Pose Estimation for a Quadrotor

MEAM 620 Project 2, Phase 1

April 10, 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. To do this we will be implementing two algorithms that were presented in the course lectures namely the complementary filter which will be used to estimate the orientation of the platform and a stereo odometry algorithm which can be used to estimate the position and orientation of the platform over time.

In order to further ground the assignment we are having you apply the algorithms 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_1` 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 `complementary_filter.py` contains stubs for the implementation of the complementary filter algorithm described in class. The file `estimate_pose_ranscac.py` contains half of the implementation of the RANSAC based stereo odometry system described in the lectures.

The remaining sandbox scripts are codes that we provide for testing your code. The script `sandbox_stereo.py` reads a pair of stereo images from the dataset and plots some figures to show you the outputs. You should be able to run this script successfully after you have run the setup script, if you get errors it means that something isn't configured correctly. You should feel free to edit this script to run the analysis on other stereo pairs in the dataset.

The other sandbox scripts test your code against the provided dataset. The `sandbox_complementary_filter.py` script tests your complementary filter implementation by running it against a complete IMU dataset and plotting the result. We show you what the final plot should look like in the file `util/MachineHall01_imu0.png`. Note that running this script can take 20 seconds or more since it runs against several thousand IMU measurements. The `sandbox_estimate_pose.py` script runs your completed RANSAC implementation against the provided stereo 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 initially both scripts should run but will produce garbage results until you complete the code.

3 Task 1: Complementary Filter

Your first task is to provide an implementation for the complementary_filter_update function found in the file `complementary_filter.py`. This file takes as input an initial rotation estimate expressed as a scipy.spatial.transform.Rotation object. It also takes two 3×1 vectors which denote the angular velocity during the interval, the measured acceleration at the end of the interval and the duration of the interval. The units are specified in the comments. The output from this function should be the new estimate for the rotation at the end of the interval again returned as a rotation object. You should follow the implementation provided in the course lecture slides.

A few things to be aware of, firstly the Rotation class that we want you to use expresses quaternions in a different format than the one used in the lecture slides, namely (x, y, z, w) as opposed to (w, x, y, z). Another important difference is that for the dataset we are testing on the x axis of the IMU is roughly aligned with the gravity direction, not the z axis. So we desire $R_{1k}g$ to be e_x not e_z which means that you will need to use a slightly different quaternion construction to come up with the rotation correction q_{acc}. Lastly the units of the acceleration measurements are expressed in meters per second squared, not in units of g's so you must remember to normalize correctly.

4 Task 2: Least Squares Solve

In order to complete the RANSAC implementation provided in `estimate_pose_ransac.py` you will need to implement a function which estimates the rotation and translation from a set of stereo correspondences. In this case the function you are asked to write, solve_w_t, which takes as input two $3 \times n$ arrays indicating the n corresponding stereo measurements in frames 1 and 2 respectively. The first row of each array contains the normalized coordinate $u' = (X/Z)$, the second row the normalized coordinate $v' = (Y/Z)$ and the last row the normalized disparity $d' = (1/Z)$. The last parameter is a rotation object corresponding to the initial base rotation R_0. Your function should return two 3×1 vectors corresponding to the recovered w and t estimates. You may find it useful to consider the numpy library function lstsq which computes a least squares estimate for a linear system.

5 Task 3: Finding inliers

Your previous function is designed to compute estimates for rotation and translation without considering inliers. As part of the RANSAC procedure we are required to identify inlier sets to a proposed solution and this is what your function find_inliers is supposed to do. This function should take as input the two 3×1 vectors corresponding to the proposed w and t values, the aforementioned matrices uvd1 and uvd2 containing the normalized stereo measurements, the initial base rotation R_0 and a threshold value which is used to decide which correspondences are inliers and which are not.

To decide which correspondences are inliers you will need to compute the discrepancy vector δ, which is described in the lecture slides, for each of the correspondences in turn. Correspondences for which $\|\delta\|$ is less than the threshold value are considered inliers otherwise they are considered outliers. Your routine should produce as output a boolean array with n entries, one for each stereo correspondence. Correspondences which are inliers should be indicated with a True value in the output array and those which are outliers should be indicated with a False entry.

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. Files `complementary_filter.py` and `estimate_pose_ransac.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.