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 3rd, 2017

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

October 27, 2017

1 Introduction

In this phase, you will implement a vision-based 3-D velocity estimator and compare against a provided dataset gathered using some certain reliable and robust method. Sections 2, 3, and 4 will give you a logical flow of the estimation procedure.

1.1 Environment

The provided data for this phase was collected with a hand-held SLAMDunk sensor module [6] (shown in Fig. 1), manufactured by Parrot®, simulating flight patterns over a floor mat of AprilTags [1] each of which has a unique ID. The data files can be downloaded from here. The data contains the rectified left camera images, IMU data, SLAMDunk's pose estimates, AprilTag [1] detections with tag size and camera intrinsics and extrinsics. All units are in m, rad, rads^{-1} and ms^{-2} if not specified.

1.2 Calibration

Camera Intrinsics and Extrinsics are given specifically in `CalibParams.mat` and has the following parameters:

- `K` has the camera intrinsics (assume that the distortion coefficients in the radtan model are zero).
- `TagSize` is in size of each AprilTag in meters.
- `qIMUToC` has the quaternion to transform from IMU to Camera frame `(QuaternionW, QuaternionX, QuaternionY, QuaternionZ)`.

- `TIMUToC` has the translation to transform from IMU to Camera frame `(TransX, TransY, TransZ)`.

1.3 Data

The `.mat` file for any of the sequence (in the format `DataNAME_OF_SEQUENCE.mat` for e.g., `DataSquare.mat` where Square is the sequence name) will contain the following data:

- `DetAll` is a cell array with AprilTag detections per frame. For e.g., frame 1 detections can be extracted as `DetAll{1}`. Each cell has multiple rows of data. Each row has the following data format:
  - `[TagID, p1x, p1y, p2x, p2y, p3x, p3y, p4x, p4y]`
    Here `p1` is the left bottom corner and points are incremented in counter-clockwise direction, i.e., the `p1x, p1y` are coordinates of the bottom left, `p2` is bottom right, `p3` is top right, and `p4` is top left corners (Refer to Fig. 2). You will use the left bottom corner (`p1`) of Tag 10 as the world frame origin with positive X being direction pointing from `p1` to `p2` in Tag 10 and positive Y being pointing from `p1` to `p4` in Tag 10 and Z axis being pointing out of the plane (upwards) from the Tag.
  - `IMU` is a cell array where each row has the following data
    `[QuaternionW, QuaternionX, QuaternionY, QuaternionZ, AccelX, AccelY, AccelZ, GyroX, GyroY, GyroZ, Timestamp]`
    You do not need the IMU readings for this project, however, if you find a creative way to use it, please feel free to use it.
  - `TLeftImgs` is the Timestamps for Left Camera Frames (Images). For e.g. Frame 1 was collected at time `TLeftImgs`(1).
  - `Pose` is an array with each row in the form
    `[PosX, PosY, PosZ, QuaternionW, QuaternionX, QuaternionY, QuaternionZ, EulerX, EulerY, EulerZ, Timestamp]`
    Here ZYX Euler angle was used which is the default for MATLAB's `quat2eul` function.
  - `PoseGTSAM` is an array of our GTSAM pose outputs from with each row in the form
    `[PosX, PosY, PosZ, QuaternionW, QuaternionX, QuaternionY, QuaternionZ]`
    Here the row number corresponds to frame number, if pose doesn't exist or is erroneous due to missed tag detections then the whole row will be zeros. You can compute valid indexes by using the following command
    `ValidIdxs = ~any(PoseGTSAM)`.
  - `PoseiSAM2` is an array of our iSAM2 pose outputs from with each row in the form
    `[PosX, PosY, PosZ, QuaternionW, QuaternionX, QuaternionY, QuaternionZ]`
    Here the row number corresponds to frame number, if pose doesn't exist or is erroneous due to missed tag detections then the whole row will be zeros.
- The provided data for comparison also has a similar format:

- `PoseGTSAM` is an array of our GTSAM pose outputs from with each row in the form `[PosX, PosY, PosZ, QuaternionW, QuaternionX, QuaternionY, QuaternionZ]` Here the row number corresponds to frame number, if pose doesn't exist or is erorneous due to missed tag detections then the whole row will be zeros. You can compute valid indexes by using the following command `ValidIdxs = ~any(PoseGTSAM)`.
- `PoseiSAM2` is an array of our iSAM2 pose outputs from with each row in the form `[PosX, PosY, PosZ, QuaternionW, QuaternionX, QuaternionY, QuaternionZ]` Here the row number corresponds to frame number, if pose doesn't exist or is erorneous due to missed tag detections then the whole row will be zeros.
- `VelGTSAM` is an array of our velocities computed from GTSAM pose outputs from with each row in the form `[Vx,Vy,Vz]` Here the row number corresponds to frame number, if velocity doesn't exist or is errorneous due to missed tag detections then the whole row will be zeros. You can compute valid indexes by using the following command `ValidIdxs = ~any(VelGTSAM)`.
- `PoseiSAM2` is an array of our velocities computed from iSAM2 pose outputs from with each row in the form `[Vx,Vy,Vz]` Here the row number corresponds to frame number, if velocity doesn't exist or is errorneous due to missed tag detections then the whole row will be zeros. You can compute valid indexes by using the following command `ValidIdxs = ~any(VeliSAM2)`.

The Images are given as a `.zip` file with the name `NAME_OF_SEQUENCEFrames.zip` where the image name is `FrameNumber.jpg` and the timestamp for each Frame Number is given in the `DataNAME_OF_SEQUENCE.mat` file's `TLeftImgs` variable, i.e., timestamp for `1.jpg` can be found as `TLeftImgs`(1).

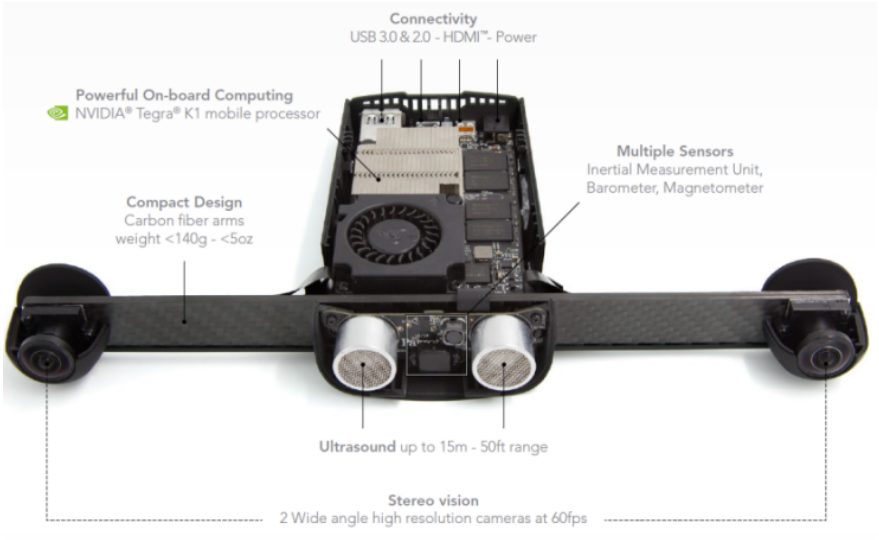

Figure 1: Parrot SLAMDunk.

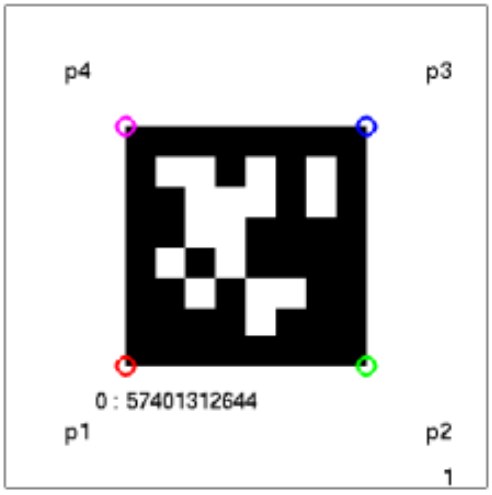

Figure 2: Corners of an AprilTag.

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 [2, 3], 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 cityscape background 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 [2, 3, 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. Again as a reminder you can get timestamps from `TLeftImgs`.

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:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} f_1(x, y, Z) \\ f_2(x, y, Z) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \\ \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \tag{1}$$

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 Provided Dataset

The provided dataset is a reasonably good output which has been gathered using some robust and reliable data capture method. You will be comparing your `PoseGTSAM, PoseiSAM2, VelGTSAM, VeliSAM2` output with the provided ground truth of the respective data. You are expected to have a plot and compute the error between the respective output and the ground truth in your report.

The optical flow-based velocity estimation will be in the frame of the camera while the `VelGTSAM` and `VeliSAM2` are in the world frame (Tag 10). You are required to transform the coordinate frames between the camera, IMU and the world to correctly compare your velocity estimates with the ground truth.

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 detailed and comprehensive `report.pdf` detailing your process and approach, as well as plots and diagrams for the same.
- A README.txt detailing anything we should be aware of. Since we are not providing strict guidelines on input/output specifications, please mention what we need to do to get your code running.
- 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.

6.1 Project Report

Primarily, your assignment will be graded on the quality and content of your submitted `report.pdf`. We expect a PDF of at most 6 pages in double-column format, complete with diagrams, plots and all other necessary explanations and details regarding your process.

Your output must include, at the very least, the following points of information:

- Describe the algorithm used along with all the equations and any assumptions made.
- Plot of estimated velocities in x, y and z directions and error for all the sequences (`Mapping, StraightLine, SlowCircle, FastCircle, Mountain` and `Square`). Provide a comparison with our reference velocity estimates with detailed analysis. You can use the function `ComputeVelError.m` provided to compute velocity error.

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.

[6] Parrot. SLAMDunk. http://developer.parrot.com/docs/slamdunk/, 2016.