RBE 549: Project 4 - Visual Inertial Odometry

Chinmay Kate
M.S. Robotics Engineering
Worcester Polytechnic Institute (WPI)
Worcester, MA 01609
Email: cskate@wpi.edu

Mandeep Singh
M.S. Robotics Engineering
Worcester Polytechnic Institute (WPI)
Worcester, MA 01609
Email: msingh2@wpi.edu

I. Overview

In this project we are going to Simultaneously Localize and Map (SLAM) the positions of an agent using data from a stereo camera and an Inertial Measurement unit (IMU sensor). The problem of SLAM can be done by using only a stereo camera - classical Structure from Motion pipeline where we reconstruct 3D points as well as find the camera pose (Localize the camera in the 3D world). But this process although accurate has high computational cost and fails in case of high speed situations (motion blur). To account for this, we also use a much cheaper sensor IMU along with the camera which compliments the limitations of using only camera system for SLAM. It can work at higher output rate and even at higher speeds and accelerations.
But still this method can be computationally expensive as we have to track too many features along the way which leads our state vector and covariance matrix to increase as the time progresses. This project takes into consideration this problem and provide an optimized solution to further reduce the computational complexity without compromising with the efficiency.
Here in place of tracking all features that the camera encounter on its way, we track the same feature with different camera poses and then do a math trick to get the relative poses of cameras with respect to each other and thus localize itself.

The implemented functions in the whole Visual inertial Odometry pipeline are discussed in detail in below sections.

II. Implemented functions

A. Initialize gravity and bias

This function is basically implemented to initialize the IMU bias and initial orientation of the IMU with respect to the world frame based on the first few readings of the IMU. For calculating gyro bias we are just taking the average of all the angular velocity values in the imu buffer and update that value in the state server imu state. For getting the orientation of the IMU with respect to the world frame we have to use gravity to our rescue. We know that gravity is always in the downward direction in the world frame, so the normalized output of the imu when kept static and being kept in any orientation should be equal to the acceleration due to gravity i.e. 9.81(approx.) So, we have a vector of gravity in inertial frame (average of imu buffer values) and we have our gravity vector, we can find the rotation between these two vectors in quaternion form easily, which will give us the initial orientation of the IMU in world frame.

B. Batch IMU processing

This function processes the messages in the imu msg buffer, executes the process model and updates the state (mentioned in Section C). We execute the process model on the imu states stored in imu msg buffer ubtil the imu time reaches the time bound.

C. Process model

In this function, we perform the steps of implementing the motion model. We write the dynamics of the IMU error state and then find the initial estimates of the state after solving the dynamics equation and thus propagate the state. This state estimate is further refined in measurement update when we have the camera observations.

First of all, the continuous dynamics of the estimated IMU state as written below:

$$
\begin{aligned}
{}^{I}_{G}\dot{\bar{q}}(t) &= \frac{1}{2}\begin{bmatrix} -\lfloor \boldsymbol{\omega}(t)\times\rfloor & \boldsymbol{\omega}(t) \\ -\boldsymbol{\omega}^{\top}(t) & 0 \end{bmatrix} {}^{I_t}_{G}\bar{q} \\
&=: \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}(t))\,{}^{I_t}_{G}\bar{q} \\
{}^{G}\dot{\mathbf{p}}_I(t) &= {}^{G}\mathbf{v}_I(t) \\
{}^{G}\dot{\mathbf{v}}_I(t) &= {}^{I_t}_{G}\mathbf{R}^{\top}\mathbf{a}(t) \\
\dot{\mathbf{b}}_{\mathbf{g}}(t) &= \mathbf{n}_{wg} \\
\dot{\mathbf{b}}_{\mathbf{a}}(t) &= \mathbf{n}_{wa}
\end{aligned}
$$

To deal with the discrete measuremrnts from the IMU we have to apply 4th order Runge-Kutta numerical integration to propagate the estimated IMU state which is explained in detailed in the next section 'Predict new state'. Once, we have the estimated IMU state, we also have to propagate the uncertainity of the state i.e. the transition matrix and the noise covariance matrix.

We follow the following steps in order to propagate the noise covariance matrix:

- Linearize the error state using the taylor's series expansion till 3rd term.
- Modify the transition matrix to get the Phi matrix.

- Now, using the Phi matrix and G matrix, the continuous time noise covariance (Q) of the system is given by the following equation:

$$\mathbf{Q}_k = \int_{t_k}^{t_{k+1}} \mathbf{\Phi}(t_{k+1},\tau)\mathbf{G}\mathbf{Q}\mathbf{G}\mathbf{\Phi}(t_{k+1},\tau)^\top d\tau$$

- Once we have the Q, Phi we can calculate the state propagated covariance denoted by P in the below written equation.

$$\mathbf{P}_{k+1|k} = \begin{pmatrix} \mathbf{P}_{II_{k+1|k}} & \mathbf{\Phi}_k \mathbf{P}_{IC_{k|k}} \\ \mathbf{P}_{IC_{k|k}}^\top \mathbf{\Phi}_k^\top & \mathbf{P}_{CC_{k|k}} \end{pmatrix}$$

Where the first matrix (top left in above matrix) is given by follwing:

$$\mathbf{P}_{II_{k+1|k}} = \mathbf{\Phi}_k \mathbf{P}_{II_{k|k}} \mathbf{\Phi}_k^\top + \mathbf{Q}_k$$

- After the IMU state and covariance propagation we just update the state correspondences to the null space.

D. Predict new state

This function is part of the above process model where we propagate the estimated IMU state using 4th order Runge-Kutta numerical integration. So, we already have the kinematic equations for the derivatives of the orientation, velocity and position as given in above Fig. . We only need to integrate those equations if we want to get the estimates of the IMU state. But the above equations are for continuous state and our system is discrete. So, we use Runge-Kutta's integration method for to estimate velocity and position.

1) Orientation: We know that q is a quaternion and to integrate a quaternion is not that direct. We use the following equation to calculate the integration of the quaternion q.

$$\begin{aligned} \mathbf{\Theta}(t,t_k) &= {}^{I_t}_{G}\bar{q}\, {}^{I_k}_{G}\bar{q}^{-1} \\ \Rightarrow \dot{\mathbf{\Theta}}(t,t_k) &= {}^{I_t}_{G}\dot{\bar{q}}\, {}^{I_k}_{G}\bar{q}^{-1} \\ &= \frac{1}{2}\mathbf{\Omega}(\boldsymbol{\omega}(t))\, {}^{I_t}_{G}\bar{q}\, {}^{I_k}_{G}\bar{q}^{-1} \\ &= \frac{1}{2}\mathbf{\Omega}(\boldsymbol{\omega}(t))\mathbf{\Theta}(t,t_k) \end{aligned}$$

2) Velocity and Position: To get the IMU state velocity and position update we use Runge-Kutta 4th order equations. The general way to obtain a Runge-Kutta integartion of 4th order is shown below:

If we plug in the functions of velocity and position in place of y, we can calculate the IMU state update for position and velocity.

$$\begin{aligned} k_1 &= f(y^*(t_0), t_0) \\ k_2 &= f\left(y^*(t_0) + k_1\frac{h}{2}, t_0 + \frac{h}{2}\right) \\ k_3 &= f\left(y^*(t_0) + k_2\frac{h}{2}, t_0 + \frac{h}{2}\right) \\ k_4 &= f\left(y^*(t_0) + k_3 h, t_0 + h\right) \end{aligned}$$

$$y^*(t_0+h) = y^*(t_0) + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}h$$

$$= y^*(t_0) + \left(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4\right)h$$

E. State Augmentation

In this function we update the camera pose and the state covariance matrix as when the camera records a new image.

1) Update Camera Pose: the camera pose can easily be obtained as we know the rotation and translation between the camera and IMU. So, we get the camera pose using the following equations:

$$ {}^{C}_{G}\hat{\bar{q}} = {}^{C}_{I}\bar{q} \otimes {}^{I}_{G}\hat{\bar{q}}, \quad \text{and} \quad {}^{G}\hat{\mathbf{p}}_C = {}^{G}\hat{\mathbf{p}}_I + \mathbf{C}_{\hat{q}}^{T}\, {}^{I}\mathbf{p}_C$$

We append these camera poses (orientation quaternion and position vector) to our state vector.

2) Augment State covariance matrix: Accordingly we also have to augment the state covariance matrix when we get new image msg. The state covariance matrix which is denoted as P is augmented as shown below:

$$\mathbf{P}_{k|k} \leftarrow \begin{bmatrix} \mathbf{I}_{6N+15} \\ \mathbf{J} \end{bmatrix} \mathbf{P}_{k|k} \begin{bmatrix} \mathbf{I}_{6N+15} \\ \mathbf{J} \end{bmatrix}^T$$

Where J is the Jacobian matrix given by th =e equation:

$$\mathbf{J} = \begin{bmatrix} \mathbf{C}\left({}^{C}_{I}\bar{q}\right) & \mathbf{0}_{3\times9} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times6N} \\ \lfloor \mathbf{C}_{\hat{q}}^{T}\, {}^{I}\mathbf{p}_C \times \rfloor & \mathbf{0}_{3\times9} & \mathbf{I}_3 & \mathbf{0}_{3\times6N} \end{bmatrix}$$

F. Add feature observations

In this function we check for the features in the images from the camera. If we get a new feature, we add it to the map server and if the feature already exists in the server we just update its new observations. This feature tracking is done by assigning a unique feature id number to each and every feature.

G. Measurement update

Measurement model is used to update the state estimates. For constructing measurement model we define residual $\boldsymbol{r}$, that linearly depends on state error $\widetilde{X}$.

Here H is Jacobian matrix which maps from state into measurement domain and noise term is zero mean which is

$$\mathbf{r} = \mathbf{H}\widetilde{\mathbf{X}} + \text{noise}$$

uncorrelated to state error for EFK framework to be applied. In measurement model of MSCKF we view static feature with multiple camera poses. Here we concatenate camera poses per feature and present this as our constraint. This is achieved without including feature pose in the filter state vector.

Writing measurement model for single feature point f_j with the equation given below:

$$\mathbf{z}_i^{(j)} = \frac{1}{^{C_i}Z_j} \begin{bmatrix} ^{C_i}X_j \\ ^{C_i}Y_j \end{bmatrix} + \mathbf{n}_i^{(j)}, \qquad i \in \mathcal{S}_j$$

Here n is the noise vector having covariance matrix. Feature position expressed in the camera frame given by equation:

$$^{C_i}\mathbf{p}_{f_j} = \begin{bmatrix} ^{C_i}X_j \\ ^{C_i}Y_j \\ ^{C_i}Z_j \end{bmatrix} = \mathbf{C}(^{C_i}_{G}\bar{q})(^{G}\mathbf{p}_{f_j} - {}^{G}\mathbf{p}_{C_i})$$

This is the 3D position of feature point in Global frame. We obtain $\widehat{P}$ estimate using least square minimization. To do this we obtain z and their filter estimates of camera poses at corresponding time instants. Once we compute estimates we can put this in our original residual $\boldsymbol{r}$ equation.

$$\mathbf{r}_i^{(j)} = \mathbf{z}_i^{(j)} - \hat{\mathbf{z}}_i^{(j)}$$

We modify a bit by linearizing the above equation and stacking the equations of all residuals into one equation given by:

$$\mathbf{r}^{(j)} \simeq \mathbf{H}_{\mathbf{X}}^{(j)}\widetilde{\mathbf{X}} + \mathbf{H}_f^{(j)}\,{}^{G}\widetilde{\mathbf{p}}_{f_j} + \mathbf{n}^{(j)}$$

Here, $\mathbf{H}_{\mathbf{X}}^{(j)}$ and $\mathbf{H}_f^{(j)}$ are Jacobians of measurement w.r.t to state and feature position jacobian respectively. As X state is used to estimate feature position, the error $\widetilde{p}$ is correlated with errors $\widetilde{X}$. Therefore we cannot use our above residual $\boldsymbol{r}$ equation directly in our measurement model and therefore we modify it by projecting $r^{(j)}$ on the left null space of the matrix $\mathbf{H}_f^{(j)}$.

$$\mathbf{r} = \mathbf{z} - \mathbf{h}(\hat{\mathbf{x}}, \hat{f}) = \mathbf{H}_x\tilde{\mathbf{x}} + \mathbf{H}_f\tilde{\mathbf{f}} + \mathbf{n}$$

$$\underset{\text{Feature Jacobian}}{\mathbf{H}_f} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix} = \mathbf{Q}_1\mathbf{R}_1$$

This residual equation is independent of the errors in the feature coordinates. From this EKF update can be performed

$$\tilde{\mathbf{z}}_{m,k} \simeq \mathbf{H}_x\tilde{\mathbf{x}}_k + \mathbf{Q}_1\mathbf{R}_1\,{}^{G}\tilde{\mathbf{p}}_f + \mathbf{n}_k$$

$$\Rightarrow \mathbf{Q}_2{}^{\top}\tilde{\mathbf{z}}_m \simeq \mathbf{Q}_2{}^{\top}\mathbf{H}_x\tilde{\mathbf{x}}_k + \mathbf{Q}_2{}^{\top}\mathbf{Q}_1\mathbf{R}_1\,{}^{G}\tilde{\mathbf{p}}_f + \mathbf{Q}_2{}^{\top}\mathbf{n}_k$$

$$\Rightarrow \mathbf{Q}_2{}^{\top}\tilde{\mathbf{z}}_m \simeq \mathbf{Q}_2{}^{\top}\mathbf{H}_x\tilde{\mathbf{x}}_k + \mathbf{Q}_2{}^{\top}\mathbf{n}_k$$

Q_1 and Q_2 are orthonormal

$$\Rightarrow \tilde{\mathbf{z}}_{o,k} \simeq \mathbf{H}_{o,k}\tilde{\mathbf{x}}_k + \mathbf{n}_{o,k}$$

which is optimal except for the inaccuracy caused by linearization. Finally we compute the Kalman gain matrix and update the covariance matrix with the equation expressed below. In order to reduce the computational complexity of the EKF update, The QR decomposition of matrix $\mathbf{H}_{\mathbf{X}}^{(j)}$ is performed. Thus the vectors Q1 and Q2 become orthonormals and we can exclude them from our final measurement equation.

$$\mathbf{K} = \mathbf{P}\mathbf{T}_H^T\left(\mathbf{T}_H\mathbf{P}\mathbf{T}_H^T + \mathbf{R}_n\right)^{-1}$$

$$\mathbf{P}_{k+1|k+1} = \left(\mathbf{I}_\xi - \mathbf{K}\mathbf{T}_H\right)\mathbf{P}_{k+1|k}\left(\mathbf{I}_\xi - \mathbf{K}\mathbf{T}_H\right)^T + \mathbf{K}\mathbf{R}_n\mathbf{K}^T$$

H. *Results and Analysis*

From the MSCKF implementation we can conclude following things:

- In recursive state estimation with camera observations, measurement model is non-linear in nature and sensitivity of camera is affected by noise causing inconsistent solutions with false local minima. This problem of linearization inaccuracies can be solved using feature depth parameterization used in measurement model.
- Most feature can be tracked over a small frames. That result in feature tracking algorithm failure. This is mostly due to noise, limited FOV, occlusions.

III. References

- https://www-users.cse.umn.edu/~stergios/papers/ICRA07-MSCKF.pdf
- https://arxiv.org/pdf/1712.00036.pdf
- https://github.com/uoip/stereo_msckf/tree/f8412830b09994ca7c681eb36a0039154a387901
- https://docs.openvins.com/pages.html
- https://arxiv.org/pdf/1704.06053.pdf

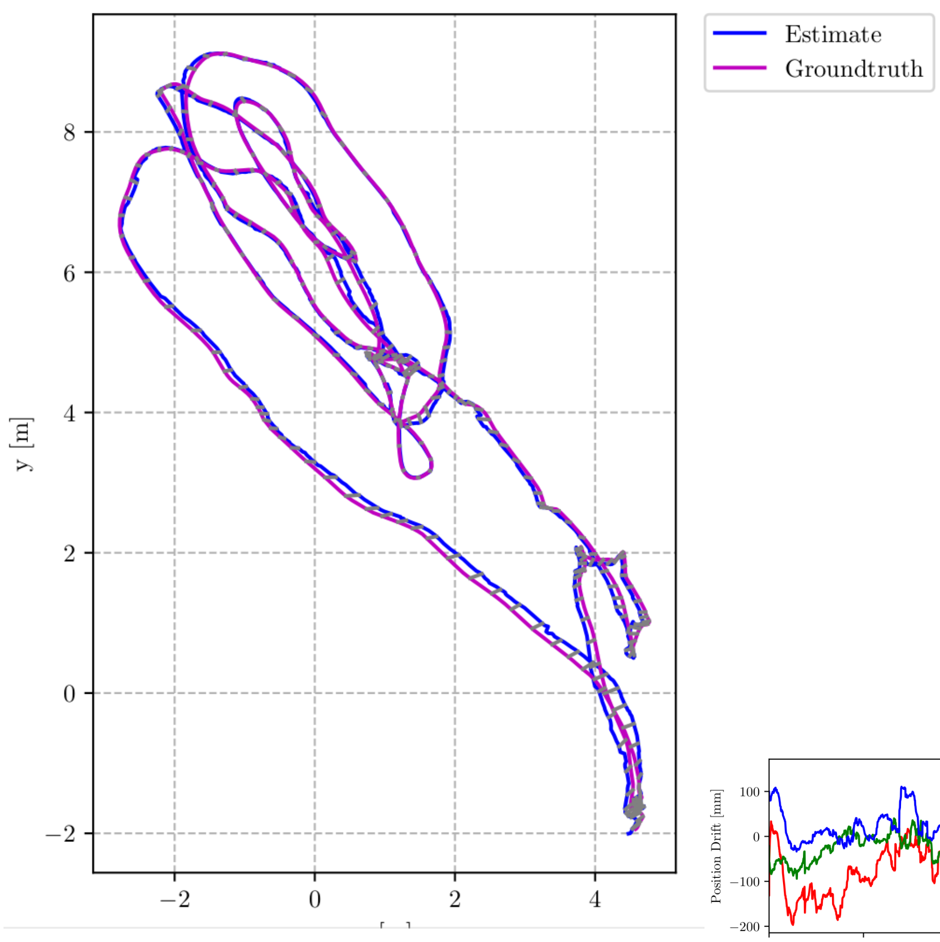


Fig. 1: Trajectories: Estimated vs Ground Truth

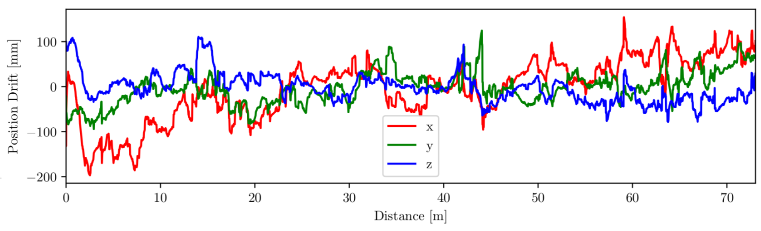
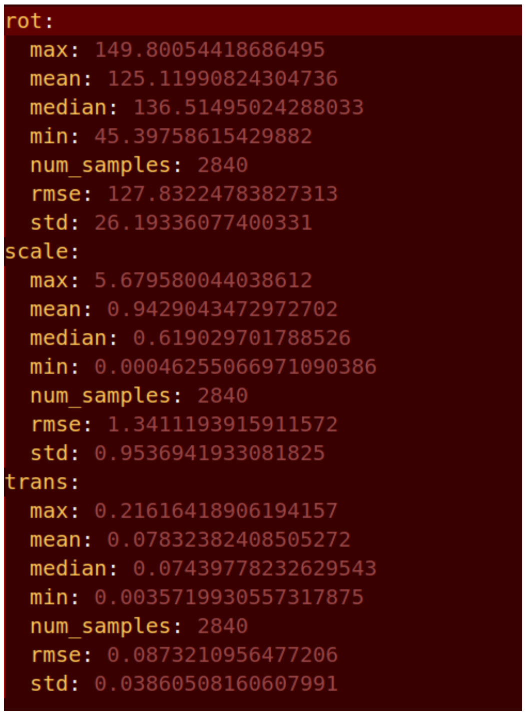


Fig. 3: Translation Error statistics



```
rot:
  max: 149.80054418686495
  mean: 125.11990824304736
  median: 136.51495024288033
  min: 45.39758615429882
  num_samples: 2840
  rmse: 127.83224783827313
  std: 26.19336077400331
scale:
  max: 5.679580044038612
  mean: 0.9429043472972702
  median: 0.619029701788526
  min: 0.00046255066971090386
  num_samples: 2840
  rmse: 1.3411193915911572
  std: 0.9536941933081825
trans:
  max: 0.21616418906194157
  mean: 0.07832382408505272
  median: 0.07439778232629543
  min: 0.0035719930557317875
  num_samples: 2840
  rmse: 0.0873210956477206
  std: 0.03860508160607991
```

Fig. 2: Absolute Error statistics