VISION-BASED POSE AND VELOCITY ESTIMATION OF QUADROTOR USING APRIL TAGS AND COMPUTER VISION METHODS

Methods and Results

Jagennath Hari – jh7454

Robot Localization and Navigation - Project 2

ROBGY-6213 Spring 2022
Dr. Loianno
April 12, 2022

1) Introduction

In this project, I implemented a Vision-based pose estimation of a quadrotor using April tags. This project has two parts, Part 1 entails using the image and distance between each April tag. The corners were extracted for all the April tags using the camera. Those April tag IDs are subsequently used to compute the pose of the drone. In part 2 we directly used the video of the camera feed each frame at a time to extract the corners. The corners are tracked in the subsequent frame to find the pose of the drone.

The primary difference between each part is the method of corner extraction, this greatly affected the graphs generated, and by further implementing RANSAC for part 2 we have improved the graph.

2) Vision-Based Pose Estimation

The data was collected by a Nano+ quadrotor that was either handheld or flown through a prescribed trajectory over a mat of AprilTags, each of which has a unique ID. We calculated the coordinates of the corners of AprilTags using the data. Then, the homography matrix has been calculated using the plane relation. We then decomposed the homography matrix using Singular Value Decomposition (SVD) to find rotation and translation.

$$\lambda(i) \begin{pmatrix} x_i' \\ y_i' \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \tag{1}$$

$$(\hat{R_1}\hat{R_2}\hat{T}) = \begin{pmatrix} \hat{r_{11}} & \hat{r_{12}} & \hat{t_1} \\ \hat{r_{21}} & \hat{r_{22}} & \hat{t_2} \\ \hat{r_{31}} & \hat{r_{32}} & \hat{t_3} \end{pmatrix} = \begin{pmatrix} f & 0 & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \tag{2}$$

$$R_1^T R_2 = 0, ||R_1|| = ||R_2|| = 1 \tag{3}$$

$$(\hat{R_1}, \hat{R_2}, \hat{R_1} X \hat{R_2}) = USV^T \tag{4}$$

$$R = U \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(UV^T) \end{pmatrix} V^T \tag{5}$$

$$T = \hat{T}/||\hat{R_1}|| \tag{6}$$

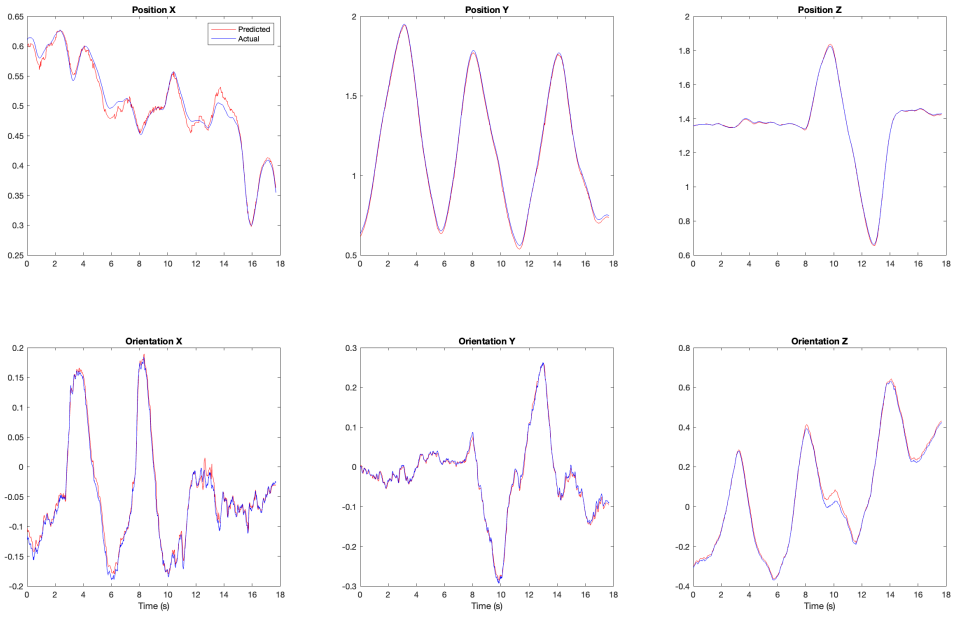

Figure 1: Pose Estimation for Dataset 1

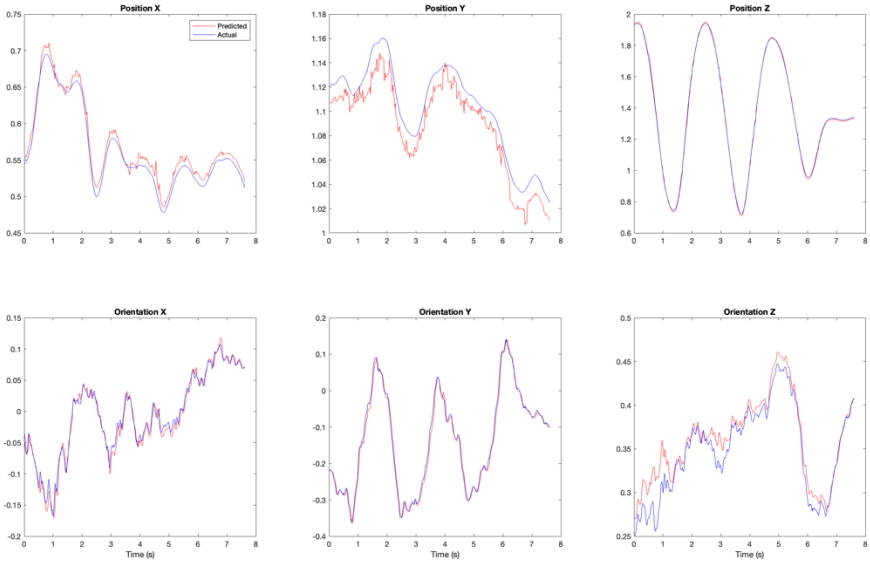

Figure 2: Pose Estimation for Dataset 4

2.1) Corner Extraction and Tracking

A Savitzky-Golay Filter was used to filter time, this ensured dt values were constant. The same filter was used on the estimated velocities to ensure the graph had minimum spikes.

Corners have been extracted using the ORB corner detector. ORB stands for oriented FAST and rotated BRIEF, which uses a modified version of FAST for finding the Keypoints and the BRIEF algorithm to compute binary descriptors. A Kande-Lucas-Tomasi Tracker was initialized to the previous frame to track those Keypoints to the next frame. This gives us the translation(distance) and using the difference in time between the frames, velocity was computed. This was used to find the linear and angular velocities of the quadrotor.

Implementation of the RANSAC algorithm has aided the estimation of velocities as the algorithm can differentiate between the inliers and outliers. These may be Keypoints that are incorrectly tracked, in general, these are those data points having a large distance from the best-fitted line. The equations for the implementation are given below –

$$\dot{\mathbf{p}} = \frac{1}{Z} A(\mathbf{p})\mathbf{V} + B(\mathbf{p})\boldsymbol{\Omega} \tag{7}$$

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{Z} \begin{pmatrix} -1 & 0 & x \\ 0 & -1 & y \end{pmatrix} \begin{pmatrix} V_x \\ V_y \\ V_z \end{pmatrix} + \begin{pmatrix} xy & -(1+x^2) & y \\ 1+y^2 & -xy & -x \end{pmatrix} \begin{pmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{pmatrix} \tag{8}$$

$$\dot{\mathbf{p}} = \frac{1}{Z} A(\mathbf{p})\mathbf{V} + B(\mathbf{p})\boldsymbol{\Omega} = \begin{pmatrix} \frac{1}{Z} A(\mathbf{p}) & B(\mathbf{p}) \end{pmatrix} \begin{pmatrix} \mathbf{V} \\ \boldsymbol{\Omega} \end{pmatrix} \tag{9}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} f_1(x, y, Z) \\ f_2(x, y, Z) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \\ \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \tag{10}$$

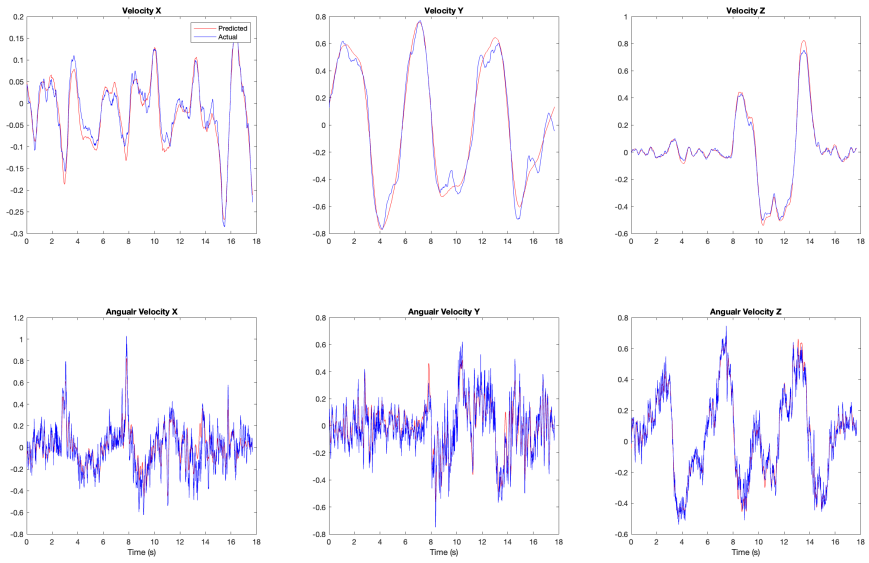

Figure 3: Velocity Estimation for Dataset 1 without RANSAC

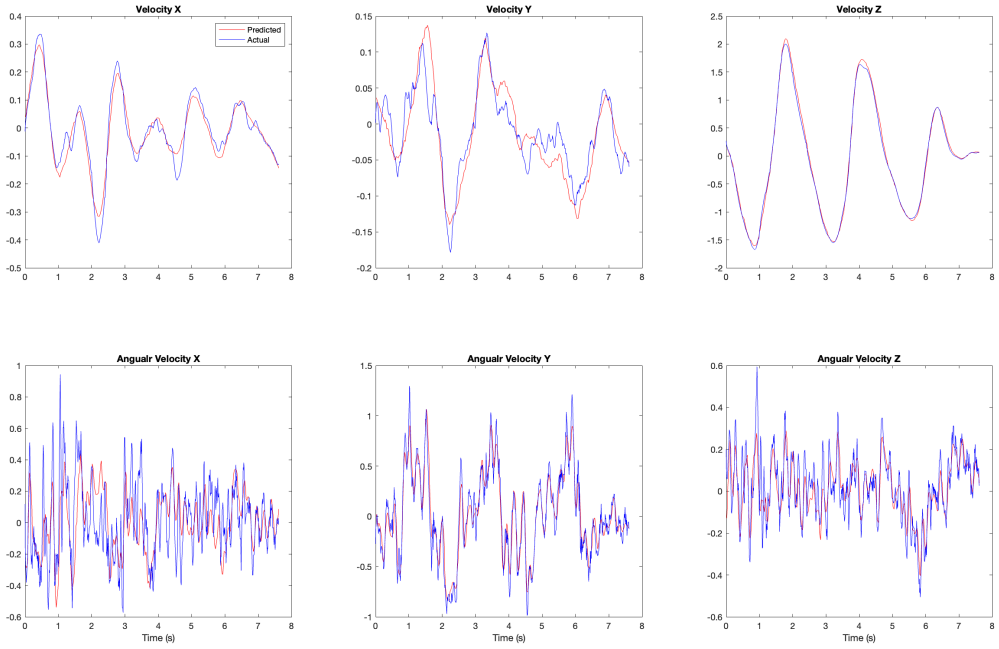

Figure 4: Velocity Estimation for Dataset 4 without RANSAC

2.2) RANSAC Implementation

RANSAC implements a best-fitted line like a regression method. This method runs for a certain number of iterations which is set by hyperparameters. This generates a fitted line for each iteration and using a certain threshold(distance) for the data points we accept the inliers and exclude the outliers. This corrects the optical velocity as incorrectly tracked Keypoints or sensor noise are excluded from the result. The below formula gives the number of iterations needed, psuccess is 0.99 and the rest are hyperparameters.

$$k = \frac{\log(1 - p_{success})}{\log(1 - \epsilon^M)} \tag{11}$$

The results are given below –

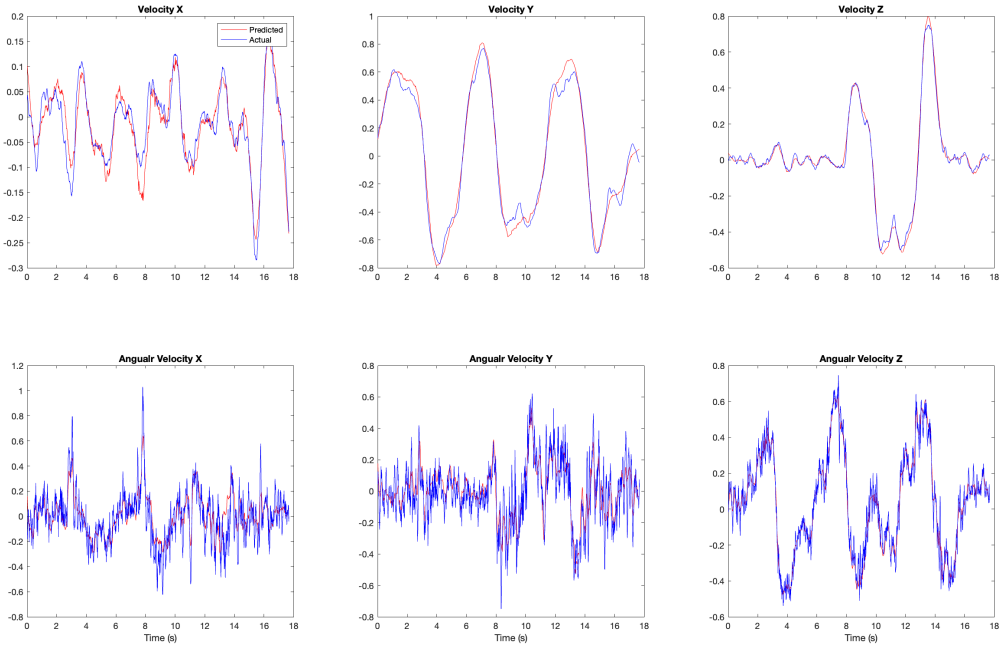

Figure 5: Velocity Estimation for Dataset 1 with RANSAC

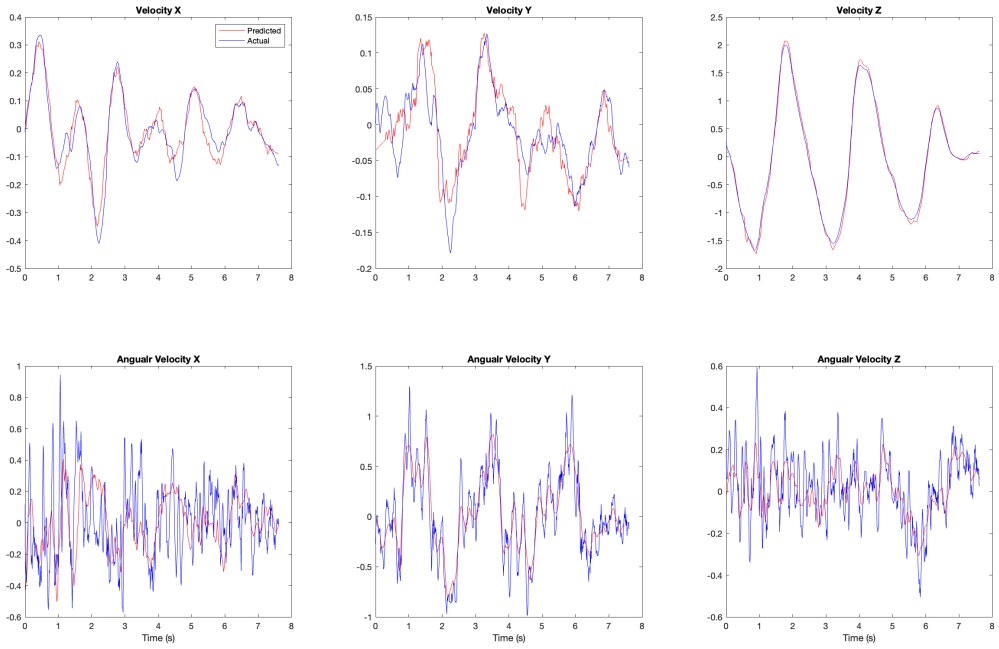

Figure 6: Velocity Estimation for Dataset 4 with RANSAC

3) Results and Conclusion

The graphs for part 1 are excellent. This can be a result of using the ID of the April tags to get the corners since these corners are computed manually rather than using any vision-based sensor (because the IDs were already given to us). Pose and Orientation are only calculated so the error is marginal being a first order function.

Part 2 graphs with RANSAC is worse because outliers are eliminated so fewer Keypoints are being tracked even though they are correct. This results in fewer estimated velocities so there are more chances of the values being wrong. The datasetNum 4 may be incorrect and has considerable noise. Furthermore, velocity estimation is a second order function so the error increases. The last limitation is the RANSAC function itself due to the lack of datasetNum not being included, fine-tuning of the threshold was not possible.