

For the Kalman gain equation to work out, the noise matrix W must be 2×2.

$$K = \hat{P}^+_{t+1} H_x^T \left(H_x \hat{P}^+_{t+1} H_x^T + H_w W H_w^T \right)^{-1}$$

We can see the portion being inverted will be 2×2, and the Kalman gain will be $(3 + 2M) \times 2$. This makes sense for our state update equations, since the innovation ν is 2×1.

$$\hat{x}_{t+1} = \hat{x}^+_{t+1} + K\nu$$
$$\hat{P}_{t+1} = \hat{P}^+_{t+1} - K H_x \hat{P}^+_{t+1}$$

Lastly, the insertion function:

$$g(x_v, z) = \begin{bmatrix} x_v + r\cos(\theta_v + \beta) \\ y_v + r\sin(\theta_v + \beta) \end{bmatrix}$$

$$\hat{x}_{t+1} = \begin{bmatrix} \hat{x}_t \\ g(x_v, z)_t \end{bmatrix}$$

So our insertion Jacobian is:

$$Y_z = \begin{bmatrix} I_{n\times n} & \mathbf{0}_{n\times 2} \\ \mathbf{0}_{2\times n} & G_z \end{bmatrix},$$

where $n = 3 + 2M$ (the state size) and the submatrix is:

$$G_z = \begin{bmatrix} \cos(\theta_v + \beta) & -r\sin(\theta_v + \beta) \\ \sin(\theta_v + \beta) & r\cos(\theta_v + \beta) \end{bmatrix}$$

These dimensions make sense when we look at our covariance update equation, where each term is a $(n + 2) \times (n + 2)$ matrix.

$$\hat{P}_{t+1} = Y_z \begin{bmatrix} \hat{P}_t & 0 \\ 0 & W \end{bmatrix} Y_z^T$$

We can then use our code from E1 and E2, as well as these derived Jacobians, to implement EKF SLAM in MATLAB. The results of this are shown in the following figure.

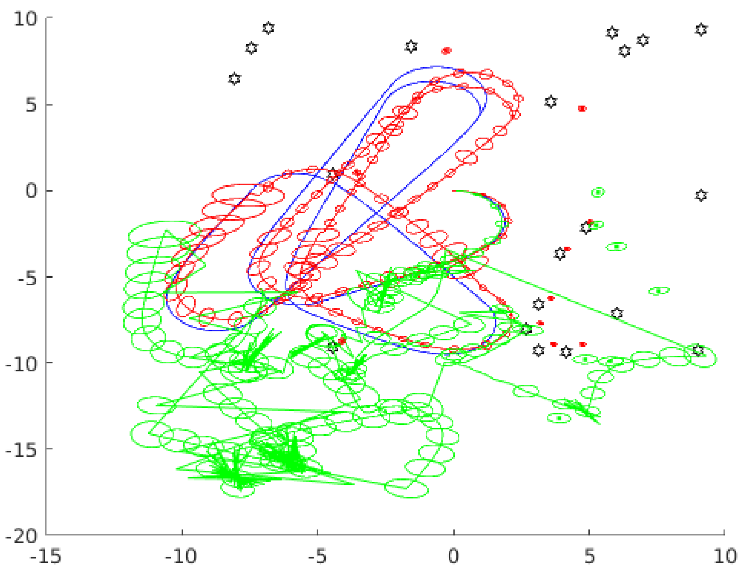


Figure 4: My EKF (red) using landmark detections for both localization and mapping. Matches the true pose and map (blue and black) fairly well. The Robotics Toolbox EKF implementation (green) fails dramatically.

E4 - SLAM with Increased Sensor Range

There are sections in the data collection during which no landmarks can be detected, meaning during these segments the uncertainty only grows. We hypothesize that increasing the sensor's maximum range will improve its performance by reducing the amount of time for which there is no detection. We reuse our code from E0 to implement the toolbox's EKF SLAM, doubling the sensor range from 4 to 8.

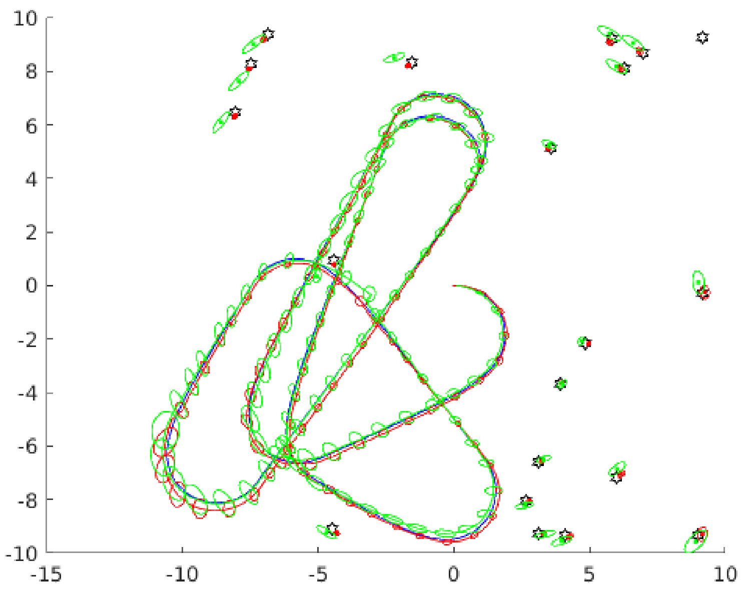


Figure 5: My EKF (red) using landmark detections for both localization and mapping, compared to the Robotics Toolbox implementation (green), both with an increased range of 8. Both match the true map very well and with low uncertainty. I would say my implementation outperforms the toolbox for mapping landmarks, while it does a better job of tracking the vehicle pose.

E5 - Noisy Measurement Generation

Up to this point, our program relies on the pregenerated data files, and only works for one specific trajectory for which we have the data. If we have a way to create new trajectories, and get measurements for odom as well as ID/range/bearing to a landmark for each timestep, our existing code will work for any input. To create these "measurements", we implement the E5 function.

My strategy involves first adding noise to all odom measurements in parallel, since this step is easy and does not depend on previous timesteps or the robot's location. Then, we keep track of the true robot position for each timestep, and compute the vector from the robot to each landmark. We can use this to find the range and relative bearing from the robot to each landmark, and then check if these fall within the provided `Range` and `fov`. Any that do have their ID, range, and bearing saved to a temporary array of visible landmarks. After this step is completed, we check the sensing mode, and for mode 'o', we randomly choose one of these visible landmarks to add to our `z` and `zind` measurement arrays. If no landmarks are visible on a given timestep, we set `zind` to 0, and save an empty array to `z`.

My implementation for this works great with my E3 SLAM implementation, producing a plot like the following. Due to the randomness of adding noise to the measurements, the results can change slightly on each run, but my E3 is still fairly consistent in how well it tracks the path and landmarks.

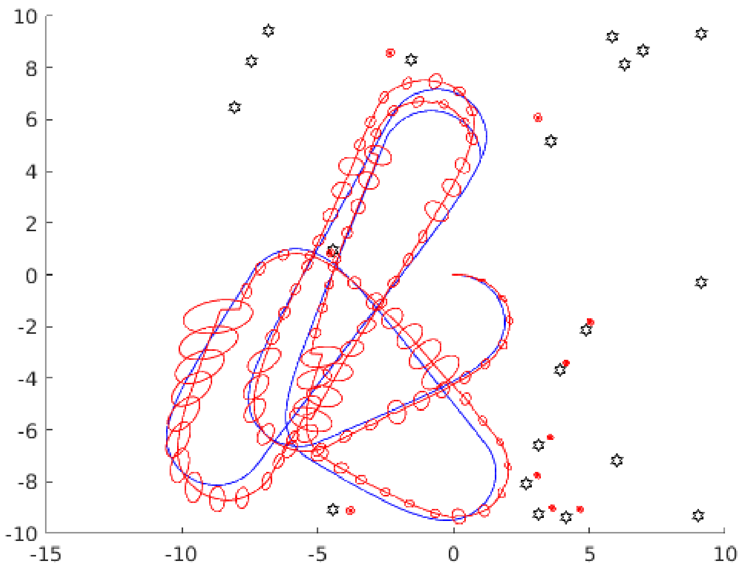


Figure 6: A result of the driver running my E5 to create data that is used by my E3 to perform SLAM. This figure looks a bit different every time it is generated, due to the random noise.

E6 - Trajectory Generation

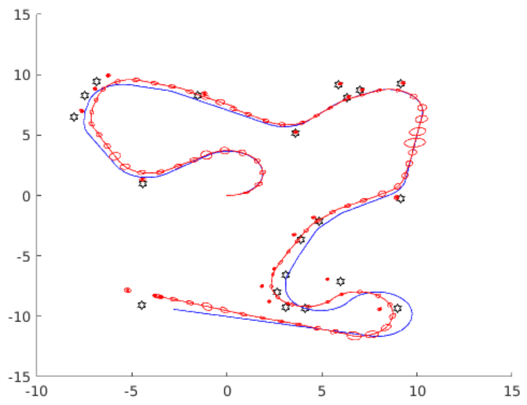
The trajectory that we've been using does not actually go near many of the landmarks, so the map is incomplete and the localization is not as good as it could be. By generating trajectories with the explicit purpose of seeing more landmarks and observing them more frequently, we can improve the performance of SLAM in our E3. This trajectory generation code should be capable, given rough locations of all landmarks, of creating trajectories which achieve the following:

- Estimate all landmarks in the map, possibly with greater error than E3.
- Estimate at least as many landmarks as E3, but with less error than E3.

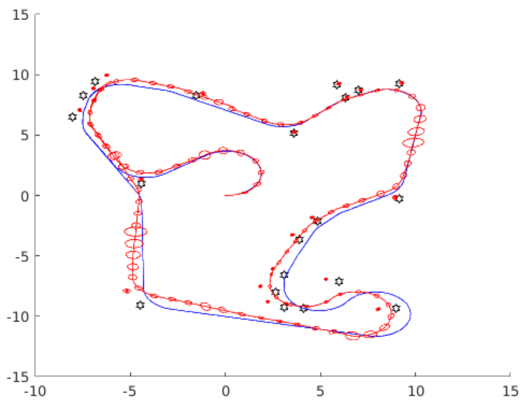
It is unlikely the same trajectory will be able to satisfy both of these expectations. To ensure a trajectory remains within the robot's constraints, any particular odom command should satisfy $\delta_d \leq 0.1$ m and $|\delta_\theta| \leq 0.0546$ radians.

Estimating all landmarks

We ensure all landmarks will be seen on the trajectory by treating the map as a travelling salesman problem (TSP), and create a directed graph using the basic nearest neighbors heuristic approach; this is non-optimal, but is perfectly fine for our purpose, where we only need the robot to pass near landmarks, not precisely through them, and we have plenty of time to do so. We cycle back to the first node in the graph after visiting all landmarks once, and continue until the desired number of timesteps has passed. If the allowable timesteps is too low for the complexity of the map, it is possible that time will run out before all landmarks have been visited, but with 1000 timesteps and this relatively small, compact map, this is not a concern for us. This strategy generates the following trajectory, which we can see is estimated fairly well, though errors do accumulate visibly towards the end of the path.



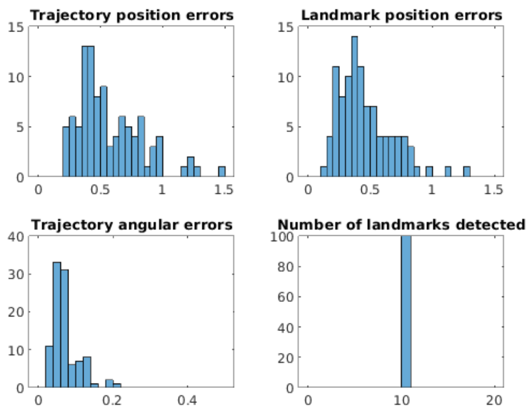
(a) Vehicle stops after visiting all landmarks. (∼ 800 timesteps)



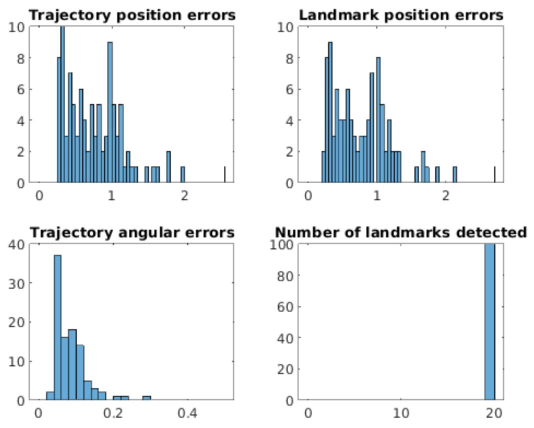
(b) Vehicle loops back to first landmark after visiting all. (1000 timesteps)

Figure 7: Trajectory generated using the TSP approach to ensure all landmarks are visited.

The trajectory generated in this way does map all landmarks, but also leads to a greater SLAM error than the default/provided trajectory. The error histograms for 100 trials of both are shown below.



(a) Default trajectory.



(b) TSP trajectory.

Figure 8: Statistics for my E3 SLAM running on different trajectories.

We can see on these plots that the default gives lower error in both position and orientation. Additionally, the console output tells us the averages (included in the following table), which confirm this. These refer to differences between the relevant statistic estimate and its ground truth.

Statistic	Default Trajectory	TSP Trajectory
Vehicle Position	0.571097	0.769604
Vehicle Orientation	0.072477	0.084432
Landmark Avg	0.461936	0.790367
Landmark Std Dev	0.214315	0.446726
# of Landmarks	10	20

Estimating fewer landmarks with greater accuracy

Since I've already written code that turns a list of waypoint IDs into a trajectory, I can reuse that by manually creating a list of waypoints that will give a more accurate (yet less complete) mapping. We can see that if the robot remains in areas with high landmark density for as long as possible, we'll have more measurements, and SLAM will perform better. I selected 7 landmarks for this path that my trajectory planner will ensure fall near the vehicle's route, and this yields the following figure.

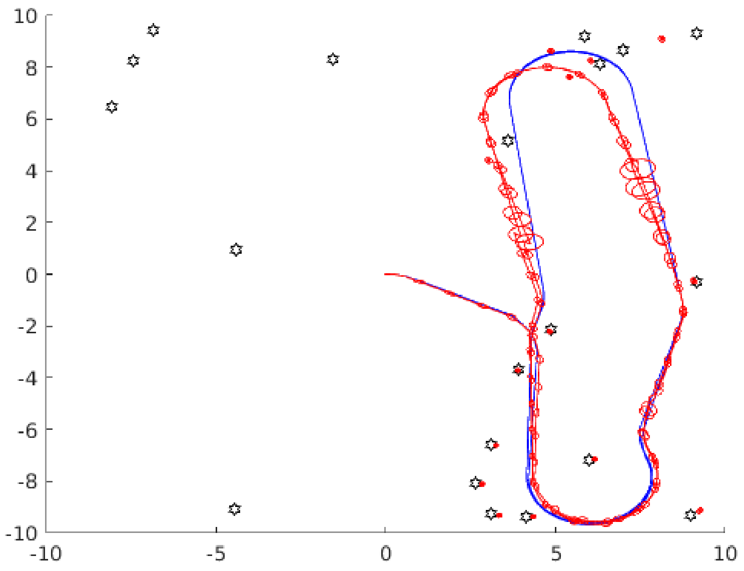
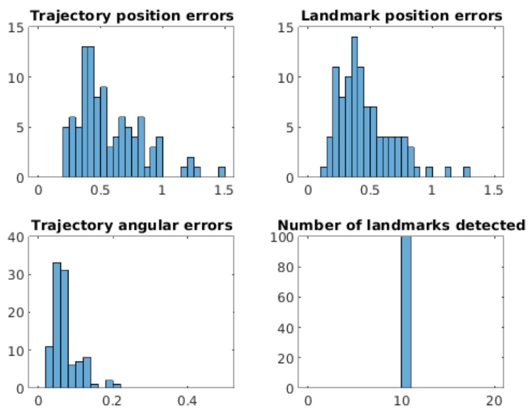
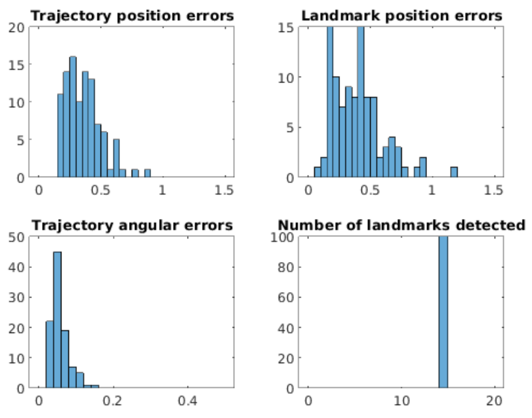


Figure 9: Path based on my manually chosen list of landmarks. Despite looking poor in the upper section, the statistics show that SLAM with this path does outperform the default trajectory.



(a) Default trajectory.



(b) New, manually chosen trajectory.

Figure 10: Statistics for my E3 SLAM running on different trajectories.

Statistic	Default Trajectory	New Trajectory
Vehicle Position	0.571097	0.356525
Vehicle Orientation	0.072477	0.056652
Landmark Avg	0.461936	0.400167
Landmark Std Dev	0.214315	0.201169
# of Landmarks	10	14

We can see this new trajectory outperforms the default in every category in the table, and even captures 4 additional landmarks in its map.

Improving the map for SLAM

The map could be improved to maximize localization effectiveness if the landmarks were arranged in a grid with granularity such that no matter the robot's position and orientation, at least one landmark is visible at all timesteps. This is because error primarily grows during stretches where the robot is driving with no landmarks in sight.

E7 - SLAM with Multi-Landmark Measurements

We now implement multi-landmark measurement via the 'a' mode in E5. This means on each timestep, our EKF has access to all visible landmarks' measurements, rather than only one chosen randomly from this set. Implementing this change in the EKF is extremely simple, as we can run the predict step as normal, then run the update step once for each landmark detected (since landmarks do not depend on one another) before setting the new state and covariance. This change yields minor improvements, but the statistics distributions look very similar to the default 'o' case.

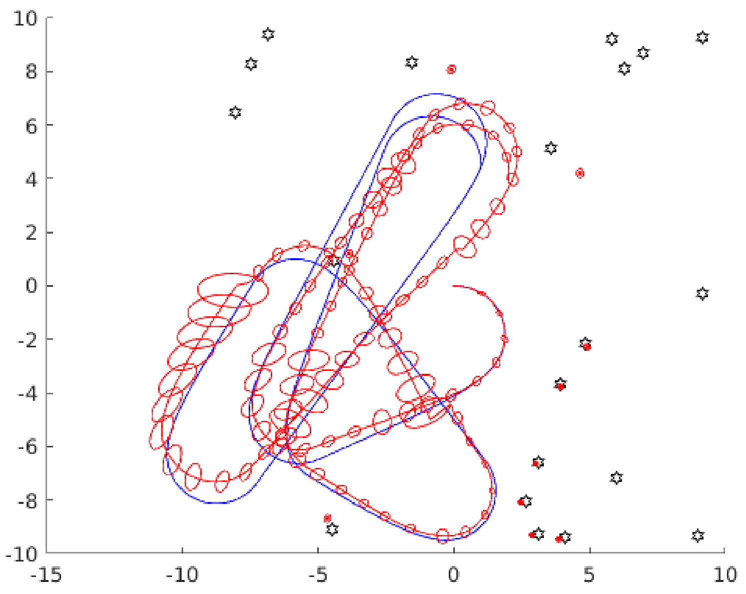
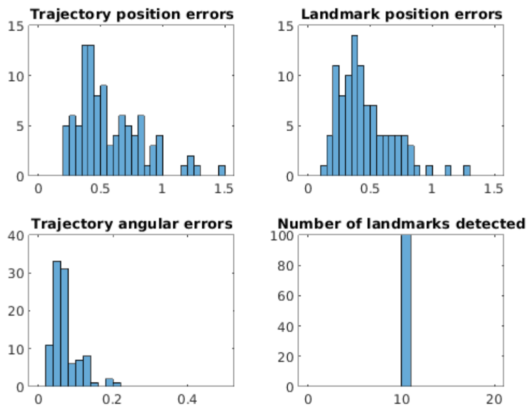
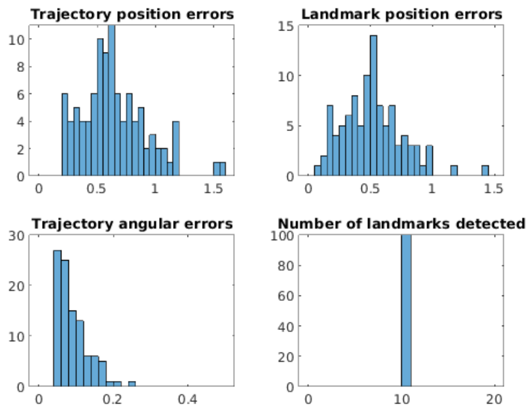


Figure 11: One run of my E7. It is still visibly imperfect.



(a) One landmark (‘o’).

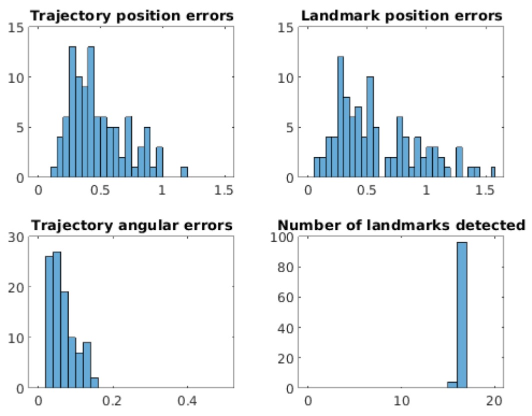


(b) All landmarks (‘a’).

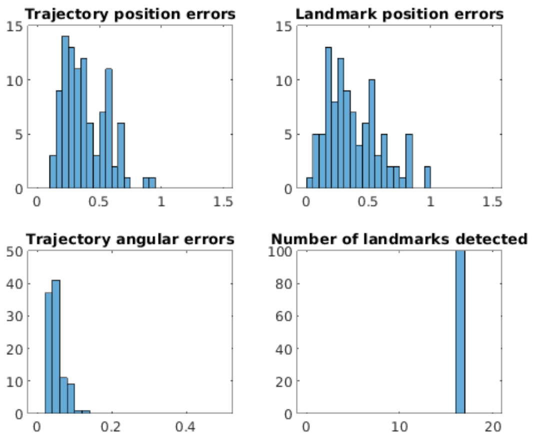
Figure 12: Statistics for my E7 SLAM running with the different landmark measurement modes.

Statistic	One Landmark (‘o’)	All Landmarks (‘a’)
Vehicle Position	0.571097	0.645565
Vehicle Orientation	0.072477	0.089526
Landmark Avg	0.461936	0.517878
Landmark Std Dev	0.214315	0.238637
# of Landmarks	10	10

We can see from the averages that the ‘a’ implementation is very similar, and is actually slightly worse than the ‘o’ mode on this run. This is because the vehicle pose estimate is updated just as frequently in both cases. The biggest cause of error is the non-negligible sections of the trajectory for which there are no landmarks to detect, and these sections are the same for both measurement modes. We can reduce the error by reducing these no-measurement segments, which can be done by increasing the sensor’s `range` and `fov` parameters, or by increasing the landmark density of the map. Increasing the range from 4 to 6 causes the ‘a’ mode to perform significantly better than the ‘o’ mode, as shown in the following figure and table.



(a) One landmark ('o'), range 6.



(b) All landmarks ('a'), range 6.

Figure 13: Statistics for my E7 SLAM running with the different landmark measurement modes, with the sensor's range increased from 4 to 6 units.

Statistic	One Landmark ('o')	All Landmarks ('a')
Vehicle Position	0.483947	0.383745
Vehicle Orientation	0.065635	0.050127
Landmark Avg	0.583157	0.385691
Landmark Std Dev	0.342905	0.220108
# of Landmarks	15.96	16

E8 - Bearing-Only SLAM

Sometimes our sensor is incapable of measuring range, such as one using the sun or stars as reference points. In our case, because our sensor has a maximum measurement range, we have an idea of the range distribution of a given measurement. Namely, we can take the mean of our measurement range (i.e., half the maximum) as our range "measurement". This will cause our insertion of this landmark's position to be very likely incorrect, so to encode our large uncertainty, we can append a large covariance matrix when updating the state covariance. Our hope is that within a finite number of observations, the true positions of these landmarks will be approached as the covariance falls lower in magnitude. We achieve this latter goal by creating our uncertainty matrix W_b as

$$W_b = \begin{bmatrix} (\text{range}/2)^2 & 0 \\ 0 & W \end{bmatrix},$$

where W is the uncertainty in the bearing measurements passed as a parameter. The upper-left quantity serves to give high uncertainty to our range "measurement", without changing the bearing's uncertainty. Tuning this higher or lower seems to lower the quality of the EKF, so this is likely near peak performance for bearing-only slam. The outcome of a single run, as well as the 100-run statistics are shown below.

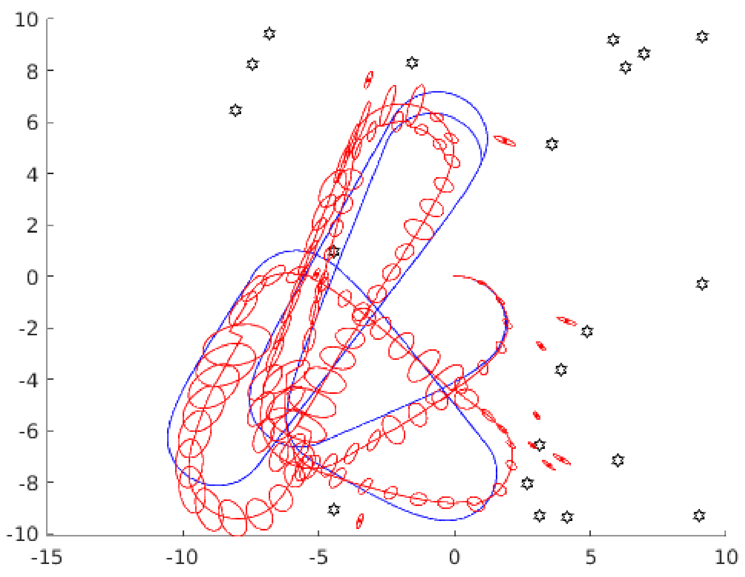
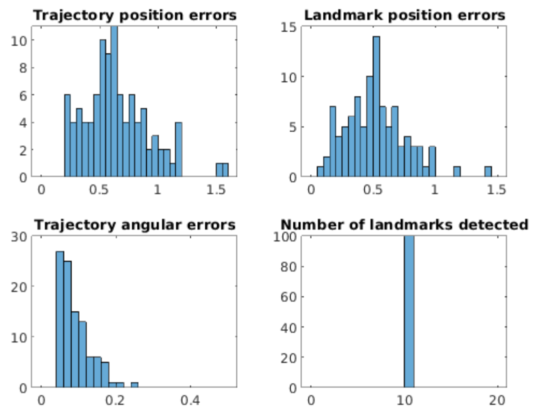
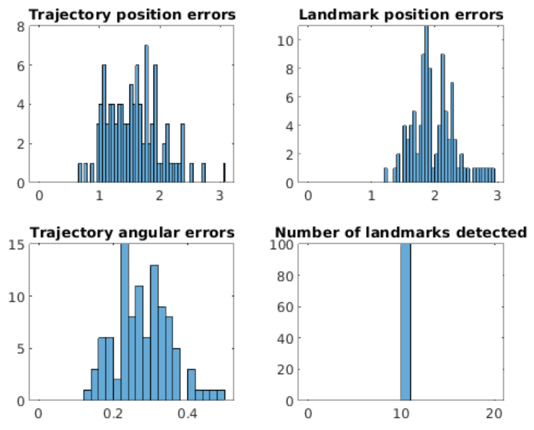


Figure 14: One run of my E8. It is clearly inferior to range+bearing SLAM, but it still follows the trajectory and map layout surprisingly well.



(a) Range and Bearing SLAM.

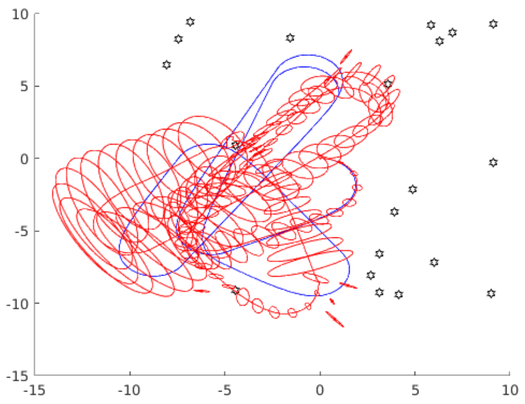


(b) Bearing-only SLAM.

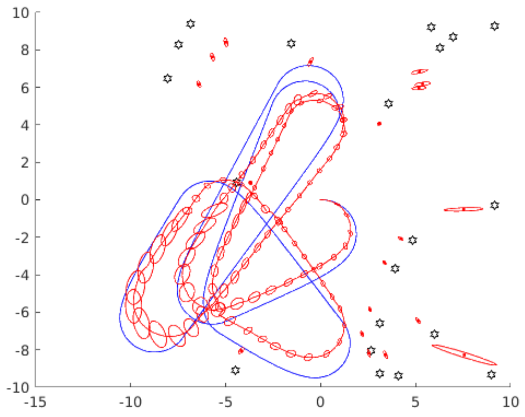
Figure 15: Statistics for SLAM with and without range measurements. Both use measurement mode ‘a’ and the default parameters for `range` and `fov`.

Statistic	Range and Bearing SLAM	Bearing-only SLAM
Vehicle Position	0.645565	1.580341
Vehicle Orientation	0.089526	0.280152
Landmark Avg	0.517878	1.996999
Landmark Std Dev	0.238637	0.344333
# of Landmarks	10	10

When altering the sensor’s parameters, we can see a trend in terms of information vs. uncertainty; if the maximum `range` is higher, more landmarks will be detected more often, but the total distance range in which a detected landmark could lie is higher, so we're less certain about every individual detection. Conversely, a lower maximum `range` means we'll be more certain about a particular measurement, but we will have fewer measurements overall. Keeping the `range` around 4 units seems like the best choice for bearing-only SLAM performance.



(a) `range` = 2. Error accumulates for long stretches where no landmarks are detected at all. Detections are more accurate since the possible range is reduced.



(b) `range` = 8. Error is kept in check by more frequent observations, but these observations tend to be further away than the mean, so our updates are underestimating the true range.

Figure 16: Bearing-only SLAM run with different sensor maximum `range` parameters.

E9 - Range-Only SLAM

Similarly to E8, sometimes we only have access to range information (not bearing) from our sensors, such as if we perform indoor mapping using various Wi-Fi signal strengths, which do not depend on bearing unless we use a directed antenna. Now our uncertainty is along the θ direction rather than the radial axis, but we can use the same philosophy to come up with a mean estimate, since we know the sensor's limited `fov` in this case; we'll use the mean of our minimum and maximum detection angles as our "measurement", and use a large covariance matrix to account for the uncertainty we're injecting into the state. We achieve this latter goal by creating our uncertainty matrix W_r as

$$W_r = \begin{bmatrix} W & 0 \\ 0 & (\text{fov\_range}/2)^2 \end{bmatrix},$$

where W is the uncertainty in the bearing measurements passed as a parameter, and the fov_range is simply the max(`fov`) - min(`fov`). The lower-left quantity serves to give high uncertainty to our bearing "measurement", without changing the bearing's uncertainty. Tuning this higher or lower seems to lower the quality of the EKF, so this is likely near peak performance for range-only slam. The outcome of a single run, as well as the 100-run statistics are shown below.

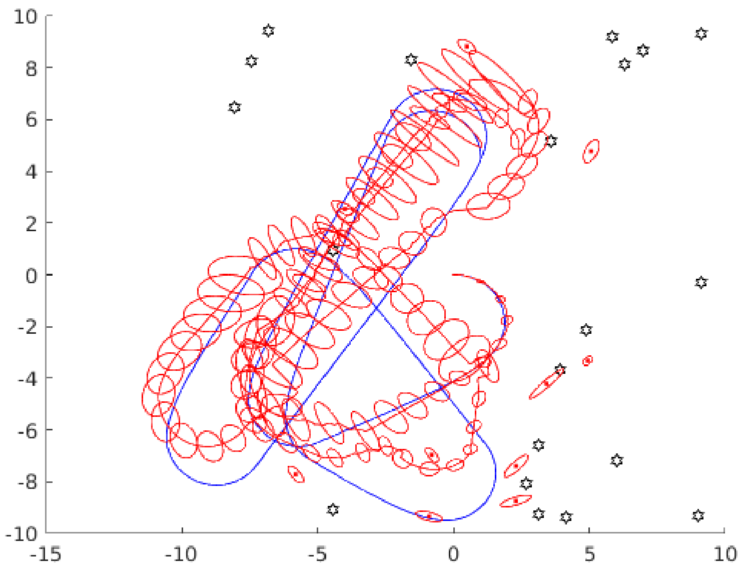
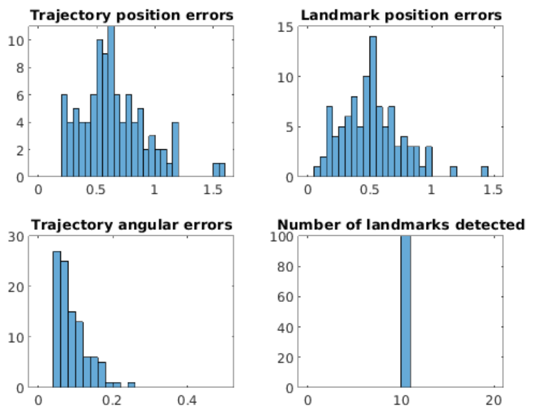
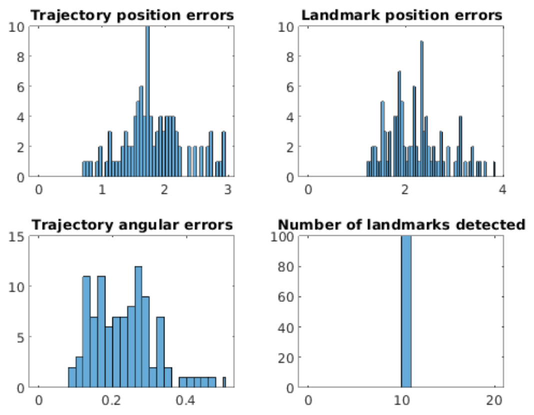


Figure 17: One run of my E9. It is clearly inferior to range+bearing SLAM, but it still follows the trajectory and map layout surprisingly well, though seems to be off on orientation more and more as time goes along.



(a) Range and Bearing SLAM.

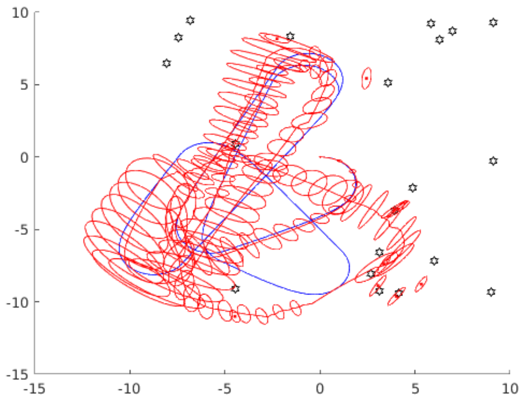


(b) Range-only SLAM.

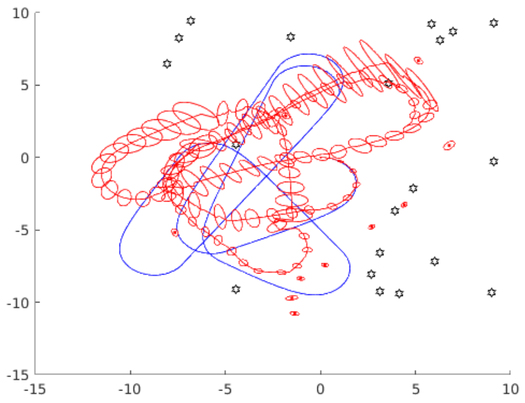
Figure 18: Statistics for SLAM with and without bearing measurements. Both use measurement mode 'a' and the default parameters for `range` and `fov`.

Statistic	Range and Bearing SLAM	Range-only SLAM
Vehicle Position	0.645565	1.820168
Vehicle Orientation	0.089526	0.231870
Landmark Avg	0.517878	2.228431
Landmark Std Dev	0.238637	0.595327
# of Landmarks	10	10

When altering the sensor's parameters, we can see a trend in terms of information vs. uncertainty which parallels the bearing-only SLAM; if the maximum `fov` is higher, more landmarks will be detected more often, but the total range of angles in which a detected landmark could lie is higher, so we're less certain about every individual detection. Conversely, a lower magnitude of `fov` means we'll be more certain about a particular measurement, but we will have fewer measurements overall. Keeping the `fov` around $[\pi/2, \pi/2]$ seems like the best choice for range-only SLAM performance.



(a) `fov` $= [\pi/4, \pi/4]$. Error accumulates for long stretches where no landmarks are detected at all. Detections are more accurate since the possible field of angles is reduced.



(b) `fov` $= [\pi, \pi]$. Error is kept in check by more frequent observations, but the measurement could be anywhere around the robot, so isolating the true bearing from measurements is very difficult. Hence, the orientation falls off dramatically as time goes on.

Figure 19: Range-only SLAM run with different sensor `fov` parameters.

Comparing bearing-only to range-only SLAM

From implementing both bearing-only and range-only SLAM, I would say that bearing-only SLAM is more effective, at least in our use case. This is because bearing encodes more useful information, and can be used quite accurately to estimate the vehicle's pose, even when range is unknown. Rotations are harder to predict using the EKF, and we rely on these bearing measurements to keep our uncertainty in check, so without them the state falls out of the range of usefulness very rapidly, as we can see by the final plots in sections E8 and E9. Additionally, as the robot moves and follows the trajectory, many landmarks are observed from several different bearings, which helps reduce the field of possible ranges to an obstacle. This is a detriment to range-only SLAM, as the range measurement may be fairly similar from different viewpoints and only adds confusion as to the true bearing to a landmark.