ROBOT LOCALIZATION AND NAVIGATION (ROB-6213)
PROJECT 1 REPORT

Rushi Bhavesh Shah (rs7236)

---

1. Introduction

Optical flow or optic flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene. Optical flow can also be defined as the distribution of apparent velocities of movement of brightness patterns in an image. Sequences of ordered images allow the estimation of motion as either instantaneous image velocities or discrete image displacements.

AprilTag is a visual fiducial system, useful for a wide variety of tasks including augmented reality, robotics, and camera calibration. Targets can be created from an ordinary printer, and the AprilTag detection software computes the precise 3D position, orientation, and identity of the tags relative to the camera.

2. Problem Statement

2.1 Vision Based Pose Estimation

The data for this phase was collected using a Nano+ quadrotor that was either held by hand or flown through a prescribed trajectory over a mat of AprilTags, each of which has a unique ID. Using the camera calibration data, the corners of the tags in the frame, and the world frame location of each tag one must compute the measured pose of the Nano+ for each packet of data.

2.2 Corner Extraction and Tracking

Corners in each image need to be extracted. Then for all corners in the image, the sparse optical flow between two consecutive images needs to be computed using the KLT tracker.

2.2.1 Velocity Estimation

Given a corner in the calibrated image frame $[x, y]^T$ and its optical flow $[\dot{x}, \dot{y}]^T$, the following linear equation holds:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \mathrm{f}_1(x, y, Z) \\ \mathrm{f}_2(x, y, Z) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \\ \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix}$$

Assuming that all corners are on the floor, Z can be computed based on the estimated height and rotation of the quadrotor using the AprilTags. Note that Z does not necessarily equal the height of the quadrotor due to nonzero roll and pitch.

2.2.2 RANSAC-Based Outlier Rejection

It is likely that there are outliers in the optical flow computation. It is needed to reject outliers using RANSAC. Three sets of constraints are required to solve the system , and thus a 3-point RANSAC can be used for outlier rejection.

3. Solution for Vision Based Pose Estimation

3.1 getCorner Function

This function aims at finding the coordinates of all the AprilTags present on the mat in the work frame by considering the given dimensions. The input to the function is the list of ids that the camera captures in one time interval, and the output gives the list of the coordinates of the 4 corners of a particular id. Hence, first of all, a zero matrix was created with 8 rows (for x and y coordinates of all 4 corners) and number of columns equal to the no. of ids present in the list. Further, the column number for each id was calculated for each tag by the following equation : *col = fix(req_id/12) + 1*. Further, x-coordinate for each corner is calculated, since there is no non-uniformity in the distances between two tags. Later, for calculating the y-coordinates, the column is taken into account since the distances between all the columns is not uniform. Later, the x and y coordinates for all the corners are stacked in a matrix.

3.2 estimatePose Function

This function aims at estimating the pose of the quadrotor based on the images that it captures of the AprilTags, and later plots the data with the data from Vicon for the comparison. To start, we first get the camera calibration matrix from the parameter.txt file. Later on, we take the list of all the id that camera captures at that particular timestamp *t*. Further, we create a null matrix A, which will contain the following matrix for the coordinates of all the corners of all the tags in that particular camera image. Where *h* is the column vector containing vertically stacked elements of a homographic transformation matrix. We can find the value of *h* by calculating the Singular Value Decomposition of the matrix A. Hence the no. of rows in the matrix A would be equal to 8*no. of ids and the no. of columns would be 9. Hence, we'll use the inbuilt MATLAB function for calculating the SVD of the vector A i.e. *[S,U,V] = svd(A)*. Hence, $h_{3x3} = V(9)$ i.e. all the elements in V make up a 3x3 *h* matrix.

$$\begin{pmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x'_i x_i & -x'_i y_i & -x'_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -y'_i x_i & -y'_i y_i & -y'_i \end{pmatrix} \boxed{h} = 0$$

Vector of unknown transformation parameters

Furthermore, we take the *h* matrix and multiply it with the

$$\begin{pmatrix} \hat{R}_1 & \hat{R}_2 & \hat{T} \end{pmatrix} = \begin{pmatrix} \hat{r}_{11} & \hat{r}_{12} & \hat{t}_1 \\ \hat{r}_{21} & \hat{r}_{22} & \hat{t}_2 \\ \hat{r}_{31} & \hat{r}_{32} & \hat{t}_3 \end{pmatrix} = \underbrace{\begin{pmatrix} f & 0 & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}}_{K^{-1}H}$$

$$\left(\hat{R}_1 \;\; \hat{R}_2 \;\; \hat{R}_1 \times \hat{R}_2\right) = USV^T$$

inverse of the camera calibration matrix *K,* to give

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

Now, to calculate the Rotation matrix we take the SVD of the
matrix on the LSH in the equation.

and to calculate the position vector T, we have $T = \hat{T}/\|\hat{R}_1\|$

But, we need the values in the body frame, hence we'll take the data from the parameters.txt file to generate a homogeneous transformation matrix to transform the values from the world from to the body frame. Furthermore, we extract the rotation matrix from this homogeneous transformation matrix and calculate the Euler angle in *ZYX* orientation and then also extract the position vector.

4. **Solution for Corner Extraction and Tracking**
   For this fort, we copy the getCorners and estimatePose functions from the previous section.

4.1 Velocity Estimation

Before starting the solution for this section, we create a row vector containing all the timestamps from the given data and pass it through a low pass filter. Then we initialize the loop to load the captured images and create two variables containing previous and current images and also we calculate the difference in the timestamp for the two images. Then we detect the *good_pts* by using the function *detectMinEigenFeatures.* Then we initialize the tracker to the last frame by using the vision.PointTracker function from the Computer Vision toolbox in Matlab. Further, we find the location of the points from the current image. Then we find the location of the next point. Further we find the value of *Z* by normalizing the values of *good_pts* and *pts* by multiplying them by the *inv(K).* Then, we calculate the velocity in camera from by taking the x and y coordinates of the corners of the images and dividing them by the difference between the timestamps to get the LHS of the following equation:

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

And, hence further, we have *(u;v), (x,y)* and *Z* and we need to find V and Omega. We find *x* and *y* by using the normalized values of *good_pts* and *pts*. We put this in a loop to calculate the *Ap* and *Bp* matrices. Which when stacked together make up the *H* matrix. Further we multiply the pseudo-inverse of the H matrix to the velocity, and the transformation matrix, we get the estimated velocity.

$$\left(\begin{pmatrix}\frac{1}{Z_1}\boldsymbol{A}(\boldsymbol{p}_1) & \boldsymbol{B}(\boldsymbol{p}_1)\\ \vdots & \\ \frac{1}{Z_n}\boldsymbol{A}(\boldsymbol{p}_n) & \boldsymbol{B}(\boldsymbol{p}_n)\end{pmatrix}\begin{pmatrix}\mathbf{V}\\ \boldsymbol{\Omega}\end{pmatrix}-\begin{pmatrix}\dot{\boldsymbol{p}}_1\\ \vdots\\ \dot{\boldsymbol{p}}_n\end{pmatrix}\right)^{T}\left(\begin{pmatrix}\frac{1}{Z_1}\boldsymbol{A}(\boldsymbol{p}_1) & \boldsymbol{B}(\boldsymbol{p}_1)\\ \vdots & \\ \frac{1}{Z_n}\boldsymbol{A}(\boldsymbol{p}_n) & \boldsymbol{B}(\boldsymbol{p}_n)\end{pmatrix}\begin{pmatrix}\mathbf{V}\\ \boldsymbol{\Omega}\end{pmatrix}-\begin{pmatrix}\dot{\boldsymbol{p}}_1\\ \vdots\\ \dot{\boldsymbol{p}}_n\end{pmatrix}\right)$$

$$\left(\mathbf{H}\begin{pmatrix}\mathbf{V}\\ \boldsymbol{\Omega}\end{pmatrix}-\dot{\boldsymbol{p}}\right)^{T}\left(\mathbf{H}\begin{pmatrix}\mathbf{V}\\ \boldsymbol{\Omega}\end{pmatrix}-\dot{\boldsymbol{p}}\right) \qquad \begin{pmatrix}\mathbf{V}^{*}\\ \boldsymbol{\Omega}^{*}\end{pmatrix}=\mathbf{H}^{\dagger}\dot{\mathbf{p}}$$

4.2 RANSAC-Based Outlier Rejection

The *velocityRANSAC* function accepts input as the *velocity* that is in the camera image frame, along with the normalized *pts*, Z, and *e* i.e the probability of one point being an inlier. So, we start by determining the probability of success of at least hitting one inlier as 0.99. Then we calculate the no. of iterations *k* using the equation on the left.

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

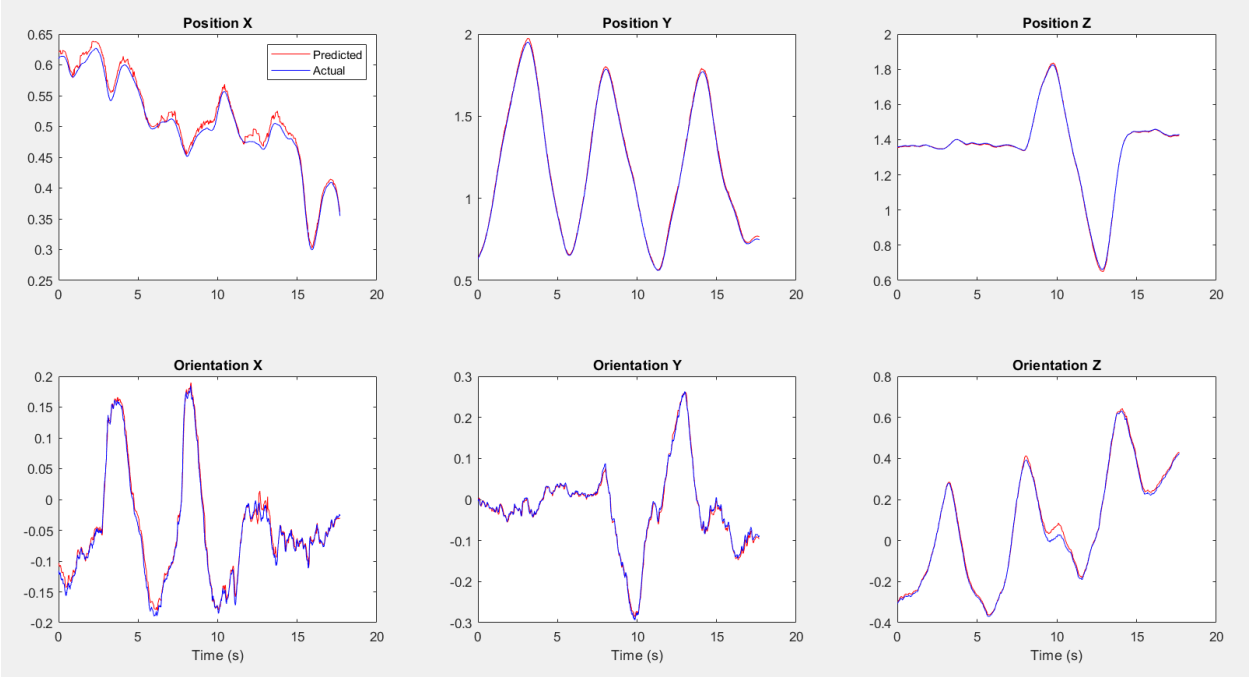
Further, the algorithm works in the following manner:

The algorithm (in pseudocode):

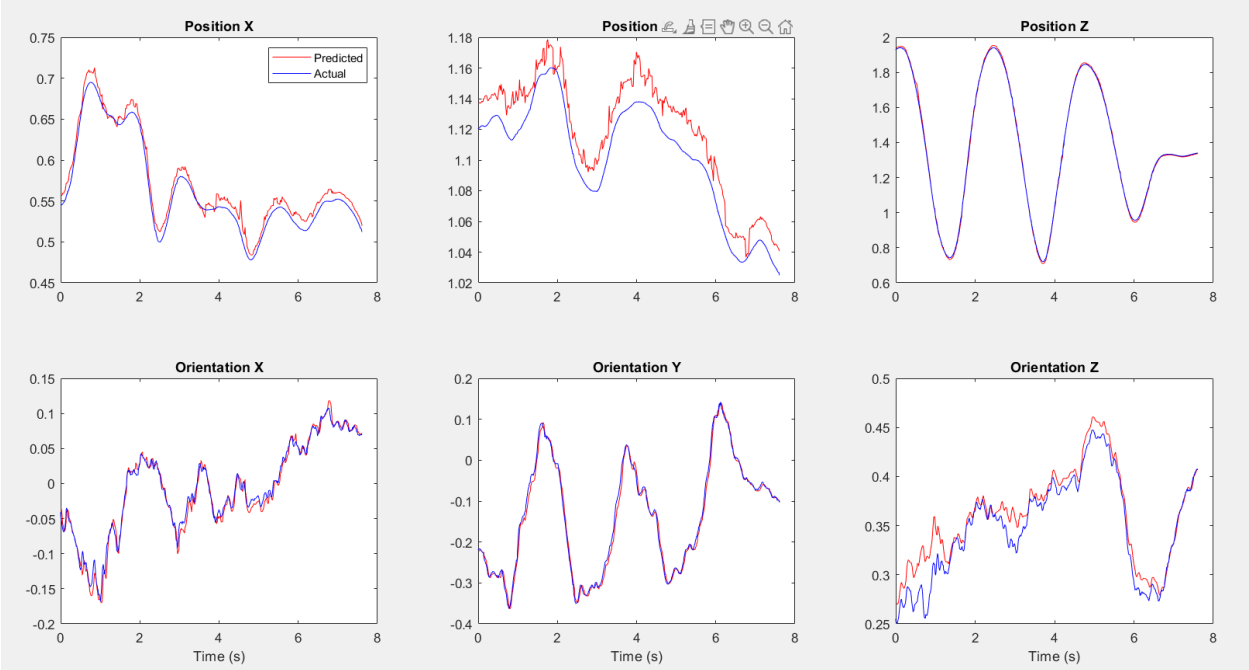
```
Repeat for k iterations
1.  Choose a minimal sample set
2.  Count the inliers for this set
3.  Keep maximum, if it exceeds a desired number of inliers
stop.
```

We start by generating a matrix with the random values of positions. Then we use the normalized *pts* to get the H matrix using the velocity given as input to calculate a temporary velocity matrix. Then we check the difference between the calculated velocity, H_i and P_i. We check the difference between the norm and the calculated value, if it is less than the threshold value, the the inlier count increments.

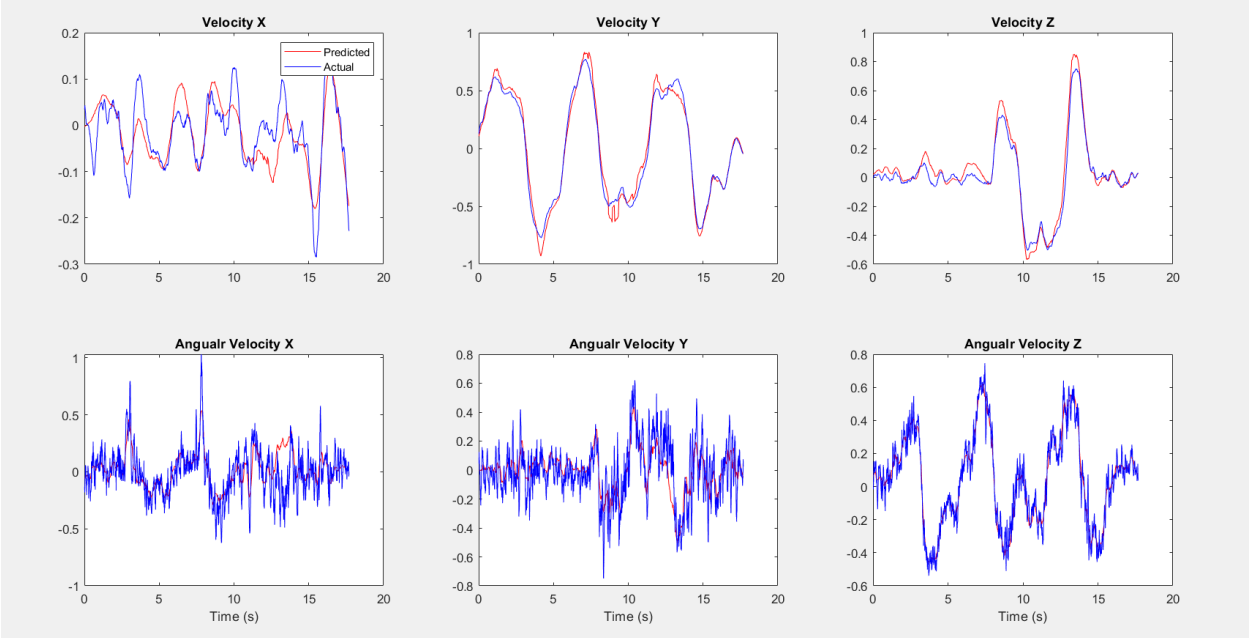
5. Plots



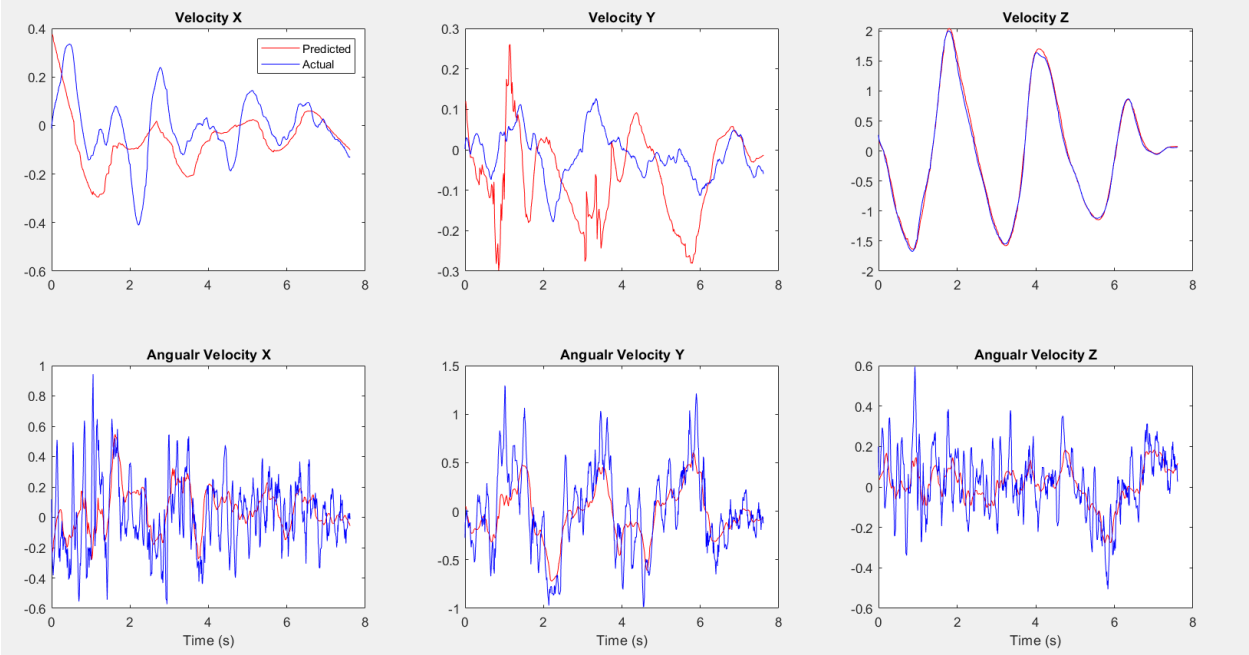
Part 1 Dataset 1



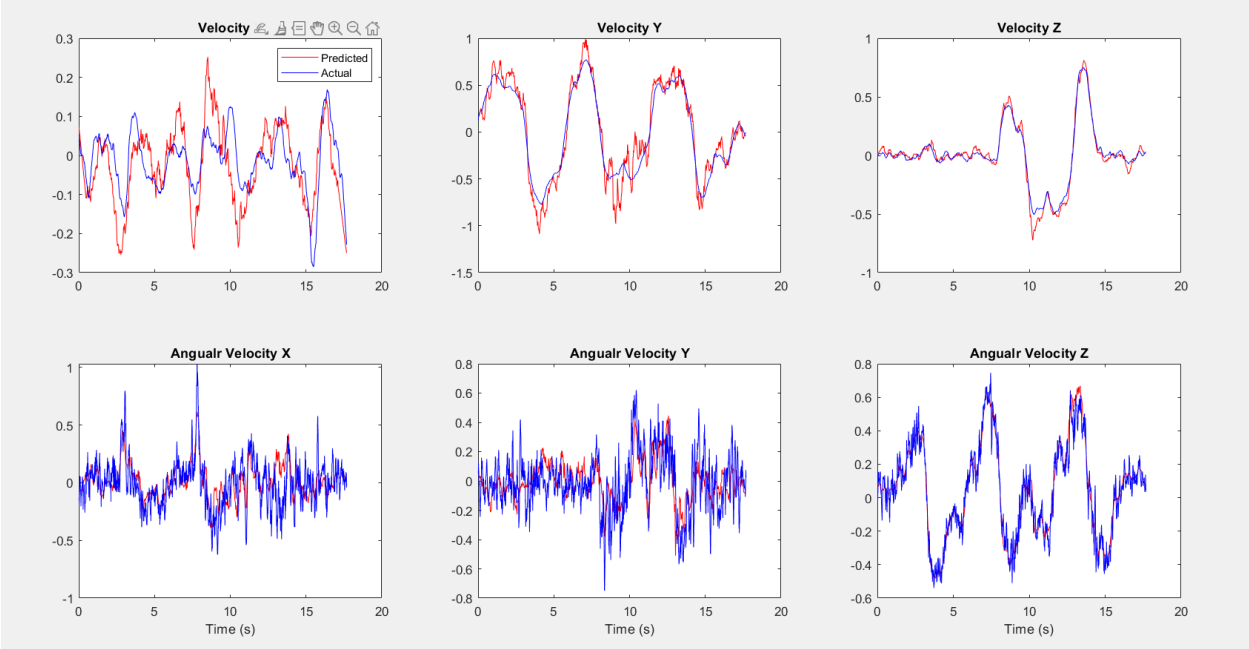
Part 1 Dataset 4



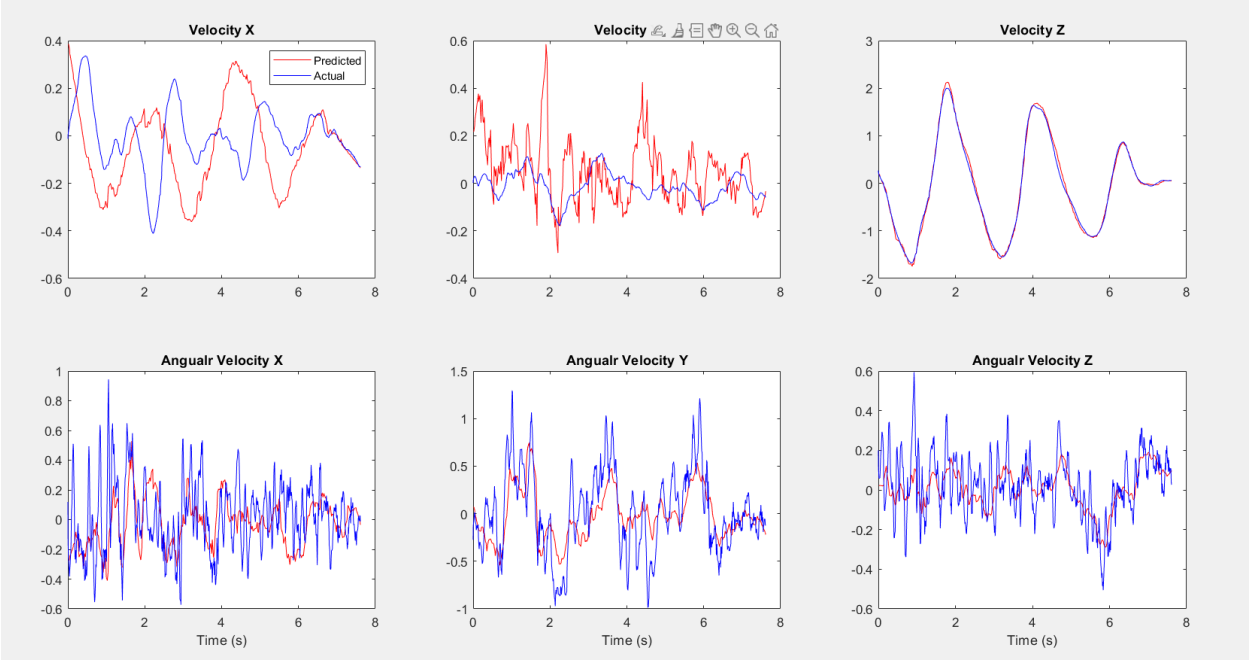
Part 2.1 Dataset 1



Part 2.1 Dataset 4



Part 2.2: Dataset 1



Part 2.2 Dataset 4

5. Conclusion

In part one, the quadrotor estimates its pose pretty accurately when compared to the ground truth. In part two, the application of RANSAC-Based Outlier rejection makes the system more robust and gives stable values.