Team 5 - QDMC - Odometry from Optical Flow

Vishnu Sashank Dorbala
University of Maryland
vdorbala@umd.edu

Tim Kurtiak
University of Maryland
tkurtiak@umd.edu

Ilya Semenov
University of Maryland
isemenov@umd.edu

Surabhi Verma
University of Maryland
sverma96@umd.edu

Abstract—**This report presents the implementation and results of an optical flow odometry and pose estimate using a greyscale stereo down facing camera. The stereo camera is used to estimate depth, and optical flow between frames is used to estimate linear and angular velocity.**

I. Problem Statement

The aim of this project is to estimate vehicle position and orientation using only the images from a downward facing stereo camera. Optical flow algorithms must be used to estimate linear and angular velocities and point depth must be estimated using the stereo camera pair. The resulting velocities are integrated over the duration of the flight to generate position and orientation estimates which are compared to the Bebop's on board estimate.

II. Feature Detection and Tracking

Optical flow vectors are calculated using the iterative Lucas-Kanade method with pyramids. A set of points in generated in a grid and used to seed the vectors, this helps keep a wide distribution of valid optical flow vectors across the entire image. The more features are generated, the more points there are to calculate movement, however it also requires more processing power. By generating a grid away from points that don't contain useful information such as near the edges of the camera that only captures a part of the camera assembly itself we can increase the density of points without comprimising processing power.

Matches which have a movement distance of greater than 15 pixels are thrown out. This effectively removes erroneous matches that would otherwise ruin optical flow calculations.

Next, the optical flow vectors $\dot{x}$ and $\dot{y}$ are calculated as the difference in x and y coordinates between the last and current images divided by the time elapsed between frames. The frame time is calculated as a rolling average (low pass filter) of the previous five frame timestamps.

III. Stereo Depth Estimate

In order to calculate optical flow, some information about the world must be known to relate image plane projections to real world measurements. In this project, the distance from the camera to a feature point (Depth), is used as the bridge between the projected image and the world. To estimate depth of a feature point, a Stereo camera pair with a known base distance B and focal length f is used to triangulate a point in both images. Reference [**?**] contains a detailed explanation of the method used to estimate depth from a stereo camera pair. Specifically, a special case of stereo cameras where the cameras are pointing in a parallel direction to each other and their baseline and have the same focal length. This detail is used to greatly simplify the math and implementation, and results in the simple formulation:

$$Z = \frac{fB}{d} \tag{1}$$

Where Z is the depth of the feature point and d is the disparity, measured in pixels, between the left and right stereo images. Additionally, the simplifying assumption above guarantees that the epipolar line between the left and right camera is a horizontal line in the image. This simplifies implementation by constraining feature point matches to one row in the corresponding stereo image.

For a given feature point, the disparity d must be found by comparing the position of the point in the left and right stereo image. For a greyscale image, this is completed by comparing the intensity of the feature point and surrounding pixels in the left image to potential matches in the right image. Potential matches are evaluated using a sumsquared error of differences in pixel intensity for a defined window of pixels around the feature point. The size of the window can vary based on images, but generally smaller windows result in a faster algorithm but are prone to noise.

Our implementation constrained the right window search area to only allow disparity results which make sense for the application of odometry on a quadrotor. For example, the disparity between left and right images must always result in a positive disparity, ie the right image shows the same feature point to the left of the projection in left image. As such, it is a waste of computation time, and allows erroneous data to search for feature point matches in that part of the image. Additionally, the quadrotor in flight will always be at least 0.5 meters above the ground when navigating. As such, the maximum disparity can be limited to about 20 pixels. These two facts reduce the computation time of searching for a match drastically. Note that changing the base distance or focal length would change the maximum disparity for a particular setup.

A set of test images were run to prove out the functionality of our stereo image depth function *stereoDepth.py* . A common example of stereo image processing is presented in figure1 below, showing left and right original stereo images of Aloe.

These images were run through the stereo depth algorithm at three different window sizes $W = 1$, $W = 15$, and $W = 25$. The first two results are shown in figure **??** below. The images show that depth is not well estimated via an exceptionally small window size of one. Instead, a window

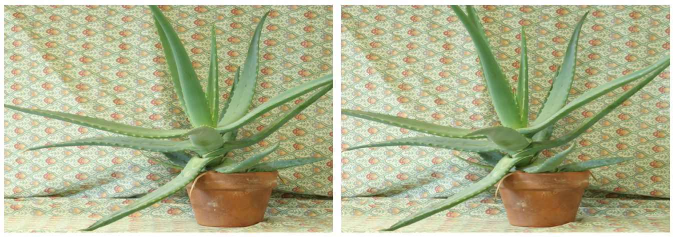
Figure 1: Left and Right Aloe Images

size of 5 generated a good depth estimate, but with some noise in the results.

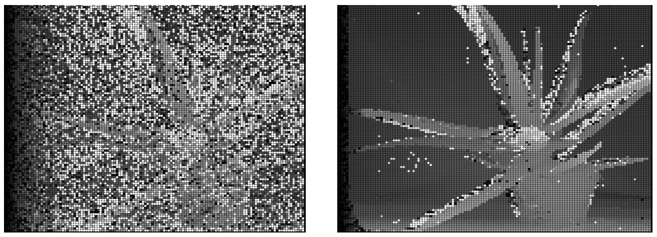
Figure 2: Left: Diagram of angular contact bearing [?]. Right: Disassembled bearing

Comparatively, a very large window size of 25 yields a nearly perfect depth image, but takes an order of magnitude longer to compute. Figure 3 shows the excellent results using a large window size.

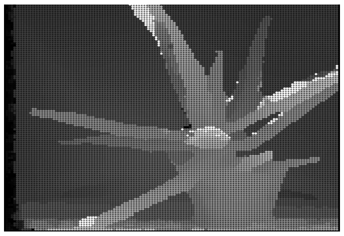
Figure 3: Aloe Depth with Window = 25

For the purposes of optical flow on a quadcopter, processing time is of the utmost importance, which prompted a secondary approach. Because it is useful to calculate disparities at points which have optical flow information, we simply use the moved positions of the grid points in the current frame as the seed for another instance of the iterative Lucas-Kanade method. These points on the current image are compared with the right camera image, returning optic flow vectors between the two cameras, which can be easily processed for horizontal discrepencies.

IV. RANSAC

Real world stereo images in flight are prone to noise in the stereo depth calculation. In order to remove erroneous depth values, a RANSAC algorithm was written to filter out bad depth points. The filtering works on the assumption that all depth estimates should lie on a plane, ie the ground.

This RANSAC implementation requires no tuning and is very simple. Given a set of x,y,z points, 3 are chosen at random N times. These three points describe a plane, and the distance from every point in the set to the plane is found. The median of these distances is saved as the score of the plane. The median is used instead of the average to avoid skewing towards outlier points. The plane with the lowest score is used to then return some number of the closest points. This helps exclude erroneous points.

V. Finding Position and Pose

The feature points and depth estimates are processed through the optical flow equation shown below. All of the feature points are combined in this formulation and the optimal solution for all optical flow points is found.

$$\begin{bmatrix} \dot{x_1} \\ \dot{y_1} \\ \vdots \\ \dot{x_n} \\ \dot{y_n} \end{bmatrix} = \begin{pmatrix} -Z_1^{-1} & 0 & x_1 Z_1^{-1} & x_1 y_1 & -(1+x_1^2) & y_1 \\ 0 & -Z_1^{-1} & y_1 Z_1^{-1} & (1+y_1^2) & -x_1 y_1 & -x_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -Z_n^{-1} & 0 & x_n Z_n^{-1} & x_n y_n & -(1+x_n^2) & y_n \\ 0 & -Z_n^{-1} & y_n Z_n^{-1} & (1+y_n^2) & -x_n y_n & -x_n \end{pmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \\ \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix}$$

A linear least squares solver is applied to the equation above to solve for the best solution of linear and angular velocities to satisfy the above equation.

VI. Estimation of Odometry

Once we obtain linear and angular velocities in camera frame, we compute the pose of the camera. This is done by integrating the velocities wrt δt i.e the time between two consecutive image timestamps. Since the angular velocities of the drone is zero, the orientation of the drone is also zero. We hence integrate only the linear velocities.

VII. Results

We visualize the odometry that we get from optical flow (in red) and the bebop odometry (in blue), in Rviz. We observe that the odometry trajectory from optical flow closely follows the ground truth trajectory obtained from bebop odometry. Fig 5. and Fig 4. show the side and top views of the helical trajectory. Fig 6. , Fig 7. are different views of the straight line trajectory.

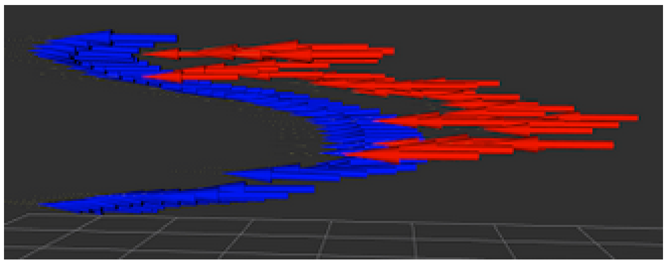
Figure 4: Helical Trajectory - 1

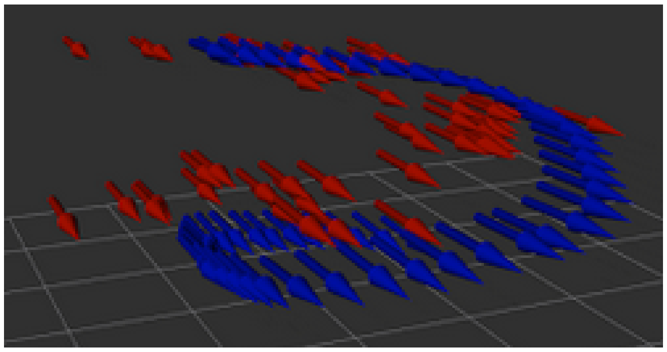
Figure 5: Helical Trajectory - 2

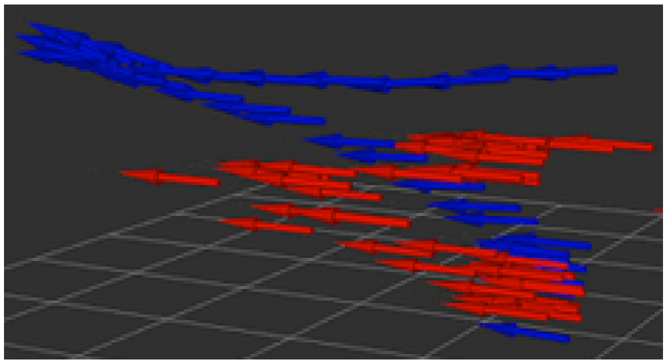
Figure 6: P2P Trajectory - 1

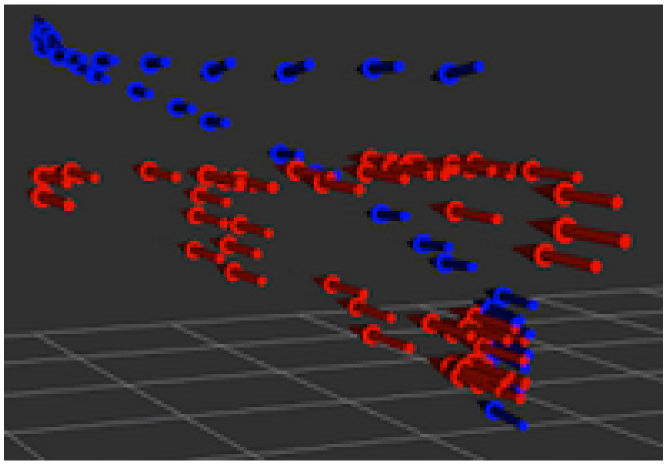
Figure 7: P2P Trajectory - 2

VIII. Lessons Learned

- Optical flow is good at linear estimation, but bad with angular estimation.
- Processing time/compute power is limited and requires optimized code.
- Additional optimization is required to get real time processing for use in flight.
- Stereo estimation using the window method is best for large resolution large baseline stereo images but is not fast enough or accurate enough for our application.

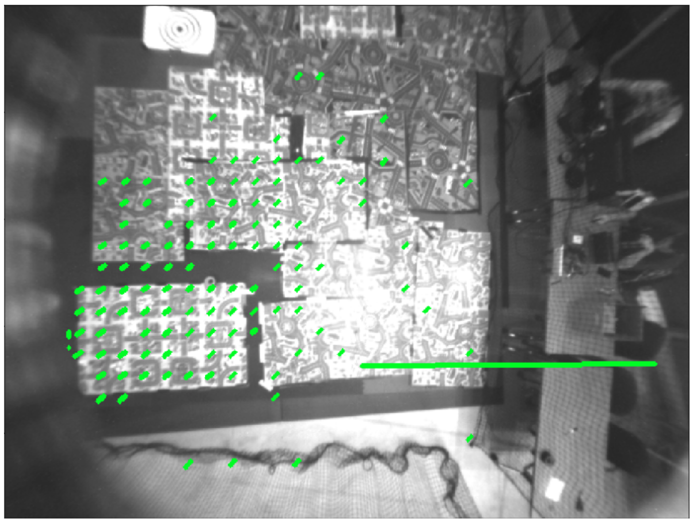
Figure 8: Helix Optical Flow

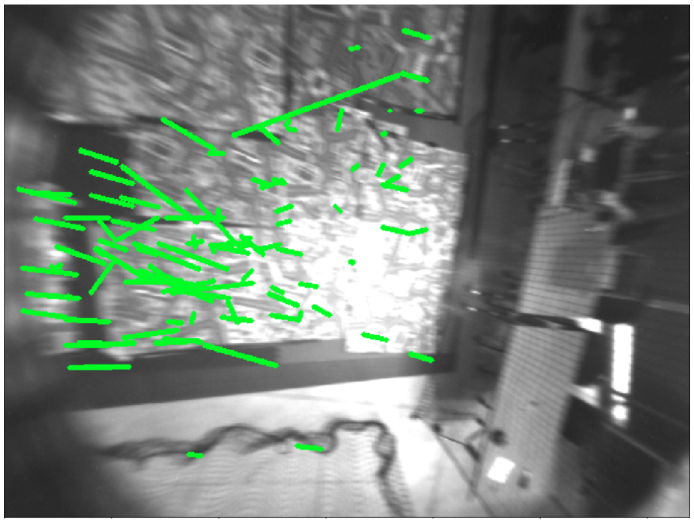
Figure 9: P2P-Optical Flow

- Due to missing frames as a result of synchronization issues, we observe our output velocity to be noisy at times. This is primarily due to a large optical flow vector being computed when frames are skipped.

IX. Conclusion

Stereo visual odometry is less accurate when compared to the odometry computed from the sonar data in the bebop. This is evident, both visually from the odometry plots that we obtain as well as from the optical flow ouputs.

X. References

- Lecture 16 Sanket, Nitin; ENAE 788M Lecture 16 https://drive.google.com/ file/d/1jcbLtuw7ptxDDn0Djl9SUXXX8ENVdTu2/view
- FlannMatcher https://opencv-python-tutroals.readthedocs.io/en/latest/pytutorials/pyfeature2d/pymatcher/pymatcher.html.