Feature-Based Localization and SLAM with EKF and Reinforcement Learning

Maryam Valipour
maryam.valipour71@gmail.com
Jan 4, 2021

Implementation Code: link to google drive

Contents

1- Introduction and Problem Statement

For this project, two closely related estimation problems, namely localization, and SLAM have been chosen to solve with both conventional extended Kalman filter and its combination with reinforcement learning. Both problems will be discussed in more detail in the following paragraphs.

Localization - In the localization problem, the robot's pose estimation in every timestep is calculated, given the mean of the estimated pose and covariance matrix of the previous timestep and the control inputs (odometry data) and sensor measurements at the current timestep. In addition, the map (the position of all the landmarks) of the environment is known beforehand in the localization problem. In this

project, it's also assumed that the correspondence between each m measurement and landmark is known too.

SLAM - The problem of simultaneous localization and mapping (SLAM) is defined as the simultaneous localization and mapping of the environment. It's an extension of the localization problem. In this case, the number of the landmarks and their correspondence in the measurement data is still known, and it's clear to which landmark each measurement corresponds. The only difference is that there is no information about the position of the landmarks in the environment. In the SLAM problem, in addition to estimating the robot's pose at every timestep, the goal is to estimate all the landmark's positions as well.

2- Related Literatures

EKF-SLAM – Literature review

The problem of SLAM has been studied for a long time and is still one of the essential issues in the robotics field. The SLAM problem has been tackled through many approaches; one method is the application of extended Kalman filters (EKF). This approach consists of two main parts, mainly prediction and correction. The correction part of the method takes more time than the prediction part. This approach can be computationally expensive, especially in larger maps with more features. EKF can be used in both visual and feature-based SLAM [1].

RL-Algorithms – Literature review

Recently, there has been lots of improvement in combining deep neural networks with reinforcement learning methods. In 2015, DeepMind proposed the DQN algorithm that used two innovations, namely replay buffer and target Q-networks that improved the performance in many reinforcement learning problems [2]. However, DQN only worked for discrete action spaces. Many real-world applications have continuous action spaces, and DQN can not be used for these situations. Even by discretizing the action space, this algorithm faces a huge problem, which is the curse of dimensionality. To solve this issue, DeepMind has proposed another algorithm named DDPG, which is basically an actor-critic method combined with DQN innovations such as replay buffer and target networks [3]. The DDPG algorithm seems to perform well in continuous action spaces and yields a robust and stable solution to the problem.

Localization-RL and SLAM-RL – Literature review

In the field of mobile robot navigation, reinforcement learning has mostly been used for the path planning part of the problem. However, recently, there have been a few attempts to use reinforcement learning with SLAM, too. In the following paragraphs, the application of reinforcement learning in the SLAM problem in recent researches has been discussed.

N. Botteghi and et al., in their research, have proposed an RL-SLAM algorithm to explore and create a map of unknown environments [4]. They have designed three different reward functions and evaluated their performance in environments with different complexities. They have shown that rewards based on map-completeness and information-gain explored the environments much faster than a simple sparse reward.

Also, their developed models were robust and able to transfer their knowledge, and as a result, generalize well to prior unseen environments.

Another significant research in this field was done by Julio A. Placed and José A. Castellanos [5]. In this research, they have used active SLAM combined with RL to decrease the localization and map's representation uncertainty, which performed much better than the classic active SLAM approach. This algorithm also generalized well to previously unseen environments. To achieve this goal, they have used a Dueling double deep q network (D3QN). Finally, they have suggested using Active SLAM with deep reinforcement learning approaches such as A3C and DDPG that deals with continuous action spaces. Also, in addition to laser measurements as input, they have suggested using this proposed algorithm on other input types such as images.

RL-EKF – Literature review

Shirli Di-Castro Shashua and Shie Mannor, in their research, have proposed a new algorithm called Deep Robust Kalman filter (Deep-RoK) [6]. This algorithm is basically a robust Bayesian method, based on the Extended Kalman filter (EKF), that intends to consider both transition probabilities and weight uncertainties in the value function in a way that the robust EKF lost function gets minimized and improve the robustness of the agent. They have experimented with their algorithm on a benchmark reinforcement learning problem, namely Cart-Pole, and have shown that their algorithm performs much better than the classic DQN algorithm. They have suggested that this method can be applied to the autonomous robotics field to reduce their uncertainties.

3- Algorithms

In this section, the conventional EKF [7] and reinforcement learning-based EKF algorithms used to solve the problem of both localization and SLAM in this project will be discussed briefly.

3-1- Localization with EKF algorithm

In the following paragraphs, the EKF method for localization is discussed. The EKF method can be split into two main parts, which are prediction and correction.

Prediction step

In the prediction step, the robot's position at timestep t is estimated based on its last pose and control commands (the odometry data) using the motion model. In this project, it is assumed that for the robot to move, it firstly has to rotate in place, then moves forward in a straight line, and then rotate in place again.

Also, in this step, the covariance for the new pose is estimated, too, which is calculated by multiplying the last timesteps' covariance with Jacobian of the motion model with respect to the previous state G_t. Also, the noise corresponded to the motion model, which is often called the process noise M_t, should be taken into account in the pose estimation's uncertainty by being added to the covariance for the new pose. In order to map the process noise to state space, it has to be multiplied by the Jacobian of the motion model

with respect to the motion parameters V_t. The equation for computing the predicted mean and covariance at timestep is t shown below.

$$\bar{\mu}_t = g(u_t, x_{t-1}) = \mu_{t-1} + \begin{bmatrix} v_t \Delta T \cos\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ v_t \Delta T \sin\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ \omega_t \Delta T \end{bmatrix}$$

$$G_t = \frac{\partial g(u_t, x_{t-1})}{\partial x_{t-1}} \begin{bmatrix} 1 & 0 & -v_t \Delta T \sin\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ 0 & 1 & -v_t \Delta T \cos\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ 0 & 0 & 1 \end{bmatrix}$$

$$\bar{\sum}_t = G_t \sum_{t-1} G_t^T + V_t M_t V_t^T$$

As it is obvious in the equations above, by incorporating the odometry data, the uncertainty around the robot's pose always increases.

Correction Step

In this step, in contrast to the prediction step, uncertainty decreases, and the robot's pose becomes corrected using data from sensors. Sensor data constitutes the range and bearing from the robot to the landmarks in its view field in each timestep.

Assuming that the estimated pose $\bar{\mu}_t$ in the prediction step was correct, the range and bearing to the landmarks in the robot's view field are computed using the measurement model. This is called the expected measurement $\hat{z}_t$ and is calculated, as shown below, where $\mu_{j,x}$ and $\mu_{j,y}$ are the jth landmark's coordinates detected at timestep t and $\mu_{j,s}$ is its ID.

$$r = \sqrt{\left(m_{j,x} - \bar{\mu}_{t,x}\right)^2 + \left(m_{j,y} - \bar{\mu}_{t,y}\right)^2}$$

$$\emptyset = \operatorname{atan2}\left(\mu_{j,y} - \bar{\mu}_{t,y}, \mu_{j,x} - \bar{\mu}_{t,x}\right) - \bar{\mu}_{t,\theta}$$

$$\hat{z}_t^i = \begin{bmatrix} \sqrt{q} \\ \operatorname{atan2}\left(\mu_{j,y} - \bar{\mu}_{t,y}, \mu_{j,x} - \bar{\mu}_{t,x}\right) - \bar{\mu}_{t,\theta} \\ \mu_{j,s} \end{bmatrix}$$

The Jacobian of the measurement model is derived by calculating its derivative with respect to the robot's pose computed at the predicted mean.

$$H_t^i = \begin{bmatrix} -\frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} & -\frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & -1 \\ 0 & 0 & 0 \end{bmatrix}$$

The overall measurement prediction uncertainty S_t is calculated by multiplying the Jacobian of the measurement model with the prior estimated covariance and adding a 3×3 constant diagonal matrix called the measurement noise that should be hand-tuned for better position estimation.

$$S_t^i = H_t^i \bar{\sum}_t \left[H_t^i\right]^T + Q_t$$

In the next step, the Kalman gain is computed using the equation shown below. The Kalman gain is basically the tradeoff between the accuracy of the sensor information and odometry data.

$$K_t^i = \bar{\sum}_t \left[H_t^i\right]^T \left[S_t^i\right]^{-1}$$

Then, the estimated pose mean and covariance are updated using the Kalman gain. Also, the difference between the actual measurement and the expected measurement is called the innovation vector. The higher this difference is, the more the robot's pose gets corrected.

$$\bar{\mu}_t = \bar{\mu}_t + K_t^i \left(z_t^i - \hat{z}_t^i\right)$$
$$\bar{\sum}_t = \left(I - K_t^i H_t^i\right) \bar{\sum}_t$$

Finally, the estimated values $\bar{\mu}_t$ and $\bar{\sum}_t$ will be considered as the pose mean and covariance of the robot at timestep t, respectively.

$$\mu_t = \bar{\mu}_t$$
$$\sum_t = \bar{\sum}_t$$

In the following section, the whole algorithm of the EKF Localization with known correspondences is illustrated**.**

EKF Localization Algorithm with known correspondences

1. $(\mu_{t-1}, \sum_{t-1}, u_t, z_t, c_t, m)$
2. $\theta = u_{t-1,\theta}$

Prediction Step

3. $G_t = \begin{bmatrix} 1 & 0 & -v_t \Delta T \, sin\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ 0 & 1 & -v_t \Delta T \, cos\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ 0 & 0 & 1 \end{bmatrix}$
4. $V_t = \begin{bmatrix} cos\left(\theta + \frac{\omega_t \Delta T}{2}\right) & -\frac{1}{2} sin\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ sin\left(\theta + \frac{\omega_t \Delta T}{2}\right) & \frac{1}{2} cos\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ 0 & 1 \end{bmatrix}$
5. $M_t = \begin{pmatrix} \alpha_1 v_t^2 + \alpha_2 \omega_t^2 & 0 \\ 0 & \alpha_3 v_t^2 + \alpha_4 \omega_t^2 \end{pmatrix}$
6. $\bar{\mu}_t = \mu_{t-1} + \begin{bmatrix} v_t \Delta T \, cos\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ v_t \Delta T \, sin\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ \omega_t \Delta T \end{bmatrix}$
7. $\bar{\sum}_t = G_t \sum_{t-1} G_t^T + V_t M_t V_t^T$

Correction Step

8. $Q_t = \begin{bmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\phi^2 & 0 \\ 0 & 0 & \sigma_s^2 \end{bmatrix}$
9. $\boldsymbol{for}\ all\ observed\ features\ z_t^i = [r_t^i \quad \phi_t^i \quad s_t^i]^T\ do$
10. $j = c_t^i$
11. $q = \left(m_{j,x} - \bar{\mu}_{t,x}\right)^2 + \left(m_{j,y} - \bar{\mu}_{t,y}\right)^2$
12. $\hat{z}_t^i = \begin{bmatrix} \sqrt{q} \\ \text{atan2}\left(\mu_{j,y} - \bar{\mu}_{t,y}, \mu_{j,x} - \bar{\mu}_{t,x}\right) - \bar{\mu}_{t,\theta} \\ \mu_{j,s} \end{bmatrix}$
13. $H_t^i = \begin{bmatrix} -\frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} & -\frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & -1 \\ 0 & 0 & 0 \end{bmatrix}$
14. $S_t^i = H_t^i \bar{\sum}_t \left[H_t^i\right]^T + Q_t$
15. $K_t^i = \bar{\sum}_t \left[H_t^i\right]^T \left[S_t^i\right]^{-1}$
16. $\bar{\mu}_t = \bar{\mu}_t + K_t^i \left(z_t^i - \hat{z}_t^i\right)$
17. $\bar{\sum}_t = \left(I - K_t^i H_t^i\right) \bar{\sum}_t$
18. $\boldsymbol{end\ for}$
19. $\mu_t = \bar{\mu}_t$
20. $\sum_t = \bar{\sum}_t$
21. $p_{z_t} = \prod_i det\left(2\pi S_t^i\right)^{-\frac{1}{2}} exp\left\{-\frac{1}{2}\left(z_t^i - \hat{z}_t^i\right)^T \left[S_t^i\right]^{-1} \left(z_t^i - \hat{z}_t^i\right)\right\}$
22. $return\ \mu_t, \sum_t, p_{z_t}$

3-2- SLAM with EKF algorithm

In the following paragraphs, the changes to the localization problem to accommodate the SLAM problem will be discussed thoroughly.

State vector and covariance matrix

Since in the SLAM problem, the goal is to estimate both landmarks and robot's positions simultaneously, the state vector's dimension change from 3×1 to $(2n + 3) \times 1$ to include landmarks x and y coordinate in the state vector as well. This affects the size of the estimated mean vector and covariance matrix and changes their size to $(2n + 3) \times 1$ and $(2n + 3) \times (2n + 3)$, respectively.

Prediction step

In this step, the robot's pose is estimated using the control commands. Clearly, control commands don't have any impact on the landmark's positions in the state vector. So, a $3 \times (2n + 3)$ matrix F_x is used to map the changes to the robot's pose mean and covariance to the full state vector. This matrix is filled by a 3×3 identity matrix and zeros elsewhere. This matrix will leave the landmarks' positions unaffected as desired.

$$F_x = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \end{bmatrix}$$

The rest of the prediction step is as indicated in the equations below.

$$\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{bmatrix} v_t \Delta T \cos\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ v_t \Delta T \sin\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ \omega \Delta T \end{bmatrix}$$

$$G_t = I + F_x^T \begin{bmatrix} 0 & 0 & -v_t \Delta T \sin\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ 0 & 0 & -v_t \Delta T \cos\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ 0 & 0 & 0 \end{bmatrix} F_x$$

$$\bar{\sum}_t = G_t \sum_{t-1} G_t^T + F_x^T R_t F_x$$

Correction step

For each measurement, only the robot's and the associated landmark's position estimation should be affected, and the rest of the landmarks' estimation should remain unchanged. So, this time a $5 \times (2n + 3)$ matrix F_{xj} is used to map the correction from one landmark and the robot's state vector to the full state vector. This vector includes a 3×3 identity matrix for the robot's pose correction and a 2×2 identity matrix for the landmark and zeros elsewhere.

$$F_{x,j} = \begin{bmatrix} 1 & 0 & 0 & 0 \dots 0 & 0 & 0 & 0 \dots 0 \\ 0 & 1 & 0 & 0 \dots 0 & 0 & 0 & 0 \dots 0 \\ 0 & 0 & 1 & 0 \dots 0 & 0 & 0 & 0 \dots 0 \\ 0 & 0 & 0 & 0 \dots 0 & 1 & 0 & 0 \dots 0 \\ 0 & 0 & 0 & 0 \dots 0 & 0 & 1 & 0 \dots 0 \end{bmatrix}$$

The rest of the correction step is demonstrated by the equations below.

$$H_t^i = \frac{1}{q} \begin{bmatrix} -\frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} & 0 & \frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & \frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} \\ \frac{m_{j,y} - \bar{\mu}_{t,y}}{q} & -\frac{m_{j,x} - \bar{\mu}_{t,x}}{q} & -1 & -\frac{m_{j,y} - \bar{\mu}_{t,y}}{q} & \frac{m_{j,x} - \bar{\mu}_{t,x}}{q} \end{bmatrix} F_{x,j}$$

$$S_t^i = H_t^i \bar{\sum}_t \left[H_t^i\right]^T + Q_t$$

$$K_t^i = \bar{\sum}_t \left[H_t^i\right]^T \left[S_t^i\right]^{-1}$$

$$\bar{\mu}_t = \bar{\mu}_t + K_t^i \left(z_t^i - \hat{z}_t^i\right)$$

$$\bar{\sum}_t = \left(I - K_t^i H_t^i\right) \bar{\sum}_t$$

In the following section, the whole algorithm of the EKF SLAM with known correspondences is illustrated.

EKF SLAM with known correspondences

1. $(\mu_{t-1}, \sum_{t-1}, u_t, z_t, c_t)$

Prediction Step

2. $F_x = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \end{bmatrix}$

3. $\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{bmatrix} v_t \Delta T \cos\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ v_t \Delta T \sin\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ \omega \Delta T \end{bmatrix}$

4. $G_t = I + F_x^T \begin{bmatrix} 0 & 0 & -v_t \Delta T \sin\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ 0 & 0 & -v_t \Delta T \cos\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ 0 & 0 & 0 \end{bmatrix} F_x$

5. $V_t = \begin{bmatrix} \cos\left(\theta + \frac{\omega_t \Delta T}{2}\right) & -\frac{1}{2}\sin\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ \sin\left(\theta + \frac{\omega_t \Delta T}{2}\right) & \frac{1}{2}\cos\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ 0 & 1 \end{bmatrix}$

6. $M_t = \begin{pmatrix} \alpha_1 v_t^2 + \alpha_2 \omega_t^2 & 0 \\ 0 & \alpha_3 v_t^2 + \alpha_4 \omega_t^2 \end{pmatrix}$

7. $R_t = V_t M_t V_t^T$

8. $\bar{\sum}_t = G_t \sum_{t-1} G_t^T + F_x^T R_t F_x$

Correction Step

9. $Q_t = \begin{bmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\phi^2 & 0 \\ 0 & 0 & \sigma_s^2 \end{bmatrix}$

10. ***for*** *all observed features* $z_t^i = [r_t^i \quad \phi_t^i \quad s_t^i]^T$ *do*

11. $j = c_t^i$

12. ***if*** *landmark j nver seen before*

13. $\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \\ \bar{\mu}_{j,s} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \\ s_t^i \end{pmatrix} + \begin{pmatrix} r_t^i \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ r_t^i \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \\ 0 \end{pmatrix}$

14. ***end if***

15. $\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{j,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{j,y} - \bar{\mu}_{t,y} \end{pmatrix}$

16. $q = \delta^T \delta$

17. $z_t^i = \begin{pmatrix} \sqrt{q} \\ \text{atan2}(\delta_y, \delta_x) - \bar{\mu}_{t,\phi} \\ \bar{\mu}_{j,s} \end{pmatrix}$

18. $\hat{z}_t^i = \begin{bmatrix} \sqrt{q} \\ \text{atan2}(m_{j,y} - \bar{\mu}_{t,y}, m_{j,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \\ \bar{\mu}_{j,s} \end{bmatrix}$

19. $F_{x,j} = \begin{bmatrix} 1 & 0 & 0 & 0 \dots 0 & 0 & 0 & 0 \dots 0 \\ 0 & 1 & 0 & 0 \dots 0 & 0 & 0 & 0 \dots 0 \\ 0 & 0 & 1 & 0 \dots 0 & 0 & 0 & 0 \dots 0 \\ 0 & 0 & 0 & 0 \dots 0 & 1 & 0 & 0 \dots 0 \\ 0 & 0 & 0 & 0 \dots 0 & 0 & 1 & 0 \dots 0 \end{bmatrix}$

20. $H_t^i = \frac{1}{q} \begin{bmatrix} -\frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} & 0 & \frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & \frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} \\ \frac{m_{j,y} - \bar{\mu}_{t,y}}{q} & -\frac{m_{j,x} - \bar{\mu}_{t,x}}{q} & -1 & -\frac{m_{j,y} - \bar{\mu}_{t,y}}{q} & \frac{m_{j,x} - \bar{\mu}_{t,x}}{q} \end{bmatrix} F_{x,j}$

21. $S_t^i = H_t^i \bar{\sum}_t \left[H_t^i\right]^T + Q_t$

22. $K_t^i = \bar{\sum}_t [H_t^i]^T [S_t^i]^{-1}$
23. $\bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \hat{z}_t^i)$
24. $\bar{\sum}_t = (I - K_t^i H_t^i) \bar{\sum}_t$
25. ***end for***
26. $\mu_t = \bar{\mu}_t$
27. $\sum_t = \bar{\sum}_t$
28. $return\ \mu_t, \sum_t$

3-3- A proposal for localization with EKF-RL

3-3-1 Localization with EKF-RL

In the following paragraphs, the details and steps of the proposed reinforcement learning-based EKF method for localization is provided in detail.

Environment and States

The agent uses the current state to decide the next action at each timestep. A state vector must contain all the relevant information for solving a given task such that the agent is able to make the optimal decision.

In this research the state is considered as a $(2 * n + 3) \times 1$ vector in which n is the number of the landmarks. This vector consists of the robot's estimated position coordinates x, y, θ and the range and bearing information from the measurements. Basically, if the robot has any measurements from any of the landmarks, it will be considered in the state vector, and if not, the respective values will be zeros. For instance, if the robot has two measurements from the landmark two and five at a timestep, the state vector will be as shown below.

$$state = [x \quad y \quad \theta \quad r_t^1 \quad \phi_t^1 \quad 0 \dots 0 \quad 0 \quad r_t^5 \quad \phi_t^5 \quad 0 \dots 0]^T$$

Action prediction using RL

The action space depends on the robot's degrees of freedom. In this project, the action space is continuous and two-dimensional. It contains the forward velocity v_t and the angular velocity ω_t. The action space at each timestep is as follows.

$$action = [v_t \quad \omega_t]$$

The estimated robot's position will be computed with the motion model as below.

$$\bar{\mu}_t = \mu_{t-1} + \begin{bmatrix} v_t \Delta T \cos\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ v_t \Delta T \sin\left(\theta + \frac{\omega_t \Delta T}{2}\right) \\ \omega_t \Delta T \end{bmatrix}$$

The termination of the episode

In this project, the data is resampled at 50Hz. In this way, data is divided into 75000 timesteps. The problem of localization is non-episodic, but for the purpose of this research, data is segmented into

episodes. Each episode is assumed to contain 100 timesteps at most, and after that, the episode terminates. Also, during the execution of the episode, if the average of the error (the error is computed by calculating the difference between the estimated robot's position and its ground-truth data at every timestep) in the last 10 timesteps is less than 1.2 times the error in the Kalman filter localization that we have implemented, the episode terminates sooner than 100 timesteps.

Reward definition using EKF

In every reinforcement learning problem, the reward function should be designed based on the problem's objective. In this research, the reward function is designed in a way that the agent would be able to estimate the robot's position accurately in every timestep. In this regard, the goal is to minimize the difference between the robot's estimated position and its ground-truth data in the localization problem. In this project, we aim to minimize the sum of distances between the robot's actual measurements and the expected measurements from the predicted position. The actual observation at each timestep is shown with z_t^i, and the expected is calculated as below, where $m_{j,x}$, $m_{j,y}$ are the coordinates of the measured landmark, and $\bar{\mu}_{t,x}, \bar{\mu}_{t,y}, \bar{\mu}_{t,\theta}$ are the coordinates of the estimated robot's position.

$$z_t^i = [r_t^i \quad \phi_t^i]^T$$

$$q = \left(m_{j,x} - \bar{\mu}_{t,x}\right)^2 + \left(m_{j,y} - \bar{\mu}_{t,y}\right)^2$$

$$\hat{z}_t^i = \begin{bmatrix} \sqrt{q} \\ \text{atan2}\left(m_{j,y} - \bar{\mu}_{t,y}, m_{j,x} - \bar{\mu}_{t,x}\right) - \bar{\mu}_{t,\theta} \end{bmatrix}$$

The reward at each non-terminating timestep is designed as the maximum number between zero and $1 - \frac{\sum_i \left\| z_t^i - \hat{z}_t^i \right\|}{N}$, which is one minus the sum of distances between the robot's actual measurements and the expected measurements from the predicted position divided by a normalizing constant N. Since the range of the sum of distances is not known accurately, the normalizing constant is chosen and tuned by hand. As the sum of the distance doesn't seem to be higher than 20, the normalizing constant is chosen to be 20.

$$reward = max\left(1 - \frac{\sum_i \left\| z_t^i - \hat{z}_t^i \right\|}{N}, 0\right)$$

In reinforcement learning problems, the final objective is to maximize the accumulated reward. In this problem, we seek to minimize the difference between the robot's actual measurement and the expected measurement. So, the chosen reward is defined in a way that by maximizing the accumulated reward, this difference gets minimized. Also, it's worth mentioning that the upper limit value of the reward in non-terminating timesteps is one. And, as the estimated robot's position reaches its real value, this difference gets closer to zero, and the reward will get to its maximum point in each timestep.

Also, as discussed in the above section, the episode terminates when the difference between the estimated robot's position and its ground-truth data in the last 20 timesteps is less than 1.2 times the error in the Kalman filter localization in the last part. When this happens, the agent receives a reward of

100. The reason behind choosing this particular reward value is that the important thing in a reinforcement learning problem is maximizing the accumulated reward. So, if the goal is reached in an episode, which is minimizing the difference between the robot's estimated and real position in this problem, the agent has to receive a maximum reward. In this regard, the maximum reward equals 100.

However, it's worth mentioning that in the dataset that was chosen for this project, not all the timesteps had measurements recorded. To tackle this issue, we have assumed the last existing actual measurement to be the actual measurement of the current timestep. Also, the expected measurement is computed with the same landmarks that were recorded in the last measurement.

Evaluation using ground-truth data

For the localization problem, the robot's ground-truth is used to evaluate the performance of the proposed algorithm.

Learning Algorithm: Deep Deterministic Policy Gradients (DDPG)

As discussed above, the state space in both the localization and SLAM problems is continuous, so a simple Q-learning algorithm can't be used here. Also, both problems have two-dimensional continuous action space, which means a deep q network can't be implemented either. As a result, a Deep Deterministic Policy Gradients (DDPG) algorithm is chosen for the purpose of this research. The method will be discussed in detail in the following paragraphs.

Deep Deterministic Policy Gradients (DDPG) is an actor-critic algorithm designed for use in environments with continuous action spaces. Despite being an actor-critic method, DDPG makes use of several innovations from deep Q learning, such as replay memory and target networks for both actor and critic networks. In total, the DDPG algorithm uses four different neural networks. Target networks are time-delayed copies of their original networks and are used to improve stability in the algorithm. In contrast to the DQN algorithm, instead of directly copying the original networks, a soft update rule is applied in this algorithm.

The replay memory innovation simply proposes that instead of only using the most recent state transition to train the networks, the agent stores all the experiences in a replay buffer and randomly samples batches from the replay buffer to update the weights of the deep neural networks.

Actor Network - The actor-network decides what action to take based on the state given as the input. In contrast to the DQN algorithm, the actor-network outputs the action values themselves and not the probabilities associated with each action in the action space.

Most reinforcement learning problems face explore-exploit dilemma, and the actor-network here is no exception. In discrete action spaces, exploration is done via policy. However, to solve this issue in the continuous action spaces, some extra noise will be added to the output of the actor-network. In the DDPG paper, the Ornstein-Unlenbeck process is used to add noise to the actions. However, in the implementation of this project, a simple Gaussian noise will be used.

The agent randomly samples states from memory, and then the actor-network is used to determine which action it thinks it should take based on those states. Then, the outputted actions form the actor-network and the sates from the memory will be plugged into the critic network to get the value for the critic network. The gradient of the actor network is taken

$$J(\theta) = \mathbb{E}[Q(s,a)|s = s_t, a_t = \mu(s_t)]$$
$$\nabla_{\theta^\mu} J(\theta) \approx \nabla_a Q(s,a) \nabla_{\theta^\mu} \mu(s|\theta^\mu)$$

Critic Network - In this network, the updated Q value is obtained by the Bellman equation, as shown below. In this algorithm, states, new states, actions, and rewards are randomly sampled. Then the new states are plugged in the target actor-network to determine actions, and finally, those actions are plugged into the target critic network to obtain target y, and this value will be multiplied by discount factor γ, and a reward for that timestep which was sampled from the memory will be added to it. The loss function error is calculated by computing the mean squared error between updated Q value and the original Q value obtained by plugging the state and action sampled from memory to the critic network.

$$y_i = r_i + \gamma \acute{Q}\left(s_{i+1}, \acute{\mu}\left(s_{i+1}|\theta^{\acute{\mu}}\right)|\theta^{\acute{Q}}\right)$$
$$Loss = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$$

Target Networks - At the beginning of the algorithm, both critic and actor networks will be initialized by some random parameters and directly copy them to target actor and target critic networks. However, in all other timesteps, the target networks will slowly track the learned network parameters via soft updates, as shown below.

$$\theta^{\acute{Q}} \leftarrow \tau^{\theta^Q} + (1-\tau)\theta^{\acute{Q}}$$
$$\theta^{\acute{\mu}} \leftarrow \tau^{\theta^\mu} + (1-\tau)\theta^{\acute{\mu}}$$
$$where\ \tau \ll 1$$

The EKF-DDPG algorithm and its block diagram is demonstrated as below.

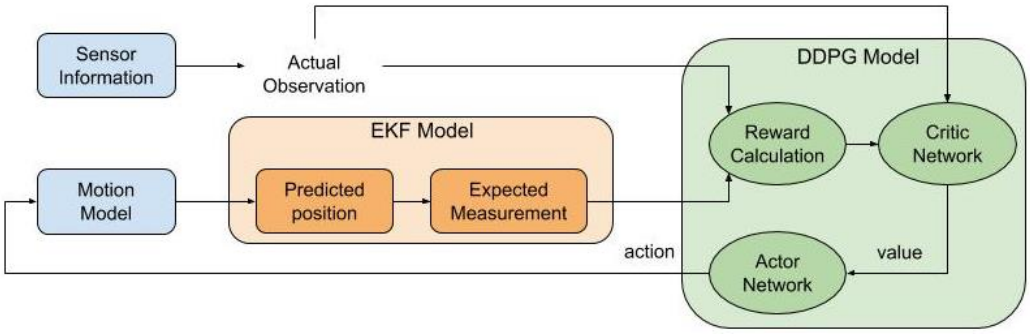

Figure 1 – The block diagram of the proposed EKF-DDPG algorithm

EKF-DDPG algorithm

1. Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
2. Initialize target network $\acute{Q}$ and $\acute{\mu}$ with weights $\theta^{\acute{Q}} \leftarrow \theta^Q, \theta^{\acute{\mu}} \leftarrow \theta^\mu$
3. Initialize replay buffer R
4. **for** episode $= 1, \mathrm{M}$ **do**
5. Initialize a random process $= \mathcal{N}$ for action exploration
6. Recieve initial observation state s_1
7. **for** $\mathrm{t} = 1, \mathrm{T}$ **do**
8. Selection action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
9.
10. action $a_t = [v_t \quad \omega_t]$
11. $\bar{\mu}_t = \mu_{t-1} + \begin{bmatrix} v_t \Delta T \cos\left(\phi + \frac{\omega_t \Delta T}{2}\right) \\ v_t \Delta T \sin\left(\phi + \frac{\omega_t \Delta T}{2}\right) \\ \omega_t \Delta T \end{bmatrix}$
12.
13. ***for*** *all observed features* $z_t^i = [r_t^i \quad \phi_t^i \quad s_t^i]^T$ *do*
14. $j = c_t^i$
15. $q = \left(m_{j,x} - \bar{\mu}_{t,x}\right)^2 + \left(m_{j,y} - \bar{\mu}_{t,y}\right)^2$
16. $\hat{z}_t^i = \begin{bmatrix} \sqrt{q} \\ \mathrm{atan2}\left(\mu_{j,y} - \bar{\mu}_{t,y}, \mu_{j,x} - \bar{\mu}_{t,x}\right) - \bar{\mu}_{t,\theta} \\ \mu_{j,s} \end{bmatrix}$
17. **end for**
18. $r_t = max\left(1 - \frac{\sum_i \| z_t^i - \hat{z}_t^i \|}{}, 0\right)$
19. observe new state s_{t+1}
20. Store transition (s_t, a_t, r_t, s_{t+1}) in R
21. Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
22. Set $y_i = r_i + \gamma \acute{Q}\left(s_{i+1}, \acute{\mu}\left(s_{i+1}|\theta^{\acute{\mu}}\right)|\theta^{\acute{Q}}\right)$
23. Update critic by minimizaing the loss: $L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$
24. Update the actor policy using the sampled policy gradient:
25. $$\nabla_{\theta^\mu} J(\theta) \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_t}$$
26. Update the target networks:
27. $$\theta^{\acute{Q}} \leftarrow \tau^{\theta^Q} + (1-\tau)\theta^{\acute{Q}}$$
28. $$\theta^{\acute{\mu}} \leftarrow \tau^{\theta^\mu} + (1-\tau)\theta^{\acute{\mu}}$$
29. **end for**
30. **end for**

4- Implementations

In the following section, the dataset is introduced, and a review of its features is given. In the next, the results of the EKF localization and EKF SLAM implementation is presented. In the next part, the result of the proposed reinforcement learning-based EKF algorithm for the problem of localization is provided.

4-1- Dataset Description

For this project, the UTIAS Multi-Robot Cooperative Localization and Mapping Dataset [8] is used. This dataset is constructed in a 15m×8m indoor space at Autonomous Space Robotics Lab (ASRL) at the

University of Toronto Institute for Aerospace Studies. It consists of 9 individual datasets, and its mainly used for multi-agent Indoor feature-based localization and SLAM problems. The 9th dataset includes some barriers that blocks the robot's view field. Each dataset includes five two-wheeled differential robots and 15 similar cylindrical tubes as landmarks. All robots have odometry, measurement, and ground-truth information in a separate text file.

All robots have odometry, measurement, and ground-truth information in a separate text file. Forward velocity along the x-axis of the robot's body frame and angular velocity about the robot's z-axis body frame is gathered at each timestep as odometry data.

At every timestep, the measurement data is captured by a camera installed on each robot. After processing the images, the range and bearing information from each robot to landmarks in its view field are recorded in a separate file. Also, the landmarks have specific barcodes printed on them so the robot can recognize their ID. In this case, it's completely known each measurement associates to which landmark, so the problem can be solved with known correspondences.

Ground-truth data is recorded for all robots and landmarks in the environment. It consists of ground-truth poses of the robot at every timestep and the mean ground-truth position of all (static) landmarks, as well as their standard deviations. For the localization problem, the landmarks' ground-truth is assumed to be known, but the robot's ground-truth will only be used to evaluate the performance of the proposed algorithm. For the case of the SLAM problem, both robots' and landmarks' ground-truth data will only be used for evaluation.

Also, in the 9th dataset gathered in the laboratory, some barriers were added to block the view of the robots.

For the purpose of this research, we only intend to solve the problem of localization and SLAM for single-agent environments. As a result, only the data from one of the robots is considered, and the rest is ignored for this project. Also, only the first dataset is used in this research.

To sum up, in this project, we have considered one robot and 15 landmarks. We have access to each robot's odometry, measurement, and ground-truth data at each timestep and information about all the landmark's ground-truth and their barcodes.

4-2- Implementation results of localization and SLAM with EKF

4-2-1 Localization with EKF

Parameter initialization

The process and measurement noise that was in this implementation are as below. M_t and Q_t are the process and measurement noise respectively. It is also worth mentioning that these parameters can be further-tuned to improve the performance of the EKF algorithm.

$$alpha = [0.2 \quad 0.2 \quad 0.09 \quad 0.08]$$

$$M_t = \begin{pmatrix} \alpha_1 v_t^2 + \alpha_2 \omega_t^2 & 0 \\ 0 & \alpha_3 v_t^2 + \alpha_4 \omega_t^2 \end{pmatrix}$$

$$\sigma_r = 2, \sigma_\phi = 3, \sigma_s = 1$$

$$Q_t = \begin{bmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\phi^2 & 0 \\ 0 & 0 & \sigma_s^2 \end{bmatrix}$$

Results

Both the ground-truth trajectory of the robot and its estimated trajectory path is shown in the figure below. The ground-truth path taken by the robot and its estimated path is illustrated by the red and green line, respectively.

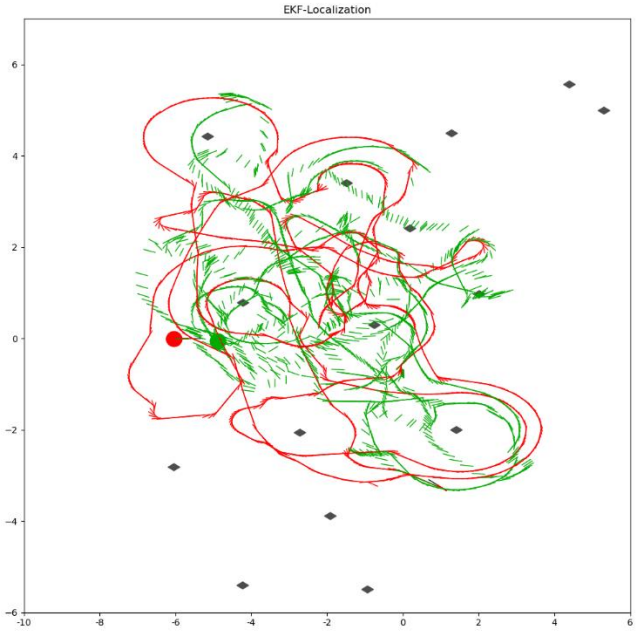

Figure 2: the ground-truth and estimated trajectory of the robot with EKF in the localization problem

Kalman gain

Kalman gain is computed at each timestep using the equations below.

$$S_t^i = H_t^i \bar{\sum}_t \left[H_t^i\right]^T + Q_t$$

$$K_t^i = \bar{\sum}_t \left[H_t^i\right]^T \left[S_t^i\right]^{-1}$$

The last computed Kalman gain is as below.

$$K = \begin{bmatrix} -0.00447702 & -0.01111291 & 0 \\ 0.02537575 & -0.00318565 & 0 \\ 0.00598972 & -0.01386602 & 0 \end{bmatrix}$$

Performance evaluation - Path loss

The robot's ground-truth is used to evaluate the performance of the proposed algorithm. In the code, we have calculated the error in the whole trajectory as follows. The error at each timestep is defined as the

distance between the robot's actual position at that timestep and the estimated robot's position. The average sum of this error in all timesteps is calculated in the code and is demonstrated by the $path\_loss$ variable. The average sum of the error in all timesteps is computed to be approximately 1.02313

$$average\ path\ loss = 1.02313$$

4-2-2 SLAM with EKF

Parameter initialization

The same as with localization, the process, and measurement noise, M_t and Q_t are defined as below, and these parameters can also be further-tuned to improve the performance of the EKF algorithm.

$$alpha = [0.1 \quad 0.01 \quad 0.18 \quad 0.08]$$

$$M_t = \begin{pmatrix} \alpha_1 v_t^2 + \alpha_2 \omega_t^2 & 0 \\ 0 & \alpha_3 v_t^2 + \alpha_4 \omega_t^2 \end{pmatrix}$$

$$Q_t = \begin{bmatrix} 11.8 & 0 \\ 0 & 0.18 \end{bmatrix}$$

Results

As already mentioned, the ground-truth path taken by the robot and its estimated path is illustrated by the red and green line in the figure below, respectively.

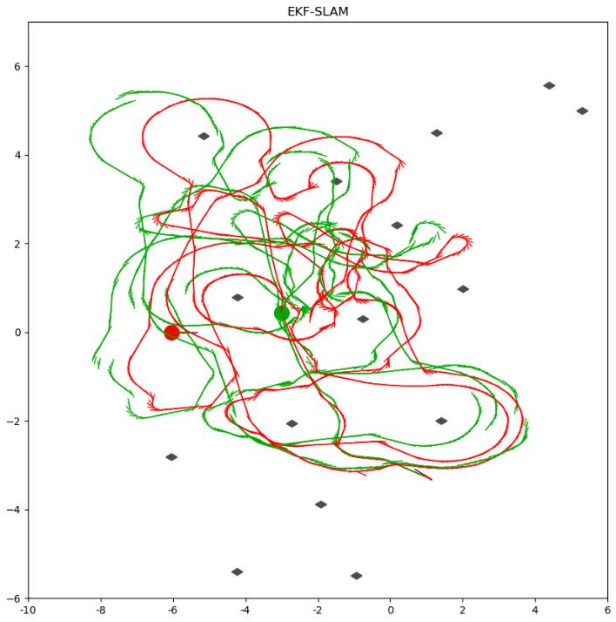

Figure 3: the ground-truth and estimated trajectory of the robot with EKF in the SLAM problem

Kalman gain

Kalman gain is computed at each timestep using the equations below.

$$S_t^i = H_t^i \bar{\sum}_t \left[H_t^i\right]^T + Q_t$$

$$K_t^i = \bar{\sum}_t \left[H_t^i\right]^T \left[S_t^i\right]^{-1}$$

The last computed Kalman gain is a 33×2 matrix, which is calculated in the code (The value is printed in the Jupyter notebook).

Performance evaluation - Path loss

The average sum of the distance between the robot's actual position and the estimated robot's position in all timesteps is calculated in the code and is computed to be approximately 0.40465.

$$average\ path\ loss = 0.40465$$

Performance evaluation - Land loss

In addition to the robot's position, the landmarks' positions are also estimated in the SLAM problem. To compute the error in the estimated landmarks' position, the landmark ground-truth data is used. This error is determined by calculating the difference between the landmarks' ground-truth positions and their respective estimated positions. The average sum of this error is demonstrated by land loss, which is computed to be 1.25568.

$$average\ land\ loss = 1.25568$$

4-3- Implementation results of localization with EKF and DDPG

Parameter initialization

In the DDPG algorithm, four different neural networks were used, two of which were target networks that are just time-delayed copies of their respective original networks. Both actor and critic networks contain three layers. The first two layers of both networks have 512 neurons. The critic network's last layer has one neuron. However, the actor network's last layer has two neurons representing the dimension of the action space.

The ReLU activation function is used for the critic and actor network's first two layers. For the last layer of the critic-network, the linear activation function was implemented, but for the last layer of the actor-network, a Tanh activation was used. The reason behind this is that the predicted action needs to be in a specific range, so a Tanh activation function is used for that purpose.

In addition, the maximum number of timesteps before termination is considered 100.

Performance evaluation - Path loss

For the implementation of the reinforcement learning-based EKF on the localization problem, the dataset is divided into two parts for training and evaluation. As already mentioned, this data is resampled at 50 HZ, and we have data for 75000 timesteps. 60000 first of samples have been used for training and the rest for evaluation.

The same as the previous parts of the implementation, the average sum of the distance between the robot's actual position and the estimated robot's position in all timesteps is calculated for evaluation and obtaining the algorithm's accuracy.

For the training and evaluation part, this average sum was calculated to be 5.71358 and 5.39422, respectively.

$$average\ path\ loss\ for\ training = 5.17358$$
$$average\ path\ loss\ for\ evaluating\ = 5.39422$$

The code for the proposed EKF-DDPG algorithm is implemented, and it's working. However, the agent was not trained properly, and the results did not converge, so the obtained results need more refinement to be accurate. This could be due to a few reasons.

First of all, the dataset that was used didn't include enough data for reinforcement learning. For reinforcement learning platforms to yield accurate results, there is a need for larger data. Also, as mentioned before, this dataset doesn't include measurement for every timestep, and to adapt to this condition, the last existing measurement was assumed to be the measurement of the current timestep. This could lead to non-accurate results.

Moreover, the reward function might need more tuning and refinement. The other parameters in the EKF-DDPG algorithm, such as the structure of the neural network the number of the layers used, and the number of the neurons in the DDPG part of the algorithm, might need tuning, too. In the next part, I have mentioned several ways to improve the obtained results here.

5- Plans and Next Steps

In the following section, I have presented several proposals and suggestions to continue this project and further improve the results and performance of the models for both localization and SLAM problems.

5-1- Continuing the implementations of EKF and EKF-DDPG for localization and SLAM

SLAM with RL-EKF

For continuing with this project, we can extend the proposed EKF-DDPG algorithm to work with the SLAM problem too. We probably have to extend the state vector to include the coordinates of landmarks and adjust the reward function accordingly.

Reinforcement learning for Continuous tasks (the introduction of the average reward and memory component)

The task of localization is continuous in nature, but for simplicity, we redefined it into an episodic task. The significant thing in continuous tasks is that the agent considers the same value for both delayed and instant rewards, so there is no need to use the discount factor γ in our formula because rewards at all timesteps are valued the same. In continuous tasks, the reward is defined differently, and the algorithm needs to make use of the average reward. The reward is defined as the difference between the instant reward and the average reward that was received in previous timesteps up till now. The reason for this kind of definition is that, in episodic tasks, by using the discount factor, the accumulative reward won't approach infinity. To compensate for not using a discount factor in continuous tasks, the average reward is introduced, so the accumulated reward won't reach infinity. By the use of average reward, only the instant rewards that are above the average value can update the weights positively. The differential semi-gradient Sarsa algorithm takes advantage of the concept of the average reward for continuous tasks [8].

Moreover, adding a memory component such as recurrent neural networks to a continuous task might be useful. Deep Recurrent Q-Networks (DRQN) [9] and Recurrent DDPG [10] both use a memory component in their reinforcement learning platform.

The introduction of the average reward and a memory network such as a recurrent neural network and LSTMs can be valuable to improve the obtained results for both the problem of localization and SLAM with RL-EKF.

Interpolation

As was discussed in the reward definition part of the report, the dataset doesn't include measurements at all timesteps, so we had to use the last existing actual measurement in the reward design. However, we can interpolate between the two measurements in a row for every timestep to improve the performance of the proposed algorithm.

Kalman filter tuning (noise)

The process and measurement noise in the motion model and measurement model in the Kalman filters have some parameters that need to be hand-tuned to obtain better and more accurate performance. The quality of our filter-tuning can be evaluated by comparing the difference between the ground-truth position of the robot and the landmarks with their respective estimated positions. This evaluation can be performed using the sum of the squared errors in the code.

RL Parameters tuning

The parameters used in the reinforcement learning algorithm, such as the maximum number of timesteps in each episode, the structure of both the actor and critic neural networks, the number of the layers and neurons, can also be tuned to further improve the performance of the proposed algorithm.

Also, the reward that was proposed in the EKF-DDPG algorithm might need a bit more tuning in order to adapt to the localization problem at hand.

Testing the proposed EKF-DDPG algorithm in larger datasets

As already discussed in the last section, reinforcement learning algorithms need more data in order to train well. And, it also didn't include measurements for all the timesteps, so we had to use the last existing measurement as the measurement of the current timestep. To continue this project, we can implement the proposed EKF-DDPG algorithm on more complicated datasets such as the Victoria park dataset, Complex Urban Dataset, San Francisco Landmark Dataset, Oxford Robotcar Dataset, and Rosario Dataset as well.

Implementing for the problem of unknow correspondences

In this project, we have assumed the associations between each measurement and landmarks are known, meaning that we know that each measurement corresponds to which landmark. In a more complicated scenario, we can assume that we don't have any prior information about the landmark's correspondences with measurement data. This problem can be tackled through the use of maximum likelihood data association. In this case, the robot computes all the probabilities of each measurement being associated with each landmark and chooses the highest probability as the landmark with that it's measuring the range and bearing data from. The maximum likelihood data association can be integrated with both conventional EKF algorithm and reinforcement learning-based EKF algorithm.

Implementing in multi-agent systems

As discussed in the dataset description, this dataset consists of information for five robots at each timestep, and it's mostly intended for multi-agent systems. So, we can extend and implement the proposed algorithm for multi-agent systems for further continuation of the project.

5-2 More research directions for exploration

In the following section, I have gathered several ideas that are worth further investigation and refinement in order to be beneficial and useful in the continuation of this research. These ideas consist of new approaches to SLAM, Kalman filters, Reinforcement learning, the combination of these methods together, more complex datasets, and useful applications for robotic simulation.

5-2-1 Ideas based on new models for localization or SLAM

Active SLAM with RL

In the problem of SLAM, the task is to simultaneously estimate the robot's pose and create a map of the environment. However, in Active SLAM, it is aimed to find the control commands for the robot to build the map of the environment faster. Reinforcement learning platforms can be used to find the optimal control commands. For the continuation of this research, we can combine Active SLAM with RL to further improve the accuracy, optimality, and speed of our algorithm [4].

FastSLAM

The EKF based SLAM method has a few disadvantages, such as quadratic complexity and sensitivity to failures in data association, which makes it inefficient to implement in large real-world environments. Also, implementing it into maps with highly unknown correspondences would lead to problems. In [9], the FastSLAM approach is introduced that samples over both robot paths and data associations and

implements both full SLAM and particle filters to better estimate the robot's and landmarks' positions. Also, FastSLAM can be used in dynamic environments. We can use this method and combine it with reinforcement learning to obtain better results.

5-2-2 Ideas based on new models for the RL algorithm

Recurrent neural network in the RL algorithm (Deep Recurrent Q-Networks or Recurrent DDPG)

The addition of a memory component such as LSTM in the neural network architectures can further improve the agents' performance in the environment. Both Deep Recurrent Q-Networks (DRQN) [10] and Recurrent DDPG [11] tackle the problem of partially observable environments by adding a recurrent neural network (RNN) to the structure of their algorithms. As is the case of many real-world applications, most environments are partially observable. As a result, these methods can be useful in these situations for this research.

Reward function Design (map completeness and information gain)

For learning to complete a map of the environment, an efficient reward function is necessary. We can design a reward function that considers map completeness and information gain (entropy difference in each timestep) and reward the agent based on these two features. In [2], it was shown that a reward function based on these two features performed much better than a sparse reward that only rewards the agent positively when it completes the map. They have used map completeness and information gain in different reward functions separately and compared their performance. In addition to individually evaluating them, we can design a new reward that takes into account both features and see if it improves performance.

Multi DDPG

The action space in this task is two-dimensional, but in many cases in the robotic application, many of these robots have multiple degrees of freedom, and their action space is very high-dimensional. So, a DDPG algorithm might not be useful in these situations. However, there is a variation of the reinforcement learning DDPG method that is proved to perform much better for multiple and high dimensional tasks in continuous action spaces [12]. We also can implement this algorithm and assess the performance of our algorithm.

5-2-3 Ideas based on the new models for Kalman filters

Another Idea for the RL-EKF algorithm

Since Kalman filters are used both as a way to estimate the state and approximate the weights in the neural networks, we can propose another idea to solve the localization and SLAM problem. We can do localization and SLAM with RL algorithms and use extended Kalman filters as an optimizer in the neural networks of our RL algorithm (DDPG) as a way to reduce the uncertainty in the weights of the neural networks**.**

Using other Kalman filter methods

1. **Mean Extended Kalman Filter (MEKF)**

The linearization made in the EKF method can sometimes lead to large errors and even divergence, specifically in real-world applications. There are some methods in the Kalman filter family that can lower this error, such as Mean Extended Kalman Filter (MEKF). This method performs much better than EKF, UKF, and even their iterative-based variations in non-linear systems. It is shown that it can effectively decrease computational costs, and at the same time, increase the accuracy of the position estimation in the SLAM problem [13]. In this project, we can combine the MEKF method with reinforcement learning and further reduce the position estimation error.

2. **Particle Filter Estimation**

This method uses a finite set of weighted particles to estimate the robot's position, and by utilizing a probabilistic observation model, a weight factor is assigned to each particle. During the next steps of the algorithm, the particles with smaller weights get omitted. This helps the filter to better estimate the real position of the robot [4].

For the problem of SLAM, a Rao-Blackwellized particle filter (RBPF) can be implemented. First, it estimates the robot's pose and then estimates the map given known poses, which is done by using assuming the map as an occupancy grid [3]. We can also implement this method into this research and even combine it with reinforcement to generate better and more accurate results.

5-2-4 Exploring other datasets and applications

UTIAS 9th Dataset

In the 9th dataset gathered in the University of Toronto's laboratory, some barriers were added to block the robots' view. Since this will lower the number of measurements in the environment, it will further complicate the problem of localization and SLAM. We can also implement our algorithm on this dataset and evaluate its performance and change the algorithm to better perform in environments with obstacles, as this is the case in most real-world applications.

Victoria park dataset

Victoria park dataset is one of the benchmarks for the problems of localization and SLAM, and since it is an outdoor dataset, it's slightly more complicated than the dataset that was used in this project, so it's worth trying the algorithms proposed in this research on this dataset considering many real-world applications are outdoor. Also, reinforcement learning algorithms usually need more data to perform better, so these kinds of datasets can be useful for this research.

Complicated Datasets

As mentioned above, reinforcement learning-based algorithms need lots of data and exploration in the environment to perform well. So, for the continuation of this project, I propose the use of more complicated datasets such as Complex Urban Dataset, San Francisco Landmark Dataset, Oxford Robotcar Dataset, Rosario Dataset and so on.

Robotic Simulation Software

We can simulate the algorithm in more complicated programs such as Webots, CoppeliaSim, Robot Operating System (Ros), and Gazebo, in which we can simulate different environments, robots, and sensors ourselves.

References

[1] L. Paull, S. Saeedi, M. Seto, and H. Li, "AUV Navigation and Localization-A Review."

[2] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," pp. 1–9, 2013, [Online]. Available: http://arxiv.org/abs/1312.5602.

[3] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," Sep. 2016, Accessed: Jan. 03, 2021. [Online]. Available: https://goo.gl/J4PIAz.

[4] N. Botteghi, B. Sirmacek, R. Schulte, M. Poel, and C. Brune, "Reinforcement learning helps slam: Learning to build maps," in *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, Aug. 2020, vol. 43, no. B4, pp. 329–336, doi: 10.5194/isprs-archives-XLIII-B4-2020-329-2020.

[5] J. A. Placed and J. A. Castellanos, "A Deep Reinforcement Learning Approach for Active SLAM," *Appl. Sci.*, vol. 10, no. 23, p. 8386, Nov. 2020, doi: 10.3390/app10238386.

[6] S. D.-C. Shashua and S. Mannor, "Deep Robust Kalman Filter," Mar. 2017, Accessed: Jan. 03, 2021. [Online]. Available: http://arxiv.org/abs/1703.02310.

[7] "Probabilistic Robotics | The MIT Press." https://mitpress.mit.edu/books/probabilistic-robotics (accessed Jan. 03, 2021).

[8] "Autonomous Space Robotics Lab: MR.CLAM Dataset." http://asrl.utias.utoronto.ca/datasets/mrclam/index.html (accessed Dec. 31, 2020).

[9] "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association - The Robotics Institute Carnegie Mellon University." https://www.ri.cmu.edu/publications/fastslam-a-factored-solution-to-the-simultaneous-localization-and-mapping-problem-with-unknown-data-association/ (accessed Jan. 04, 2021).

[10] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable MDPs," in *AAAI Fall Symposium - Technical Report*, Jul. 2015, vol. FS-15-06, pp. 29–37, Accessed: Jan. 01, 2021. [Online]. Available: www.aaai.org.

[11] "(1) (PDF) Recurrent Network-based Deterministic Policy Gradient for Solving Bipedal Walking Challenge on Rugged Terrains." https://www.researchgate.net/publication/320296763_Recurrent_Network-based_Deterministic_Policy_Gradient_for_Solving_Bipedal_Walking_Challenge_on_Rugged_Terrains (accessed Jan. 01, 2021).

[12] "Multi-task deep reinforcement learning for continuous action control | Proceedings of the 26th International Joint Conference on Artificial Intelligence." https://dl.acm.org/doi/10.5555/3172077.3172350 (accessed Jan. 01, 2021).

[13] W. Zhou, C. Zhao, and J. Guo, "The study of improving Kalman filters family for nonlinear SLAM," *J. Intell. Robot. Syst. Theory Appl.*, vol. 56, no. 5, pp. 543–564, 2009, doi: 10.1007/s10846-009-9327-9.

[14] "UTIAS-practice/common at master · 1988kramer/UTIAS-practice." https://github.com/1988kramer/UTIAS-practice/tree/master/common (accessed Jan. 04, 2021).