Single Beam EKF SLAM on a Mobile Robot

Harrison Espino
espinoh@uci.edu

Abstract

This project aimed to implement Extended Kalman Filter Simultaneous Localization and Mapping (EKF-SLAM) on OpenBot, a low cost mobile robot platform that utilizes a single beam sonar sensor for range finding. This was first tested in simulation using an agent that possesses the same movement and measuring capabilities as the OpenBot. The same implementation was then tested on real robot data collected in a number of different environments. Results indicate that sparse landmark observations in the form of single distance measurements allow for reasonably accurate landmark and pose estimations in simulation. In practice, the robot captures the general shape of landmarks placed, but pose estimations were very inaccurate. Due to odometry error and unforeseen trajectory changes from wheel slippage, cheaper platforms may require a more complicated motion model or additional motion sensors.

1 Introduction

In order to facilitate the use of SLAM and other navigational algorithms, robotic platforms often utilize an extensive sensor suite. This includes LiDAR or RGB-D cameras for determining distances to landmarks as well as IMU or gyroscopic sensors for precise odometry measurements. Many research oriented robotic platforms invest in high quality sensors that produce accurate high dimensional data, resulting in a very expensive overhead cost.

By contrast, recent work has tried to leverage the use of Android phones and widely available micro controllers such as Arduino or Raspberry Pi in order to lower the cost. In this work, we utilize the OpenBot [Müller and Koltun, 2021], which uses a 3-D printed chassis containing an Arduino connected to a Google Pixel XL mounted above the robot. The only range-finding sensor consists of a single-beam sonar, which returns a singular distance measurement for objects in a narrow cone in front of the robot. The robot platform is shown in Fig. 1.

The primary goal of this work is to analyze the feasibility of the EKF SLAM algorithm for low cost mobile robotic platforms. First, this will be done in simulation by replicating the movement and measurement behavior of the OpenBot in an ideal environment. Then, this SLAM system will be used on data collected on the OpenBot in multiple environments. This work concludes by analyzing the performance and shortcomings of this SLAM implementation, and suggests a future direction for SLAM in low cost settings.

1.1 Related Work

SLAM is a very well-studied problem in robotics that has been approached in a number of different ways. Methods such as ORB-SLAM or MONO-SLAM function only on monocular camera data. This is achieved through tracking visual features across multiple viewpoints to triangulate the position of the robot and landmarks [Mur-Artal et al., 2015] [Davison et al., 2007]. In order to replicate the behavior and energy efficiency of natural organisms, methods such as RAT-SLAM model spiking neurons that mimic place and head direction cells found in the rodent hippocampus [Milford et al.,

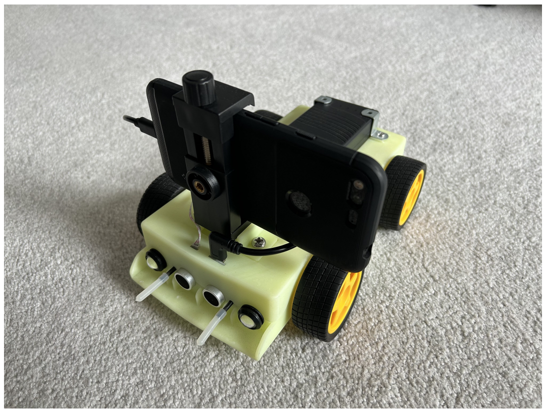


Figure 1: The OpenBot robot platform.

2004]. For limited sensor capabilities, vision based solutions taking advantage of the mounted phone camera data may serve as an alternative approach to traditional sensor fusion SLAM.

SLAM as it relates to learning in graphical models is generally divided into two categories. EKF SLAM and similar methods iteratively estimate a single posterior probability distribution for the robot and landmark positions (more details on EKF SLAM in a later section) [Smith and Cheeseman, 1986]. This allows for the use of these algorithms in an online setting, however it does not implement loop closure to retroactively correct path estimates. GraphSLAM and similar methods instead represent the entire pose and map history as a factor graph under certain motion and measurement constraints [Thrun and Montemerlo, 2006]. Loop closure is inherently implemented in this method, however the high computational cost generally confines this to an offline setting. The relatively weak computational power of our hardware suggests EKF SLAM is a more favorable option.

Lastly, previous work has designed robotic navigation algorithms with the intent of using them on low-cost platforms. Zou et al. [2020] used energy efficient neuromorphic hardware to implement online terrain classification for an android based robot. Swarm robotics, which focuses on cooperation between multiple robots, often employ several cheap "micro-robots", which can be centimeters in length [Na et al., 2021]. The OpenBot platform has been used for indoor localization, taking advantage of the phone's IMU and Wi-Fi round trip time (RRT) encoders [Zhou et al., 2023]. These works show promising applications for cheaper robotic hardware.

1.2 SLAM

In this section, EKF SLAM is explained in greater detail. We represent the the world as the hidden Markov model illustrated in Fig. 2. In this case, z_t represents the state of the world at time t. "State" consists of the location and rotation of the robot (also known as the "pose", represented as x, y, θ) and the location landmarks. This means that the state can be represented as a vector of size $2n + 3$, where n is the number of landmarks being tracked. Some implementations of EKF SLAM allow for a variable state size as the robot observes new landmarks, however for this implementation the number of unique landmarks is initially defined. In the context of the hidden Markov model, z_t is *hidden*.

Observable variables are denoted as u_t and x_t, and represent the robot's movement commands and sensor data, respectfully. Both of these may take different forms depending on the robot's drive configuration and available sensors- it is partially the goal of EKF SLAM to model how a specific robot's movement and sensor data interact with the state estimate. In general, the movement u_{t-1} effects the resulting state z_t, which then produces the sensor observation x_t. The objective of SLAM is then to predict the state z_n given z_{n-1}, u_{n-1} and x_n.

EKF SLAM approaches this as a problem of probabilistic estimation. Rather than representing the best state prediction as a singular vector, EKF SLAM holds a probability distribution across all possible state configurations. In Kalman filtering the state is estimated to be Gaussian, with mean z and covariance Σ. The Extended Kalman filter then estimates these parameters in two distinct steps known as the *prediction step* and the *update step*.

In the prediction step, the state estimate z_n and corresponding covariance Σ_n are updated using u_{n-1} according to the following equations:

$$\bar{z_t} = f(z_{t-1}, u_{t-1}) \tag{1}$$
$$\bar{\Sigma_t} = F_t \Sigma_{t-1} F_t^T + R_t \tag{2}$$

$f(z, u)$ is the motion model which, as previously mentioned, is meant to translate the movement command u to a corresponding change in state. In contrast to a regular Kalman filter which represents motion dynamics linearly in the form of a transition matrix, the motion model of an extended Kalman filter allows for non-linear movement. The covariance estimate is then updated according to the Jacobian matrix of this motion model, denoted F_t as well as a modeled noise matrix for motion R_t.

This estimate (denoted $\bar{z_n}$ and $\bar{\Sigma_n}$) is then refined using the observed measurement x_n. The update step is carried out from the following:

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1} \tag{3}$$
$$z_t = \bar{z}_t + K_t(x_t - h(\bar{z}_t)) \tag{4}$$
$$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t \tag{5}$$

Similar to the motion model, the Extended Kalman filter uses a generative measurement model $h(z)$ which returns the predicted measurement given the current state estimate. The Kalman gain K_t in Eqn. (3) is calculated using the estimated covariance, the Jacobian matrix of the measurement model H_t and a modeled noise matrix for measurements Q_t. This Kalman gain together with the difference between the actual measurement and predicted measurement is added to the state estimate from the prediction step to generate a final estimate and covariance.

Applying EKF SLAM to a robotic platform requires a unique implementation of the motion model, measurement model, and the corresponding noise matrices according to the supplied motion commands and available sensors. Specific implementation of these for the OpenBot are detailed in the Methods section.

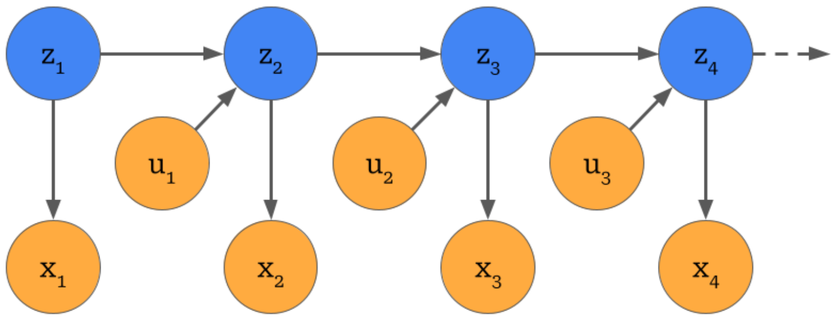


Figure 2: The hidden Markov model describing the evolution of the world state in SLAM problems.

2 Methods

In this section, the specific details of the EKF SLAM implementation as it relates to the motion and measurement capabilities of the OpenBot are explained in greater detail. Then, the methods for using this SLAM formulation in simulation as well as the procedure for collecting real data is covered.

2.1 SLAM Implementation

The OpenBot receives movement commands in the form of specifying a designated wheel rotation per minute (rpm). The chassis operates using differential drive, meaning the left and right wheels on each axis are controlled independently from each other. Turns are executed by specifying the wheel rpm of the turn direction to be smaller than the opposite direction. To convert these drive commands into a form that could be used to express a change in position over time, movement commands are converted into linear and angular velocity through the following process.

Algorithm 1: Conversion of wheel rpm to linear and rotational velocity

```
1 function rpmToVel ($rpm_l, rpm_r, WHEELRAD, WHEELBASE$);
  Input  : Left and right rotations per minute in radians/second, wheel radius, and wheel base
           distance
  Output : Motion command $u$ converted to linear and angular velocity
2 $vel_l = rpm_l * 2\pi * WHEELRAD$
3 $vel_r = rpm_r * 2\pi * WHEELRAD$
4 $v = (vel_l + vel_r)/2$
5 $w = (vel_r - vel_l)/WHEELBASE$
6 $u = [v, w]$
```

This movement command is supplied together with a time step length dt and the previous state estimation to create the following motion model.

Algorithm 2: OpenBot motion model

```
1 function motionModel ($z, u, dt$);
  Input  : Current state estimate, motion command, and time step length
  Output : Next state estimate $\bar{z}$
2 $d = u_v * dt$
3 $r_i = u_w * dt$
4 $\Delta x = d * cos(\theta_t + r_i)$
5 $\Delta y = d * sin(\theta_t + r_i)$
6 $\Delta\theta = r_i + r_f$
7 $\Delta z = [\Delta x, \Delta y, \Delta\theta]$
8 $\bar{z} = z + \Delta z$
```

Note that the movement command includes a "final" rotation r_f in order to introduce non-linear motion into the model. When motion error is introduced, this can account for changes in pose due to friction or wheel slippage. Additionally, this pseudocode assumes that an appropriate amount of of zeros are appended to the change in state Δz in order to match the number of landmarks. The Jacobian matrix F and motion noise R_t are modeled as the following:

$$F = \begin{bmatrix} 0 & 0 & -d * sin(\theta_t + r_i) \\ 0 & 0 & d * cos(\theta_t + r_i) \\ 0 & 0 & 0 \end{bmatrix} \tag{6}$$

$$R_t = \begin{bmatrix} 0.05 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} \tag{7}$$

Greater error is assumed to occur for rotations due to the concern of wheel slippage using differential drive on surfaces with high friction. Together, these three components form the prediction step of the model.

For the update step, we first convert our measurement model into estimated x and y coordinates given the measured distance and the current estimated pose of the robot.

```
Algorithm 3: Conversion of sonar measurement to landmark location estimate
1 function sonarToLoc ($d, x_r, y_r, \theta_r$);
  Input  : Distance measurement by the sonar $d$, estimated pose of the robot $x_r, y_r, \theta_r$
  Output : Measurement converted to landmark $x_{lm}$ and $y_{lm}$ coordinates
2 $\Delta x = d * cos(\theta_r)$
3 $\Delta y = d * sin(\theta_r)$
4 $x_{lm} = x_r + \Delta x$
5 $y_{lm} = y_r + \Delta y$
```

The measurement model can then be formulated to get expected measurement of a landmark by proceeding backwards, converting estimated x and y coordinates to a sonar measurement.

```
Algorithm 4: OpenBot Measurement Model
1 function measurementModel ($x_{lm}, y_{lm}, x_r, y_r, \theta_r, \theta_{thresh}$);
  Input  : Estimated location of a landmark, estimated pose of the robot, angle thershold
  Output : Expected sonar measurement $d$ and $a$
2 $\Delta x = |x_r - x_{lm}|$
3 $\Delta y = |y_r - y_{lm}|$
4 $\gamma = \Delta x^2 + \Delta y^2$
5 $a = \arctan(\frac{\Delta y}{\Delta x}) - \theta_r$
6 $d = \sqrt{\gamma}$ if $|a| < \theta_{thresh}$ else $d = 0$
```

Note that, similar to the movement model, an additional parameter is added here that is does not occur in measurements. Due to the all-or-nothing nature of a single point sonar measurement, the measurement model also outputs an expected angle of measurement in order to make proper adjustments when comparing the expected and observed measurement. This can also account for noise in measurements due to the sonar not being perfectly represented by a beam, but rather a narrow cone in practice. Lastly, the Jacobian matrix of the measurement model H and measurement noise Q_t are modeled as the following (For ease of understanding, same notation $\Delta x, \Delta y, \gamma$, and d is used):

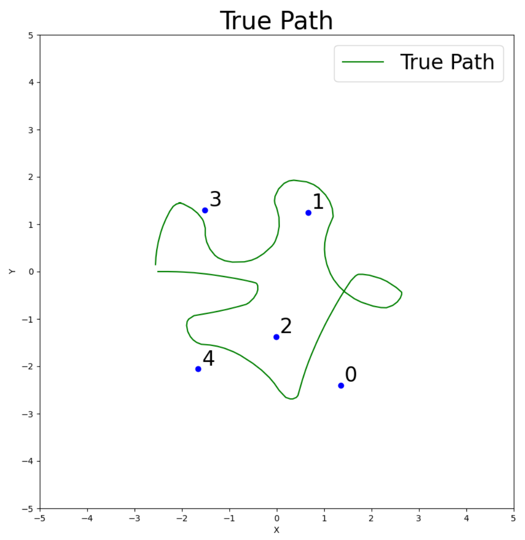


Figure 3: The true path taken by the robot in simulation.

$$H = \begin{bmatrix} \frac{-d\Delta x}{\gamma} & \frac{-d\Delta y}{\gamma} & 0 & \frac{d\Delta x}{\gamma} & \frac{d\Delta y}{\gamma} \\ \Delta y & -\Delta x & -\gamma & -\Delta y & \Delta x \end{bmatrix} \tag{8}$$

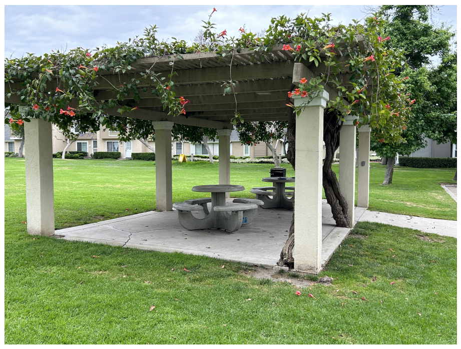
$$Q_t = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.01 \end{bmatrix} \tag{9}$$

In this case, we assume relatively low error for both the distance and angle measurements (angle measurements will always be 0). We also assume that, upon observing a landmark, the landmark correspondence is known. The motion model, measurement model, and noise covariance matrices supply the necessary components for EKF SLAM on the OpenBot. Next, the experimental protocols for simulated and real experiments are described.

2.2 Simulation

To test if this formulation of EKF SLAM can successfully perform the task of localization and mapping, it was tested in a simulated environment using an agent which responds to movement commands and makes observations in the same manner as the OpenBot. Landmarks were randomly placed in the environment, represented as singular (x, y) coordinates. A series of movement commands were hand designed in order to simulate varying movement patterns (left and right turns, loops, abrupt stops), and were used as input to the agent. Movement commands were received at a time window of 0.1 seconds.

At each timestep, the simulated agent moved according to the left right wheel rpm. This was converted to linear and angular velocity as previously described, using a wheel radius of 0.03 meters and a drive base length of 0.15 meters. Gaussian noise was added to the robot's final position to simulate movement errors due to wheel slippage or friction. The robot then sensed the first landmark within a $\frac{\pi}{32}$ window projected outward from the robot. This was roughly measured as the range that the real OpenBot sonar would detect landmarks. The true path and placement of landmarks are shown in Fig. 3.


(a) Gazebo environment

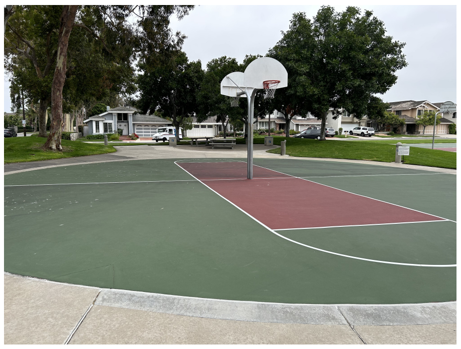
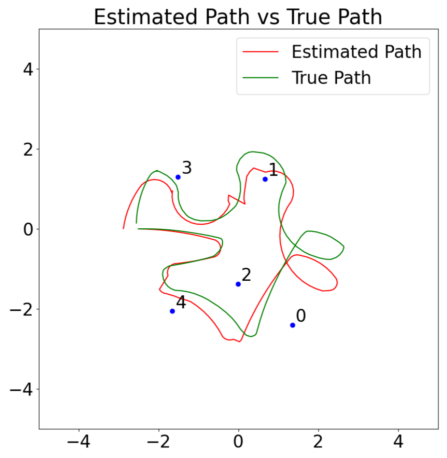

(b) Basketball court environment

Figure 4: Environments used for real robot data collection

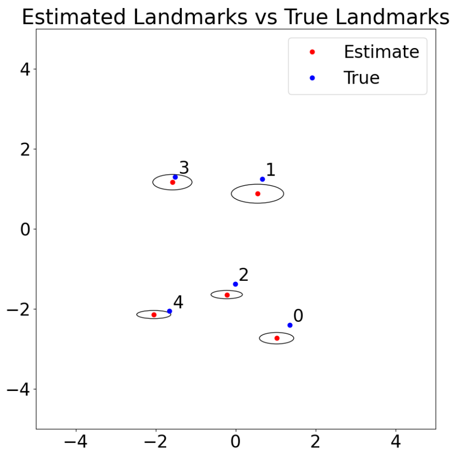
2.3 Robot Data Collection

For real data collection, the OpenBot was tested in a number of real world environments. First is the gazebo shown in Fig. 4a. Six pillars arranged in a rectangular pattern served as the landmarks for the OpenBot to detect. The OpenBot was driven in circles three times, observing each of the pillars once per lap. The terrain was notably very uneven due to cracks in the pavement. The second environment is shown in Fig. 4b. This was a basketball court which was much larger and featured smoother terrain. Five landmarks were placed evenly around the perimeter of the court, and the robot was similarly tasked with driving around the perimeter three times and observing each of the landmarks once per lap.

Both sonar measurements and motion commands were published at a rate of 0.1 seconds. This data was collected at regular intervals, as well as the current frame of the camera. In order to handle landmark correspondence, the data was preprocessed by hand by scanning through the camera frames and marking where a landmark was observed.



(a) Estimated agent pose against true pose.



(b) Estimated landmark locations against true locations.

Figure 5: Estimated agent pose and landmark location.

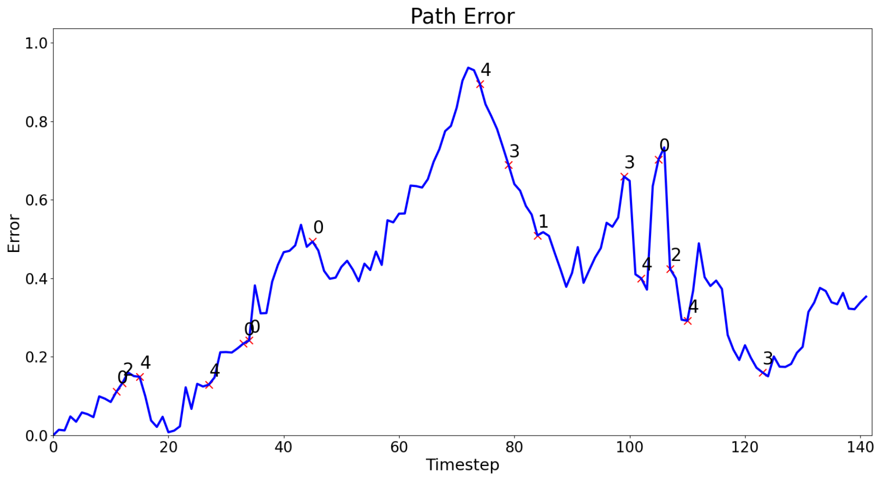


Figure 6: Squared error of the estimated path over time.

3 Results

3.1 Simulation

The estimated path of the agent against the true path, as well as the estimated locations of landmarks are shown in Fig. 5. In both cases, the agent is able to maintain a reasonably accurate map and pose estimate over time. In certain sections, such as near landmark "1", sudden jumps in the estimated path indicate significant adjustments to the state vector due to discrepancies in the measurement model versus the observed sensor readings.

This can be better interpreted using the squared path error plotted in Fig. 6. As expected, error in estimation reaches its peak at around timestep 70, in which the agent performs a loop during a long period without landmark observations. Error then begins to reduce as previous landmarks are observed, with notable reductions in error after the observation of landmarks 4, 3, and 0 following the peak error. This demonstrates that the formulation of EKF SLAM is making appropriate adjustments given observations, which is consistent with the qualitative analysis of the path and landmarks.

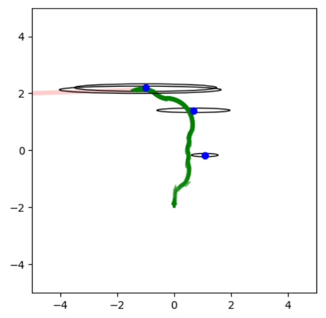
3.2 Real Data

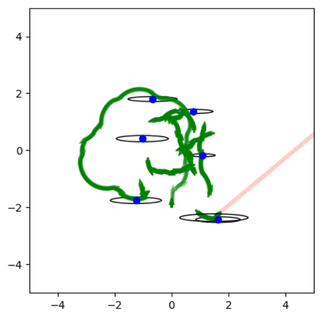
Using data collected in real environments, it was not possible to determine the true path for comparison. Instead, the path of the agent can be analyzed qualitatively as in the simulation results. The final path estimation history and landmark estimations are shown in Fig. 7 and 8

In the gazebo environment, the robot was able to determine the general rectangular shape of the pillars. Unfortunately, EKF SLAM was not nearly as successful in estimating the pose as in simulation. We can observe several large jumps in the robot pose after landmark observations, which suggest the movement model is not capturing the motion dynamics for the real robot. Fig. 7c shows the path and landmark estimates updating over early, middle, and late exploration timesteps.

During data collection, it was observed that the supplied motion command often did not match the actual motion of the robot. For example, a motion command of maximal left rpm and zero right rpm should ideally result in a stationary turn. However, due to the low torque of the drive mechanism combined with the friction from the ground, the right wheel would often begin to turn as well resulting in a wide turn instead.

The worse performance in the larger court environment supports the idea that errors are mainly present in the motion model (see Fig. 8c). Since distances between landmark observations are larger in this environment, EKF SLAM relies the motion model more to maintain a pose estimate without an update step. This means that errors in the motion model have a longer amount of time to accumulate.


(a) Pose and landmark history at timestep 100 (Roughly halfway through the first lap).


(b) Pose and landmark history at timestep 410 (Two laps completed).

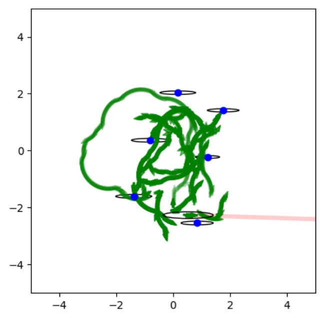
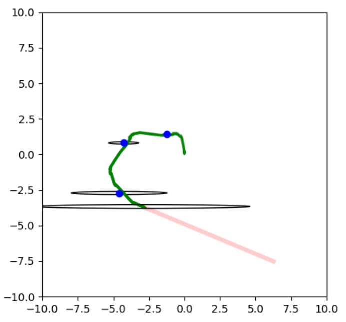
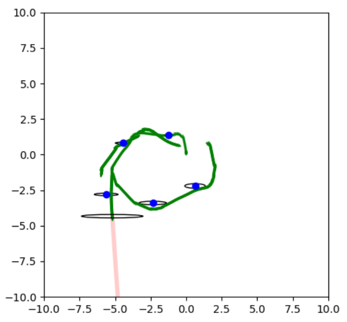

(c) Pose and landmark history at timestep 814 (Final timestep).

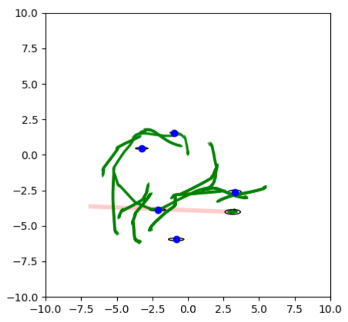
Figure 7: Examples of pose and landmark estimates for OpenBot data in the gazebo environment



(a) Pose and landmark history at timestep 260 (Roughly halfway through the first lap).



(b) Pose and landmark history at timestep 650 (Roughly halfway through the second lap).



(c) Pose and landmark history at timestep 1232 (Final timestep).

Figure 8: Examples of pose and landmark estimates for OpenBot data in the basketball court environment

4 Conclusion

In this work, EKF SLAM is configured for OpenBot, a low cost robotic platform with limited sensor capabilities. The motion model takes advantage of the option of non-linear motion dynamics by introducing a secondary rotation to model potential movement errors. Similarly, the measurement model adds an additional dimension of measurement angle to allow for proper pose adjustments. A simulated OpenBot is able to estimate the location of obstacles as well as the path taken without the accumulation of error.

However, from these results it's clear that several considerations need to be made to successfully move from simulation to real robotics applications. From observations during data collection, it is hypothesized that the robot may not be able to reach the desired rpm dictated by motion commands. This can be due the drive mechanism being unable to support certain wheel speed configurations given the friction of the ground. The inaccurate motion model subsequently caused error to accumulate in the path estimate over time.

To further investigate the use of EKF SLAM on low cost devices, future work could take advantage of other available sensors used for mobile phones (accelerometer, gyroscope, and magnetometer among others) to refine the motion model to be robust to differences in motion commands and actual execution. Given the sparsity of landmark measurements, it might also be necessary to choose paths which are intentionally designed to repeatedly observe previous landmarks. In either case, this work illustrates that EKF SLAM presents additional challenges when transitioning to real data.

5 Teammate Contribution and References Used

Implementation and writing were done by me (Harrison Espino) as the sole team member on this project. All parts of SLAM for simulation and real data were done in Python and was adapted from a mathematical description of EKF SLAM using odometry and a high dimensional LiDAR range finder described in Choset et al. [2005]. A very simple Android app to broadcast data using ROS from the OpenBot was also adapted from the original app provided by Müller and Koltun [2021], and the receiving script was implemented in Python as well.

References

Howie M Choset, Kevin M Lynch, Seth Hutchinson, George A Kantor, Wolfram Burgard, Lydia E Kavraki, and Sebastian Thrun. *Principles of robot motion*. Intelligent Robotics and Autonomous Agents series. Bradford Books, Cambridge, MA, May 2005.

Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007. doi: 10.1109/TPAMI.2007.1049.

M.J. Milford, G.F. Wyeth, and D. Prasser. Ratslam: a hippocampal model for simultaneous localization and mapping. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 1, pages 403–408 Vol.1, 2004. doi: 10.1109/ROBOT.2004.1307183.

Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. doi: 10.1109/TRO.2015.2463671.

Matthias Müller and Vladlen Koltun. Openbot: Turning smartphones into robots. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9305–9311, 2021. doi: 10.1109/ICRA48506.2021.9561788.

Seongin Na, Yiping Qiu, Ali E Turgut, Jiří Ulrich, Tomáš Krajník, Shigang Yue, Barry Lennox, and Farshad Arvin. Bio-inspired artificial pheromone system for swarm robotics applications. *Adaptive Behavior*, 29(4):395–415, 2021. doi: 10.1177/1059712320918936.

Randall C. Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986. doi: 10.1177/027836498600500404.

Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006. doi: 10.1177/0278364906065387.

Baoding Zhou, Zhiqian Wu, Zhipeng Chen, Xu Liu, and Qingquan Li. Wi-fi rtt/encoder/ins-based robot indoor localization using smartphones. *IEEE Transactions on Vehicular Technology*, 72(5):6683–6694, 2023. doi: 10.1109/TVT.2023.3234283.

Xinyun Zou, Tiffany Hwu, Jeffrey Krichmar, and Emre Neftci. Terrain classification with a reservoir-based network of spiking neurons. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2020. doi: 10.1109/ISCAS45731.2020.9180740.