Visual Inertial Odometry for a Quadrotor

MEAM 620 Project 2, Phase 2

April 27, 2020

1 Introduction

This assignment investigates the problem of estimating the pose of a flying robotic platform using information obtained from its onboard sensors, in this case an onboard inertial measurement unit (IMU), and an onboard stereo pair. In this assignment we will be fusing both sources of information together in an Error State Kalman Filter based on the notes and the paper we discussed in class. Please download the latest version of the lecture slides since changes have been made to improve clarity. Note that we are also providing a link to the underlying paper which describes the techniques you will be implementing.

In order to further ground the assignment we are having you apply the code you write to real data acquired by a quadrotor platform. For this we are using portions of the EuRoc dataset collected at ETH Zurich. We are providing a paper describing this dataset and the platform on which it was collected as one of the documents associated with this assignment. I would definitely reccomend taking a quick look at this paper.

2 Code Organization

The code packet that you are being provided with as part of this assignment is organized in the usual manner. In the top level directory there is a file entitled `setup.py` which you should run in order to install all of the needed packages. The `proj2_2` package is divided into 3 subdirectories.The `util` directory contains a set of tests you can run on your code via the unit test script `test.py`. The `dataset` directory contains a small portion of the EuRoc dataset which is comprised of stereo and IMU data.

The code directory contains a set of code files and sandbox files which constitute the coding assignment. The file `stereo.py` contains a series of utilities that we are providing for reading information from the dataset, you should not modify this file. The file `vio.py` contains function stubs to implement the key steps in the ESKF filter algorithm described in class. This is the only file that you should modify, it contains comments indicating the parts we expect you to fill in.

The `sandbox_vio.py` script runs your completed ESKF implementation against the provided stereo and IMU dataset to estimate the pose of the aircraft. After it runs it produces a plot showing the estimated position and orientation over time. Note that this script will produce garbage results until you complete the code. Note that running this script can take 20 seconds or more since it runs against several thousand stereo and IMU measurements. We show you what the final plot should look like in the PDF files that we provide in the `util` directory.

3 Task 1: Nominal State Update

One core step in the ESKF algorithm involves updating the nominal state based on the prevailing measurements from the IMU. This function takes in a tuple which represents the current nominal state, two

3×1 vectors that represent the angular velocity and linear acceleration and a scalar value dt indicating the duration of the timestep. The tuple that you are provided contains all of the elements of the state: the current position, $p \in \mathbb{R}^3$, the current velocity, $v \in \mathbb{R}^3$, the current rotation, q, which is provided as a Rotation object, the accelerometer bias, $a_b \in \mathbb{R}^3$, the gyroscope bias, $\omega_b \in \mathbb{R}^3$, and the current estimate for the gravity vector, $g \in \mathbb{R}^3$. Note that all of these inputs, except q, are provided as 3×1 vectors.

The output of this function should be a new tuple in the same format indicating the nominal state of the system at the end of the timestep.

4 Task 2: Covariance Update

The next function is used to update the error state covariance matrix in response to an IMU update. The inputs to this function include all of the inputs to the previous function as well as an initial error state covariance matrix in $\mathbb{R}^{18\times18}$. In order to apply the covariance updates specified in the slides you will need a few parameters that define the noise covariance matrix.

The following table gives the name of the parameter and the quantity in the slides it refers to:

accelerometer_noise_density	$\sigma_{\tilde{a}_n}$
gyroscope_noise_density	$\sigma_{\tilde{\omega}_n}$
accelerometer_random_walk	σ_{a_w}
gyroscope_random_walk	σ_{ω_w}

The mysterious term $R^T\{(\omega_m - \omega_b)\Delta t\}$ refers to the transpose of the rotation matrix formed by taking the vector, $(\omega_m - \omega_b)\Delta t$, and viewing it as specifying a rotation about that axis by an angle corresponding to the vector length in radians. Note that the Rotation class provides a member function called from_rotvec that you may find useful for this purpose.

The notation $[a_m - a_b]_\times$ denotes the 3×3 skew symmetric matrix associated with the vector $(a_m - a_b)$

5 Task 3: Measurement Update Step

Your final task is to implement a function that updates the nominal state and the error state covariance matrix based on a single image measurement provided as a 2×1 vector in the parameter uv. The parameter $P_w \in \mathbb{R}^3$ denotes the coordinates of the point with respect to the world coordinate system.

The first thing that you should compute is the innovation vector which is simply the difference between the measured image coordinates, uv, and the predicted value, $(X_c/Z_c, Y_c/Z_c)$. The result should be a 2×1 vector. In order to account for outliers we compare the magnitude of this vector to a specified error threshold, if the vector magnitude is greater than this quantity no update is performed and the original state and covariance are returned, otherwise we compute updated versions of the nominal state and error state covariance according to the prescription given in the lecture slides. Basically we end up performing a modified version of the Extended Kalman Filter updating step. Note that once we calculate the error state update, $\delta\mathbf{x}$, we use this vector to update the nominal state estimate.

We also require that when computing the updated covariance matrix that you use the form specified in the slides that is guaranteed to produce a symmetric positive definite output namely:

$$\Sigma_t = (I - K_t H_t)\Sigma'_t(I - K_t H_t)^T + K_t Q_t K_t^T$$

The final output is a tuple containing the new nominal state, the new error covariance and the computed innovation vector. Note that this is returned whether or not the innovation magnitude is within bound, if no update is performed the resulting nominal state and covariance matrix are the same as the ones provided as inputs.

6 Submission

When you are finished, submit your code via Gradescope which can be accessed via the course Canvas page. Your submission should contain:

1. A `README` file detailing anything we should be aware of (collaborations, references, etc.).
2. The file `vio.py` as well as any other Python files you need to run your code

Shortly after submitting you should receive an e-mail from `no-reply@gradescope.com` stating that your submission has been received. Once the automated tests finish running the results will be available on the Gradescope submission page. There is no limit on the number of times you can submit your code.

Please do not attempt to determine what the automated tests are or otherwise try to game the automated test system. Any attempt to do so will be considered cheating, resulting in a 0 on this assignment and possible disciplinary action.