RBE/CS 549 Computer Vision

P4 - Virtual Inertial Odometry

Uthiralakshmi Sivaraman
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, USA
usivaraman@wpi.edu

Noopur Koshta
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, USA
nkoshta@wpi.edu
Using 1 late day

***Abstract*—This project implements a filter-based stereo visual inertial odometry that uses the MultiState Constraint Kalman Filter (MSCKF).**

***Index Terms*—VIO, MSCKF**

I. Introduction

The objective of the project is to obtain depth from an image and thus estimate scale. However, obtaining depth from a single camera without any prior information about the environment is not possible. One simpler alternative solution is to utilize a stereo camera with a known pose, where depth can be directly estimated by matching features. But, matching is expensive and hard, and it does not work when there is motion blur, which is common on robots. To solve this, we can use an Inertial Measurement Unit (IMU) which measures linear and angular acceleration. The complementary nature of IMU and camera can be used for multi-modal fusion to estimate accurate camera pose and backtrack depth. This methodology has been used in the DARPA Fast Light Autonomy (FLA) project, where a team achieved 20m/s autonomous flight using a VIO approach. It is important to note that prior information is necessary even for deep learning networks to estimate depth from a single image, and training on a diverse dataset is necessary for out-of-domain generalization. IN this report, we talk about the classical approach of solving the Visual Inertial Odometry using Multi State Kalman Filter.

II. Implementation

A. *Initialize Gravity and Bias*

This function initializes the bias and initial orientation based on the starting IMU reading. Initially, the angular and linear velocity are obtained by averaging the first few readings in the imu msg buffer. The gyro bias is then initialized using the average angular velocity, and gravity is obtained from the linear velocity. The normalized gravity vector is passed as the IMU state. We then use these two vectors to initialize the initial orientation, ensuring that the estimation is consistent with the inertial frame. Finally, the quaternions are converted to rotation matric and are passed as the orientation state of the IMU.

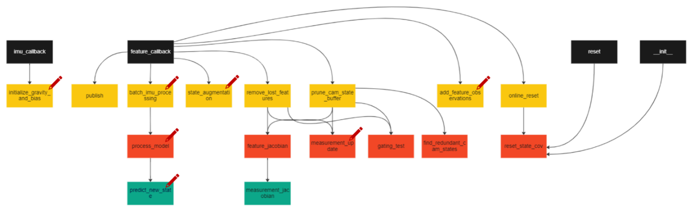


Fig. 1. Function structure of the msckf.py. A red pencil icon indicates that we need to implemented that function

$$X_I = ({}^I_G\mathbf{q}^T \ \ \mathbf{b}_g^T \ \ {}^G\mathbf{v}_I^T \ \ \mathbf{b}_a^T \ \ {}^G\mathbf{p}_I^T \ \ {}^I_C\mathbf{q}^T \ \ {}^I\mathbf{p}_c^T)^T$$

B. *Batch Imu Processing*

This function processes the imu messages in the imu msg buffer based on the specified time bound. We initiate the process model for every imu input in each time bound and repeat the process until the time bound is reached. Following this, the current imu id is updated to the next state imu id. Finally, all the unused imu messages are removed from the imu msg buffer.

C. *Process Model*

This function computes the dynamics (pose) of the camera module state based on the most recent IMU state update. The function takes in the time, m gyro (current angular velocity), and m acc (current linear acceleration) as arguments. Initially, we obtain the IMU state updates and evaluate the linearized continuous dynamics for the error IMU state. We then calculate the discrete transition matrices F and Q. Then, we approximate the matrix exponential to the 3rd order. We propagate to the state and predict new state using the 4th order Runge-Kutta method.

$$\tilde{X}_I = ({}^I_G\tilde{\theta}^T \ \ \tilde{\mathbf{b}}_g^T \ \ {}^G\tilde{\mathbf{v}}_I^T \ \ \tilde{\mathbf{b}}_a^T \ \ {}^G\tilde{\mathbf{p}}_I^T \ \ {}^I_C\tilde{\theta}^T \ \ {}^I\tilde{\mathbf{p}}_c^T)^T$$

$$\dot{\tilde{X}}_I = F\tilde{X}_I + Gn_I$$

$$F = \begin{bmatrix} -\lfloor\hat{\omega}_\times\rfloor & -I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ -C({}^I_G\mathbf{q}^T)\lfloor\hat{a}_\times\rfloor & 0_{3\times3} & 0_{3\times3} & -C({}^I_G\mathbf{q}^T) & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix}$$

$$G = \begin{bmatrix} -I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & -C({}^I_G\mathbf{q}^T) & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & I_3 \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix}$$

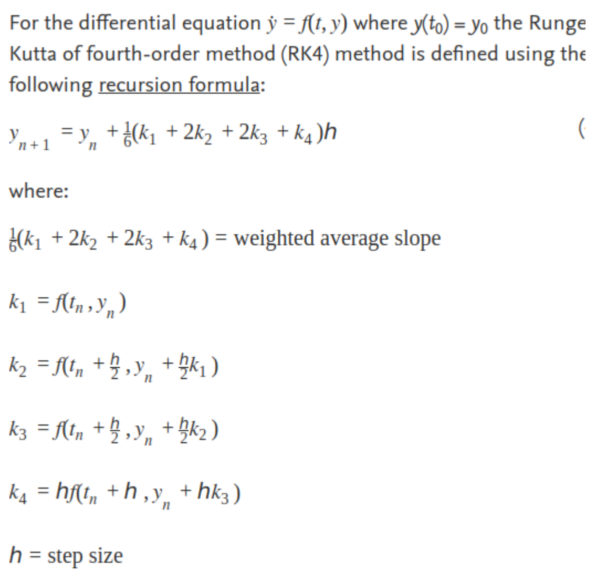
$$\phi = I_{21\times21} + F(\tau).d\tau + \frac{1}{2} * (F(\tau).d\tau)^2 + \frac{1.0}{6.0} * (F(\tau).d\tau)^3$$

D. Predict New State

The 4th order Runge-Kutta method is utilized in this function to propagate the state. The function takes in the time step, gyro, and acceleration for the given state as input. Initially, we calculate the normalized value of the error state of angular velocity (gyro). Next, we obtain the current orientation, velocity, and position values from the imu state server. Utilizing the current values and the omega values, we compute the angular velocity and the angular acceleration. These values are further approximated utilizing the Runge-Kutta method.

E. State Augmentation

We will use this function to calculate the state covariance matrix in order to propagate the state's uncertainty. Initially, we obtain the IMU and camera state values, which include



For the differential equation $\dot{y} = f(t, y)$ where $y(t_0) = y_0$ the Runge Kutta of fourth-order method (RK4) method is defined using the following recursion formula:

$$y_{n+1} = y_n + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h \qquad ($$

where:

$\tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ = weighted average slope

$k_1 = f(t_n, y_n)$

$k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1)$

$k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2)$

$k_4 = hf(t_n + h, y_n + hk_3)$

h = step size

$$P_{k|k} = \begin{bmatrix} J_{21+6N} \\ J \end{bmatrix} P_{k|k} \begin{bmatrix} J_{21+6N} \\ J \end{bmatrix}^T$$

$$K = PT_H^T(T_H PT_H^T + R_n)^{-1} \qquad \Delta X = Kr_n$$

the rotation from the IMU to camera and the translation vector from the camera to the IMU. Then, we add a new camera state to the state server by utilizing the initial IMU and camera state. Following that, we resize the state covariance matrix and propagate the covariance of the IMU state. The resulting augmented covariance matrix is provided.

F. Feature Observations

In this function, the feature message is obtained as input, and the current IMU state ID is retrieved along with the number of features. Each feature is then added to the map server one by one, if it does not already exist in the server. The count of the number of tracked features is also updated for every given state, and the map server is continuously updated and all features are tracked. The tracking rate is then calculated as the ratio of the number of tracked features to the total number of current features available. Overall, this function is responsible for adding new features to the map server and continuously tracking existing features, allowing for accurate and efficient mapping.

G. Measurement Update

This function employs a measurement model to update the state estimates. A residual r is defined that depends linearly on the state errors. Subsequently, we aim to simplify the Jacobian matrix by utilizing QR decomposition. We then calculate the Kalman gain and state error.

Using this state error, we update the IMU state first, followed by the camera states. Finally, the state covariance is updated, and the covariance matrix is modified to be symmetric.

III. Results

This project utilizes input data from the MH01easy subset of the EuRoC dataset. The output generated by the project for this data is depicted in the accompanying figure and is consistent with the anticipated output.

References

[1] https://rbe549.github.io/fall2022/proj/p4/
[2] https://arxiv.org/pdf/1712.00036.pdf
[3] https://www-users.cse.umn.edu/ stergios/papers/ICRA07-MSCKF.pdf
[4] https://github.com/KumarRobotics/msckf vio