CMSC 828T
Vision, Planning and Control in Aerial Robotics
Project 2 Phase 2 (P2Ph2):
Vision-based 3-D Velocity Estimator
Due on 11:59:59PM on Oct 27$^{\text{th}}$, 2017

Prof. Yiannis Aloimonos, Nitin J. Sanket,
Kanishka Ganguly, Snehesh Shrestha

October 22, 2017

1 Introduction

In this phase, you will implement a vision-based 3-D velocity estimator and compare against the ground truth given by motion capture system by `Vicon`. Sections 2, 3, and 4 will give you a logical flow of the estimation procedure.

1.1 Environment

The data for this phase was collected using a quadrotor that was either held by hand or flown through a prescribed trajectory over a mat of AprilTags, each of which has a unique ID that can be found in `parameters.txt`. The `map.pdf` shows the layout of the AprilTag mat. The tags are arranged in a 12 x 9 grid. The top left corner of the top left tag should be used as coordinate (0, 0) with the `X` coordinate going down the mat and the `Y` coordinate going to the right. Each tag is a 0.152 m square with 0.152 m between tags with the exception of the space between columns 3 and 4, and 6 and 7, which is 0.178 m. Using this information you can compute the location of every corner of every tag in the world frame.

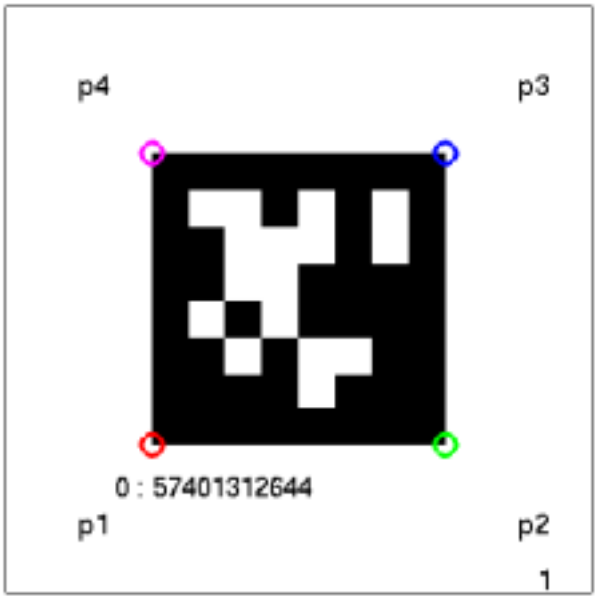
1.2 Calibration

The intrinsic camera calibration matrix and the transformation between the camera and the robot center are given in `parameters.txt`. Two photos (`top_view.jpg` and `side_view.jpg`) are included to visualize the camera-quadrotor transform. You will need to transform your camera-based pose estimate from the camera frame to the robot frame so that you can compare it against the ground truth `Vicon` data.

1.3 Data

The data for each trial is provided in a `mat` file. The file contains a struct array of image data called `data`, which holds all of the data necessary to do pose estimation. The format of image data struct contains the following data:

1. Time stamp (`t`) in seconds.
2. ID of every AprilTag that is observed in the image (`id`).
3. The center (`p0`) and four corners of every AprilTag in the image. The corners are given in the order bottom left (`p1`), bottom right (`p2`), top right (`p3`), and top left (`p4`) as shown in the figure. The *ith* column in each of these fields corresponds to the *ith* tag in the ID field and the values are expressed in image coordinates.



4. Rectified image (`img`).
5. IMU data with Euler angles (`rpy`), angular velocity (`omg`), and linear acceleration (`acc`).

The mat file also contains Vicon data taken at 100 Hz, which will serve as our ground truth measurements. The Vicon data is stored in two matrix variables, `time` and `vicon`. The `time` variable contains the timestamp while the `vicon` variable contains the Vicon data in the following format:

$$\begin{bmatrix} x & y & z & roll & pitch & yaw & v_x & v_y & v_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^T \tag{1}$$

2 Corner Extraction and Tracking

You will first need to extract corners in each image. You may choose to use MATLAB's built in corner detectors [3, 2], external MATLAB-based computer libraries [4], find implementations on the Internet, or write your own corner detector. You are safe to assume that all detected corners will be on the ground plane. Note that the extracted corners should include not only the AprilTag [1] corners, but also corners from the filler images between the AprilTags.

For all corners in the image, you will then compute the sparse optical flow between two consecutive images using the KLT tracker [5]. You may again choose between [3, 2, 4], or find the tracker implementation on the Internet, or write your own implementation. This sparse optical flow, together with the time stamp of the image, will give you $[\dot{x}, \dot{y}]^T$ in the calibrated image frame. Note that the timestamps can have a bit of jitter, which adds noise to the time measurement. You can assume that the images were taken approximately at a constant rate and simply use a low-pass filter on the ΔT to get better results.

3 Velocity Estimation

Given a corner in the calibrated image frame $[x, y]^T$ and its optical flow $[\dot{x}, \dot{y}]^T$, the following linear equation holds:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} f_1(x, y, Z) \\ f_2(x, y, Z) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \\ \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \tag{2}$$

where $\begin{bmatrix} V_x, V_y, V_z, \Omega_x, \Omega_y, \Omega_z \end{bmatrix}^T$ are linear and angular velocities to be estimated. Both the functions $f_1(.)$ and $f_2(.)$ return 1×6 vectors. Z is the depth of the corner. Assuming that all corners are on the floor, Z can be computed based on the estimated height and rotation of the quadrotor using the AprilTags. Note that Z does not equal to the height of the quadrotor due to nonzero roll and pitch.

4 RANSAC-Based Outlier Rejection

It is likely that there are outliers in the optical flow computation. You will need to reject outliers using RANSAC. Three sets of constraints are required to solve the system above equation (1), and thus a 3-point RANSAC can be used for outlier rejection. It is recommended to recompute the velocities using equation (1) with all inliers.

5 Compare Against Ground Truth

Using the ground truth data provided, which are available as two variables, `time` and `vicon`. The `time` variable contains the timestamp for all the Vicon data while the `vicon` variable contains the actual data.

Note that the optical flow-based velocity estimation will be in the frame of the camera, while the Vicon velocity is given in the world frame. You will need to transform the coordinate frames between the camera, the quadrotor, and the world to properly compare your velocity estimates against the ground truth. The Vicon velocity estimate can be noisy at some point, but you should be able to get a reasonable ground truth during majority of the flight time.

6 Submission

When you are finished you will submit your code via CMSC 828T website submit section. You should create a folder called `code` and copy all the files into it, zip it, and submit `code.zip`. Please note the zip file needs to be `.zip` format. Any other format is not valid. Please note:

- Do NOT add or submit any sub-folders.
- Do NOT submit any visualization code. If you have any either remove them or comment them out.
- Do NOT print out any outputs. If you have any debug code printing outputs to the console, please remove them or comment them out.
- Only include the files that are listed below and any new dependent m-files you might have created.
- Do NOT submit any other files that are not necessary and was created only for your testing/ debugging.

Your submission should contain:

- A README.txt detailing anything we should be aware of.
- A LateDays.txt with a single number letting us know how many of the late days you wish you exercise for this submission.
- All necessary files inside one folder `code` such that we can just run the automated testing script to see your results. The expected files for this assignments are:
  - `init_script.m`
  - `estimate_velocity.m`
  - And any other new m-files you might have created that is necessary for running your code.

7 Acknowledgement

The project has been adapted from the University of Pennsylvania MEAM 620 course.

8 Collaboration Policy

You can discuss with any number of people. But the solution you turn in MUST be your own. Plagiarism is strictly prohibited. Plagiarism checker will be used to check your submission. Please make sure to **cite** any references from papers, websites, or any other student's work you might have referred.

References

[1] AprilTags. https://april.eecs.umich.edu/software/apriltag.html.

[2] MATLAB Computer Vision System Toolbox. http://www.mathworks.com/products/computer-vision/.

[3] MATLAB Image Processing Toolbox. http://www.mathworks.com/products/image/.

[4] VLFeat. http://www.vlfeat.org/.

[5] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.