ECE 276A W21: PR3 Visual-Inertial SLAM

Patrick Youssef
University of California, San Diego
psyoussef@ucsd.edu

1 INTRODUCTION

In this paper we will discuss an approach to solving the simultaneous localization and mapping (SLAM) problem for a mobile vehicle and attempt to understand the shortcomings of the given implementation. Possible improvements will be suggested along the way alongside the state of the current results.

1.1 Importance of SLAM

The problem of localizing a mobile robot within an unknown map is one that has many novel uses in modern robotics. In many applications we don't have the luxury of a predetermined map, but the usage of landmarks and correspondence can help to maintain a correct localization. A solution to this problem falls into the family of problems called SLAM. A utility of this method is for navigation and creation of a map for any unknown or changing space where a map is either not available or needs to be updated realtime.

1.2 Summary of Approach

There are a variety of ways to solve this family of problems but we are tasked to solve this using an extended kalman filter (EKF) and visual landmarks. We will apply the method to a given dataset of stereo camera features and inertial measurements from an IMU. The process of creating the map and maintaining a position within the map is as follows:

(1) Initialize a map and vehicle state together.
(2) Compute a predicted vehicle state using control input.
(3) Project landmarks into relevant coordinate systems.
(4) Compute the Jacobian over observations.
(5) Update landmark state and covariance.
(6) Update vehicle state and covariance.

2 PROBLEM FORMULATION

Here we will discuss more precisely the problem we are trying to solve and the associated mathematical quantities that will help us.

2.1 Problem Statement

In this problem we are given visual landmarks from stereo cameras and IMU measurements and are tasked with predicting the state of the vehicle over time using a map of its surroundings that we generate along the way from the landmarks. We are given the data over time for both the landmarks and IMU measurements as well as other helper data structures that will be discussed more later.

2.1.1 Localization Problem. Localization deals with understanding the position of a robot in a given map.

- Input: A given map m and a sequence of controls u
- Objective: Obtain a sequence of states x of the robot's path

2.1.2 Mapping Problem. Mapping deals with logging the landmarks of a robot's surroundings.

- Input: A sequence of states x and observations z
- Objective: Obtain a map m of the robot's surroundings

2.1.3 SLAM Problem. SLAM is doing both localization and mapping with a limited understanding of both.

- Input: A sequence of controls u and observations z
- Output: A sequence of states x and a map of surroundings m

In our particular case we can attribute the input measurements for u and z to be coming from the IMU and stereo camera rig respectively.

2.2 Assumptions

2.2.1 Markov Independence. We assume Independence in the joint distribution $P_j = p(x_{0:T}, z_{0:T}, u_{0:T-1})$.

$$P_j = p(x_0)\prod_{t=0}^{T} p_h(z_t|x_t)\prod_{t=1}^{T} p_f(x_t|x_{t-1}, u_{t-1})\prod_{t=0}^{t-1} p(u_t, x_t) \quad (1)$$

2.3 Bayes Filter

The Bayes filter is the root of probabilistic filtering techniques that we often use in robotics. It consists of two steps:

2.3.1 Prediction Step. In this step, we compute the predicted probability density over x given our priors.

$$p_{t+1|t}(x) = \int p_f(x|s, u_t)p_{t|t}(s)ds \quad (2)$$

2.3.2 Update Step. In this step, we take the aforementioned predicted density $p_{t+1|t}(x)$ and our observations z_{t+1} with out observation model to incorporate out measurement information and correct the predicted density.

$$p_{t+1|t+1}(x) = \frac{p_h(z_{t+1}|X)p_{t+1|t}(x)}{\int p_h(z_{t+1}|s)p_{t+1|t}(s)ds} \quad (3)$$

An extended kalman filter is a realization of the Bayes Filter.

2.3.3 Extended Kalman Filter. An extended kalman filter (EKF) is an extension to the base kalman filter. The base kalman filter is a realization of the bayes filter than is computationally and memory efficient as it makes two primary assumptions that simplify operations.

- The world (our system) is linear
- PDFs are gaussian

If these two assumptions are met it can be shown that the kalman filter is the optimal estimator, but the world is typically not linear and hence we extend into the EKF.

The EKF works by linearizing the nonlinear models through the use of a multivariate Jacobian. These linearized forms can be used very similarly to the base equations in the base KF. We will detail more about our particular implementation later.

2.3.4 What I Would Like. Other than our intuition there is no real way to check against the true path or true map. I would have liked to have a ground truth path to gather an MSE of the filtered path.

3 TECHNICAL APPROACH

Here I will discuss in depth how the methods mentioned prior are implemented to solve the visual-inertial SLAM problem.

3.1 Sensor Details

Here I will give a brief summary as to the structure and information from the aforementioned visual and inertial sensors.

3.1.1 Visual Stereo Pair. The vehicle is equipped with a pair of cameras separated by a baseline b in meters. Using the stereo pair we can find correspondences between landmarks in both cameras and derive information about the 3d coordinates of the landmarks. I will detail more about this moving into the mapping section. The features we are given are already matched and we are given the image coordinates of where the landmarks lies in each of the image planes for the left and right camera. This leads to a single observation of $z \in \mathbb{R}^4$.

3.1.2 Inertial Measurement Unit (IMU). The IMU gives us measures of both the linear velocity $v \in \mathbb{R}^3$ but also the angular velocities $\omega \in \mathbb{R}^3$.

3.2 Visual Feature Matching

One of the first steps to solving any of the following problems regarding the EKF and primarily mapping is to have correspondence of visual features between each camera and over time. Here I discuss my attempt at deriving these features to be used in the following sections. The overall approach is as follows:

- Detect corners using the Harris corner detector in the stereo image pair.
- Stereo match these corners between both images for stereo correspondence.
- Extend the correspondence temporally into the next pair of images.

3.2.1 Harris Corner Detector. The Harris corner detector is a common corner detector used in classical computer vision. To apply the corner detector you first convert the image to a grayscale image and apply a gaussian blur. The blur helps the corner detector reject weak corners. Next I compute two images: an x and y image gradient by applying the following convolution kernels:

$$Kx = \begin{bmatrix} 0 & 0 & 0 \\ -0.5 & 0 & 0.5 \\ 0 & 0 & 0 \end{bmatrix} \tag{4}$$

$$Ky = Kx^T \tag{5}$$

With the gradient images with intensities I_x and I_y I can compute a G matrix for each pixel:

$$G_{i,j} = \begin{bmatrix} I^2_{x_{i,j}} & I_{x_{i,j}} I_{y_{i,j}} \\ I_{x_{i,j}} I_{y_{i,j}} & I^2_{y_{i,j}} \end{bmatrix} \tag{6}$$

We can then compute the two eigenvalues for each of these G matrices and use that to compute a score with as follows:

$$C = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{7}$$

Where k is a constant we tune for the Harris corner. We can then rank the C scores across the whole image to pick out the n strongest corners.

3.2.2 Stereo Matching. Stereo matching assume that I have two known temporally matched images from a stereo rig. I apply the corner detector to both images. The process is as follows:

- For each corner in the left image check the normalized cross correlation (score) between the left corner and all right corners to get n scores.
- Pick the best score as a set of matched corners if the score is greater than a preset threshold (reject weak correspondence).
- Repeat for all corners in the left image.

Normalized cross correlation is defined as:

$$NCC = \sum_{i,j} \tilde{W_1}(i,j) \tilde{W_2}(i,j) \tag{8}$$

$$\tilde{W} = \frac{W - \bar{W}}{\sqrt{\sum_{k,l}(W(k,l) - \bar{W})^2}} \tag{9}$$

Where W is a window from the grayscale image of a determined size around the corner and $\bar{W}$ is the mean pixel value of the window (thanks CSE 252A!).

3.2.3 Temporal Matching. After tinkering with this for a bit I opted not to continue, proceed to conclusion for more information.

3.2.4 Conclusion. Although I made some progress, for the sake of consistency between peers and the course I opted to dedicate more time to the SLAM as opposed to here and ended up using the included feature set. You can refer to any file with `featurematch` in the name to see more of what I did.

3.3 Data Cleaning

A good first step is to process the data into a cleaner and more usable form as often times real world data can not start in the greatest form. For this dataset there wasn't too much preprocessing.

3.3.1 Splitting Input Data. The first step I took was to separate data from the one large list into separate variables with their context more easily attached.

3.3.2 Times. The time vector was in the UNIX time standard which is great because there is not need to convert to seconds but the baseline time is relative to January 1, 1970 so I subtract the minimum time of the set on all values so that I start at $t = 0$.

3.4 Vehicle Setup

Here I will discuss how the visual features can be matched to the IMU and what we will focus on to define the center of focus on the vehicle.

3.4.1 Camera to IMU Transform. We are given an $SE(3)$ transform that takes observations and maps them from the left camera optical frame (denoted as o in transforms) and the IMU frame (denoted as i in transforms) as iTo.

3.4.2 Vehicle Pose. Our pose estimation of the vehicle will actually be pose estimation of the IMU so we will often use the prior transform to get to the IMU space and then world space.

3.5 Extended Kalman Filter

I will give a general overview out of context of the specific problem as to how an EKF is formulated and at a high level how it works. As mentioned prior the EKF can be summarized as a realization of the general Bayes Filter that assume the PDFs are gaussian and linearizes our nonlinear models of the world. Let's cover some of the steps within an EKF.

3.5.1 EKF Prediction. The EKF prediction step advances our means and covariances based on the prior means, covariances, and a control input. It yields a prediction of both the new means and new covariances that will be corrected later. This looks like:

$$\bar{\mu}_t = g(u_t, \mu_{t-1}) \tag{10}$$

Where μ is our means and g is our motion model that advances the means based on the prior means and a control input.

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t \tag{11}$$

Where Σ is our covariances, G_t is our linearization, and R_t is our process noise. This completes the prediction step and now we need to move forward with correcting these estimates.

3.5.2 Kalman Gain. The Kalman Gain represents the weights for a weighted sum that will proceed further. It encapsulates our trust or distrust in our observations.

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1} \tag{12}$$

Where H_t is another linearization and Q_t is our observation noise. The kalman gain is used to determine the weight of updates relative to an error signal in the following step.

3.5.3 EKF Update. The EKF update step, often called the correction step, is where we tweak our prior estimates of the means and covariances.

$$\mu_t = \bar{\mu_t} + K_t(z_t - h(\bar{\mu_t})) \tag{13}$$

Where z_t is our observation and h is a function that maps the estimated state to a predicted observation $\bar{z}$.

$$\Sigma_t = (I - K_t H_t)\bar{\Sigma_t} \tag{14}$$

3.5.4 How To Apply This. These steps detailed are very general and in practice many helper functions are needed to get the information detailed here. This is detailed in the following sections as we consider the linearizations, mapping functions, etc.

3.6 Localization via EKF Predict

We begin applying the EKF by just implementing the prediction step highlighted before. Using the IMU inputs we essentially dead-reckon the means through the control inputs u that come from the IMU.

3.6.1 Known. It is assumed that we know the following moving into the implementation:

- Control inputs $\mathbf{u_t}$ over time from the IMU
- Timesteps τ_t used in our discrete integration
- Process noise W

3.6.2 Implementation. The implementation is quite simple as we do not need to implement the covariances as there is no update step that takes advantage of this. We are merely predicting with no correction. Regardless I will detail the covariances advance here as it's needed in following sections.

$$\mu_{t+1|t} = \mu_{t|t} \exp(\tau \hat{\mathbf{u}}_t) \in SE(3) \tag{15}$$

$$\Sigma_{t+1|t} = \exp(\tau \overset{\curlywedge}{u}_t) \Sigma_{t|t} \exp(\tau \overset{\curlywedge}{u})_t + W \in \mathbb{R}^{6x6} \tag{16}$$

We apply the first equation recursively until we are done with all of our samples. In later parts we also advance the covariances.

3.6.3 Explanation of Operators and Terms.

$$\mathbf{u_t} = \begin{bmatrix} \mathbf{v_t} \\ \omega_\mathbf{t} \end{bmatrix} \in \mathbb{R}^6 \tag{17}$$

$$\hat{\mathbf{u}}_t = \begin{bmatrix} \hat{\omega}_t & \mathbf{v_t} \\ \mathbf{0}^\mathbf{T} & 0 \end{bmatrix} \in \mathbb{R}^{4x4} \tag{18}$$

$$\overset{\curlywedge}{u}_t = \begin{bmatrix} \hat{\omega}_t & \hat{\mathbf{v}}_t \\ \mathbf{0} & \hat{\omega}_t \end{bmatrix} \in \mathbb{R}^{6x6} \tag{19}$$

3.7 Landmark Mapping via EKF Update

We continue our process to full Visual-Inertial SLAM by working on the update side of an EKF. In this part we rely on IMU pose over time that we get from the localization step done before. As the landmarks are assumed to be static we do not need a prediction step to track them, but we do need an update step.

3.7.1 Known. It is assumed that we know the following moving into the implementation:

- Camera calibration matrix M
- IMU to camera transform $oTi \in SE(3)$
- IMU pose $T_t \in SE(3)$
- New observations z_{t+1}
- Oberservation noise V

3.7.2 Implementation. This implementation is more involved than the prediction was. We first need to get predicted observations based on our prior estimate of the landmark mean. We do this just for the landmarks observed. For each observed landmark:

$$\bar{\mathbf{z}}_{t+1} = M\pi(oTiT_{t+1}^{-1}\underline{\mu}_t) \in \mathbb{R}^4 \tag{20}$$

Next we need to compute the Jacobian H of the predicted z with respect to the world coordinate $\mathbf{m}$ of the landmark for each of the landmarks observed. For each observed landmark:

$$H_{t+1} = M\frac{d\pi}{d\mathbf{q}}(oTiT_{t+1}^{-1}\underline{\mu}_t)oTiT_{t+1}^{-1}P^T \in \mathbb{R}^{4x3} \tag{21}$$

Where P is the canonical projection. Next we need to apply the updates given these terms. The following steps are done for each observed landmark individually. First we compute the Kalman Gain.

$$K_{t+1} = \Sigma_t H_{t+1}^T (H_{t+1}\Sigma_t H_{t+1}^T + I \otimes V)^{-1} \in \mathbb{R}^{3x4} \tag{22}$$

Next we apply all the prior computations to update the means and covariances of the landmarks.

$$\mu_{t+1} = \mu_t + K_{t+1}(\mathbf{z}_{t+1} - \bar{\mathbf{z}}_{t+1}) \in SE(3) \tag{23}$$

$$\Sigma_{t+1} = (I - K_{t+1}H_{t+1})\Sigma_t \in \mathbb{R}^{4x4} \tag{24}$$

3.7.3 Explanation of Operators and Terms. There are a new new operators and terms in the update steps, let's go over them.

M contains all the intrinsic camera and stereo parameters needed for stereo feature calculations.

$$M = \begin{bmatrix} fs_u & 0 & c_u & 0 \\ 0 & fs_v & c_v & 0 \\ fs_u & 0 & c_u & -fs_u b \\ 0 & fs_v & c_v & 0 \end{bmatrix} \tag{25}$$

π is a simple projection function.

$$\pi(\mathbf{q}) = \frac{1}{q_3}\mathbf{q} \in \mathbb{R}^4 \tag{26}$$

$\frac{d\pi}{d\mathbf{q}}$ is the derivative of the projection function with respect to $\mathbf{q}$.

$$\frac{d\pi}{d\mathbf{q}} = \begin{bmatrix} 1 & 0 & \frac{-q_1}{q_3} & 0 \\ 0 & 1 & \frac{-q_2}{q_3} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{-q_4}{q_3} & 1 \end{bmatrix} \in \mathbb{R}^{4x4} \tag{27}$$

$\otimes$ is the Kronecker product which can just be `np.kron()`. In practice for us it's just $I \times$ Noise.

3.8 Visual-Inertial SLAM

The final step is to incorporate the steps taken in localization and mapping and bring them together here. A few considerations to mention first regarding this integration. I attempted to do an inclusive update step for all parameters but had issues where means would drift well outside of the typical value. Because of this I opted instead to continue the individual landmark updates as done in the mapping step and to add on top of that a separate update for the robot pose. To summarize, each iteration:

- EKF Prediction is applied to just the IMU pose means and covariances
- We check for what landmarks are in view
- Landmarks are updated individually based on individual Hs,Ks,etc.
- We compute H for the IMU pose over each landmark and stack them
- IMU means and covariances are updated based on a different H,K,etc. relative to the landmarks

3.8.1 Known. It is assumed that we know the following moving into the implementation:

- Control inputs $\mathbf{u_t}$ over time from the IMU
- Timesteps τ_t used in our discrete integration
- Process noise W
- Camera calibration matrix M
- IMU to camera transform $oTi \in SE(3)$
- New observations z_{t+1}
- Observation noise V

Note: IMU pose is not know anymore as it's updated as part of each iteration now and not just relied upon from localization.

3.8.2 Implementation. We require a few new additions to the prior set of equations, notably in regards to IMU pose updates as that has not been done prior. For each observed landmark we compute H as:

$$H_{t+1,i} = -M\frac{d\pi}{d\mathbf{q}}(oTi\mu_{t+1|t}^{-1}\underline{\mathbf{m}}_t)oTi(\mu_{t+1|t}^{-1}\underline{\mathbf{m}}_t)^{\odot} \in \mathbb{R}^{4x6} \tag{28}$$

for $i = 1, ..., N_{observed}$, where $\mathbf{m}$ is the current estimate of the landmark position in the world. We then stack all of these Jacobians to form one H.

$$H_{IMU} = \begin{bmatrix} H_{t+1,1} \\ . \\ . \\ . \\ H_{t+1,N_{observed}} \end{bmatrix} \in \mathbb{R}^{4N_{obs.}x6} \tag{29}$$

Now we can perform the updates to the IMU, first compute the Kalman gain.

$$K_{t+1} = \Sigma_t H_{t+1}^T (H_{t+1}\Sigma_t H_{t+1}^T + I \otimes W)^{-1} \in \mathbb{R}^{6x4N_{obs.}} \tag{30}$$

Now for the IMU means and covariances update.

$$\mu_{t+1|t+1} = \mu_{t+1|t}\exp((K_{t+1}(\mathbf{z}_{t+1} - \bar{\mathbf{z}}_{t+1}))^{\wedge}) \in SE(3) \tag{31}$$

$$\Sigma_{t+1|t+1} = (I - K_{t+1}H_{t+1})\Sigma_t \in \mathbb{R}^{6x6} \tag{32}$$

Note: $(\mathbf{z}_{t+1} - \bar{\mathbf{z}}_{t+1})$ is the same as from landmark updates.

3.8.3 Explanation of Operators and Terms. There is only one new operator in the IMU pose updates.

$\underline{s}^{\odot}$ is as follows:

$$\underline{s}^{\odot} = \begin{bmatrix} I & -\hat{\mathbf{s}} \\ 0 & 0 \end{bmatrix} \tag{33}$$

4 RESULTS

The results section serves to present the performance of the solution and talk about shortcomings.

4.1 Visual Feature Matching

4.1.1 Parameters. Here I discuss all the parameters I used to produce the stereo matching results shown below.

4.1.2 Corners & Matching. Shown below is an example of finding corners in a sample image as shown in Figure 1 and an example of stereo matching in Figure 2. As we can see many of the original corners are not present in the matching result, this is due to the highly restrictive NCC cutoff that only matches a pair if the correlation is above that cutoff. I found that varying k did not change the matching results too much.

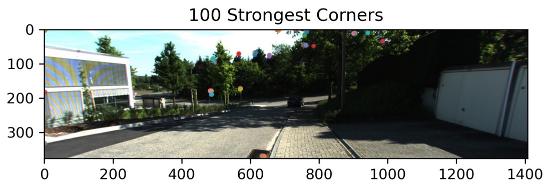

Figure 1: 100 strongest corners in sample image

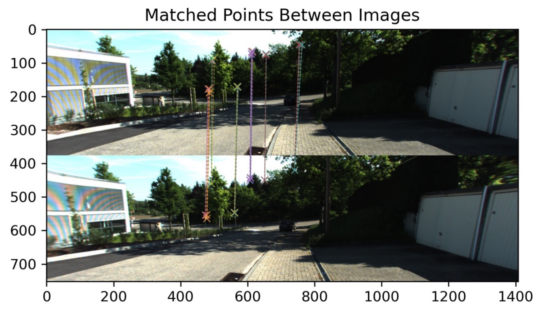

Figure 2: Corner matches between sample images

4.2 Landmark Subset

To aid in debugging and to improve compute time, I picked every k_{sub} landmarks from the original dataset to reduce the number of observations in a given timestep. In Figure 3 you can see the original number of observations over the samples. For most of my runs I opted to use $k_{sub} = 2$ to get observations over time like that of Figure 4.

4.3 Localization via EKF Predict

4.3.1 Localized Path. Based on the process highlighted in the technical approach section we get localization results as follows in Figure 5.

Table 1: Visual Matching Parameters

Parameter	Value
N Corners	100
Blur Std.	1
Window Size	5x5
k	0.01
NCC Radius	40
NCC Cutoff	0.75

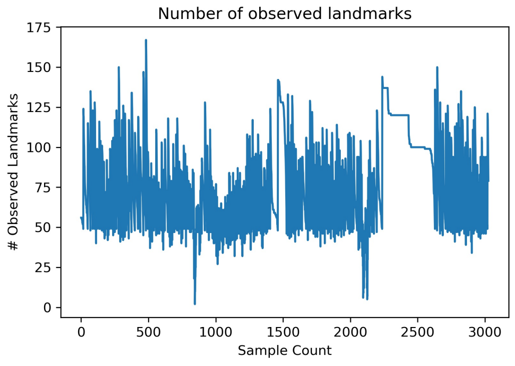

Figure 3: Original amount of observations

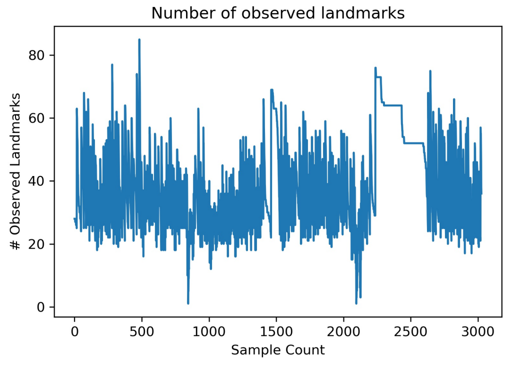

Figure 4: Subset of observations

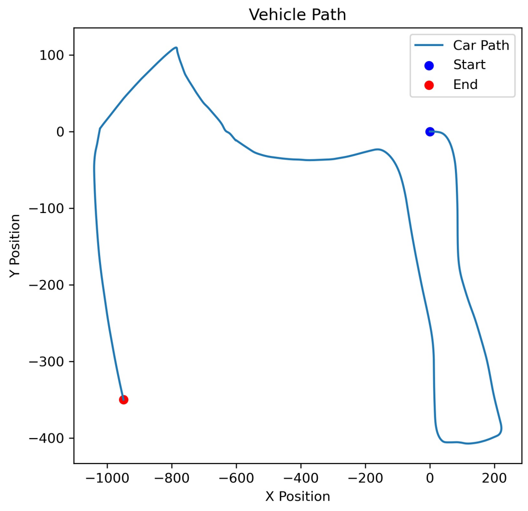

Figure 5: Localization via EKF predict path

After going through the images of the drive taken by the vehicle, this path seems very reasonable. An additional check I would like to have been able to do is to check against a ground truth to get a more numerical idea of the accuracy.

4.4 Landmark Mapping via EKF Update

Here are the results for my mapping attempt using the EKF update steps to keep track of and update the positions of the landmarks in the world coordinates.

4.4.1 Parameters. Here are the parameters used to create these plots, my code has these same values used already but they're there for you to check against.

Table 2: Landmark Mapping Parameters

Parameter	Value
k_{sub}	2
Observation Noise	10

As a reminder, k_{sub} here is how much we reduce the number of landmarks by to speed up execution.

4.4.2 Mapped Landmarks. Based on the technical approach we get the mapping results in Figure 6. We can see that the path matches that of the localization only which is to be expected as we essentially just integrated the computations from (a) directly into this work. Later we will see the plots stacked and you'll notice they are completely overlapped on one another. Mapping seems reasonable given the range of objects in the viewport and the surrounding scenery from the images.

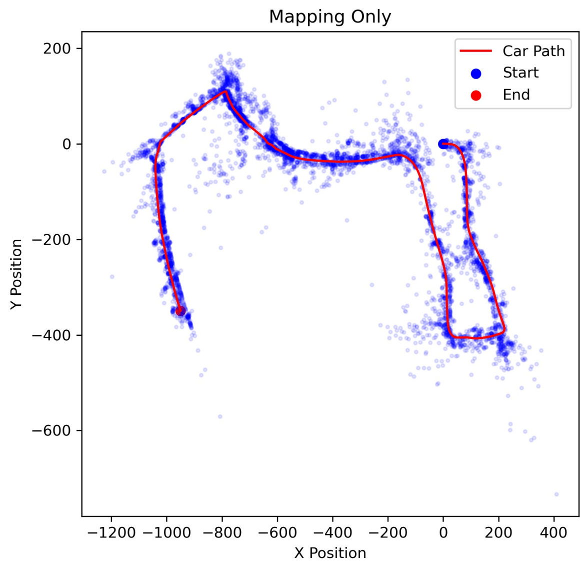

Figure 6: Landmark Mapping via EKF Update

4.5 Visual-Inertial SLAM

Here are the results of my mapping and localization done as the integration of the prior work with the addition of the IMU update steps.

4.5.1 Parameters. Here are the parameters used to create these plots, my code has these same values used already but they're there for you to check against.

Table 3: Visual-Inertial SLAM Parameters

Parameter	Value
k_{sub}	2
Observation Noise	10
Process Noise	0.001

As a reminder, k_{sub} here is how much we reduce the number of landmarks by to speed up execution.

4.5.2 EKF SLAM Path and Map. Based on the technical approach we get the mapping results in Figure 7. We can see that the path does not seems correct, this is covered down below in the issues subsection.

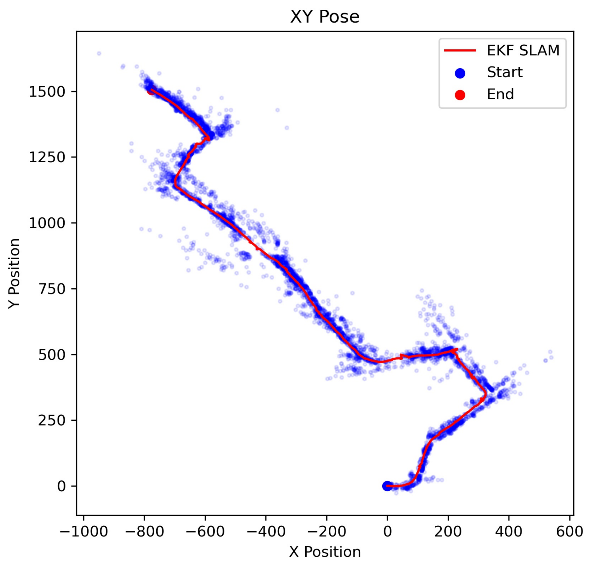

Figure 7: Vehicle Path and Map via Visual-Inertial SLAM

4.6 Issues

It is apparent that EKF results do not match the mapping results, you can see this more clearly in Figure 8, but there does seem to be one primary issue. It seems like ω_z is flipped when looking at the mapping only compared to the visual-inertial SLAM. When looking at the overlaid results it seems like Visual-Inertial SLAM is

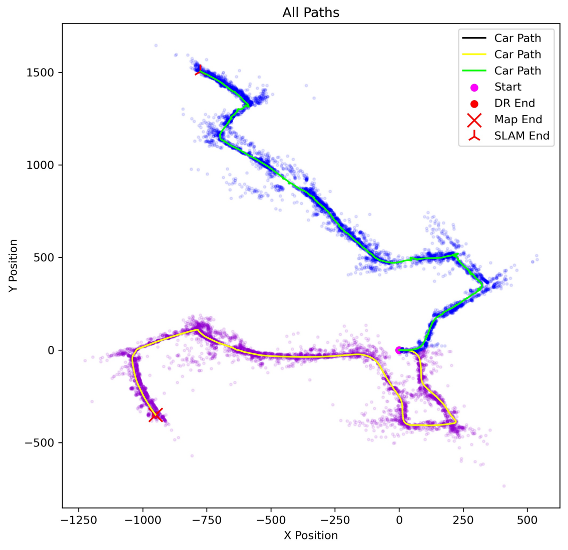

Figure 8: All Maps Overlaid

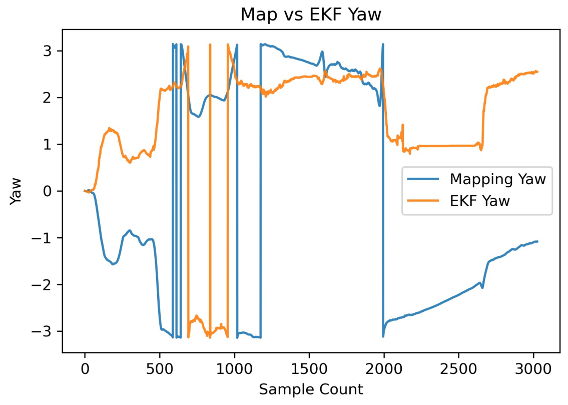

Figure 9: Yaw for Mapping and Visual-Inertial SLAM

mirrored across a near horizontal axis. I attempted to debug this for a few hours but to no avail. This is an active issue I am investigating post submission but as of now it's the results that I get.

To further corroborate this, Figures 9 and 10 show that when comparing the original yaw values for the visual-inertial SLAM and the mapping only we see they are very nearly mirrored and when one is inverted are nearly the same.

I also believe this may be due to my implementation where I update the IMU pose and landmarks all are independent updates. For a while I was working on an inclusive update but the results I got followed Figure 11 where occasional landmarks would drift very far from the true position. You can see more of this code in any file with `inclusive` in the name.

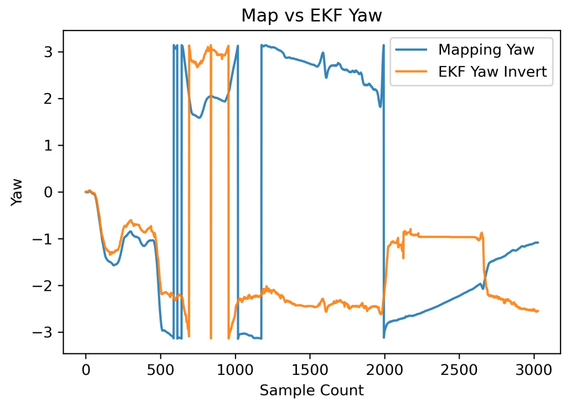

Figure 10: Yaw for Mapping and Inverted Visual-Inertial SLAM

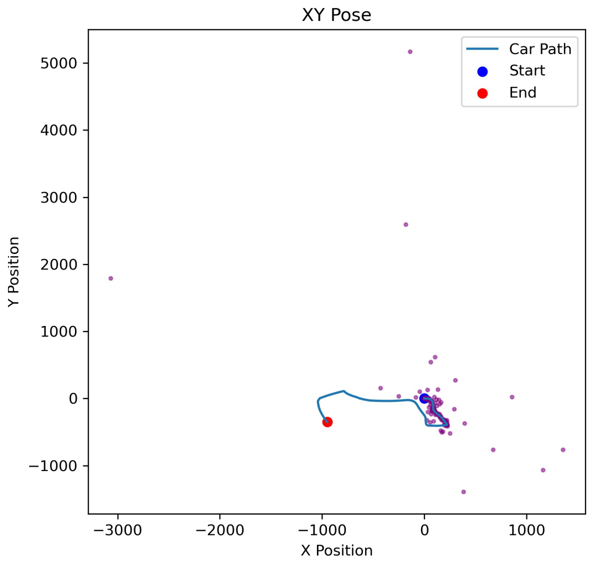

Figure 11: Landmark Drift