Visual Inertial SLAM with an Extended Kalman Filter

William Argus[1]
Department of Electrical and Computer Engineering, UCSD
wargus@eng.ucsd.edu

I. INTRODUCTION

Simultaneous localization and mapping is a problem that involves determining the state and environment of a robot at the same time. This presents a challenge because typically mapping the environment requires knowledge of the state and mapping the state requires knowledge of the environment. The approach employed by this author in this paper is to implement an Extended Kalman Filter, which allows for continuous improvement in the state estimate and the map, since it takes into account differences between the robot's odometery and its observations in order to correct for errors.

The Extended Kalman Filter is chosen since the robot's motion and observations models are completely linear. This approach approximates them in order to apply the filter. This filter corrects the robot and observation states each time the robot iterates, meaning each time it takes new data. This can help to improve the localization and mapping that the robot does, particularly when the data used for one or both of these processes is subject the noise or inaccuracies.

The problem is defined in mathematical terms in the Problem Formulation section, the approach with an EKF mentioned in the previous paragraph is detailed in the Technical Approach section, and the results are presented and discussed in the Results section.

II. PROBLEM FORMULATION

A. Localization Problem

In order to solve the localization problem in this project, the concept of a Markov assumptions in robotics must be discussed. This refers to the concept that the state of a robot x_{t+1} depends only on the the previous input u_t and state x_t. Hence the motion model of a robot with Markov assumptions can be described in equation 1, where w_t is noise.

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

This means that the given a map and a sequence of control inputs, the robot state can be determined as long as there is a map and history of the control inputs. It will be discussed in the following sub-sections, however, that this problem is formulated such that there is no map of the environment, so this problem is not formulated as a simple problem of solely localizing a robot under Markov assumptions.

B. Mapping Problem

There is no available map of the robot's environment. Therefore this means that mapping is also included as a part of this problem. In order to map the environment using visual mapping, the observations for a given time step can be found with the following equation.

$$\mathbf{z}_t = M\pi({}_OT_I U_t \mathbf{m}) + v_t \tag{2}$$

Where $\mathbf{m}$ is a vector of all points being observed, M is a stereo camera matrix, $\mathbf{z}_t$ is defined as a vector of observations at time t, π is the projection function $\pi(q) = \mathbf{q}/q_3$, U_t is the inverse IMU pose, ${}_OT_I$ is the transformation from IMU frame to optical frame, and v_t is noise, defined as $v_t \sim \mathcal{N})), I \otimes V$.

C. SLAM Problem

As discussed previously, neither the robots location, nor the map of the environment is known, so the problem is formulated as a SLAM, (simultaneous localization and mapping) problem. The general solution to SLAM uses the fact that since the robot is assumed to be Markov, the observation and motion models can be treated as separate probability density functions, seen in Equation 3.

$$p(x_{0:T}, m, z_{0:T}, u_{0:T-1}) = p_{0|0}(X_0, m) \prod_{t=0}^{T} p_h(z_T | x_t, m) \prod_{t=1}^{T} p_f(x_t | x_{t-1}, u_{t-1}) \tag{3}$$

Here, x is the robot state, m is the map of the environment, z is the observations, and u is the control input. The inputs to the SLAM problem are the robot's observations, z, and the control input, sometimes put through a motion model and viewed as odometry, is u. The outputs are the robots state at all time instances, x, and the map of the environment, m. Here, p_f, p_h, and $p_{0|0}(x_0, m)$ are the probability distributions of the motion and observations models, and the prior probability, respectively.

III. TECHNICAL APPROACH

In the approach to solving this problem that this paper takes, the Extended Kalman Filter (EKF) is used to complete the SLAM. Since this is just one approach of many that can be used to solve the visual intertial SLAM problem, it is detailed in this section, Technical Approach. This section

details the approach used to solve the problems listed in the problem formulations. It will describe first the technical approach and supporting mathematical equations used to solve this problem and then the specific approach and techniques used within the code to get the desired result.

A. Theoretical Description

1) Extended Kalman Filter: Using an EKF, the motion model in equation 1 can be approximated with the Taylor series, to give the following equation for the distribution of the robot state, x_t:

$$f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \approx f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{0}) + F(\mathbf{x}_t - \boldsymbol{\mu}_{t|t}) + Q\mathbf{w}_t \tag{4}$$

The quantities in this equation are defined as follows: w_t is noise, independent of x_t, $Q = \frac{df}{dw}(x_t, u_t, 0)$, $F_t = \frac{df}{dx}(\mu_{t|t}, u_t, 0)$; note that Q and F are Jacobian matrices. The EKF approximates the distribution of x_t, $f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t)$, as a Gaussian, with mean and variance, used in the prediction step of the EKF, given by equations 5 and 6, respectively.

$$mean : \mu_{t+1|t} = f(\mu_{t|t}, u_t, 0) \tag{5}$$

$$variance : \Sigma_{t+1|t} = F_t \Sigma_{t|t} (F_t)^T + Q_t W (Q_t)^T \tag{6}$$

The second part of the EKF, done after the prediction step described above, is the update step of the EKF. equation 7, below, describes the distribution of the observation model, with $\mathbf{v}_t \sim \mathcal{N}(0, V)$ being the normally distributed observation noise, and equations 8 and 9 defining the Jacobian matrices of equation 7.

$$h(\mathbf{x}_{t+1}, \mathbf{v}_{t+1}) \approx \\ h(\boldsymbol{\mu}_{t+1|t}, \mathbf{0}) + H_{t+1}(\mathbf{x}_{t+1} - \boldsymbol{\mu}_{t+1|t}) + R_{t+1}\mathbf{v}_{t+1} \tag{7}$$

$$R_{t+1} = \frac{dh}{dv}(\mu_{t+1|t}, 0) \tag{8}$$

$$H_{t+1} = \frac{dh}{dx}(\mu_{t+1|t}, 0) \tag{9}$$

To compute this update step for the EKF, the following three equations are needed, the Kalman Gain (eq. 10), the updated mean (eq. 11), and updated variance (eq. 12) of the probability distribution of x_{t+1}, given z_{t+1}. In other words this updated the probability distribution of the state at some time $t+1$ conditioned on the observation for that time $t+1$.

$$K_{t+1|t} := \Sigma_{t+1|t} H_{t+1}^T (H_{t+1} \Sigma_{t+1|t} H_{t+1}^T + R_{t+1} V R_{t+1}^T)^{-1} \tag{10}$$

$$\boldsymbol{\mu}_{t+1|t+1} = \boldsymbol{\mu}_{t+1|t} + K_{t+1|t}(\mathbf{z}_{t+1} - h(\boldsymbol{\mu}_{t+1|t}, \mathbf{0})) \tag{11}$$

$$\Sigma_{t+1|t+1} = (I - K_{t+1|t} H_{t+1}) \Sigma_{t+1|t} \tag{12}$$

These equations are used in the solution to the problem described in the Problem Formulation Section. The following subsections of this Technical Approach section will reference these equations in its description of the Python program which utilized these equations in its problem solution.

B. Description of Code

The code used to achieve the solution presented here will be discussed in this sub-section of the report. First, all of the functions used will be explained, and then after that the main code will be explained, referencing the use of the program functions. Note that more information on the code can also be found in the code itself, as it is commented, as well as in the Readme file submitted with the code. The first function, $load_data$, was given to the students as part of the project. It was unaltered and serves to extract the data from the given dataset and place it into the data dictionary structure. The second function, $visualize_trajectory_2D$, was also given as part of the project. It was modified slightly such that it takes as input the history of robot states as well as the landmarks, and outputs a map for visualizing these two things. The third function, $vect2hat$, takes a 3 element vector and returns the hat map matrix for that vector. It essentially implements the following equation.

$$[\mathbf{x}]_\times = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \tag{13}$$

The fourth function, $IMU2world$ takes the inverse IMU pose (since this program works with the inverse IMU pose on the advice of the professor) in the IMU frame, and converts it to the world frame of reference. The fifth function, $createM$ is intended to create the stereo camera matrix, M, seen in the equation below, using the given intrinsic calibration matrix K, and baseline, b. Both the equation used to create M, as well as the equation that can be used to create M from camera characteristics had the aforementioned quantities above not been given are listed below.

$$M = \begin{bmatrix} K[0,0] & 0 & K[0,2] & 0 \\ 0 & K[1,1] & K[1,2] & 0 \\ K[0,0] & 0 & K[0,2] & -bK[0,0] \\ 0 & K[1,1] & K[1,2] & 0 \end{bmatrix} \tag{14}$$

$$M = \begin{bmatrix} fs_u & 0 & c_u & 0 \\ 0 & fs_v & c_v & 0 \\ fs_u & 0 & c_u & -fs_u b \\ 0 & fs_v & c_v & 0 \end{bmatrix} \tag{15}$$

The sixth function, $piProjMatrix$, divides the input vector by its third element. The purpose of this is to remove the depth element, Z. The seventh function, $dPidQ$, is intended to take the derivative of the projection function of a vector that is passed into it. It essentially executes the following equation.

$$\frac{d\pi}{d\mathbf{q}} = \frac{1}{q_3}\begin{bmatrix} 1 & 0 & -\frac{q_1}{q_3} & 0 \\ 0 & 1 & -\frac{q_2}{q_3} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{q_4}{q_3} & 1 \end{bmatrix} \tag{16}$$

The eighth function, $GetHzHat$ returns the Jacobian matrix for the stereo camera, and the Z-hat vector for the time instance, and state passed into it. It utilizes the following two equations to achieve this result. Note that in equation 18, $v_{t+1,i}$ is a 4 by 1 vector with all values being the observation noise value, found experimentally of 20. It was noticed after submitted the code for this project that had accidentally uploaded the version that did not include this, but adding it is as simple as adding the 4 by 1 vector with the noise value of 20 mentioned above to z-Hat. Note this change pertains only to the

$$H_{t,i,j} = M\frac{d\pi}{d\mathbf{q}}({}_oT_i\mu_{t+1|t}\boldsymbol{m}_j)_oT_i(\mu_{t+1|t}\boldsymbol{m}_j)\cdot P \tag{17}$$

Where P is defined originally as P^T.

$$\tilde{z}_{t,i} = M\pi({}_oT_i\mu_{t+1|t}\boldsymbol{m}_j) + v_{t+1,i} \tag{18}$$

The ninth function, $UpdateMarxMean$, was intended to compute the updated mean of the landmarks during the update step. It implemented the following equation and returned the value of the updated mean of the landmark at the given time.

$$\boldsymbol{\mu}_{t+1|t+1} = K_{t+1|t}(\mathbf{z}_{t+1} - \hat{\mathbf{z}}_{t+1}) + \boldsymbol{\mu}_{t+1|t} \tag{19}$$

The tenth function, $getUHat$, was used to compute the matrices for the hat map of the velocity vector, as well as the twist of the velocity vector, $\hat{u_t}$ (a 4x4 matrix), and (U-curly-hat) $\tilde{u_t}$ (a 6x6 matrix), respectively.

$$\hat{\mathbf{u}}_t = \begin{bmatrix} \hat{\boldsymbol{\omega}}_t & \hat{\boldsymbol{\lambda}}_t \\ \mathbf{0} & 0 \end{bmatrix} \tag{20}$$

$$\tilde{\mathbf{u}}_t = \begin{bmatrix} \hat{\boldsymbol{\omega}}_t & \hat{\boldsymbol{\lambda}}_t \\ 0 & \hat{\boldsymbol{\omega}}_t \end{bmatrix} \tag{21}$$

The eleventh function, $vect2hat46$, created the hat map for a 6 element vector. It used the following equation.

$$[\mathbf{x}]_\times = \begin{bmatrix} 0 & -x[6] & x[5] & -x[1] \\ x[6] & 0 & -x[4] & -x[2] \\ -x[5] & x[4] & 0 & -x[3] \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{22}$$

This concludes the description of the functions used. These functions will be referenced in the following code describing the main part of the program. The first 50 lines of the main loop are straight forward; loading data, defining variables to store the car's and landmark's mean, co-variance, and trajectory history. Additionally, noise matrices are defined here. The first is matrix W used as Gaussian noise with variance σ_w added to the covariance in the EKF covariance prediction when doing localization only.

$$W = \begin{bmatrix} \sigma_w & o & 0 & 0 & 0 & 0 \\ 0 & \sigma_w & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_w & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_w & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_w & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_w \end{bmatrix} \tag{23}$$

Where σ_w was experimentally found to be 100.

Also created is the matrix for the additive noise used in the calculation of the Kalman Gain, vI, which was Gaussian with variance σ_v.

$$vI = \begin{bmatrix} \sigma_v & o & 0 & 0 & 0 & 0 \\ 0 & \sigma_v & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_v & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_v & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_v & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_v \end{bmatrix} \tag{24}$$

Where σ_w was experimentally found to be 10^4.

The variables used to store the the car's and landmark's mean, co-variance, and trajectory history during the implementation of the full SLAM are the last variables defined in this seciton.

Next the program looped through all time stamps, this same loop being used for the implementation of Parts A, B and C. It should be noted that in this case, the index variable for the loop is j, meaning that j refers to the current stamp. Before any of the code for the following parts were written, the first thing done was to find the difference in time between loops, dt, for convenience later. As noted in the Readme file, the program performs best and fastest when either Parts A and B are run, or Part C is run. Running all three parts together is quite slow so does not give best results and should not be done.

1) Part A: First $\hat{u}$ and $\tilde{u}$ (u curly-hat) are found with the previously mentioned function. Next, the predictions for car mean and car covariance are found with the following two equations, taken from the lecture slides mentioned in the reference section of this report.

$$\boldsymbol{\mu}_{t+1|t} = exp(-\tau\hat{\mathbf{u}}_t)\boldsymbol{\mu}_{t|t} \tag{25}$$

$$\Sigma_{t+1|t} = exp(-\tau\tilde{\mathbf{u}}_t)\Sigma_{t|t}exp(-\tau\tilde{\mathbf{u}}_t)^T + w \tag{26}$$

Finally, the car and landmarks states were saved in the appropriate variables.

2) Part B: In part b, all the landmarks were looped through. The current feature was grabbed for convenience, and then it was checked that the current feature was in the frame, with the loop moving onto the next feature if it was not. If the feature hadn't been initialized yet, it was initialized using the $updateMarxMean$ function. If the feature had already been initialized, then it was updated by finding the Jacobian of z_{t+1}, $\hat{z_{t+1}}$, and using the following equations, which are derived from equations 10, 11, and 12 specifically for this instance of an EKF.

$$K_{t,j} := \Sigma_{t,j} H_{t,j}^T (H_{t,j} \Sigma_{t,j} H_{t,j})^T + V)^{-1} \quad (27)$$

$$\boldsymbol{\mu}_{t+1,j} = \boldsymbol{\mu}_{t,j} + K_{t,j}(\mathbf{z}_{t,j} - \hat{\mathbf{z}}_{t,j}) \quad (28)$$

$$\Sigma_{t+1,j} = (I - K_{t,j} H_{t,j}) \Sigma_{t,j} \quad (29)$$

Where the matrix V is given by vI previously.

3) Part C: In part C, first the matrix $WPrime$, the covariance for the Gaussian movement noise is defined,

$$WPrime = \begin{bmatrix} \sigma_1 & o & 0 & 0 & 0 & 0 \\ 0 & \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_2 \end{bmatrix} \quad (30)$$

Where σ_1 and σ_2 were found experimentally to be 10^{-6},, and 10^{-7}, respectively. Next, the same steps as detailed in Part A were followed to predict the car mean and covariance, with the former, as well as the updated matrix means from the previous iteration saved (this was formatted this way such that all the saving lines of code would be in 1 block). To update while performing SLAM, this student referenced the algorithm from Probabilistic Robotics, by Sebastian Thurn in section 10.2. This involved updating all of the means and covariances at one time by keeping track of which features were being updated in the current time step and then forming a diagonal Jacobian matrix with dimensions 4Nt by (3M+6) and landmark covariance matrix (3M+6) by (3M+6). However this created memory problems so the implementation this student used worked in the same way as part b seen previously, with the exception of also performing the update step on the car means and covariance as well. The equations for the update steps used on the car mean and covariance are seen below as equations 31, 32, and 33.

$$K_{t+1|t} := \Sigma_{t+1|t} H_{t+1|t}^T (H_{t+1|t} \Sigma_{t+1|t} H_{t+1|t}^T + I \otimes V)^{-1} \quad (31)$$

$$\boldsymbol{\mu}_{t+1|t+1} = exp((K_{t+1|t}(\mathbf{z}_{t+1} - \hat{\mathbf{z}}_{t+1}))^\wedge) \boldsymbol{\mu}_{t+1|t} \quad (32)$$

$$\Sigma_{t+1|t+1} = (I - K_{t+1|t} H_{t+1|t}) \Sigma_{t+1|t} \quad (33)$$

The results of this approach will be discussed in the following section. The final part of the code is simply using the $visualize_t rajectory_2 d$ to show the results.

IV. Results of Parts A and B, Map 22

All results will be presented first, then discussion will follow for ease of formatting. These are the results of Parts A and B, map 22, mapping the trajectory and updating the landmarks, but not doing SLAM.

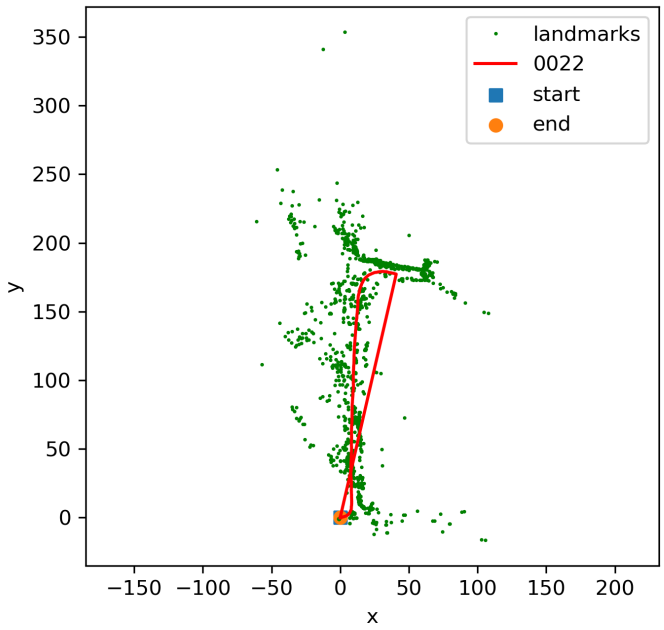

Fig. 1. Parts A and B, Map 22, timestamp: 300

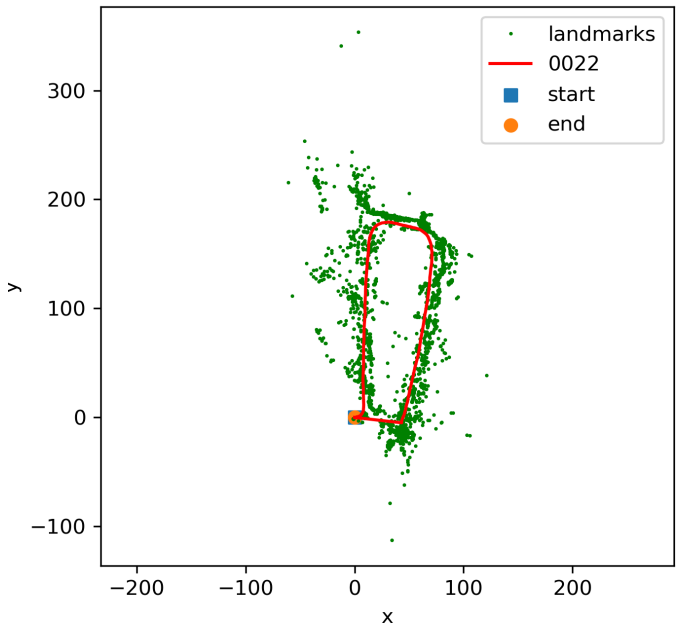

Fig. 2. Parts A and B, Map 22, timestamp: 600

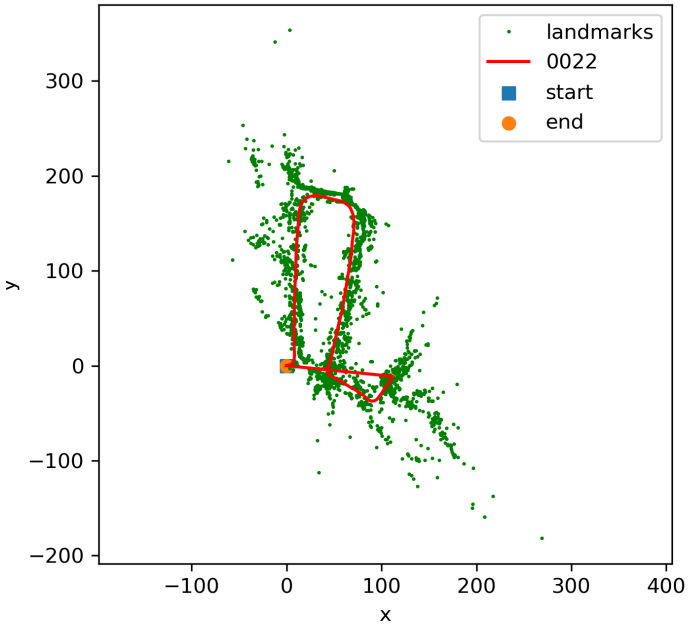

Fig. 3. Parts A and B, Map 22, timestamp: final

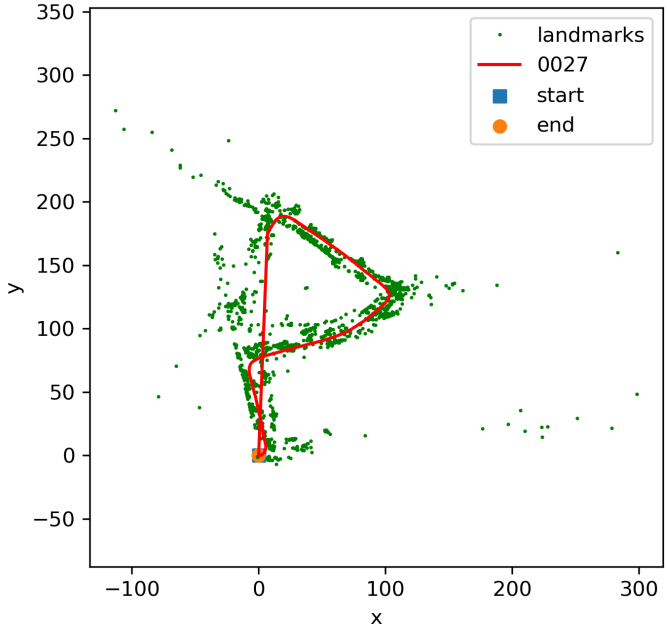

Fig. 4. Parts A and B, Map 27, timestamp: 500

V. Results of Parts A and B, Map 27

These are the results of Parts A and B, map 27, mapping the trajectory and updating the landmarks, but not doing SLAM.

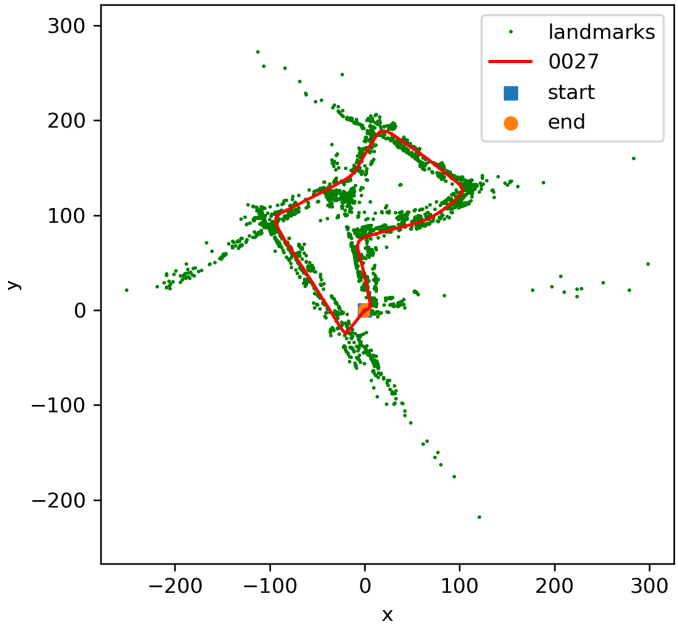

Fig. 5. Parts A and B, Map 27, timestamp: 900

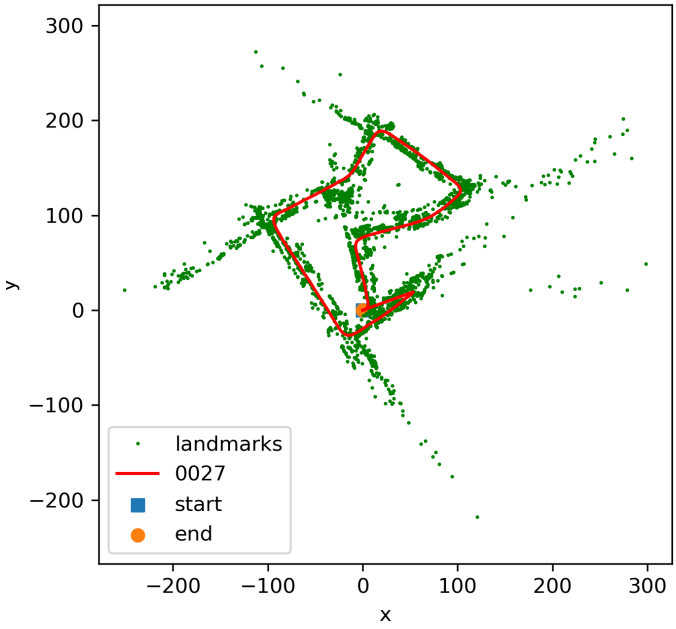

Fig. 6. Parts A and B, Map 27, timestamp: final

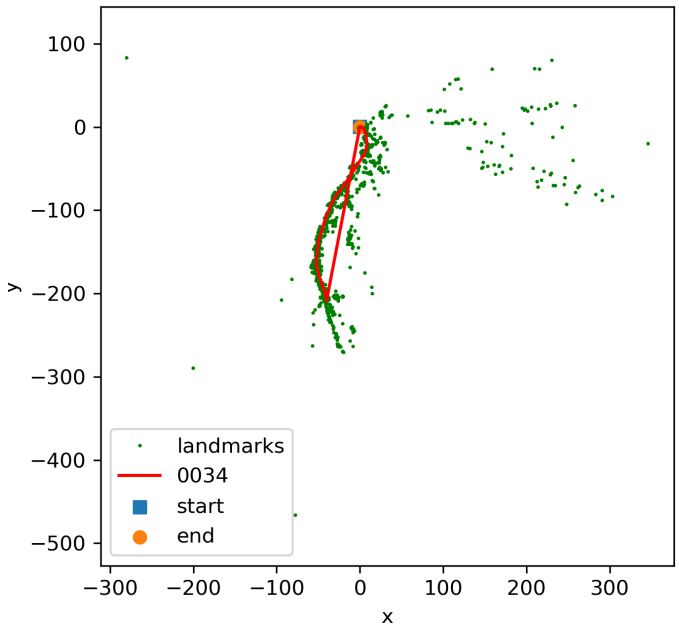

Fig. 7. Parts A and B, Map 34, timestamp: 300

VI. Results of Parts A and B, Map 34

These are the results of Parts A and B, map 34, mapping the trajectory and updating the landmarks, but not doing SLAM.

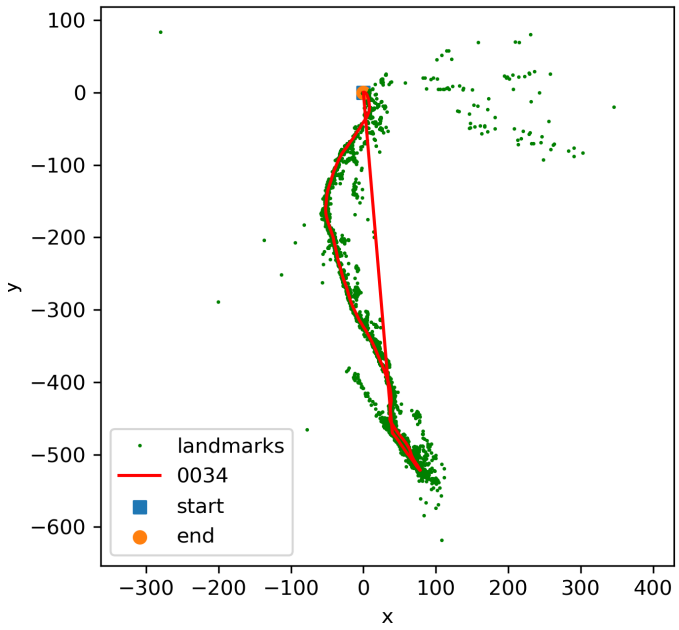

Fig. 8. Parts A and B, Map 34, timestamp: 1000

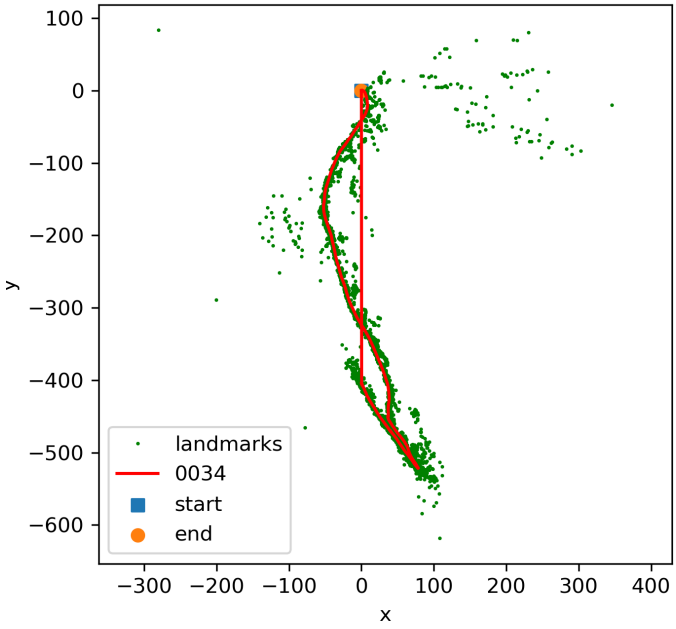

Fig. 9. Parts A and B, Map 34, timestamp: final

VII. Results of Parts C, Map 22

All results will be presented first, then discussion will follow for ease of formatting. These are the results of Part C, map 22, using SLAM to predict the car mean and then update the car and landmarks.

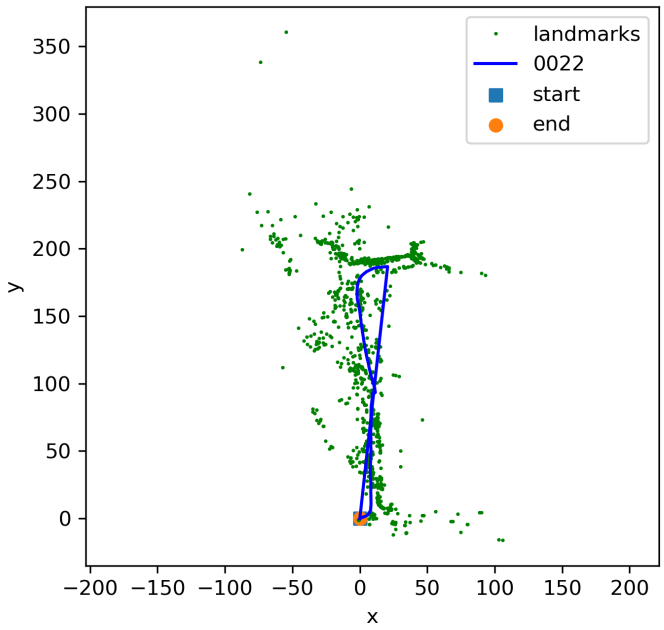

Fig. 10. Part C, Map 22, timestamp: 300

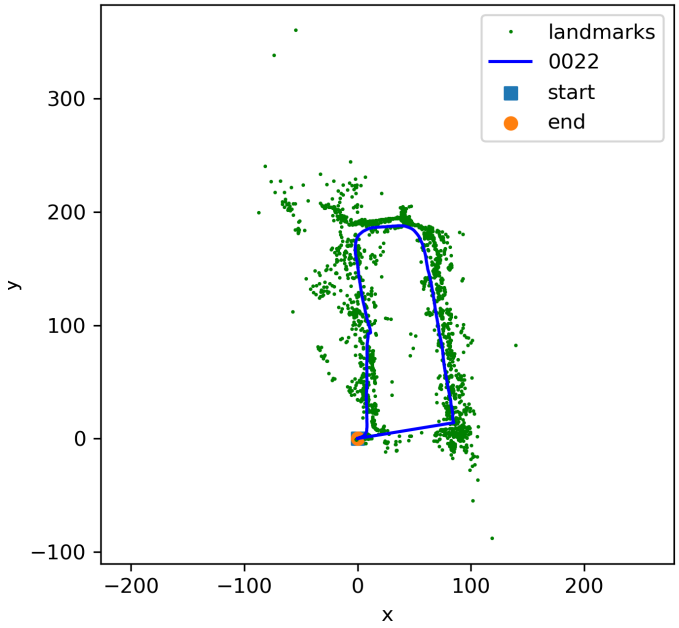

Fig. 11. Part C, Map 22, timestamp: 600

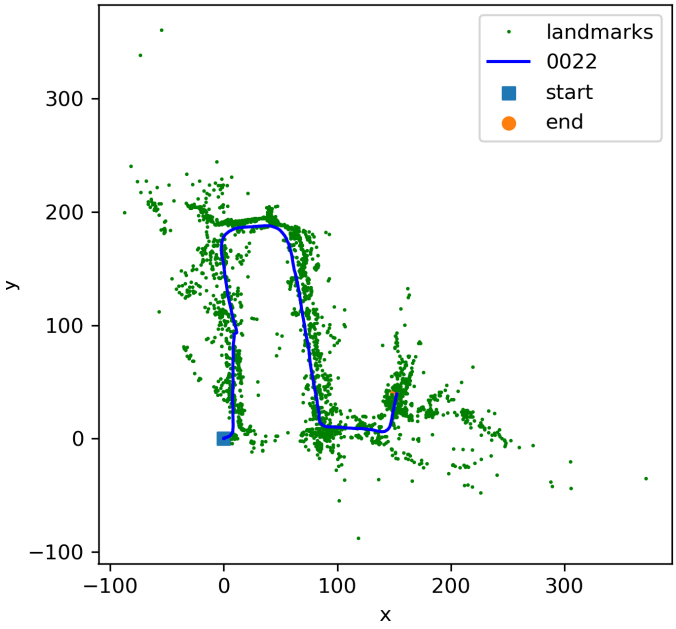

Fig. 12. Part C, Map 22, timestamp: final

VIII. Results of Part C, Map 27

All results will be presented first, then discussion will follow for ease of formatting. These are the results of Part C, map 27, using SLAM to predict the car mean and then update the car and landmarks.

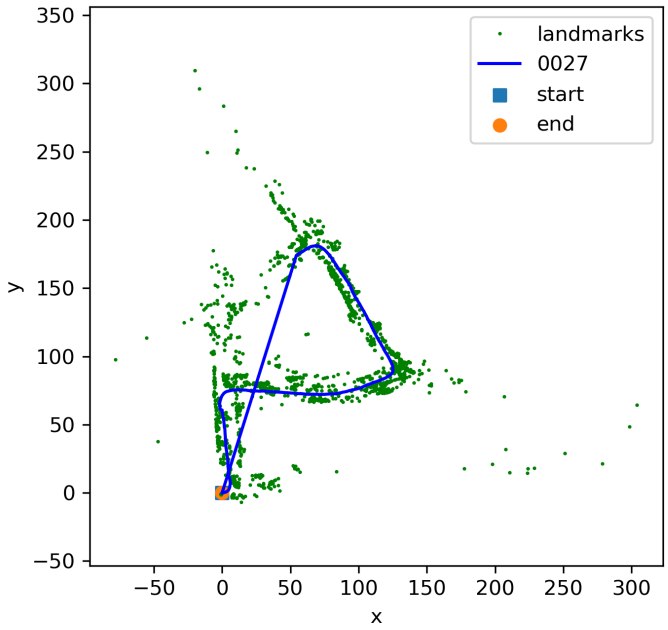

Fig. 13. Part C, Map 27, timestamp: 500

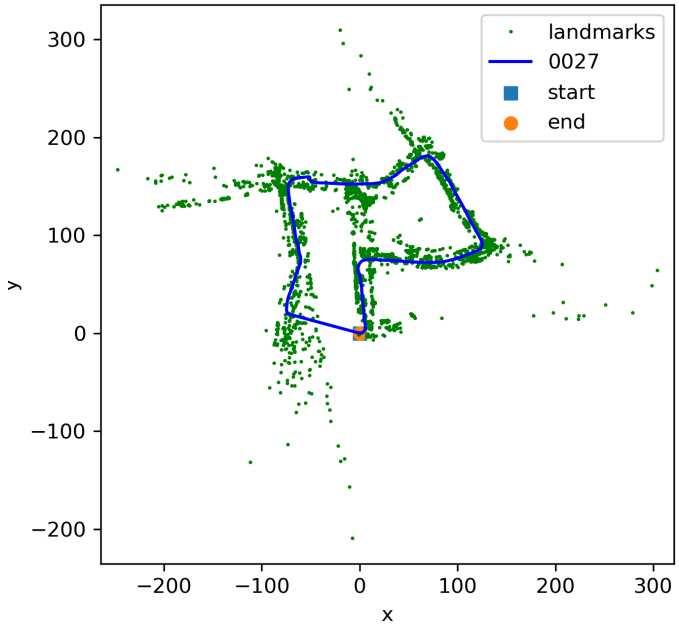

Fig. 14. Part C, Map 27, timestamp: 900

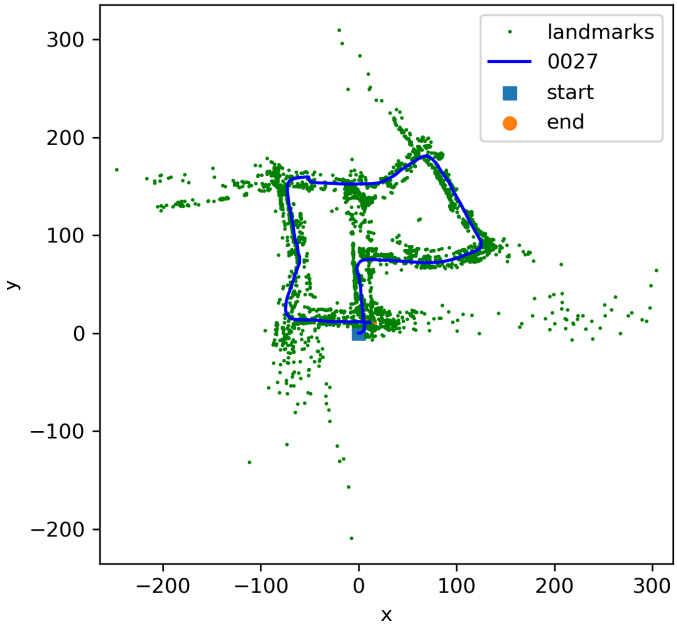

Fig. 15. Part C, Map 27, timestamp: final

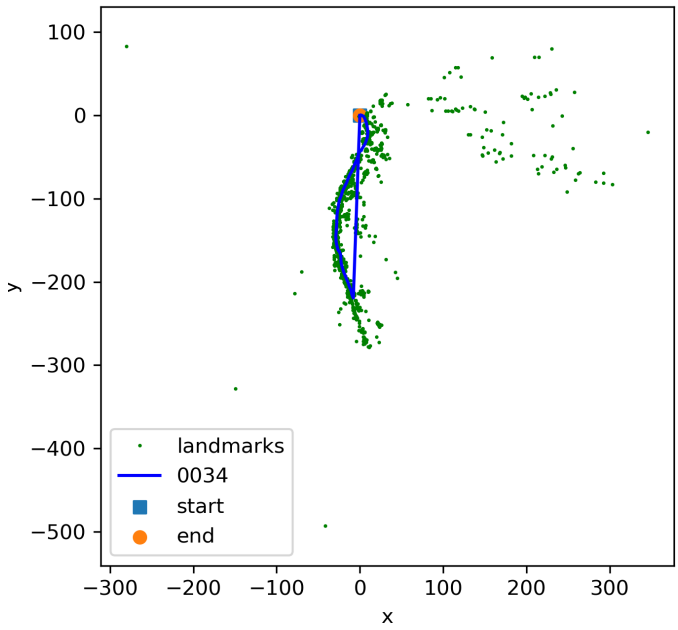

Fig. 16. Part C, Map 34, timestamp: 300

IX. Results of Part C, Map 34

All results will be presented first, then discussion will follow for ease of formatting. These are the results of Part C, map 34, using SLAM to predict the car mean and then update the car and landmarks.

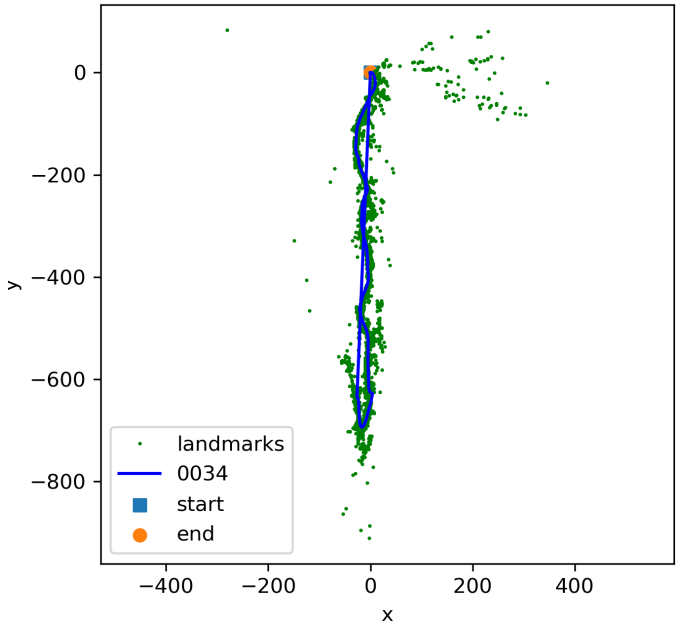

Fig. 17. Part C, Map 34, timestamp: 1000

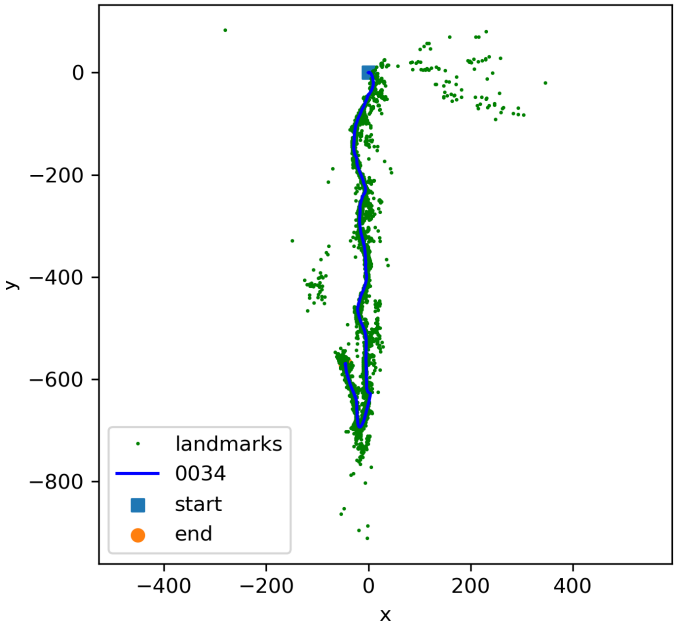

Fig. 18. Part C, Map 34, timestamp: final

X. Results Discussion

In all three of the test sets showcased in this report, using the EKF to perform visual inertial SLAM gives better results than doing simply prediction of the car's trajectory and updates on the landmark's locations. It should be noted that a large factor in the performance of the EKF is the movement and measurement noise used in the prediction and update equations. This noise could only be found via experimentation, and since this author was pressed for time, the optimal noise values were not able to be found. However, with more time, better values could have been found, resulting in even better results. Essentially the code approach is good, the author simple lacked the necessary time to tune the noise values for optimal EKF SLAM. For an example of this, included in this submission is a folder of SLAM with poorly tuned noise values. In order to avoid overwhelming the report with pictures these were not included, but the reader is invited to view those to see the improvement gained by a small amount of tuning on the noise values; more tuning would have lead to an even better result.

A. Map 22

Here the results of Map 22 will be discussed, comparing the SLAM results to the results obtained without using SLAM. In timestamp 300, both look very similar - the effect of SLAM cannot yet be seen. In the 600th timestamp, the results from SLAM are clearly better, the lines are straighter, and the result more closely tracks with what is expected based on the video of this map. By the final timestamp, the results can clearly be seen, with the SLAM correctly following the path seen in the video, and non-SLAM showing a completely incorrect path towards the end.

B. Map 27

In Map 27 in timestamp 500, both look very similar - the effect of SLAM cannot yet be seen. With timestamp 900, however, the improvement of the SLAM over non-SLAM is clear. Mainly the turns taken by the SLAM algorithm are more accurate, resulting in the map being "rotated" as seen in non-SLAM. In timestamp 34, the SLAM algorithm has improved even more, nearly completing the path, as opposed to the non-SLAM ending up off course. Another observation is that in addition to the accurate rotation, the SLAM implementation displays the turns more accurately, something that seems to cause the non-SLAM a lot of problems.

C. Map 34

In Map 34 in timestamp 300, both look very similar - the effect of SLAM cannot yet be seen. In timestamp, 1000, however, the SLAM implementation shows improvement in that it shows a straighter path, the length of the path is correct, as opposed to the non-SLAM's path, which is not straight at all. In the final timestamp, the SLAM implementation shows not a perfect path of turn at the end, but a definite improvement over the non-SLAM results.

D. Discussion Wrap-Up

As seen above, the SLAM algorithm clearly produces better results than the non-SLAM. Included with this report are images of both the SLAM and non-SLAM algorithms every 100 timestamps for each of the three maps, if the reader wishes to look at the results more closely. Once again it will be noted that with more tuning time, the SLAM results will improve and get better simply by narrowing down on better noise values. In particular, finding noise values that work well for all maps would be possible, which would be good since Map 34 was not used for tuning at all, and it arguably has the least convincing SLAM, so more tuning time would allow for the ability to find noise values that work well across all maps, leading to better results.

XI. Discussion with Fellow Students

This student regularly studies in a group with other students including Roumen Guha, Stephen West, and Maria Fatima and it is acknowledged here that this student discussed this project with these students in capacities such as: the math behind the frame transformations, and the math seen in the lecture slides pertaining to the Extended Kalman Filter Transformations.

References

[1] Atanasov, Nikolay. "ECE 276A." UCSD ECE276A: Sensing Estimation in Robotics (Winter 2020), Github, 1 Jan. 2020, natanaso.github.io/ece276a/.
[2] Thrun, Sebastian, et al. Probabilistic Robotics. MIT Press, 2010.