MEAM 620 Project 2 Phase 2

Due: Thursday, April 4th, 2019 at 11:59pm

1 Overview

In this phase you will implement a vision-based 3D velocity estimator complete with feature detection and tracking as well as outlier rejection using RANSAC. The dataset and calibration parameters from Phase 1 will also be used in this phase and, as before, your results will be compared against ground truth measurements provided by VICON. The following sections detail the various components of the estimation procedure.

2 Corner Extraction and Tracking

The first step towards estimating velocity is extracting features in the form of corners from the provided camera imagery. MATLAB's computer vision toolbox provides a variety of methods for accomplishing this task including: `detectFASTFeatures`, `detectHarrisFeatures`, and `detectMinEigenFeatures` (see the documentation on point feature types for more information [1]). While other approaches exist, you must use one of these three in your implementation. For this project you can safely assume all detected corners lie on the ground plane. Also note that detected corners may belong to the inter-tag-scribles in addition to the AprilTag corners.

Armed with a set of corners from the detector, the next step is to compute the sparse optial flow of these features as they propagate through consecutive images. To do this, you will make use of MATLAB's implementation of the KLT algorithm [2][3]. The computed 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:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1(x, y, Z) \\ \mathbf{f}_2(x, y, Z) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \\ \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \tag{1}$$

where $[V_x, V_y, V_z, \Omega_x, \Omega_y, \Omega_z]^T$ are the linear and angular velocities to be estimated, $\mathbf{f}_1(.)$ and $\mathbf{f}_2(.)$ return 1×6 vectors, and 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 (part of the results from Phase 1). Note that Z does not necessarily equal the height of the quadrotor due to nonzero roll and pitch.

4 Outlier Rejection with RANSAC

It is likely you will encounter outliers in the optical flow computation which will will greatly reduce the accuracy of your results. You will need to reject these outliers using RANSAC. Since three sets of constraints are required to solve Equation 1, a 3-point RANSAC can be used for outlier rejection. After running RANSAC, it is recommended to recompute the velocities using Equation 1 with all the inliers.

5 Compare Against Ground Truth

As in Phase 1, the VICON data is present as two variables, `time` and `vicon`. The `time` variable contains the timestamp for the VICON data while the `vicon` variable contains the actual data in the following format:

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

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 points, but you should be able to get a reasonable ground truth estimate during majority of the flight time.

6 Code

You must implement the estimation steps outlined above in `estimate_vel.m`, provided to you as a function template. As before, the first input to `estimate_vel.m` will be a `sensor` struct including a grayscale image, captured AprilTag ids, corresponding AprilTag image coordinates, and the current timestamp (see the header description in `estimate_vel.m` for complete information). You are free to specify additional inputs using `init_script.m`. Since the sensor struct passed to `estimate_vel.m` contains a single image and you are interested in computing optical flow between successive images, you will need to make use of persistent variables [4] to ensure pertinent information survives to the next function call. Similar to Phase 1, if the data contained in `sensor` is invalid or no AprilTag ids are present you must return empty velocity vectors.

7 Submission

As always, we'll use the `turnin` system for submission. The project name for this phase is called `proj2phase2`, so the command you'll use for `turnin` would be

```
$ turnin -c meam620 -p proj2phase2 -v *
```

Your submission should include:

- A `README` file containing anything specific we should be aware of.
- Files `init_script.m`, `estimate_vel.m` and any additional files your code requires to run.

Shortly after submitting using `turnin` you should receive an email from `meam620@seas.upenn.edu` stating that your submission has been received. After that, you can check the status of your submission on the monitor webpage at https://alliance.seas.upenn.edu/~meam620/monitor/.

Once the automated tests finish for your code, you should receive an email containing your test results. This email will contain plots of your estimates compared to the ground truth and also tell you how long each iteration of your code takes. You should write your own functions for evaluating the error between your estimate using vision and the ground truth. Your grade would depend on the time taken per iteration and the error between your estimate compared to ground truth.

References

[1] Point Feature Types, https://www.mathworks.com/help/vision/ug/point-feature-types.html

[2] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in Proc. of the Intl. Joint Conf. on Artificial Intelligence, Vancouver, Canada, Aug. 1981, pp. 24-28.

[3] Matlab vison.PointTracker class (KLT implementation), https://www.mathworks.com/help/vision/ref/vision.pointtracker-system-object.html

[4] persistent, https://www.mathworks.com/help/matlab/ref/persistent.html