Project 4: Visual Inertial Odometry (Using 1 Late Day)

Karter Krueger
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA 01609
Email: kkrueger2@wpi.edu

Tript Sharma
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, 01609
Email: tsharma@wpi.edu

I. Introdution

This project is to demonstrate how we can combine camera and IMU data to compute pose and orientation in realtime with high precision leveraging the fast and precise IMU data along with slow but drift-resistant camera data. We implement the Stereo MSCKF algorithm to achieve this task.

II. Code Implementation

The code for this project followed the structure of the original stereo MSCKF written in C++, as it was converted for this class over to Python for the template/starter code with 7 key functions missing for us to implement based on the MSCKF paper. We describe our code and some of the math that went into it from the paper in the following sections below.

A. Initialize Gravity and Biases

This method defines the initial gravity and bias using the first 200 IMU data-points. Since the camera-IMU pair is stationary during this period, we can compute the gravity as an average of the acceleration measurements for the given timestamps. Setting the initial gravity vector is important for us to reference throughout the rest of the code to determine the change in orientation of the IMU. Orientation of a (non-accelerating) IMU can be determined by finding the angles between the current accelerometer values and the initial gravity vector values. The gyro bias terms are also important as we must subtract the bias to get an accurate angular velocity reading from the gyro. If we didn't remove bias, then there could be a small constant value added to the readings that would make the robot think it is constantly rotating even when it is stationary, which would cause drift.

B. Batch IMU Processing

This method processes the IMU buffer messages to compute the motion model. It checks whether the IMU recordings (in the buffer) are within the current camera feature recording timestamp. If so, we compute the process model for the current frame using all the unused IMU recordings till now.

C. Process Model

The process model is one of the key functions of the Kalman Filter as it runs the dynamics on the IMU error state to update the state covariance matrix and set the null-space values for the orientation, position, and velocity of the IMU. This method follows the equations outlined in section III.A of the MSCKF paper. First, we take the raw gyro and accelerometer readings and subtract the gyro and accelerometer biases, respectively. We also determine a dt as the change in time between the current IMU message and the last timestamp processed for the IMU-state. Next, we build the F and G matrices from the appendix A section of the paper. The F matrix is of size (21, 21) and is comprised of the screw matrix of the gyro omega vector, several identity matrices, and rotated acceleration vectors, as shown in the excerpt from the paper in Fig. 1.

$$\mathbf{F} = \begin{pmatrix} -\lfloor \hat{\boldsymbol{\omega}}_\times \rfloor & -\mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ -C\left({}^I_G\hat{\mathbf{q}}\right)^\top \lfloor \hat{\mathbf{a}}_\times \rfloor & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -C\left({}^I_G\hat{\mathbf{q}}\right)^\top & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{pmatrix}$$

Fig. 1. F matrix from the MSCKF paper

The G matrix is constructed of size (21, 12) with several identity matricies and a rotation matrix of the orientation, as seen in the paper Fig. 2 below.

$$\mathbf{G} = \begin{pmatrix} -\mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -C\left({}^I_G\hat{\mathbf{q}}\right)^\top & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_3 \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{pmatrix}$$

Fig. 2. G matrix from the MSCKF paper

Once we have the F matrix, we estimate Phi using the 3rd order matrix exponential using $(F_{dt} = F{*}dt)$, (F_{dt}^2), and (F_{dt}^3) as a Taylor-series expansion. We then run PredictNextState on the IMU values to propagate the state using the 4th order Runge-Kutta. We then follow math from the paper and reference code to modify the transition matrix using the null-orientation and orientation to rotate the gravity vector and modify the Phi matrix exponential. Once Phi is ready, we propagate the state covariance matrix using Phi and Q computed from Q = Phi * G * cov-noise * G.T * Phi.T * dt. Lastly, we force the covariance to be symmetric as ((cov + cov.T) / 2.0), and update the null-space values for orientation, position, and velocity of the IMU-state.

D. Predict New State

Since the states are defined as error w.r.t the previous state, we need to integrate all the previous states to predict the new state. The continuous dynamics of the estimated IMU state using equations given in the figure 3 has been linearized in Section II-C. To deal with the discrete time measurements of IMU, we propagate the state using a numerical integration function i.e. 4th order Runge-Kutta (RK4).

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

Fig. 3. Continuous State Dynamics Equations from the MSCKF paper

The orientation, pose and velocity are computed along with $[\omega_X]$. Using equations in Figures 3 and 4 we compute $\delta\bar{q} * \delta t$ which defines the new state orientation. It is also necessary to compute RK4 of velocity and pose.

$$
\begin{aligned}
{}^{I}_{G}\bar{q} &= \delta\bar{q} \otimes {}^{I}_{G}\hat{\bar{q}} \\
\delta\bar{q} &= \begin{bmatrix} \hat{\mathbf{k}}\sin(\frac{1}{2}\tilde{\theta}) \\ \cos(\frac{1}{2}\tilde{\theta}) \end{bmatrix} \simeq \begin{bmatrix} \frac{1}{2}\tilde{\boldsymbol{\theta}} \\ 1 \end{bmatrix}
\end{aligned}
$$

Fig. 4. Quaternion error state from OpenVINS documentation

E. State Augmentation

When a new image is captured, the camera pose is estimated and appended to the state vector. We first collect and compute all the rotation and translation matrices between World, IMU, and Camera coordinate frames. We also create a new Camera State to add to the state-server for later processing. Then we compute the matrix J, from the paper appendix B, shown in Fig. 5, which is required to update the state covariance matrix. J is comprised of the rotation from IMU to Camera frame, identity, and a skew matrix of the rotation from IMU to World with the translation from cam to IMU. We then resize the state covariance matrix to add 6 rows and columns for the new camera state of 6 values. The state covariance matrix is then updated by setting the bottom left section to $(J * IMU - cov)$, top right section to the transpose of the bottom left (symmetrical for IMU-Cam and Cam-IMU), and the bottom right section to the Cam-Cam covariance, computed from $(J * CIJ.T)$ with C being top-right section covariance. We lastly enforce overall covariance symmetry with State-Cov = (State-Cov + State-Cov.T) / 2.

$$
\mathbf{J}_I = \begin{pmatrix} C\left({}^{I}_{G}\hat{\mathbf{q}}\right) & \mathbf{0}_{3\times 9} & \mathbf{0}_{3\times 3} & \mathbf{I}_3 & \mathbf{0}_{3\times 3} \\ -C\left({}^{I}_{G}\hat{\mathbf{q}}\right)^{\top}\lfloor {}^{I}\hat{\mathbf{p}}_C\times\rfloor & \mathbf{0}_{3\times 9} & \mathbf{I}_3 & \mathbf{0}_{3\times 3} & \mathbf{I}_3 \end{pmatrix}
$$

Fig. 5. Matrix J from the MSCKF paper appendix

F. Add Feature Observations

Here we take in a new feature message and add the features to the map server. We loop over all features (stereo points from the images) in the message, check if each feature is already in the map server, and update the observations for the corresponding (matching ID) IMU message. Then we update the tracking-rate based on $\frac{trackedFeatures}{stateFeatures+0.00001}$.

G. Measurement Update

The measurement update is another crucial component to the Kalman filter, as it computes the Kalman gain and updates the state values such as position, velocity, and orientation. First, we decompose the Jacobian matricies H and r using QR decomposition (with the Numpy function in "reduced" mode for sparcity) to reduce the computational complexity. We compute the Kalman gain following $K = linearSolve(S, H * P).T$ using $S = H * P * H.T + obsCov * I$. This formulation is in-line with the following Kalman gain from equation 29 of the original MSCKF paper, in Fig. 6.

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

Fig. 6. Equation 29 from original MSCKF paper to compute K for Kalman Gain

The correction to the state is then computed using equation 30, as $\Delta X = K * r$. The ΔIMU is the first 21 values of the ΔX. We then get the IMU-DQ using the quaternion small-angle formula on the first 3 IMU state components. We update orientation using quaternion multiplication of the previous orientation and the new IMU-DQ. The rest of the IMU state components are updated by adding the respective ΔIMU components for GyroBias, Velocity, AccelerometerBias, and Position. We then update the IMU rotation and translation components of the IMU to Camera frames using the small-angle quaternion of latter components 15-18 of the ΔIMU. After updating the IMU state components, we now update the camera states by looping through them and applying the respective rotation and translation components of the ΔX that correspond to each camera state. We again use the small-angle quaternion form and multiplication to compute the updates. Lastly, we force the state covariance matrix to be symmetric again using $stateCov = (stateCov + stateCov.T)/2$.

III. Results

Our project upload contains a video of the full flight, with a view of the full-flight path below in Fig. 8. You can see our flight path successfully tracks the ground truth very closely. We computed the ATE as 0.08m translation, shown in Table I.

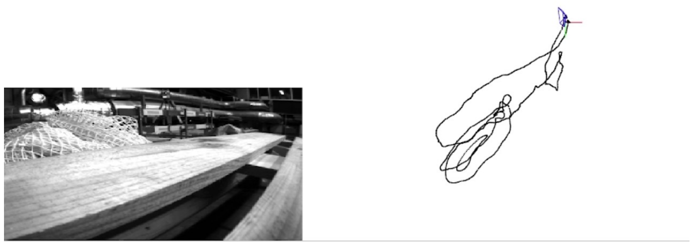

Fig. 7. Estimated Trajectory

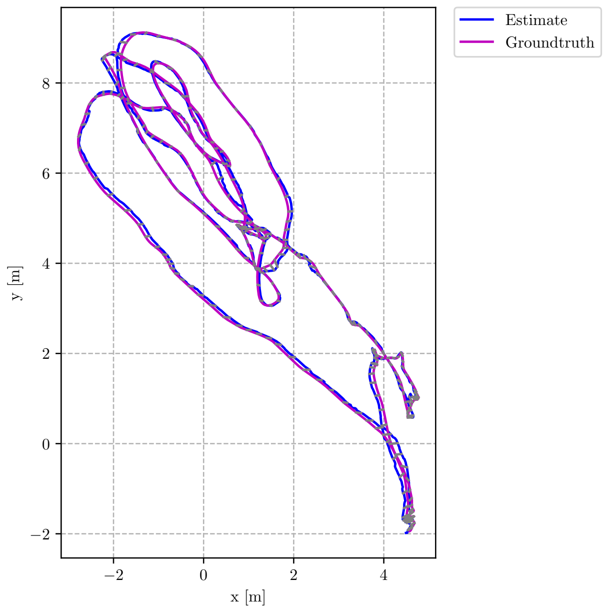

Fig. 8. Estimated Trajectory vs Groundtruth (via VICON)

The Absolute trajectory Error (ATE) and Relative Pose Error (RPE) using RMSE for the Rotation and Translation values have been mentioned in the Table below:

	ATE	RPE
Rotation	1.780931681713913	3.033415405486056
Translation	0.08175968722637134	0.22720598410625495

TABLE I
RMSE ATE AND RPE

This project was a great learning experience to teach us first-hand how a Kalman filter works through all of the important components, especially with predicting the next state, computing the process model, and updating the covariance and state. We are thankful for the OpenVINS resource (https://docs.openvins.com/pages.html) for showing the derivations through challenging parts. We also give credit to the code provided by this class to show us how the rest of the steps work, and credit to the original MSCKF author's Github repository (https://github.com/KumarRobotics/msckf_vio/) which gave us guidance through a few tricky parts that weren't fully clear in the paper.