Stereo Visual Odometry - USING 2 LATE DAYS

Mrinalgouda Patil
Alfred Gessow Center of Excellence
University of Maryland
College Park, Maryland 20742
Email: mpcsdspa@gmail.com

Curtis Merrill
Alfred Gessow Center of Excellence
University of Maryland
College Park, Maryland 20742
Email: curtism@umd.edu

Ravi Lumba
Alfred Gessow Center of Excellence
University of Maryland
College Park, Maryland 20742
Email: rlumba@umd.edu

***Abstract*—This paper examines the problem of visual odometry - estimating the pose of a moving camera relative to a static ground. First, two different methods for estimating depth are described. Next, a method for estimating optical flow using feature matching and RANSAC is described. Finally, these two methods are integrated and applied on two real trajectories flown by a bebop drone.**

I. Introduction/Problem Statement

The goal of this project is to compute the 3D camera trajectory of a stereo sensor. A downward facing DUO camera is attached to our quadcoptor and the images from each camera (right and left) can be used to calculate instantaneous height at one time. By comparing the image from one camera at time t to an image from the same camera at time t-dt, the optical flow can be computed which can be used to solve for the 3D world velocities using the following equation.

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}\frac{-1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y\\ 0 & \frac{-1}{Z} & \frac{y}{Z} & (1+y^2) & -xy & -x\end{bmatrix} \begin{bmatrix}V_X\\ V_Y\\ V_Z\\ \Omega_X\\ \Omega_Y\\ \Omega_Z\end{bmatrix} \quad (1)$$

In this project, two trajectories were flown with our quad: a straight line and a helix. The method used above was used to compute the estimated trajectory for each run, and was compared to the bebop odometry. Two videos in Rviz were created for each trajectory - the first video contains the estimated trajectory vs. the bebop odometry and the second contains detected features at each frame.

II. Height Calculation

The first major step in calculating the 3D camera trajectory is calculating the depth at each time step using the stereo camera. Because we have two images with very similar features of view, we can find the same feature in both images. Based on the pixel difference between the feature location in each image, and since the camera planes are parallel with a known offset, the depth can be calculated using the following equation.

$$Z = \frac{fB}{disparity} \quad (2)$$

,where f is the focal length of the camera (200 pixels), B is the offset between the camera (.03m for use), and the disparity is the pixel difference between the same feature in the difference images.

To summarize, the disparity, or pixel difference between the same image in the two difference frames, is the only variable needed to calculate the depth. The next two subsections will go over two different ways attempted to find the disparity.

A. Sliding Window Method

The first attempt at calculating the disparity was using the sliding window method. First, features are identified in one of the camera images (we choose the left image), as seen in Figure 1. This feature identification was done using SIFT.

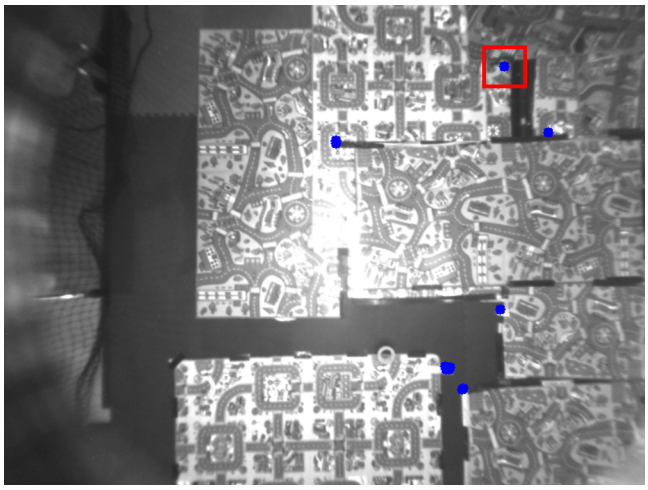

Fig. 1. After the intial features are found (blue), each feature has a box fit around it.

Next, we place a window around this feature (Figure 1), and store the pixel values at these points. Then, we recreate the same window in the right image, and try to slide it along the epipolar line to find the feature. Since the cameras are mounted parallel, this should just involve sliding the window along the x axis of the image frame, as seen in Figure 2.

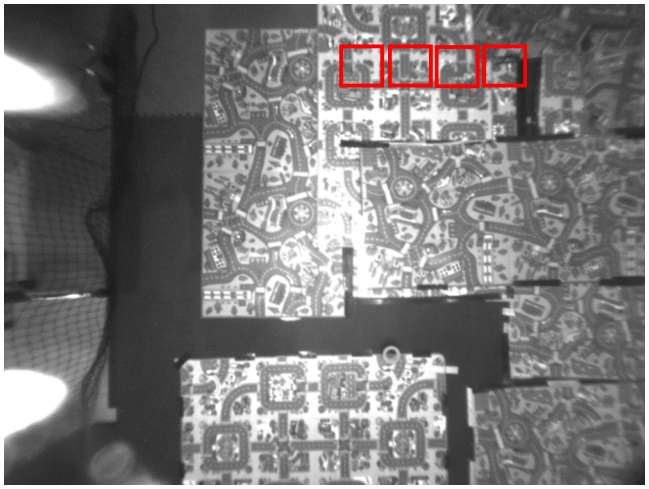

Fig. 2. A window is moved across at a constant y to attempt to find the feature.

Each time the window is moved, the error for each pixel is calculated by squaring the difference between the left and right window. This error is then summed up and stored along with the disparity (step size * step number), and the process is computed a certain number of steps. After all of the steps, disparity with the lowest error is assumed to correspond to the actual feature, and is used to determine the height at that feature. After all feature matches are found, only a subset of those are used to calculate the depth. The user can specify how many matches to be used, and only the top matches (based on error) will be used. This method works reasonably well for clean images. The depth was calculated for each feature and then averaged to calculate the overall depth of the quad. This was done under the assumption that the pitch and roll angles experienced by the quad are small, and therefore the camera depth is roughly equal to the quadcopter depth.

1) Refinements: Although the original sliding window method worked well, the height measurements were still inconsistent frame-to-frame. Some of this was due to factors outside of our control, such as the fact that distance between the cameras was small, resulting in decreasing performance above 2m. However, one other problem we noticed was that the feature in the left camera did not have the exact same y coordinate in the right image, so moving the window along the x axis only would never find the exact same feature. This would lead to lots of noise in the height measurement. To get around this we had the window move in both the x and y directions, which allowed us to increase our feature matches significantly.

Even with this change, there were still instances where we would not be able to detect the corresponding feature in the right camera frame. We believe that this is due to the fact that even at the same feature in both images, the pixel values are different. We checked this manually and found discrepancies in the pixel values of up to 15. Although it was thought that this error should be present in all windows and therefore not affect the feature matching, we observed otherwise. To get around this, we decided to change our error equation from the difference in pixels squared to the difference in pixels cubed (and then take the absolute value). By changing to the cube root, we are placing more importance on large pixel differences and reducing the effect of the offset previously noted. Using this method also helps us be more sensitive to gradients, such as edges and corners that will be present in the window.

After implementing both of these methods, our height measurements improved for most cases, as seen in Figures 3 and 4. For this example, a small number of intial features and final matches was specified to make it easier for the reader. In practice, many more features/matches were used.

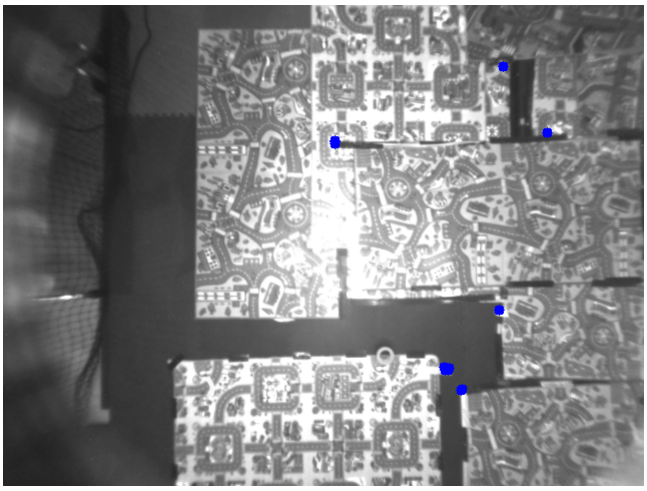

Fig. 3. Initially, a certain number of features was found by SIFT.

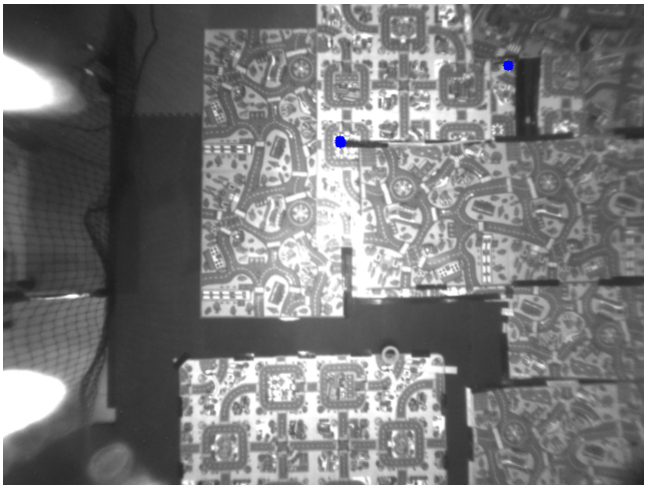

Fig. 4. After running the full sliding window, these are the best matches. We see that they match very well with features from 3.

However, certain image frames with very poor lightening conditions, such as the one shown in Figure 5, still gave very

poor and inconsistent results.

Fig. 5. Some of the frames in the straight line trajectory had very poor lighting and the sliding window method would not work well.

For increased robustness, we decided to try to utilize feature matching algorithms to calculate our disparity instead.

B. Feature Matching for Disparity

We used feature matching to replace the window method for robustness. Like with the window method, SIFT was used to detect the features in the left and right window. Then a flann based feature matcher was used to find the feature matches between the two windows. Next, RANSAC was used to refine the matches and remove the outliers. Finally, Eq. 2 was used to calculate the depth based on the refined matches from RANSAC.

1) Added Filters: Even after we use RANSAC, there was still some noise in our depth calculation. Two different methods were implemented to reject these "bad" measurements.

First, a filter was implemented on the disparity for each feature match. Because we know the points for the right and left cameras, we know what sign our disparity should be when comparing features between the two images. This allows us to filter out disparities that would result in negative depths. Also, based on Eq. 2 we know what values of disparity to expect to give us reasonable heights, so we can filter out very high disparities (greater than 40) that would lead depths of below .15m. Implementing both of these helped reduce the noise for each depth estimate.

To further filter the data, a second filter was implemented on the calculated depths for each feature. For one time instance, we had between 10-15 matched features that passed the previous filters (RANSAC, disparity magntitude/sign), which meant we had 15 depth measurements. Sometimes, there could still be one or two outliers that would cause problems when averaging the depths. The median and standard deviation were found for the depth measurements, and only the depths within 2 standard deviations of the median were kept. For this, the median was chosen instead of the mean to help eliminate the effect of outliers during the filtering process. The depth measurements that fell within 2 standard deviations of the median were averaged and that gave the depth estimate for our quad.

C. Optical Flow

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera. The pixel velocities were calculated using the equations given in the class notes. The pixel coordinates were normalized using the intrinsic parameters. The obtained pixel velocities were used to compute the angular and translational velocities using linear least square solver, which was used to update the position of the quadrotor using integration.

D. RANSAC

RANSAC or random sampling and consensus, is an algorithm which is used to fit data and remove outliers. It works by taking some random set of points, fitting a model to those points, and then checking how many of the total points fit that model within some specified error tolerance (these points are called inliers). Many sets of random points are taken and the set of points that contains the most inliers is kept, and a new model is made by taking a least squares fit with all of the inliers of the best fit model, and the outliers are removed. In this project, RANSAC was used to remove bad feature matches from Open CV feature matching algorithm. The sets of points considered feature matches were input as data, and a homography transformation was used as the model to evaluate inliers and outliers. RANSAC provided less noisy data by removing some of the false feature match detections.

III. Integration

The components mentioned above were integrated together as follows.

For each sample, the right and left camera frames were compared to get the depth. Next, the current left camera image and the previous left camera image are used to calculated the optical flow ($\dot{x}$ and $\dot{y}$) for each feature. These two values, plus the coordinates of that point and the depth, are used to calculate the 6 velocities (linear and angular at that feature).

After the velocity is calculated for each feature, it is filtered similarly to how the depth as filtered - only measurements that fall within 2 standard deviations are used. The velocities for that instance are taken to be the average of the filtered velocities. Finally, these velocities are integrated to find the accumulated position of the quad. The Δt used during the integration was found from the image messages.

IV. Results

A. Depth Measurements

Compare sliding window and feature matcher if have time.

B. Straight Line

The first trajectory that was used was a straight line that went from (0,0,0) to (2,2,2). The full 3D trajectory for our predicted position vs. the bebop autonomy are shown below.

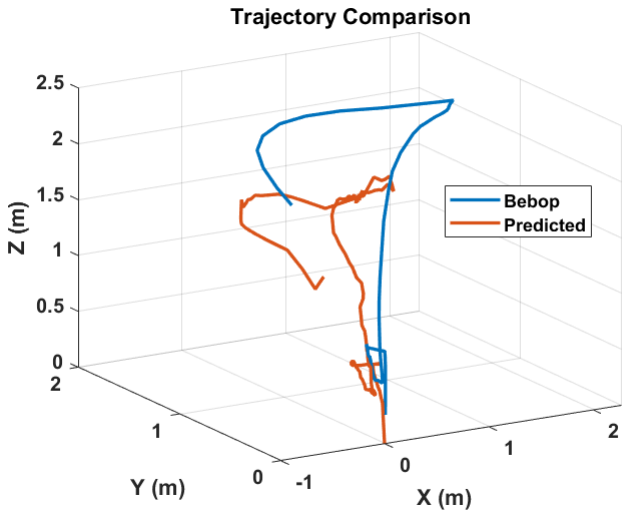

Fig. 6. Predicted Trajectory vs. Bebop Odometry for the Straight Line Trajectory.

It is difficult to get a good idea of accuracy from the 3D plot, as there is no time synchronization. Therefore, we present the time histories of the position below.

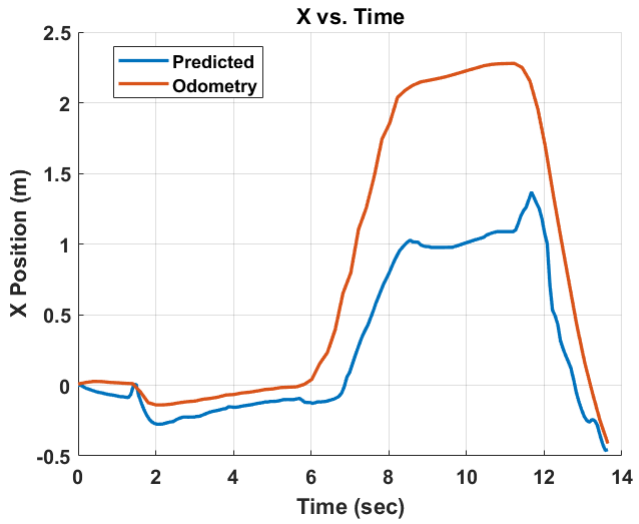

Fig. 7. Predicted X vs. Bebop Odometry for the Straight Line Trajectory.

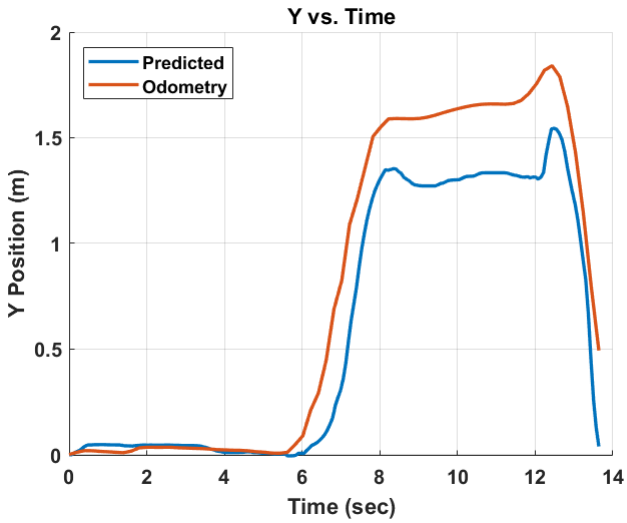

Fig. 8. Predicted Y vs. Bebop Odometry for the Straight Line Trajectory.

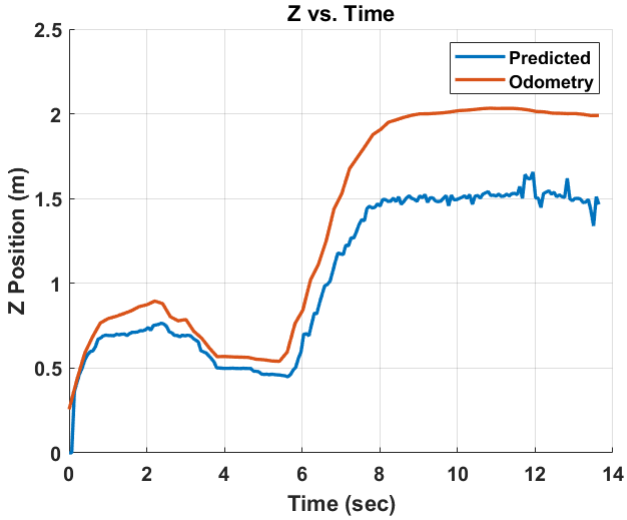

Fig. 9. Predicted Z vs. Bebop Odometry for the Straight Line Trajectory.

Looking at these results, we see that we are able to capture the same general trends as the odometry, however our prediction struggles when the magnitude of the displacement becomes very large. From Eq. 1, we can see that if Z is under predicted, then our linear velocities will be under predicted. From figure 9, we can see that our predicted Z matches the odometry very well until the depth gets above 1.5m, at which our prediction levels off.

We believe that some of this error is due to our camera specs making it hard to accurately obtain Z. From the depth equation, we see that a disparity of 4 pixels gives us 1.5. When obtaining many features, tracking, and trying to filter them, it is harder to obtain good estimates when the disparity value is so small (for example 4 vs. 8). Some of the error could also be due to camera blur. The straight line run was

done very quickly, and therefore there is significant camera blur at certain frames throughout the run. At the end of the run especially, when the quad stops, there is some blur and also some pitch to stop the quad. This might also induce some error as we assumed that there was minimal pitch and roll when writing our code.

The second trajectory that was used was a helix. Below are the time histories of our estimated trajectory vs. the bebop odometry, followed by the 3D trajectory comparison.

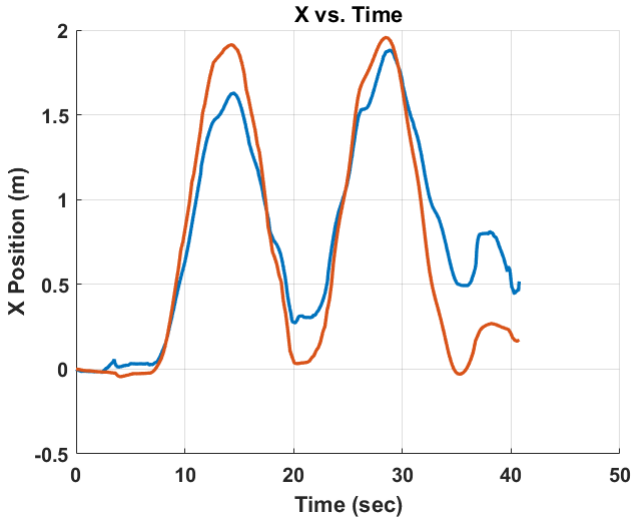

Fig. 10. Predicted X vs. Bebop Odometry for the Helix Trajectory.

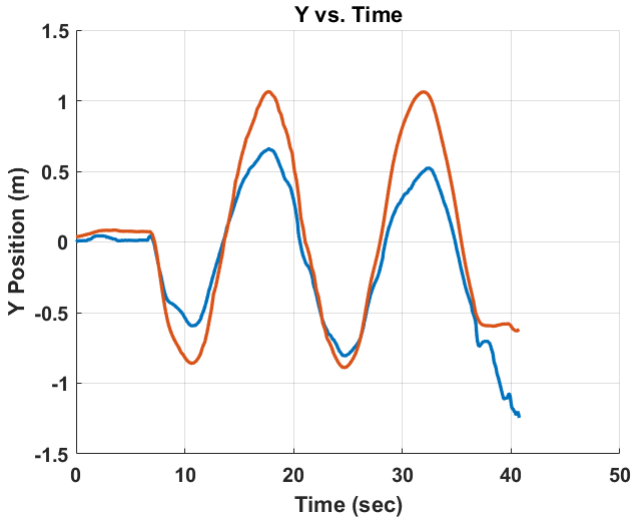

Fig. 11. Predicted Y vs. Bebop Odometry for the Helix Trajectory.

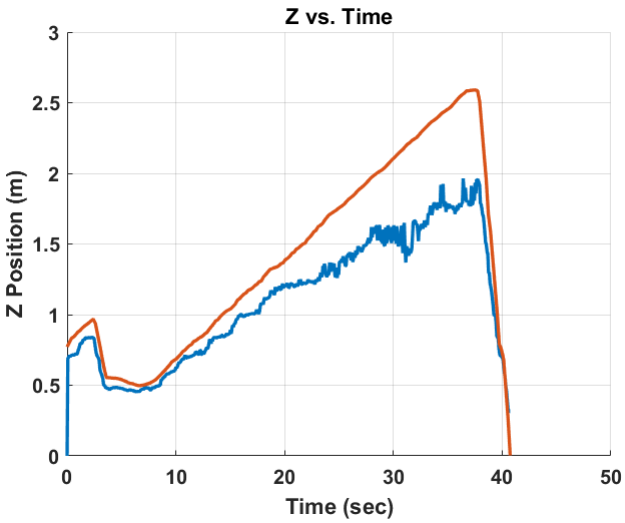

Fig. 12. Predicted Z vs. Bebop Odometry for the Helix Trajectory.

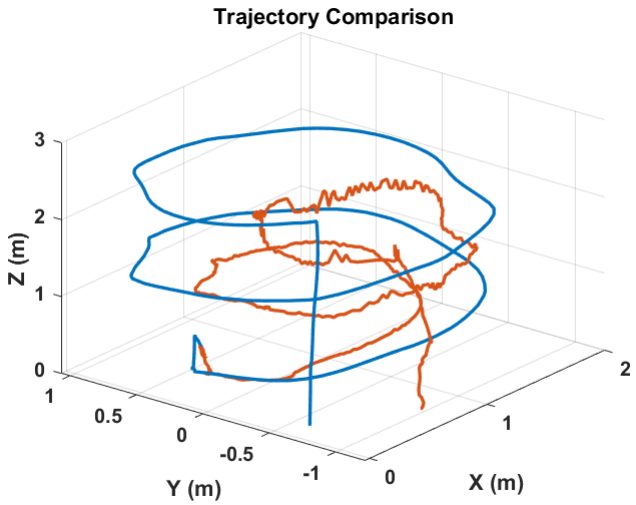

Fig. 13. Predicted Trajectory vs. Bebop Odometry for the Helix Trajectory.

From these plots, we see the same trends that we saw in the straight line run. Again, our Z position is under predicted as the magnitude increases. However, one interesting thing is that the X and Y are also under predicted, even when the Z is pretty accurate (basically the first turn of the helix. This indicates that the struggles in the previous section might not have been completely related to the Z, but instead that our code is not able to predict higher velocities.

V. Conclusion and Lessons Learned

From this project, we learned that depth measurement is very important when computing optical flow. Having a small baseline distance between the two stereo cameras (.03m) and a low resolution means that depth measurements deteriorate as the Z increases. If a bigger stereo camera can not be used, a different method, such as sonar, should be used for more

accurate Z measurements.

The second major lesson learned was about computational time.

REFERENCES

[1] ENAE788 Class 5 Slides
[2] Some Code taken from learnopencv.com/rotation-matrix-to-euler-angles/