16-833: Robot Localization and Mapping, Spring 2021

Homework 2 - SLAM using Extended Kalman Filter (EKF-SLAM)

Due: Wednesday Mar 17, 11:59pm, 2021

Your homework should be submitted as a **typeset PDF file** along with a folder including **code only (no data)**. The PDF must be submitted on **Gradescope**, and code submitted on **Canvas**. In your implementation, please only fill in the functions marked by `TODO`, and play with parameters, and keep other parts unchanged. If you have questions, post them on Piazza or come to office hours. Please do not post solutions or codes on Piazza.

This homework must be done **individually**, and plagiarism will be taken seriously. You are free to discuss and troubleshoot with others, but the code and writeup must be your own. Note that you should list the name and Andrew ID of each student you have discussed with on the first page of your PDF file.

1 Theory (40 points)

In this part we are going to guide you through some steps of EKF-SLAM in math. This will be helpful in your implementation in the next problem.

A robot is moving on the 2D ground plane. In each time step t, the robot is controlled to move forward (the x-direction of the robot's coordinates) d_t meters, and then rotate α_t radian. The pose of the robot in the global coordinates at time t is written as a vector $\mathbf{p}_t = \begin{bmatrix} x_t & y_t & \theta_t \end{bmatrix}^\top$, where x_t and y_t are the 2D coordinates of the robot's position, and θ_t is the robot's orientation.

1. Based on the assumption that there is no noise or error in the control system, predict the next pose $\mathbf{p}_{t+1}$ as a nonlinear function of the current pose $\mathbf{p}_t$ and the control inputs d_t and α_t. (5 points)

$$\mathbf{p}_{t+1} = \begin{bmatrix} x_t + d_t * cos(\theta_t) \\ y_t + d_t * sin(\theta_t) \\ \theta_t + \alpha_t \end{bmatrix}$$

2. However, in reality there are some errors when the robot moves due to the mechanism and the terrain. Assume the errors follow Gaussian distributions: $e_x \sim \mathcal{N}(0, \sigma_x^2)$ in x-direction, $e_y \sim \mathcal{N}(0, \sigma_y^2)$ in y-direction, and $e_\alpha \sim \mathcal{N}(0, \sigma_\alpha^2)$ in rotation respectively (all in robot's coordinates). For details, please see Fig. 1. Now if the uncertainty of the robot's pose at time t can be represented as a 3-dimensional Gaussian distribution $\mathcal{N}(0, \Sigma_t)$, what is the predicted uncertainty of the robot at time $t+1$? Please express it as a Gaussian distribution

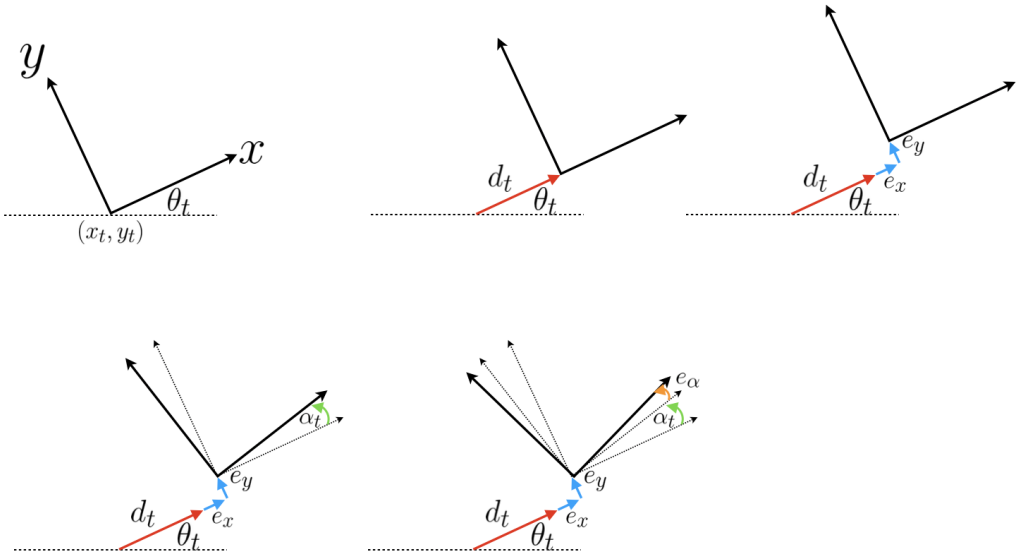


Figure 1: 0) Robot state; 1) robot moves d_t along its x-axis; 2) due to noise it shifted e_x and e_y along its x and y axises respectively; 3) robot rotates α_t after shifting; 4) due to noise the rotation is disturbed by e_α.

with zero mean. (5 points)

After adding the error terms, we get the following:

$$\mathbf{p}_{t+1} = \begin{bmatrix} x_t + (d_t + e_x)cos(\theta_t) - e_y sin(\theta_t) \\ y_t + (d_t + e_x)sin(\theta_t) + e_y cos(\theta_t) \\ \theta_t + \alpha_t + e_\alpha \end{bmatrix}$$

$$\Sigma_{t+1} = G_t \Sigma_t G_t^T + L_t R_t L_t^T$$

where R_t is the measurement covariance, and G_t and L_t are the jacobians of the motion equations w.r.t. p and to the error terms, evaluated at the linearization point. They are defined as follows:

$$G_t = \begin{bmatrix} 1 & 0 & -(d_0 + e_x)sin(\theta_0) - e_y cos(\theta_0) \\ 0 & 1 & (d_0 + e_x)cos(\theta_0) - e_y sin(\theta_0) \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -d_t sin(\theta_t) \\ 0 & 1 & d_t cos(\theta_t) \\ 0 & 0 & 1 \end{bmatrix}$$

$$L_t = \begin{bmatrix} cos(\theta_t) & -sin(\theta_t) & 0 \\ sin(\theta_t) & cos(\theta_t) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. Consider a landmark l being observed by the robot at time t with a laser sensor which gives a measurement of the bearing angle β (in the interval $(-\pi, \pi]$) and the range r, with noise $n_\beta \sim \mathcal{N}\left(0, \sigma_\beta^2\right)$ and $n_r \sim \mathcal{N}\left(0, \sigma_r^2\right)$ respectively. Write down the estimated position (l_x, l_y) of landmark l in global coordinates as a function of $\mathbf{p}_t$, β, r, and the noise terms. (5 points)

Assuming the robot has x, y, and θ position noises n_x, n_y, n_θ, which have the covariance matrix Σ, and are different from the e terms in the previous question, we write down the equations as follow.

$$l_x = x + n_x + (r + n_r)cos(\theta + n_\theta + \beta + n_\beta)$$

$$l_y = y + n_y + (r + n_r)sin(\theta + n_\theta + \beta + n_\beta)$$

Note again, these error terms are defined differently from that of the previous question.

4. According to 1.3, if we now know that l is at (l_x, l_y) in the global coordinates, please predict the measurement of bearing and range based on l_x, l_y, $\mathbf{p}_t$, and the noise terms (use functions $np.arctan2(.)$ and $warp2pi(.)$ that warps an arbitrary angular value into the range $(-\pi, \pi]$ if needed). (5 points)

   Given l_x, l_y, $\mathbf{p}_t$, and the noise terms, we can write β and r as follows:

$$\beta = warp2pi(np.arctan2(l_y - y_y, l_x - x_t) - \theta_t - n_\beta)$$

$$r = \sqrt{(l_x - x_t)^2 + (l_y - y_t)^2} - n_r$$

5. An important step in EKF-SLAM is to find the measurement Jacobian H_p with respect to the robot pose. Please apply the results in (d) to derive the analytical form of H_p (Note: H_p should be a 2×3 matrix represented by $\mathbf{p}_t$ and l). (10 points)

$$H_p = \begin{bmatrix} \frac{l_y - y_t}{(l_x - x_t)^2 + (l_y - y_t)^2} & \frac{x_t - l_x}{(l_x - x_t)^2 + (l_y - y_t)^2} & -1 \\ \frac{x_t - l_x}{\sqrt{(l_x - x_t)^2 + (l_y - y_t)^2}} & \frac{y_t - l_y}{\sqrt{(l_x - x_t)^2 + (l_y - y_t)^2}} & 0 \end{bmatrix}$$

6. For each measurement of bearing and range, we also need a measurement Jacobian H_l with respect to its corresponding landmark l. Please again derive the analytical form of H_l (Note: H_l should be a 2×2 matrix represented by $\mathbf{p}_t$ and l). Why do we not need to calculate the measurement Jacobian with respect to other landmarks except for itself (Hint: based on what assumption)? (10 points)

   For each landmark, we get the following jacobian matrix:

$$H_l = \begin{bmatrix} \frac{y_t - l_y}{(l_x - x_t)^2 + (l_y - y_t)^2} & \frac{l_x - x_t}{(l_x - x_t)^2 + (l_y - y_t)^2} \\ \frac{l_x - x_t}{\sqrt{(l_x - x_t)^2 + (l_y - y_t)^2}} & \frac{l_y - y_t}{\sqrt{(l_x - x_t)^2 + (l_y - y_t)^2}} \end{bmatrix}$$

   We don't need to calculate the measurement Jacobian between the landmarks based on the assumption that measurement noises are zero-mean Gaussians, meaning that if we model each landmark w.r.t the robot, we have modeled them w.r.t. each other.

Note in this section, in addition to conventional linearization with additive noise (in the form of $x_{t+1} = g(x_t, u_t) + \epsilon_t$), you may encounter non-additive noise (in the form of $x_{t+1} = g(x_t, u_t, \epsilon_t)$). Derivation of the linearization for this formulation can be helpful. Also, all the noise reading (ϵ, n) can be regarded as infinitismal. Second and higher order terms comprised by their products can be ignored.

2 Implementation and Evaluation (45 points)

In this part you will implement your own 2D EKF-SLAM solver in Python. The robot in problem 1 starts with observing the surrounding environment and measuring some landmarks, then executes a control input to move. The measurement and control steps are repeated several times. For simplicity, we assume the robot observes the same set of landmarks in the same order at each pose. We want to use EKF-SLAM to recover the trajectory of the robot and the positions of the landmarks from the control input and measurements.

1. Find the data file `data/data.txt` that contains the control inputs and measurements. Open the file to take a look. The data is in the format of measurements= $\begin{bmatrix} \beta_1 & r_1 & \beta_2 & r_2 & \cdots \end{bmatrix}$, where β_i, r_i correspond to landmark l_i, and control= $\begin{bmatrix} d & \alpha \end{bmatrix}$ (refer to problem 1 for the notations). The lines are in sequential time order.

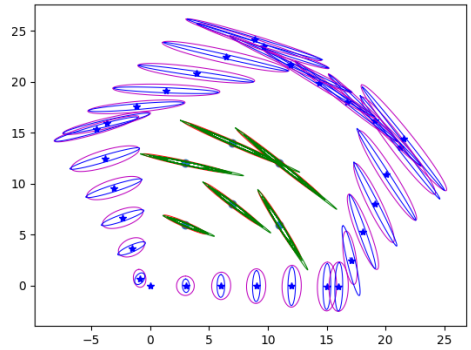
   What is the fixed number of landmarks being observed over the entire sequence? (5 points)

   There are 6 landmarks based on data.txt.

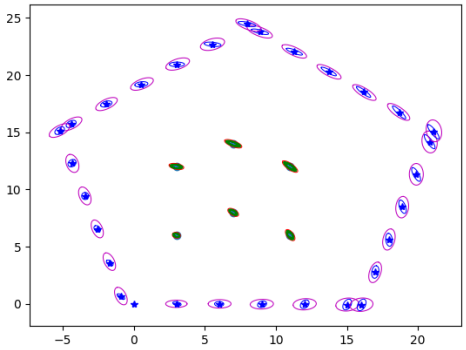
2. The `code/` folder contains the Python code file `ekf_slam.py`. Follow the comments and fill in the functions `warp2pi`, `init_landmarks`, `predict`, `update` to enable the system. The visualization will show the trajectory and the landmarks with their covariances in the shape of 3-σ ellipses. It will be updated sequentially, and you may process the next step by clicking your mouse or hitting any key.

   Attach a clear figure of your visualization in the write up once all the steps are finished. (25 points)

   The following visualizations (Figures 2 and 3) are for using the default parameters and using one tenth the $init_\theta$ standard deviation respectively.



(a) Default Parameters



(b) 0.01 init_theta_std

3. In the output figure, the magenta and blue ellipses represent the predicted and updated uncertainties of the robot's position (orientation not shown here) at each time respectively. Also, the red and green ellipses represent the initial and all the updated uncertainties of the landmarks, respectively.

   Describe how EKF-SLAM improves the estimation of both the trajectory and the map by comparing the uncertainty ellipses. (5 points)

   Each motion step, we increase the uncertainty of the robot's position due to errors in movement; however, the uncertainty of the location of the landmarks as well as the measurement uncertainties remain the same. Therefore, we can use this measurement to lower our trajectory uncertainty.

   With each new measurement from a different robot position, we can also use that extra information to update the landmark's locations, and the question becomes how much do we trust the robot location v.s. the measurements, and that is encapsulated in the Kalman Gain term.

   Depending on our current uncertainty of the robot position and the measurement variance, the Kalman gain scales in a way such that we trust each the correct amount.

4. Now let's evaluate our output map. Suppose we have the ground truth positions of all the landmarks:

   $$\begin{bmatrix} l_1^\top & l_2^\top & \cdots & l_k^\top \end{bmatrix} = \begin{bmatrix} 3 & 6 & 3 & 12 & 7 & 8 & 7 & 14 & 11 & 6 & 11 & 12 \end{bmatrix}.$$

   Plot the ground truth positions of the landmarks in the output figure (code already provided in function `evaluate`) and attach it below. Is each of them inside the smallest corresponding ellipse? What does that mean?

   Compute and report the *Euclidean* and *Mahalanobis* distances of each landmark estimation with respect to the ground truth. What do the numbers tell you? (10 points)

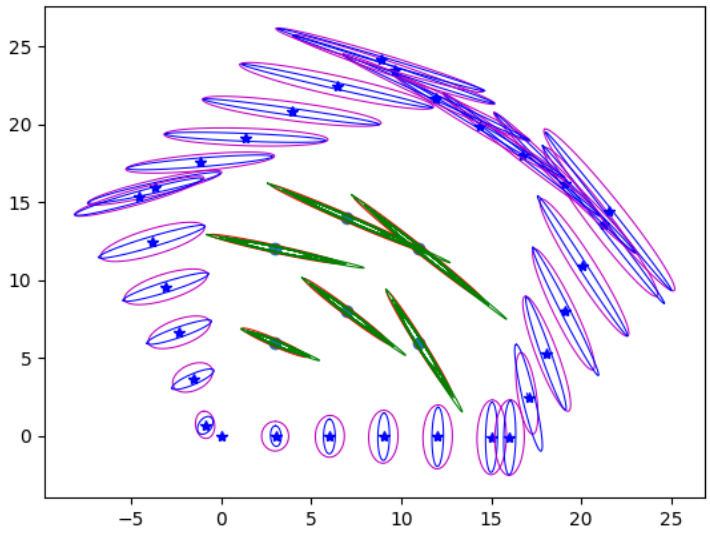


Figure 3: Default Parameters

Each of them is indeed inside the smallest corresponding ellipses. That means each landmark is within 3-σ of our best estimate location. The algorithm was able to perform satisfactory SLAM.

The distances are as follows:
Euclidean: [0.3056792 0.57610011 0.48502988 0.72333471 0.57254086 0.75206466]
Mahalanobis: [0.38070632 0.3917905 0.38169119 0.3931668 0.38463699 0.40154727]

In general, both error distances are small, with the Mahalanobis slightly smaller than the Euclidean distance in all but one landmark. This is likely due to the x and y directions of the covariance matrix differing so much, and the Euclidean not considering this. When the initial theta uncertainty is set lower, the Euclidean becomes smaller than the Mahalanobis.

3 Discussion (15 points)

1. Explain why the zero terms in the initial landmark covariance matrix become non-zero in the final state covariance matrix (print out the final P matrix to check it). Additionally, when setting the initial value for the full covariance matrix P (line 201 in `ekf_slam.py`) an assumption is made regarding certain cross-covariances that is not necessarily correct. Can you point out what that is? (5 points)

   The covariance matrix terms between landmarks and between landmarks and the location state terms become non-zero as we update because we look at all measurements with respect to the entire system, and cross-correlate the variances of our estimation.

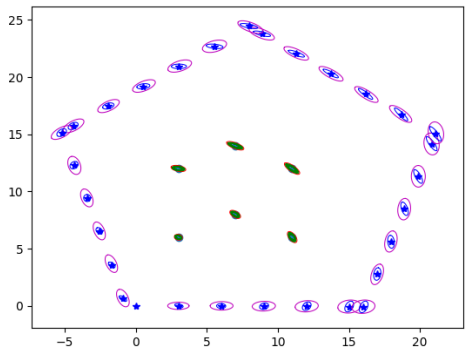
   In other words, knowing the location of one landmark will tell us something about the robot's position as they were used in each other's update. Same goes for knowing the robot position and estimating the landmark positions.

   The assumption we made is that the covariance between the initial robot position and the landmark positions are 0, which is incorrect as we used the robot's initial location to measure the landmark locations.

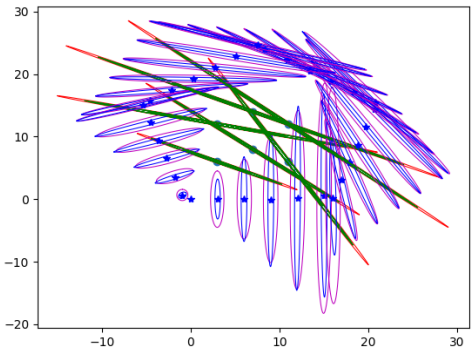
2. Play with the parameters (line 163-167). Modify each parameter σ_x, σ_y, $\sigma_\alpha\sigma_b$, σ_r at a time and fix the others. In particular, you should set each parameter to be 10 times larger/smaller to the original parameters to discuss how each parameters influence the results. Attach figures for better explanation. (10 points)

   First off, as shown in Figure 4, reducing the initial theta standard deviation to 0.01 drastically reduces our uncertainty of the landmark positions over the entire process, which increasing it to 0.5 drastically increases our uncertainty. This shows that our initial estimation of the robot's position has a very large cascading effect in EKF SLAM use cases.

   In changing σ_β and σ_r, we observe that if we increase these values, meaning we distrust our measurements more, the constraining effect of the measurement update step on our robot's location uncertainty decreases. This is evident in



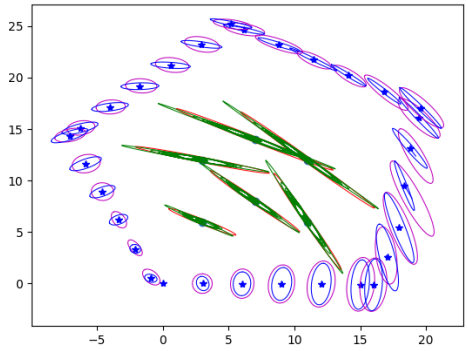
(a) 0.01 init_theta_std



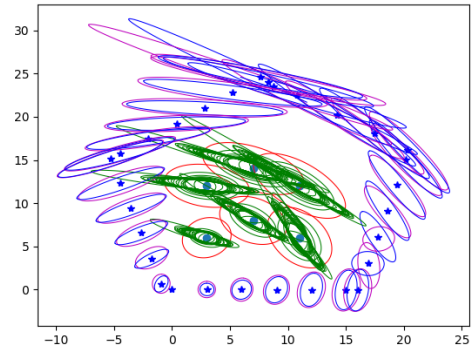
(b) 0.5 init_theta_std

Figure 4: Initial Theta Standard Deviation

the blue circles of the robot location covariances almost the same size as the red circles of the location covariances right after robot movement. This also has an effect on the landmark locations, making our estimations poorer.



(a) 10 times σ_β

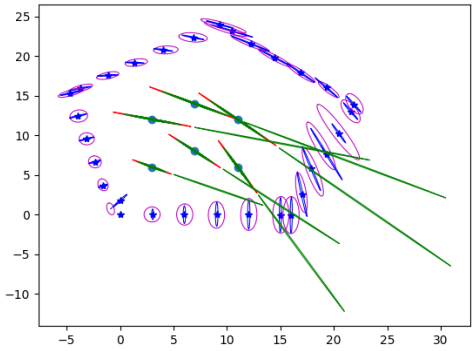


(b) 10 times σ_r

Figure 5: Distrusting the Measurements

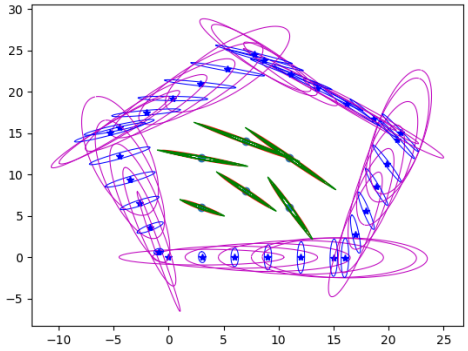
On the other hand, if we trust the measurements too much such that it cannot represent the inherent variance in our data, we sometimes get incorrect mean measurements, as shown in Figure 6.

In changing the robot action uncertainties, making σ_x and σ_y larger makes localizing the robot harder. As shown in Figure 7, the red circles of robot state uncertainty after an action is taken becomes quite large, however, with accurate measurements the EKF SLAM algorithm is able to correct for that and not lose track in the long run, as shown by the much smaller blue circles. Finally for the angle uncertainty, because the robot doesn't make a lot of drastic turns and only five times throughout, large σ_α's do not effect the resulting estimations much, and the graph looks very similar to a normal σ_α, as shown in Figure 8.

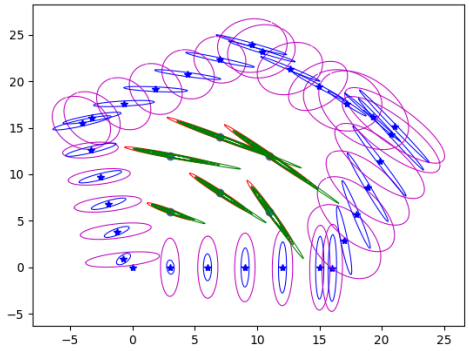


(a) 0.1 times σ_r

Figure 6: Trusting the Measurement too much

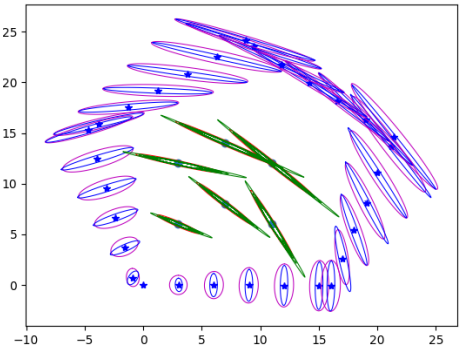


(a) 10 times σ_x



(b) 10 times σ_y

Figure 7: Higher Action Uncertainty



(a) 100 times σ_α

Figure 8: Higher Action Uncertainty - α

3. With the same set of landmarks being observed all the time, the EKF-SLAM system runs in constant time in each iteration. However, for real-world problems, the total number of landmarks can grow unbounded if the robot keeps exploring new environments. This will slow down the EKF-SLAM system a lot and become a crucial problem for real-time applications. What changes can

we make to the EKF-SLAM framework to achieve constant computation time in each cycle or at least speed up the process (list as many possible solutions as you can think of)? (Bonus 5 points)

With a growing set of landmarks, a lot of the older landmarks will become obsolete unless we come full circle back to them. As a result, we can forget all landmarks in our algorithm calculations if they are not observed in a set number of time steps.

Additionally, the matrices will be very sparse, and we can look for many computational methods that exploit this sparsity.

4 Code Submission

Upload your `ekf_slam.py`. Please do not upload the `data/` folder or any other data or image.