# Project 4
# Phase 1 - Classical Visual-Inertial Odometry

Manoj Velmurugan*, Rishabh Singh†
Robotics Engineering
Worcester Polytechnic Institute
Email: *v.manoj1996@gmail.com, †rsingh8@wpi.edu

***Abstract*—The Multi-State Constraint Kalman Filter (MSCKF) is a visual-inertial odometry algorithm that utilizes an Extended Kalman Filter (EKF). This project presents a Python version of S-MSCKF (Stereo-MSCKF). MSCKF overcomes the complexity of traditional algorithms by not holding the feature positions of all detected features as a state in Kalman filter formulation.**

## I. Introducion

Figure 1 gives the overview of the implemented functions. We have developed the seven core functions of the stereo Multi-State Constraint Kalman Filter (MSCKF). This implementation is a direct conversion of the original C++, which is referenced here.

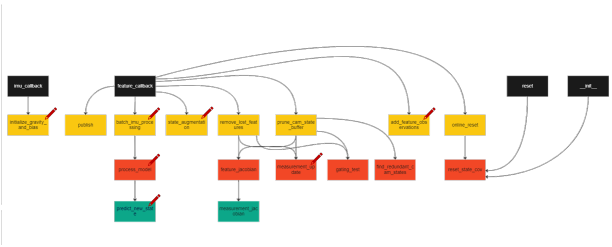

Fig. 1: Overview of the implemented functions

## II. INITIALIZE GRAVITY AND BIAS

To initialize the filter. gravity and gyroscope biases are set using the IMU messages we have in the buffer which we assumed to be captured while the robot remains static. The gyroscope bias (bg) is determined by calculating the average of these gyroscope readings. Gravity (g) is initialized to the vector [0, 0, $g_{norm}$], where $g_{norm}$ represents the magnitude of the gravity vector derived from the norm of the accelerometer readings in the buffer. The initial orientation of the robot is established using a quaternion that aligns the negative gravity vector (-g) with $g_{norm}$.

## III. BATCH IMU PROCESSING

We process the IMU messages kept in the buffer at a certain frequency. We check if the timestamp of the message is too old, and reject it. Using the new data, we update state estimation using the process model.

## IV. PROCESS MODEL

We model the IMU with the following equations:

$$
\begin{aligned}
{}^I_G\dot{\hat{\boldsymbol{Q}}}(t) &= \frac{1}{2}\Omega(\boldsymbol{\omega}(t)){}^I_G\hat{\boldsymbol{Q}}(t),\\
{}^I_G\boldsymbol{V}(t) &= {}^I_G\boldsymbol{C}\boldsymbol{a}(t),\\
{}^I_G\dot{\boldsymbol{P}}(t) &= {}^I_G\boldsymbol{V}(t),
\end{aligned}
$$

Here ${}^I_G\boldsymbol{q}(t)$ is a unit quaternion describing the rotation from global frame $\{G\}$ to IMU frame $\{I\}$. $\boldsymbol{\omega}(t) = [\omega_x, \omega_y, \omega_z]^T$ is the rotational velocity in the IMU frame. Also we have,

$$
\Omega(\boldsymbol{\omega}) = \begin{bmatrix} -\hat{\boldsymbol{\omega}} & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^T & 0 \end{bmatrix} \tag{1}
$$

$$
\lfloor\hat{\boldsymbol{\omega}}\rfloor = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{2}
$$

The final model for IMU error-state following the EKF formulation is:

$$
\dot{\tilde{\boldsymbol{x}}}_{IMU} = \boldsymbol{F}\tilde{\boldsymbol{x}} + \boldsymbol{G}_{IMU} \tag{3}
$$

Where $\boldsymbol{F}$ and $\boldsymbol{G}$ are

$$
\boldsymbol{F} = \begin{bmatrix} -\hat{\boldsymbol{\omega}}_x & -\boldsymbol{I}_3 & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & -\boldsymbol{C}(\tilde{\boldsymbol{q}})^T \\ -\boldsymbol{C}(\tilde{\boldsymbol{q}})^T\boldsymbol{A}_x & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{I}_3 & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \end{bmatrix} \tag{4}
$$

$$
\boldsymbol{G} = \begin{bmatrix} -\boldsymbol{I}_3 & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{I}_3 & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & -\boldsymbol{C}(\tilde{\boldsymbol{q}})^T & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{I}_3 \end{bmatrix} \tag{5}
$$

## V. PREDICT NEW STATE

The state is propagated by applying a fourth-order Runge-Kutta method, which provides a numerical solution for the differential equations governing the system's dynamics.

## VI. STATE AUGMENTATION

In the state augmentation function we incorporate the new camera into the existing state. This is done through the following equations:

$$\boldsymbol{J}_r = \begin{bmatrix} -\boldsymbol{C}(\boldsymbol{q}^{IG})^T & \boldsymbol{0}_{3x9} & \boldsymbol{I}_3 & \boldsymbol{0}_{3x3} \\ -\boldsymbol{C}(\boldsymbol{q}^{IG})^T \boldsymbol{T}_{\boldsymbol{p}^{cx}} & \boldsymbol{0}_{3x9} & \boldsymbol{I}_3 & \boldsymbol{0}_{3x3} \end{bmatrix} \tag{6}$$

The covariance matrix is propagated:

$$\boldsymbol{P}_{k+1|k} = \begin{bmatrix} \boldsymbol{P}_{II,k+1|k} & \boldsymbol{\Phi}_k \boldsymbol{P}_{IC,k} \\ \boldsymbol{P}_{IC,k}^T \boldsymbol{\Phi}_k^T & \boldsymbol{P}_{CC,k|k} \end{bmatrix} \tag{7}$$

$$\boldsymbol{P}_{k|k} = \boldsymbol{J} \begin{bmatrix} \boldsymbol{P}_{k+1|k} \\ \boldsymbol{P}_{k+1|k}^T \end{bmatrix} \boldsymbol{J}^T \tag{8}$$

## VII. ADDING FEATURE OBSERVATIONS

This method integrates the newly detected features from the latest frame into the existing feature map. Each feature is cataloged within the feature map alongside its unique feature ID and the ID of the current state.

## VIII. MEASUREMENT UPDATE

This method defines the heart of our estimation pipeline. It performs the following tasks,

- Reduce the residual state space by projecting the residuals to a lower dimensional space. This is performed by computing the QR decomposition of the residual. Premultiplying the residual with the transpose of Q matrix would make the problem computationally more efficient.
- Compute the Kalman gain.
- Perform Kalman filter update step using the reduced form of residuals.
- Add the residuals to the integrated IMU states. This will correct the integration errors. Update the state covariance matrix while ensuring that it is remaining symmetric.

The Kalman gain is obtained using Eqn. 9 and the state errors are computed using Eqn. 10.

$$K = PT_H^T(T_H P T_H^T + R_n)^{-1} \tag{9}$$

$$\Delta X = K r_n \tag{10}$$

## IX. RESULTS

The MS-EKF algorithm worked as expected and the trajectory of UAV was visualized live using Pangolin Fig. 2.

RPG Trajectory evaluation toolbox was used to analyze our outputs against ground truth.

The MSE errors are as follows,

- Rot RMSE - 1.8977
- Scale RMSE - 1.714
- Translation RMSE - 0.1122

The translation error over time is given in Fig. 3

The top view of the trajectory was compared against the ground truth. It shows good agreement as shown in Fig. 4

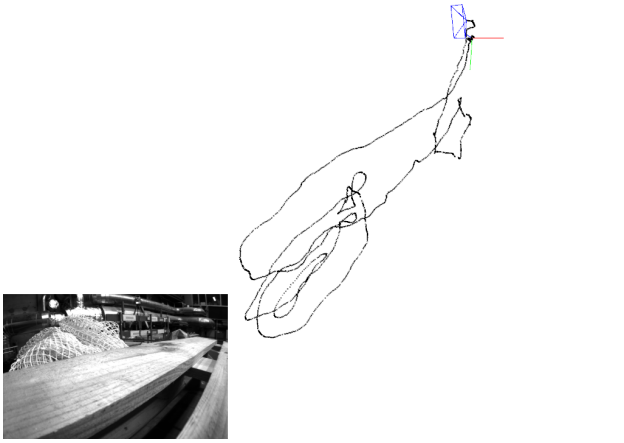

Fig. 2: Trajectory of the Estimated UAV Pose visualized in Pangolin

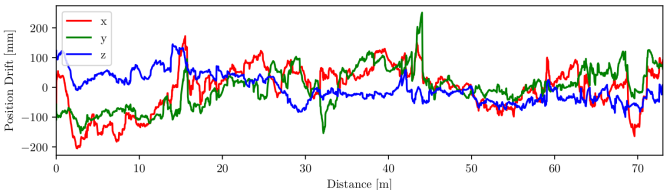


Fig. 3: Translation Error over time

## X. CONCLUSION

In conclusion, this project has introduced a Python implementation of the Stereo-MSCKF (S-MSCKF), a variant of the Multi-State Constraint Kalman Filter (MSCKF) visual-inertial odometry algorithm. By leveraging an Extended Kalman Filter (EKF), the MSCKF approach offers a streamlined solution that avoids the computational burden associated with traditional algorithms. Notably, the MSCKF framework does not treat all detected feature positions as individual states within the Kalman filter formulation, thereby simplifying the computational complexity while maintaining robust visual-inertial odometry performance.

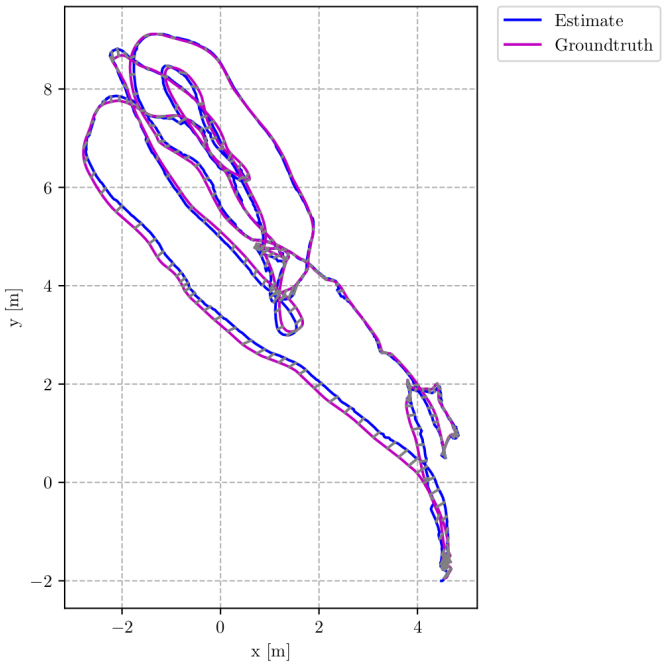


Fig. 4: Ground Truth Vs Position Estimate