Stereo Visual Odometry - USING 1 LATE DAY

Team `sudo rm -rf *`

Abhishek Shastry
Department of Aerospace Engineering
University of Maryland
College Park 20742
Email: shastry@umd.edu

Animesh Shastry
Department of Aerospace Engineering
University of Maryland
College Park 20742
Email: animeshs@umd.edu

Nicholas Rehm
Department of Aerospace Engineering
University of Maryland
College Park 20742
Email: nrehm@umd.edu

***Abstract*—In this report, odometry and pose estimation is achieved through the use of a stereo camera on the PRG Husky for ENAE788M: Hands on Autonomous Aerial Robotics. Features are detected and matched at time t and t-dt to calculate the relative change in x and y. The same features are then matched between left and right video frames at time t to calculate the depth of those features using the disparity, focal length, and baseline of the camera set. The full set of detected features and their corresponding coordinates are fit to an optical flow model using RANSAC. Results show that this method is good at estimating the vehicle pose at low altitude, but suffers at higher altitude due to the small baseline between the stereo cameras.**

I. Introduction

Downward-facing stereo video feed can be used to obtain an estimate of relative camera velocity which can be integrated with time to compute relative camera location in a fixed inertial frame. This method of odometry can be used to provide feedback for outer-loop control schemes on a quadrotor to carry out a desired flight path. Temporal feature matching allows for a displacement estimate while spatial feature matching of the stereo image pair allows for a distance estimate which is why a stereo image pair is required to obtain a full solution. This method is prone to outliers which must be rejected in order to create an accurate model of the depth and pose estimation. Link to the result videos: Click Here

II. Basic Implementation Method

A. Feature Detection and Matching

Temporal and spatial feature matching is achieved by generation of ORB features in two images and their brute-force matching. This implemented in OpenCV by creating the objects `orb = cv2.ORB_create()` and `bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck = True)` along with using the functions `orb.detectAndCompute` and `bf.match`. The features detected in the left stereo camera's image at time t are matched with the features from the previous image obtained from the left stereo camera at time $t-dt$ to get the temporally matched features. A sample temporal feature matching is shown in Figure 1. To get the spatially matched features, a similar feature matching is performed between the left and right images of the stereo camera at time t. A sample spatial feature matching is shown in Figure 2.

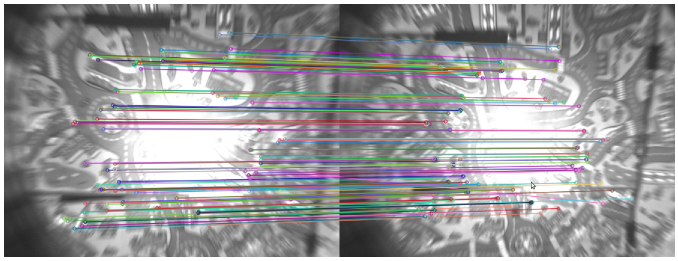

Fig. 1. Temporal ORB feature matching between current left frame at time t (plotted on left) and previous left frame obtained at $t-dt$ (plotted on right)

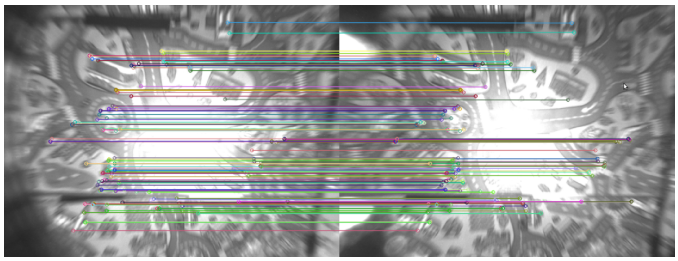

Fig. 2. Spatial ORB feature matching between left and right stereo image frames

B. Feature Sorting and Trimming

Both the spatially and temporally matched feature set are sorted according to their matched features' distances and the bottom 50% of the matches are trimmed. This results in the matched feature set being more accurate as we are removing the "bad" matches.

C. Removing "lonely" matches

The intersection of the Temporal and Spatial matched feature set gives us the common features that are present in the 3-image set $\{\text{Right Image}(t), \text{Left Image}(t), \text{Left Image}(t-1)\}$. The Left Image$(t)$ is set as the Query Image for both Temporal and Spatial match set as it is the common image. Thus, the Right Image(t) becomes the Train Image for the Spatial match set and Left Image$(t-1)$ becomes the Train Image for the

Temporal match set. Intersection of the two sets is generated, by first sorting both the matched sets according to their "Query Index" and then looking at the Query Indexes of both the sets and extracting out the matches that have the same Query Index. Both the depth value and the temporal displacement value of these common matches/features can now be computed as described in the following subsections.

D. Depth Estimation

The depth of every i^{th} spatially detected feature relative to the camera can be calculated as:

$$Z_i = \frac{f * B}{d_i} = \frac{f * B}{(x_i)_R - (x_i)_L} \tag{1}$$

where f is the known normalized focal length (in pixel units) of the camera, B is the baseline of the stereo cameras, and d_i is the disparity of the i^{th} feature in pixel coordinates between the left and right camera frames. The disparity doesn't has the y pixel coordinates as all the epipolar lines are parallel in both the images, which results from the fact that there is no relative orientation between the Left and Right camera of the Duo3D Stereo rig. The individual feature depths must be calculated separately as we are assuming non-zero pitch and roll of the quadrotor.

E. Optical Flow

A sparse optical flow is computed for every feature from the temporal feature matching. The difference in the i^{th} features' pixel coordinates at times t and $t-dt$ multiplied by the inverse of dt gives the optical flow in pixel coordinates $[\dot{u}_i\ \dot{v}_i]^T$. Some outliers in the feature matching lead to incorrect optical flow estimates, but this is taken care of with RANSAC. The features in pixel coordinates can be transformed in the normalized image coordinates by using the following transformation.

$$x_i = \frac{u_i - c_x}{f_x} \tag{2}$$

$$y_i = \frac{v_i - c_y}{f_y} \tag{3}$$

Differentiating the above equation gives the optical flow in the normalized image coordinates.

$$\dot{x}_i = \frac{\dot{u}_i}{f_x} \tag{4}$$

$$\dot{y}_i = \frac{\dot{v}_i}{f_y} \tag{5}$$

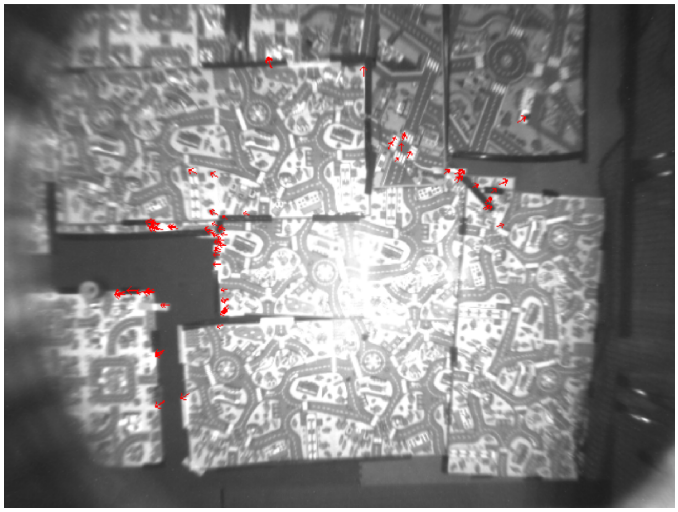

Fig. 3. Computed optical flow between current and previous left camera frames. Typically, there are no mismatched features left after sorting the matches by distance followed by trimming process.

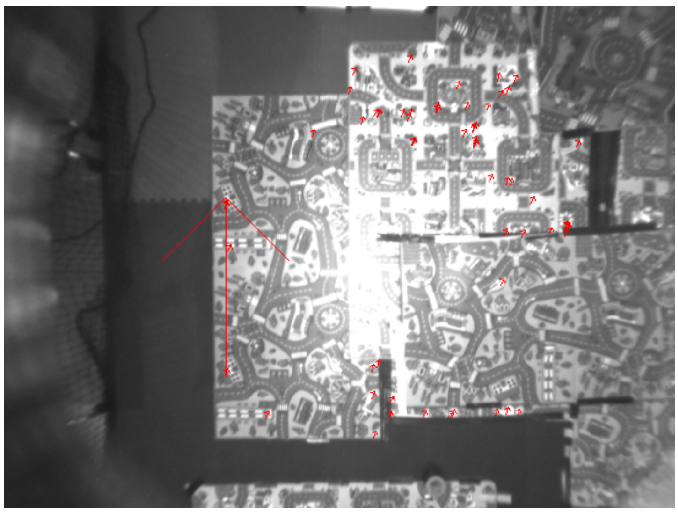

Fig. 4. A rare case when an outlier due to wrong feature matching is present in the computed optical flow.

F. Velocity and Pose Estimation with Linear Least-Squares

For every i^{th} feature the equation relating the optical flow and the camera's linear and angular velocity is written as follows.

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix}_i = \begin{bmatrix} -\frac{1}{Z_i} & 0 & \frac{x_i}{Z_i} & x_i y_i & -1-x_i^2 & y_i \\ 0 & -\frac{1}{Z_i} & \frac{y_i}{Z_i} & 1+y_i^2 & -x_i y_i & -x_i \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \\ \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \tag{6}$$

The equation is linear in nature and can be molded into the form $y = Ax$ when all the data for the i^{th} pixel are stacked on top of each other. A standard linear least-squares algorithm will then solve for x which is the camera's linear and angular velocity. This velocity is in the camera frame and has to be converted into the body frame. The body frame velocity is

then numerically integrated to get the pose estimate of the quadrotor.

G. Velocity and Pose Estimation with RANSAC

Feature matchings are never always perfect and therefore the optical flow dataset will have some outliers as shown in Figure 4. Hence, to robustly estimate the quadrotor's velocity from the optical flow equation (6), a RANSAC algorithm has been implemented. A general view of the algorithm is described below.
A set of three matched points are randomly selected and a model is generated using equation (6). Every other set of matched points are then compared to this model to compute the total number of inliers based on a tuned error tolerance. This is done k times to find the set that includes the most number of inliers, which are then used to compute the "best" model. The number of iterations k is selected using equation (7) which describes the ratio of log-likelihood of never selecting a full inlier set in k iterations to selecting a set that has at least one outlier. The parameter n(=3 for model (6)) represents the minimum number of points (randomly sampled from y) required to define the model (x). Parameter w is an estimate of the number of inliers to total number of points and it represents the probability of randomly selecting an inlier in y. Parameter p represents the desired probability of selecting at least one inlier set in k iterations. For example, assuming that there is a 75% chance that a randomly sampled point will belong to an inlier, 5 iterations will guarantee that the probability of selecting at least one full inlier set and correspondingly the "best model" is around 93.54%

$$k = \frac{log(1-p)}{log(1-w^n)} \tag{7}$$

III. Additional Implementation Methods

A. Point Cloud Generation and plane fitting

A sparse point cloud is generated by triangulating every feature from the spatially matched feature set. Assuming that the quadrotor flies over a level surface, a plane is fitted on the point cloud using RANSAC and its orientation and distance from origin gives us the roll, pitch and height information of the left stereo-camera with respect to the fitted plane, which is an estimate of the ground plane (can also be considered as the inertial frame). The roll and pitch angles are then transformed into the body frame to get the quadrotor's roll and pitch angles. Since, the baseline of the stereo-camera setup is low (30mm) the height estimates of the individual points degrade with increase in height of the quadrotor. This can be observed in the Figures 5, 6, 7.

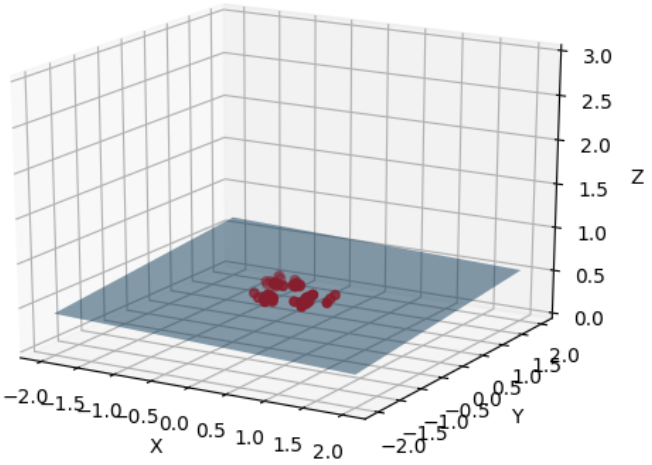

Fig. 5. Point Cloud from triangulation of ORB Stereo features at low height. There are no outliers and the height estimates are fairly accurate.

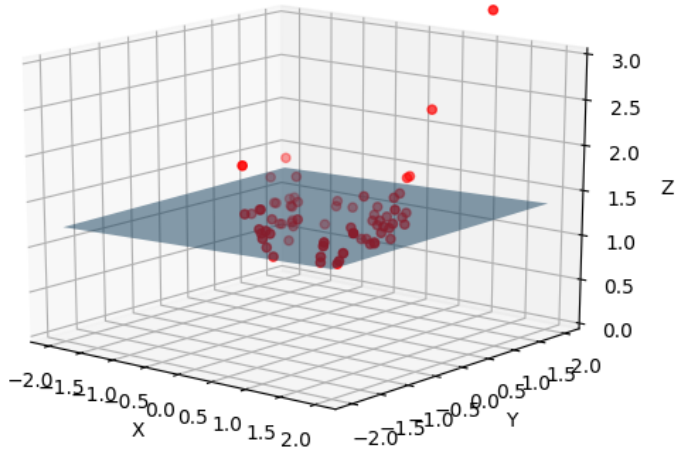

Fig. 6. Point Cloud from triangulation of ORB Stereo features at medium height. There are a few outliers and the height estimates are doable.

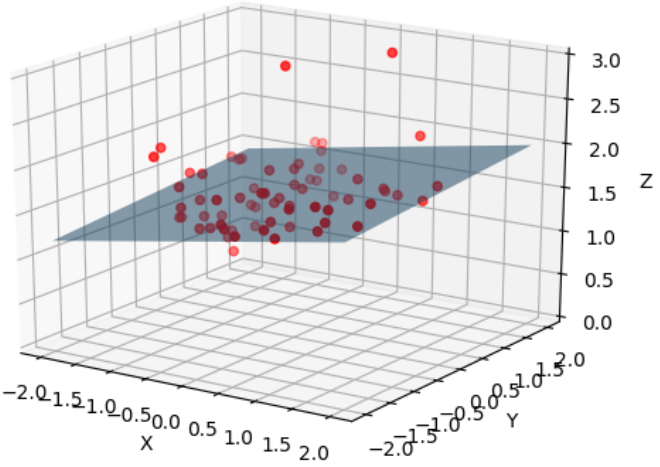

Fig. 7. Point Cloud from triangulation of ORB Stereo features at high height. There are many outliers and the height estimates are not good enough.

B. Orientation from IMU

The orientation obtained by integrating the angular velocity estimates from Optical Flow will diverge from the ground truth with time as there is no direct measurement. Hence, in practice an IMU is used along with the Stereo-camera pair in a tightly/loosely integrated method to generate better estimates of pose. Since, the dataset didn't have IMU measurements from the Duo3D, we used the Bebop's Odometry coming at 5Hz to get the orientation. As the frequency was low, an increasing integration error was still observed in the pose estimates. Hence, we used the angular velocity estimates from the optical flow, integrated over time, to "fill" in the gaps between two measurements.

IV. Results for Point to Point Dataset

Point to point dataset had a large jerks in the orientation and hence a lot of the images were blurry which resulted in bad feature detection and matching. Hence, in this case RANSAC method slightly performs better than Linear least-squares. The pose estimates coming from Bebop is plotted in blue, from Linear least-squares method given in Section II-F is plotted in red, from RANSAC method in Section II-G is plotted in yellow, from plane fitting method in Section III-A is plotted in green and from IMU method in Section III-B is plotted in purple.

A. Position Estimates

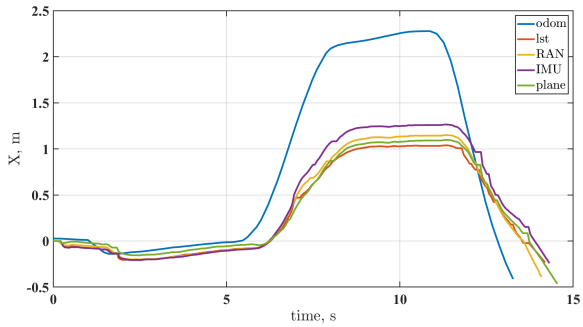

Fig. 8. Quadrotor's X-position computed from various methods.

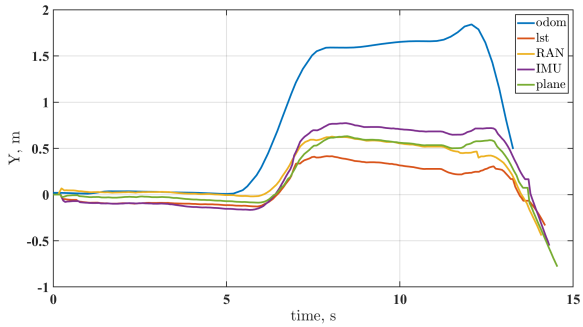

Fig. 9. Quadrotor's Y-position computed from various methods

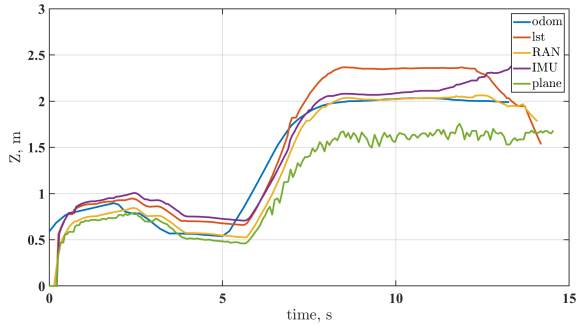

Fig. 10. Quadrotor's Z-position computed from various methods

B. Velocity Estimates

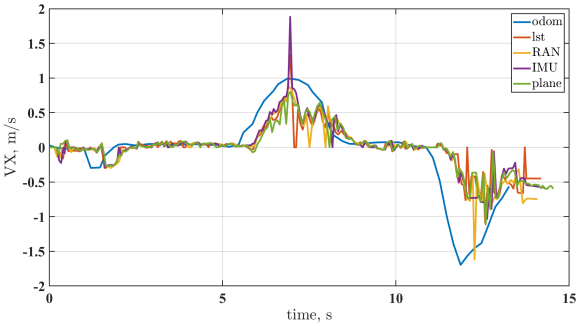

Fig. 11. Quadrotor's X velocity computed from various methods

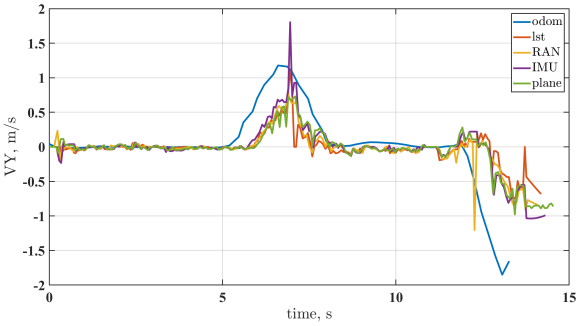

Fig. 12. Quadrotor's Y velocity computed from various methods

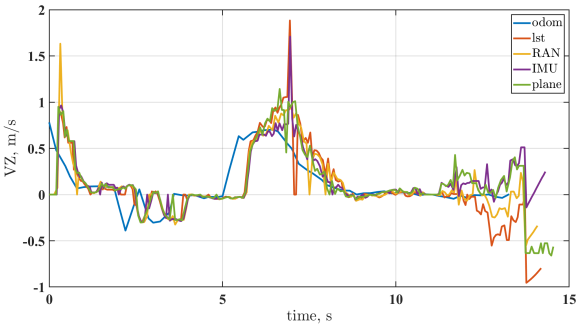

Fig. 13. Quadrotor's Z velocity computed from various methods

C. Orientation Estimates

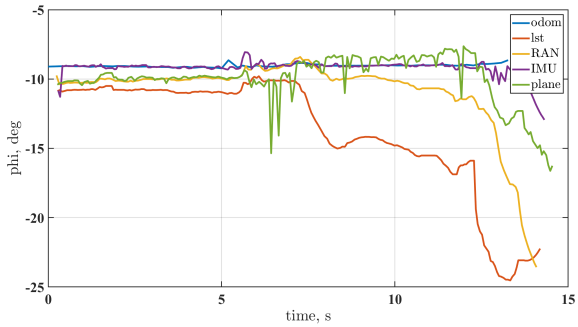

Fig. 14. Quadrotor's roll angle (ϕ) computed from various methods

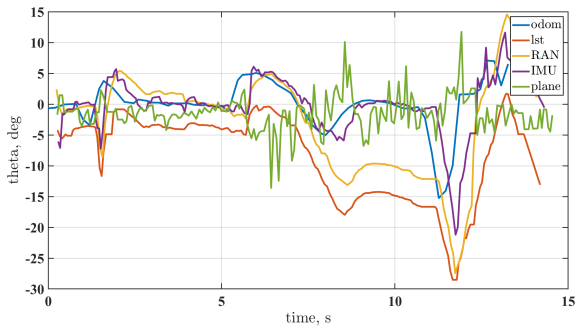

Fig. 15. Quadrotor's pitch angle (θ) computed from various methods

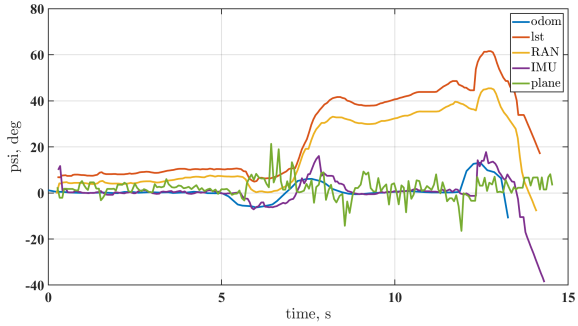

Fig. 16. Quadrotor's yaw angle (ψ) computed from various methods

V. Results for Helix Dataset

Helix dataset had the quadrotor flying at higher altitudes and therefore, the position estimates worsened at increasing rates as height increased. However, the number of false matches were low, and hence, in this case Linear least-squares method slightly performs better than RANSAC which takes more computation time and runs slower which leads to higher accumulation of numerical integration errors. The pose estimates coming from Bebop is plotted in blue, from Linear least-squares method given in Section II-F is plotted in red, from RANSAC method in Section II-G is plotted in yellow, from plane fitting method in Section III-A is plotted in green and from IMU method in Section III-B is plotted in purple.

A. Position Estimates

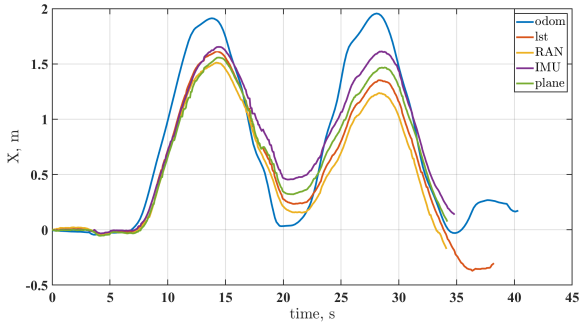

Fig. 17. Quadrotor's X-position computed from various methods.

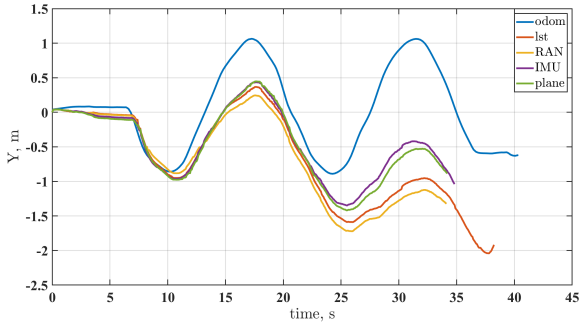

Fig. 18. Quadrotor's Y-position computed from various methods

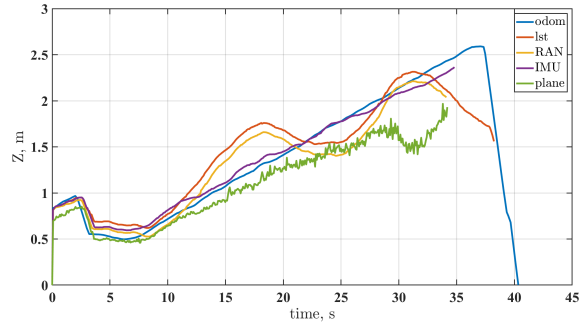

Fig. 19. Quadrotor's Z-position computed from various methods

B. Velocity Estimates

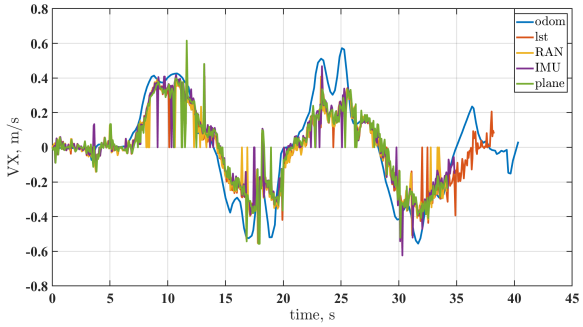

Fig. 20. Quadrotor's X velocity computed from various methods

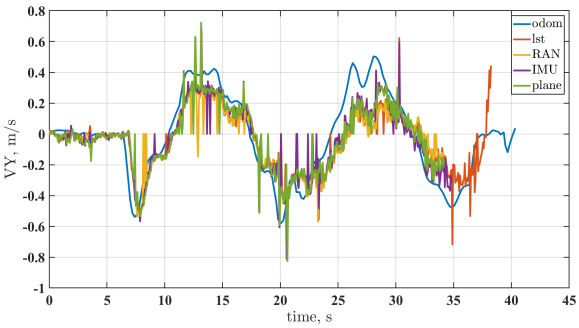

Fig. 21. Quadrotor's Y velocity computed from various methods

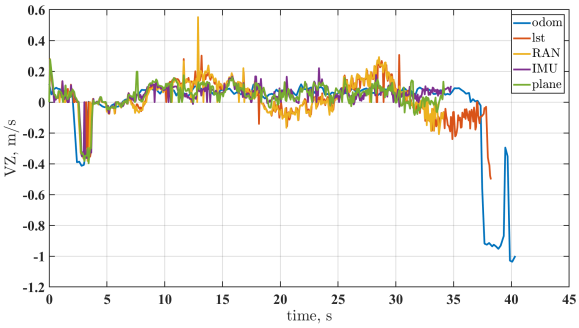

Fig. 22. Quadrotor's Z velocity computed from various methods

C. Orientation Estimates

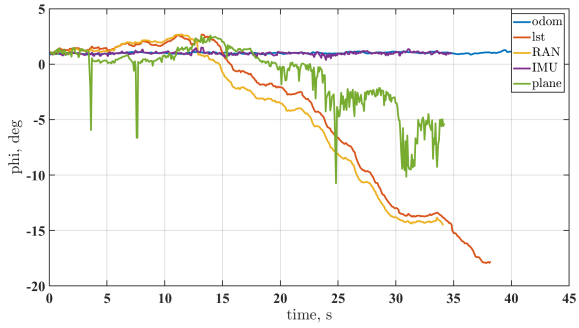

Fig. 23. Quadrotor's roll angle (ϕ) computed from various methods

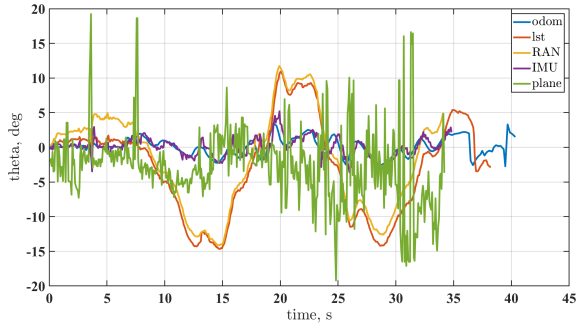

Fig. 24. Quadrotor's pitch angle (θ) computed from various methods

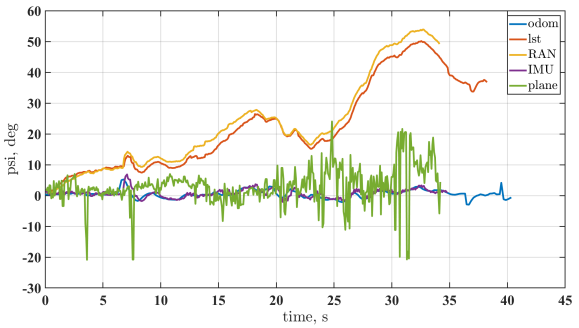

Fig. 25. Quadrotor's yaw angle (ψ) computed from various methods

VI. Conclusion

Accuracy of stereo visual odometry was found to be heavily dependent on altitude due to the fixed and small baseline of the stereo camera set. Depth computations are only accurate for a small range of altitudes, in our case a maximum of about 1.5 meters. The use of alternative depth estimates such as an ultrasonic distance sensor to estimate vehicle altitude would significantly improve the optical flow estimates. Additionally, IMU data could be used in place of the odometry orientation estimate to improve performance because a ground plane can easily be extracted when combined with a known camera depth. In this case, only a monocular downward facing camera would be required to temporally match the scene features to compute velocity. This scheme is probably what is already implemented on the bebop which is why our performance across all altitudes is inconsistent and worse than the bebop's.