ENAE788M Project 4a
Team Bouncing Rainbow Zebras

Erik Holum
Graduate Student
University of Maryland
Email: eholum@gmail.com

Edward Carney
Graduate Student
University of Maryland
Email: carneyedwardj@gmail.com

Derek Thompson
Graduate Student
University of Maryland
Email: derekbt@yahoo.com

***Abstract*—This project present results from determining position and orientation of a platform with a mounted stereo camera. Results are presented for altitude using stereopsis, and for full position and orientation using optical flow. Data from two flight profiles were captured and processed using the techniques described below and compared to onboard odometry captured by the same system. Overall performance characteristics as well as significant lessons learned from implementation are discussed.**

I. Introduction

In this project, we are given the task of determining position and orientation of the Bebop drone using optical flow and stereopsis from the bottom-facing Duo camera. Data was captured via ROS bags for two unique flight profiles: (1) a straight-line point-to-point flight path, and (2) and helical flight path (similar to the flight pattern for Project 2. This data included the Bebop odometry and the Duo camera capture; this was used to develop and test against.

Several OpenCV libraries [1] are used for feature detection and tracking between subsequent frames to determine translational velocity and angular rates, which are then integrated with respect to time to obtain position and orientation. Differences in feature detection between the left and right images is used to determine the altitude of the Bebop directly at every timestep.

Unless otherwise mentioned all code is implemented using ROS [2] and Python. Testing is done with ample usage of Python Jupyter notebooks.

II. Altitude from Stereopsis

We use feature detection and matching to compute altitude from the left and right images of the camera. We experimented with different types of detectors offered by the OpenCV library, including $BRIEF$, $FAST$, ORB, and the Shi-Tomasi detector. In general what we found was that quality was far more important than quantity when it came to altitude detection - since we found matches to have a great deal of noise in them.

Our first thought, given that we had already implemented it, was to try to use RANSAC to find the true altitude. We recall that given base distance, B, focal length, f, pixel distance d, then the altitude Z is given by,

$$Z = \frac{fB}{d} \implies d = \frac{fB}{Z}.$$

With this simple model and method of error checking, it was trivial to test using our RANSAC implementation to remove outliers. However, this was wildly unsuccessful do to the large differences in distance between matches. basically every subset of points produced an extremely different value for Z. We abandoned trying to use RANSAC to filter outliers.

In the end, we used the ORB matcher with brute force detection, but only on a limited set of high quality features. We found ORB to be slightly slower, but producing far more reliable features than other detectors in CV2. The brute force matcher was handy, as it gave estimated quality distances for matched pixels. We applied two basic filters for these matches, $[x, y]$ and $[x', y']$:

1) y and y' must be approximately equal (they must lie on the save line).
2) We only take the 20 closest matches as determined by the brute force matching algorithm.

The results of feature detection and matching are presented in figures 1 and 2. Given our top 20 matches and distances d_i, we compute our altitude as

$$Z = \frac{1}{20} \sum_i \frac{Bf}{d_i}.$$

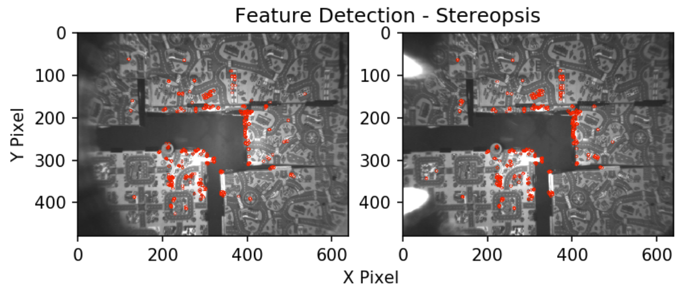

Fig. 1. Feature detection for left and right frames for stereopsis.

III. Position/Orientation from Optical Flow

The linear and angular velocities for the Bebop were obtained via optical flow by first extracting the unique identifying features in the first of two subsequent frames. In terms of

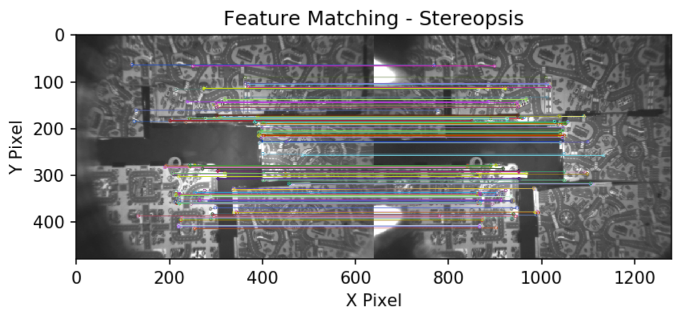

Fig. 2. Feature matching between left and right frames for stereopsis.

feature detection, we found that speed and quantity tended to be more important than quality. Given the testing done previously for Z, we opted to use the Shi-Tomasi detector implemented in OpenCV's $goodFeaturesToTrack$ function. A sample of the feature tracking done between subsequent frames during an altitude descent is shown in Figure 3; as expected for an altitude descent, the features here appear to radiate outward from the center of the image.

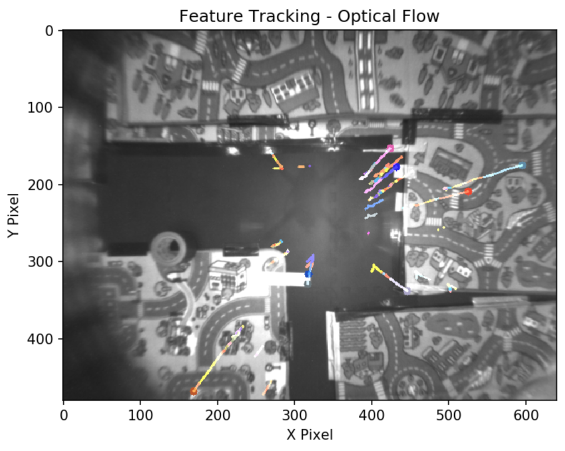

Fig. 3. Feature tracking from subsequent frames using optical flow during an altitude descent.

Thus, given two sequential images I_i and I_{i+1} and feature points $\bar{p}_i$, we use OpenCV's built in iterative Lucas-Kanade tracker $calcOpticalFlowPyrLK$ function to produce flow coordinates in I_{i+1}, $\bar{p}_{i+1}$. Given these features and flow points, together with a time stamp Δt, we compute an estimate for pixel velocity at each point as,

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix}_i = \dot{\bar{p}}_i = \bar{p}_{i+1} - \bar{p}_i.$$

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

Where $\mathbf{f_1}$ and $\mathbf{f_2}$ are given by the matrix:

$$\begin{bmatrix}-\frac{f}{Z} & 0 & \frac{x}{Z} & xy/f & -(1+x^2/f) & y\\ 0 & -\frac{f}{Z} & \frac{y}{Z} & (1+y^2/f) & -xy/f & -x\end{bmatrix}$$

Note that A will convert conventional units (e.g. m/s) to pixel velocities.

Then, given 3 different tracked pixel velocities from the set of $\dot{\bar{p}}_i$, we can solve for the 6 element camera velocity at time i using a system of 3 linear equations (of two variables each). Namely, let

$$A = \begin{bmatrix}\mathbf{f_1}(x_1,y_1,Z)\\ \mathbf{f_2}(x_1,y_1,Z)\\ \mathbf{f_1}(x_2,y_2,Z)\\ \mathbf{f_2}(x_2,y_2,Z)\\ \mathbf{f_1}(x_3,y_3,Z)\\ \mathbf{f_2}(x_3,y_3,Z)\end{bmatrix}$$

Be a matrix with three chosen pixel velocities computed from flow points. Then we have a simple system of equations

$$\begin{bmatrix}\dot{x}_1\\ \dot{y}_1\\ \dot{x}_2\\ \dot{y}_2\\ \dot{x}_3\\ \dot{y}_3\end{bmatrix} = AX$$

where X is the velocity state vector.

Since optical flow calculations are inherently noisy, we had to use some method to separate inliers from outliers in our flow coordinates in order to produce a reasonable estimate of the state, X. We used a 3-Point RANSAC to iteratively test randomly-selected models. The model was fit using linear least squares. We initially were using a linear equation solver, but found that least square fitting was more reliable (no singluarities!). Errors were computed using a simple vector 2-norm. E.g., given a point $[x, y]$, computed pixel velocity $[\dot{x}, \dot{y}]$, and solved state X, the error is given by,

$$e = \left\| \begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} - \begin{bmatrix}\mathbf{f_1}(x,y,Z)\\ \mathbf{f_2}(x,y,Z)\end{bmatrix} X \right\|.$$

An inlier for our RANSAC model is defined as any point with an error below a certain threshold, e_t. A sample frame with inliers and outliers is provided in figure 4.

The result of this process was a reliable velocity vector X. Position and orientation for the Bebop were then obtained by integrating the linear and angular velocities at each time step. The time step was taken as the difference between the second and first frame for each subsequent pair of frames. This process was repeated for every frame in the data set, with the result being a full flight profile for the position and orientation of the Bebop.

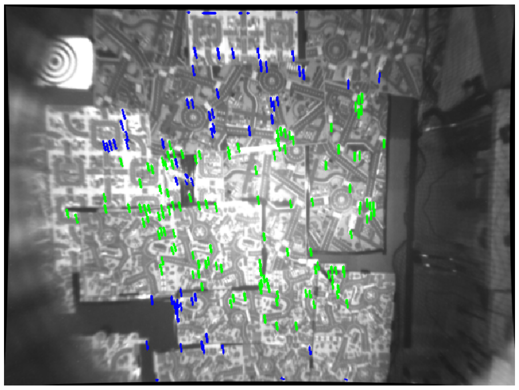

Fig. 4. Inliers and outliers of matched optical flow points computed using our RANSAC implementation. Inliers are green, outliers are blue.

IV. Results

Results for the altitude estimation from stereopsis for both the point-to-point (P2P) and helix trajectories can be seen in Figures 5 and 6.

Results for the position and orientation estimation from optical flow for both the point-to-point (P2P) and helix trajectories can be seen in Figures 7 - 10. Three-dimensional comparisons of the trajectory from the Bebop odometry and the optical flow are shown in Figures 11 and 12.

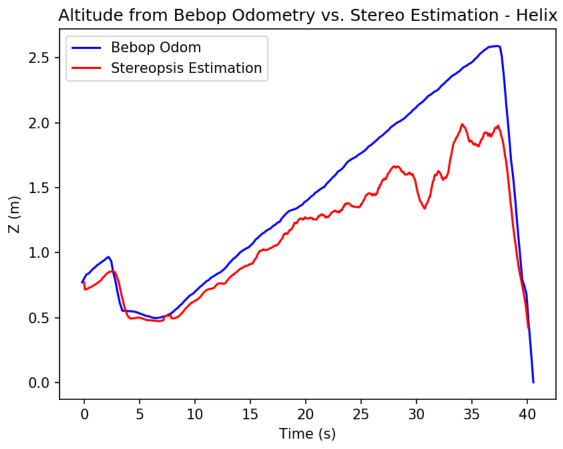

Fig. 5. Altitude estimation from stereopsis - Helix.

V. Videos

Point to Point Trajectory:

1) Image Processing: https://youtu.be/aCv6CwIvVEo
2) Rviz Positioning: https://youtu.be/qpy5V5gLPhI

Helix Trajectory:

1) Image Processing: https://youtu.be/xfIgtud$_UcM$
2) Rviz Positioning: https://youtu.be/WnD6Io0eyB8

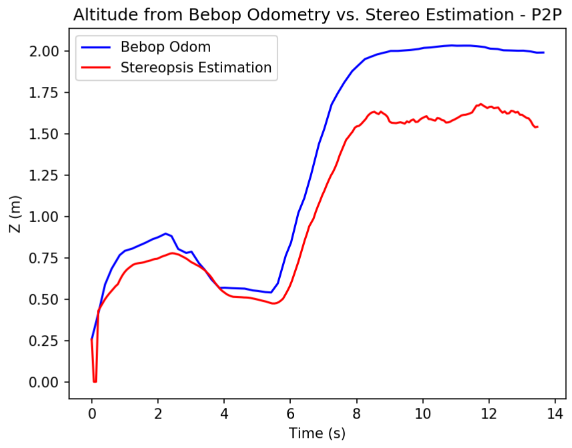

Fig. 6. Altitude estimation from stereopsis - P2P.

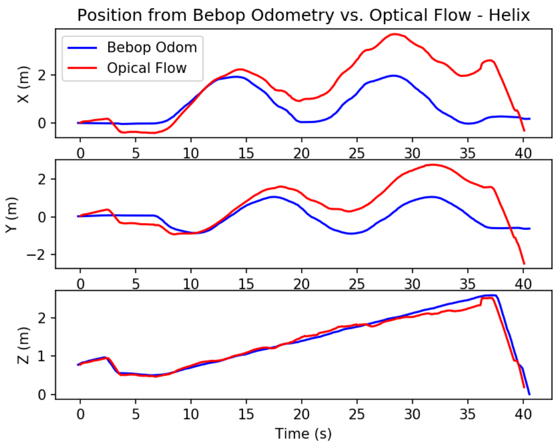

Fig. 7. Position estimation from optical flow - Helix.

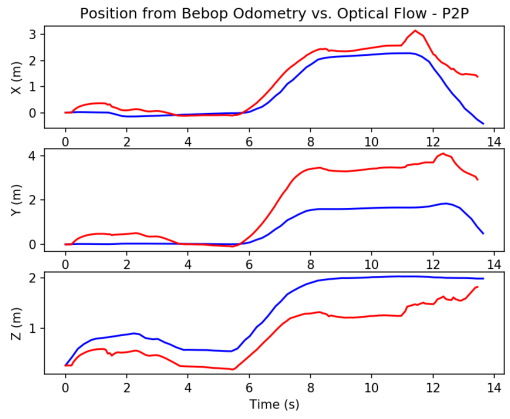

Fig. 8. Position estimation from optical flow - P2P.

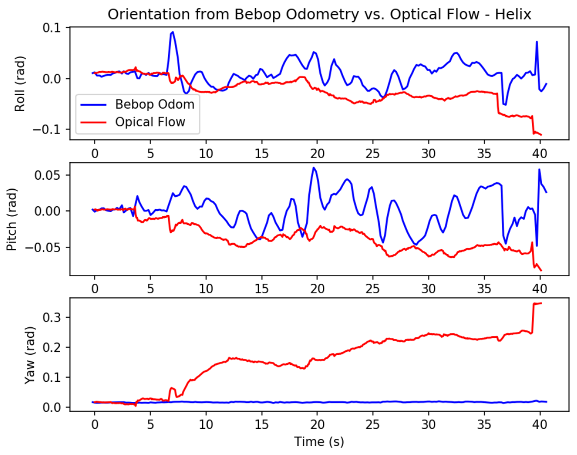

Fig. 9. Position estimation from optical flow - Helix.

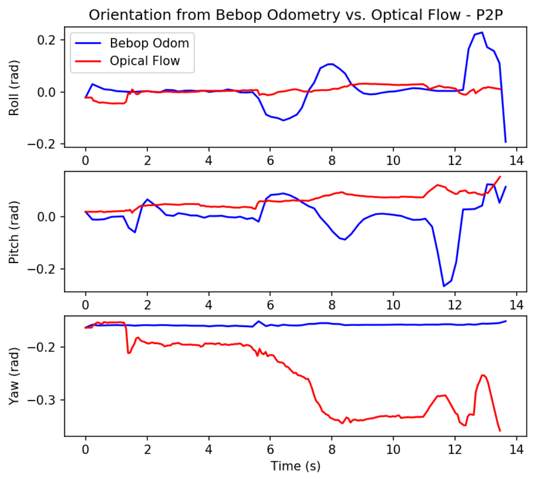

Fig. 10. Position estimation from optical flow - P2P.

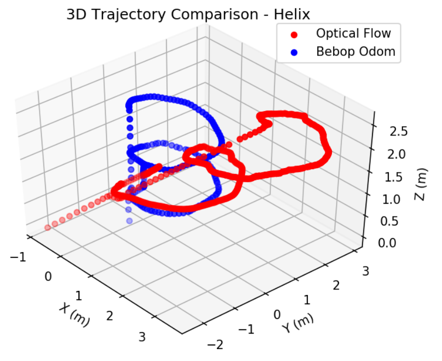

Fig. 11. 3D trajectory comparison for optical flow and Bebop odometry - Helix.

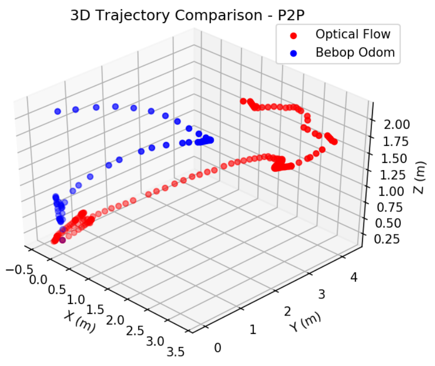

Fig. 12. 3D trajectory comparison for optical flow and Bebop odometry - P2P.

VI. Important Lessons Learned

A. *Processing Requirements*

It was clear (although somewhat unexpected) from processing and testing on our local workstations that without significant consideration for constructing fast, lean code that the processing time for optical flow can be substantial. Our initial implementation implementation ran at 1 Hz and worked remarkably well when post-processing the data. This was also on our workstations which included 32GB of RAM and Intel i7 Processors (8th gen). Once we had a functional algorithm, this project turned into turning down dials and breaking things until we could run at reasonable rates with ok data.

Given more time, there are many places to make improvements. At this point primary bottleneck is in our implementation of our 3-point RANSAC algorithm. This slowdown could be substantially reduced in two ways: (1) vectorizing various sections of the algorithm (especially in the error determination), and (2) relaxing the algorithm implementation. Due to time constraints, we focused primarily on the second method by reducing the number of iterations, increasing the error tolerance to call a point valid for a given model, and reducing the total number of valid points required to determine that a model fits a set of data. It works, but does not nearly have the performance and reliability as an implementation with no Hz constraints.

Even now, we are heavily constrained in our ability to get reasonable data when running on an UP board. We will have to make significant changes to get this to run reliably at higher data rates.

B. *Parameter Sensitivity*

We have a huge number of parameters available to us to tune, including for feature detection, matching, optical flow, RANSAC, etc. While some of these parameters are simple to understand or have minimal impact on our computation, others are tremendously sensitive and drastically affect performance. In trying to increase the frame rate processing time, as mention above, we found that increasing by even a single Hz required re-tuning many of the parameters we had.

C. Noise in Readings

The raw measurements obtained from both stereopsis and optical flow were seen to suffer from substantial noise. Even with the implementation of the moving average for stereopsis and RANSAC algorithm for optical flow as detailed above, the results were still relatively noisy and suffered from some drift and lag. As such, prior to actual implementation and use on the Bebop, we would need to improve the filtering done for both of these methods. It may be necessary to either increase the strictness of the RANSAC implementation (although this would require improvements in the RANSAC processing requirements as discussed below) or an additional filter to further smooth or average the readings obtained from stereopsis and optical flow. It should be noted that these measurements could likely be combined with other pose and orientation estimation measurements (e.g. from the IMU and/or accelerometer) via an Unscented Kalman Filter to improve the net state estimation.

D. Frame Differences

Although this is similar to a lesson learned in previous Projects, it is worth noting that any frame differences between the Bebop odometry definition and the convention used for an optical flow implementation can result in apparent data discrepancies, even when the source of the discrepancy is actually the result of axes swapping or values being negative in one definition as opposed to another. We encountered this in our initial implementation of optical flow and spent a non-trivial amount of time examining the code for fundamental logical flaws. After reviewing the optical flow diagrams, we came to the conclusion that the definitions of the x and y coordinates for the Bebop and our implementation of the optical flow equations was reversed; simply swapping the coordinates immediately yielded significantly better results.

Acknowledgment

The authors would like to thank the professors for this course, Nitin J. Sanket and Chahat Deep Singh, as well as Dr. Inderjit Chopra.

References

[1] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[2] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.