Robot Localisation and Navigation Project 2 Report:

Name: Santosh Srinivas Ravichandran
Net ID: sr6411

Abstract: This report presents a vision-based 3D pose and velocity estimator for a quadrotor using AprilTags. The pose estimation leverages camera calibration data, AprilTag corners, and world frame locations to compute the quadrotor's pose for each data packet. Velocity estimation is achieved by extracting corners, computing optical flow, and rejecting outliers using RANSAC. The approach, mathematical formulations, and algorithms used for pose and velocity estimation are discussed. Results are compared against ground truth Vicon data, demonstrating the effectiveness of the implemented system.

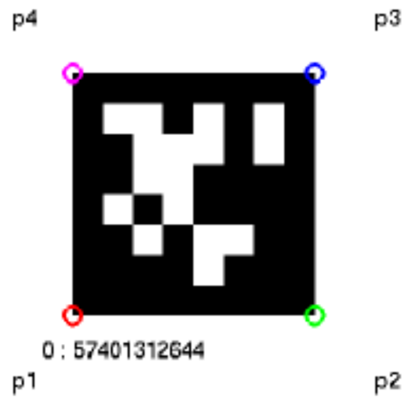
Dataset Description:

A 12*9 grid of AprilTags used for pose estimation is given in the world frame. Corner of every tag is computed using the getcorners function.

Parameters.txt: IDs for every tag arrangement, Camera calibration matrix, Rotation and translation matrix between camera and imu frame is given in the file.

Each data packet contains:

- Timestamp
- April Tag Id
- Corner and center locations (p4 p43 p2 p1 p0)



Project Methodology:

The project is divided into 2 parts and the methodology for each is discussed below.

Part 1 Methodology:

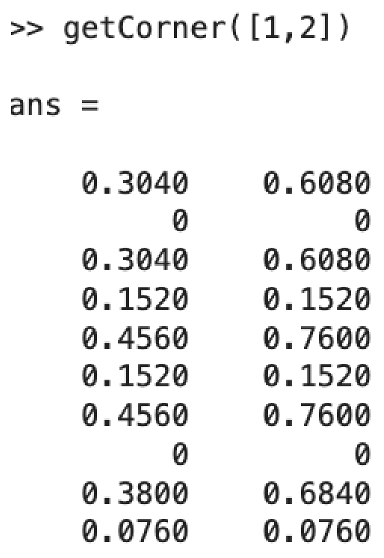
Part 1 involves estimating the pose of the quadrotor using AprilTags, homography and camera calibration.

The first step is to compute the coordinates of the corners in the world frame for which the getCorners.m file was coded.

getCorners.m

Function definition: It takes a list of AprilTag IDs as input and returns the coordinates of the four corners and the center of each AprilTag in the world frame.

Description: Calculates the row and column indices based on the AprilTag's position in the 12x9 grid, considering the tag size and spacing. It then computes the x and y coordinates of the top-left corner and derives the coordinates of the other corners and center. The output is a matrix containing the coordinates of the corners and center for each detected AprilTag, column-wise. Column i has the corner and center coordinates of AprilTag of ID- i in the order of p4,p3,p2,p1,p0.



```
>> getCorner([1,2])

ans =

    0.3040    0.6080
         0         0
    0.3040    0.6080
    0.1520    0.1520
    0.4560    0.7600
    0.1520    0.1520
    0.4560    0.7600
         0         0
    0.3800    0.6840
    0.0760    0.0760
```

Pseudo-code:

```
Iteration over all  Ids:
        For each id compute its row index and column index based on the grid given;
        Based on the index compute top left corner
        Based on the top left corner, derive remaining corners and center
        Place all in the ith column
```

<u>EstimatePose.m:</u>

Function definition: The primary goal of the function is to compute the drone's pose, including both its position and orientation, at a specific time frame t within the dataset. This computation relies on the identification of AprilTags within the visual data collected by the drone's camera.

1. Initialization:

The function begins by initializing an empty matrix `A` and setting up the camera's intrinsic matrix `k` given in the question.

2. AprilTag Processing:

For each detected AprilTag in the dataset at time `t`, the function retrieves the tag's corners in both the world frame and the camera frame, essential for constructing the transformation equations.

3. Matrix Construction:

Using the corner coordinates, the function constructs a series of linear equations encapsulated in the matrix `A`. This matrix bridges the spatial information between the camera and world frames.

$$A = \begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x'_i . x_i & -x'_i . y_i & -x'_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -y'_i . x_i & -y'_i . y_i & -y'_i \end{bmatrix}$$

$where\, x_i,\, y_i$ co-ordinates of points in world frame and $x'_i,\, y'_i$ are image co-ordiates in camera

4. Singular Value Decomposition (SVD):

The function applies SVD on matrix `A` to extract the homography matrix `H`, which represents the transformation from the world frame to the camera frame.

$$A = USV^T$$

$$h = V(:,9)$$

$$H = \operatorname{sign}(V(9,9)) \cdot \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

5. Pose Extraction:
Through matrix manipulation involving `H` and the camera intrinsic matrix `k`, the function calculates the rotation matrix `R` and the translation vector `T`, effectively estimating the drone's pose.

$$R = K^{-1}.H$$

$$R1 = R(:,1)$$

$$R2 = R(:,2)$$

$$T = R(:,3)$$

$$(R1 \quad R2 \quad R1XR2) = USV^T$$

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

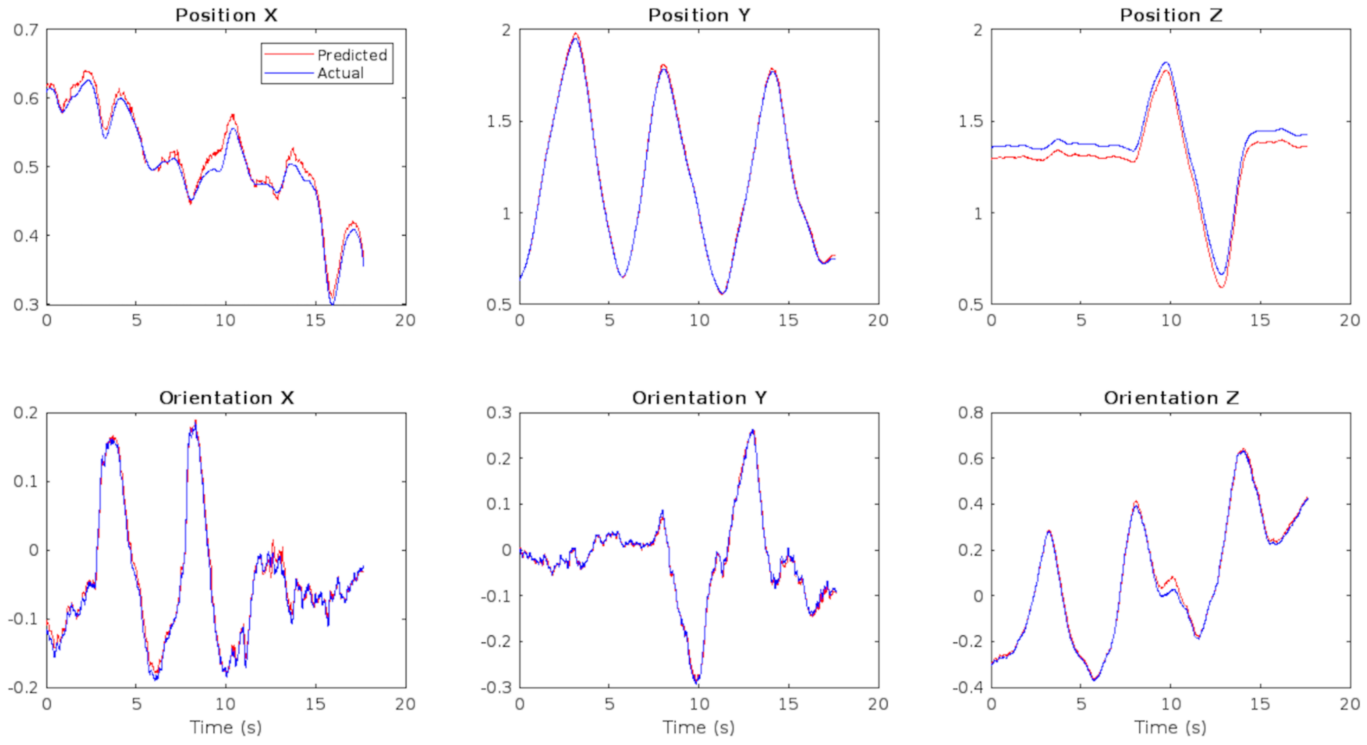
$$T = T/norm(R1)$$

6. World Frame Transformation:

The function then applies additional transformations to convert the pose from the camera frame to the world frame, using the known relationship between the camera and the drone's IMU.

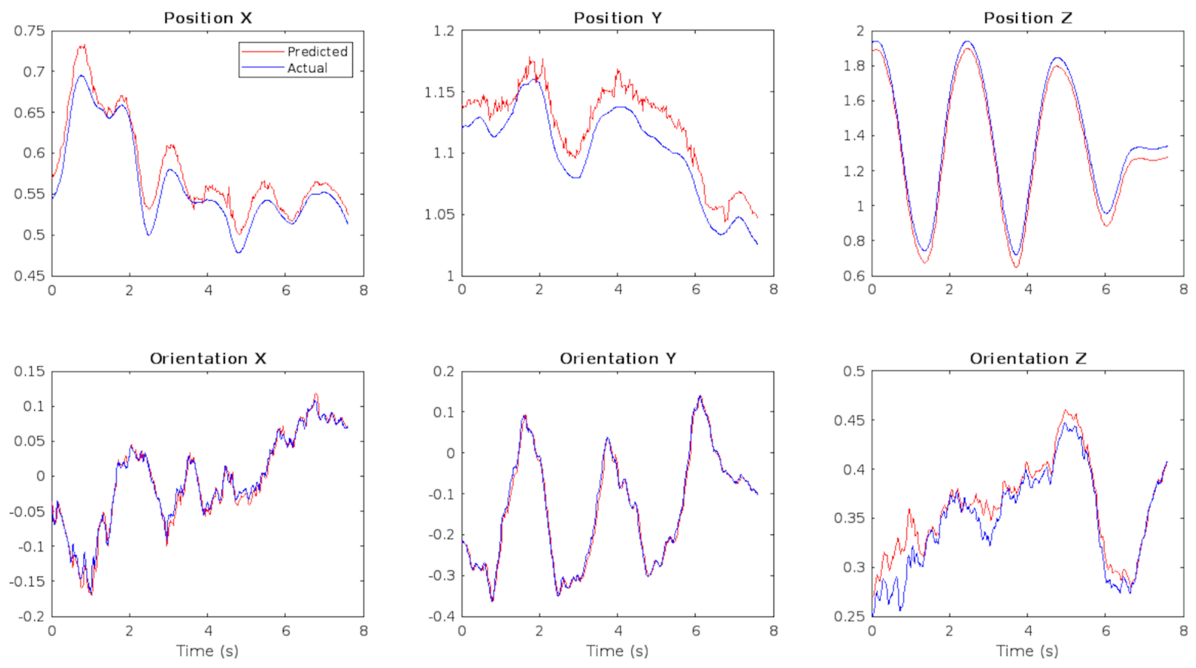
Results:

Dataset 1:



As can be seen above, the predicted and the actual positions for both position and orientation reasonably match very well. There is a slight offset in position z and position x predictions. But the orientations match very well.

Dataset 4:



As can be seen above, all the predicted and actual values are very similar. There is a small difference in position x and an offset in position Y. Position z and all the orientation results are very good. Dataset 4 results are slightly less accurate compared to dataset 1.

Part 2 Methodology:

Part 2 involves estimating the linear and angular velocity of the quadrotor using corner detection, optical flow estimation and RANSAC.

1. Corner Extraction and Tracking: The corner points are extracted using the **FAST Algorithm** from MATLAB's Computer Vision Toolbox, with the 100 strongest points being selected for tracking.

2. Tracking Points in the Second Image: The corner points are tracked into the second image using the KLT tracker. Both the initial corner locations and the tracked points are calibrated to camera coordinates using the camera matrix.

$$calibrated\,location = K^{-1}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

where x, y are corner image co-ordinates detected by the CV toolbox

3 . Optical Flow Estimation: The optical flow is determined by the displacement of tracked points from their initial positions. The optical velocity is computed by dividing the optical flow by the time difference (dt), which is refined using a low-pass filter to reduce noise.

$$optical\,flow\ =\ \Delta calibrated\,location(x,y) = \begin{pmatrix}\Delta x\\ \Delta y\end{pmatrix}$$

$$optical\,velocity\ =\ p^{\cdot}\ =\ \begin{pmatrix}\Delta x\\ \Delta y\end{pmatrix}/\Delta t$$

4. Depth Calculation of Corners: The drone's pose is estimated with the `estimatePose` function, providing necessary spatial orientation and position data. Transformation matrices that link the drone's body, camera, and world frames are derived. The depth of each corner is calculated based on geometric relationships established from the pose and camera perspective.

$$\begin{pmatrix}X_w\\ Y_w\\ 0\end{pmatrix}\ =\ ({}^{w}R_c)\cdot K^{-1}.\,\lambda.\begin{pmatrix}X_c\\ Y_c\\ 1\end{pmatrix}\ -\ ({}^{w}R_c)\cdot({}^{C}P_w)$$

$$For\,\text{finding }\lambda\text{, we use the last element of LHS that is 0 to RHS}$$

$$B1\ =\ {}^{w}R_c.\,K^{-1}\begin{pmatrix}X_c\\ Y_c\\ 1\end{pmatrix};\ A\,1 = ({}^{w}R_c)({}^{c}P_w)$$

$$\lambda\ =\ A1(3,1)/B1(3,1)\ =\ Z_C\ =\ depth\,of\,corner$$

5. Optional RANSAC for Velocity Estimation: If enabled, the velocity of the camera relative to the world frame is robustly estimated using the `velocityRANSAC` function. In the absence of RANSAC, velocity is calculated directly using all points, providing a straightforward method that may be more susceptible to outliers.

$$H\ =\ \begin{pmatrix}(1/Z_1).\,A(p_1) & B(p_1)\\ (1/Z_n).\,A(p_n) & B(p_n)\end{pmatrix}$$

$$\begin{pmatrix}V^{\star}\\ \omega^{\star}\end{pmatrix} = \text{pseudoinverse }(H)\cdot(p^{\cdot})$$

$$\text{where H is obtained as the general form for all points and } p^{\cdot}\ is\,the\,calibrated\,optical\,flow$$

6. Velocity Storage: The calculated velocity is stored in the `estimatedV` variable, conforming to a specified structure that details linear and angular velocity components.

Pseudo code:

```
1. Initialize and load dataset.
2. Define camera intrinsic matrix 'k' and RANSAC parameters 'e', 'ransac_flag'.
3. For each sample 'n' in the dataset:
   a. Load consecutive images (img1, img2) for the current and previous frames.
   b. Calculate time difference 'dt' between frames.
   c. Detect FAST features in img1 and select the strongest corners.
   d. Initialize a KLT tracker with these corners on img1.
   e. Use the tracker to find corresponding points in img2.
   f. Calibrate corner and point locations from image to camera coordinates.
   g. Calculate optical flow as the difference between corresponding points.
   h. Filter valid optical flows and their corresponding corner points.
   i. Estimate the flow velocity by dividing the valid optical flow by 'dt'.
   j. Estimate drone's position, orientation, and transform matrices.
   k. Calculate the depth 'Z' for each corner point.
   l. If 'ransac_flag' is 0, use all points to estimate camera velocity in the camera frame, then transform it to the world frame.
   m. If 'ransac_flag' is 1, use the RANSAC algorithm to estimate velocity.
   n. Store the estimated velocity in 'estimatedV'.
4. Plot the estimated velocity alongside ground truth data.
```

<u>RANSAC Implementation:</u>

1. Parameter Initialization: The algorithm sets up essential variables like the camera matrix `k`, success probability `P_success`, and RANSAC threshold `e`.

2. Iterative Sampling: It randomly selects minimal data points (three) in each iteration to form a hypothesis model based on the defined probability of success.

$$K = \log(1 - P_{success}) / \log(1 - \in^{m})$$

Where K is the number of iterations in which random sampling is done.

3. Model Hypothesis: The function constructs a matrix `H_opt` that relates optical flow velocities to the camera's hypothesized velocity, using selected sample points.

$$H_{opt} = \begin{pmatrix} (1/Z_1).\,A(p_1) & B(p_1) \\ (1/Z_2).\,A(p_2) & B(p_2) \\ (1/Z_3).\,A(p_3) & B(p_3) \end{pmatrix}$$

$$\begin{pmatrix} V^{\star} \\ \omega^{\star} \end{pmatrix} = H^{-1} \cdot (p^{\cdot})$$

where H is obtained as the general form for 3 randomly chosen points and $p^{\cdot}\ is\ the\ calibrated\ optical\ flow\ \ velocity for\ those$ 3 points

4. Inlier Detection: The process evaluates each data point against the hypothesized model, classifying points as inliers if they fall within a predefined error threshold.

$$\left\| \begin{pmatrix} (1/Z(i).\,A(p_i) & B(p_i) \end{pmatrix}. \begin{pmatrix} V^{\cdot} \\ \omega \end{pmatrix} - p^{\cdot}_i \right\| \le \beta$$

5. Optimal Model Selection: The algorithm identifies the iteration with the highest number of inliers, indicating the best fit among all hypothesized models.

6. Final Velocity Estimation: The function refines and computes the camera's velocity in the camera frame using inliers from the best model, and then transforms this velocity to the world frame.

$$H = \begin{pmatrix} (1/Z_1).\,A(p_1) & B(p_1) \\ (1/Z_n).\,A(p_n) & B(p_n) \end{pmatrix}$$

$$\begin{pmatrix} V^{\star} \\ \omega^{\star} \end{pmatrix} = \text{pseudoinverse } (H) \cdot (p^{\cdot})$$

where H is obtained the general form for all inliers and $p^{\cdot}\ is\ the\ calibrated\ optical\ flow$

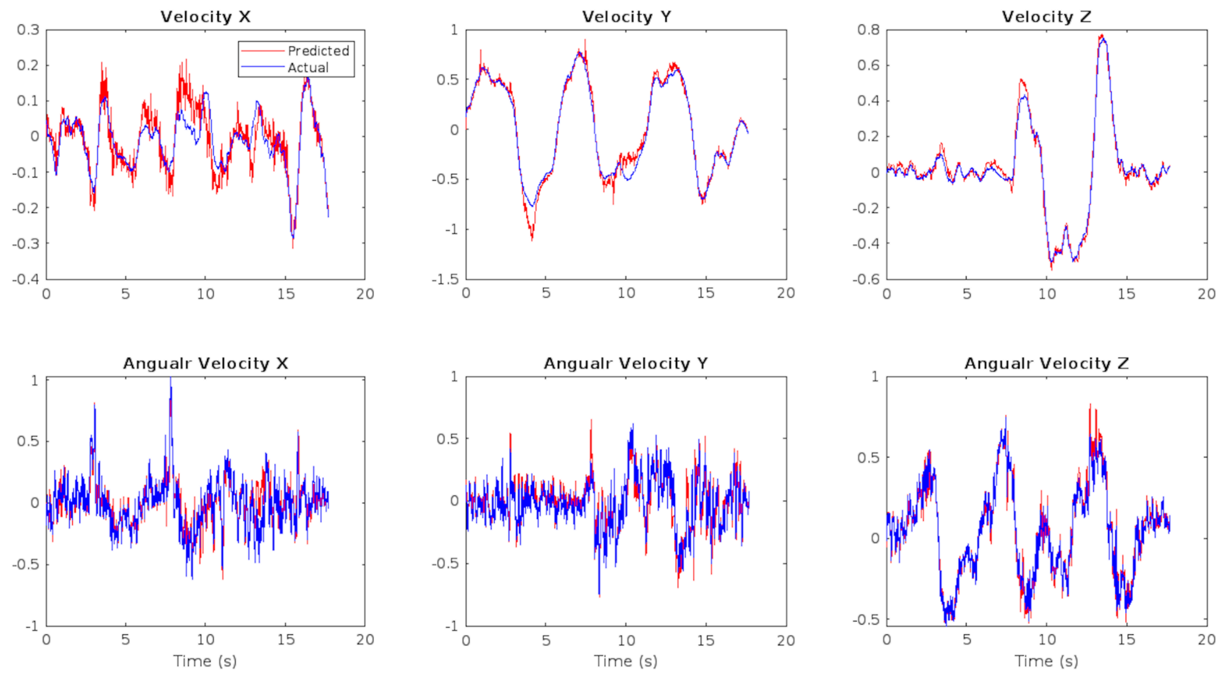
7. Output: The function outputs a robustly estimated velocity vector, representing both linear and angular velocities of the drone in the world frame.

Pseudocode:

```
function velocityRANSAC(optV, optPos, Z, R_c2w, e)
    Initialize P_success, M (minimum number of points), beta (threshold)
    Calculate iterations needed based on P_success and e

    Set total_num_points from optPos
    Initialize max_inlier_count to 0

    for each iteration
        Randomly select M indices from total_num_points
        Extract velocities, positions, and depths for selected points
        Construct H_opt matrix for the selected points

        Compute optimal_camera_velocity using H_opt and selected velocities

        Initialize inlier_indices and inlier_count

        for each point in total_num_points
            Construct H_j matrix for the current point
            Predict velocity for the current point
            Calculate error between predicted and observed velocity

            if error < beta
                Mark point as inlier and update inlier_indices and inlier_count

        if inlier_count > max_inlier_count
            Update max_inlier_count
            Extract positions, velocities, and depths of inliers
            Construct H matrix using all inliers
            Compute final camera velocity using H and inlier velocities
            Transform final velocity to world frame using R_c2w

    Return final velocity vector
end
```

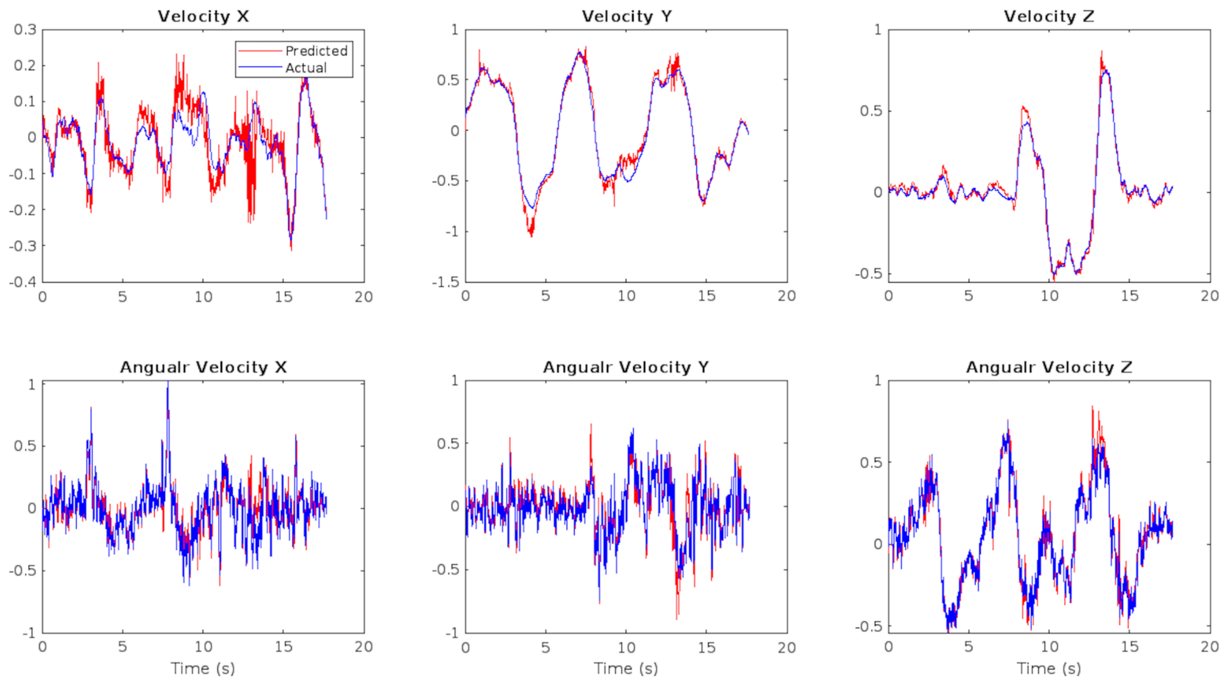
Results:

Dataset 1 Ransac off:



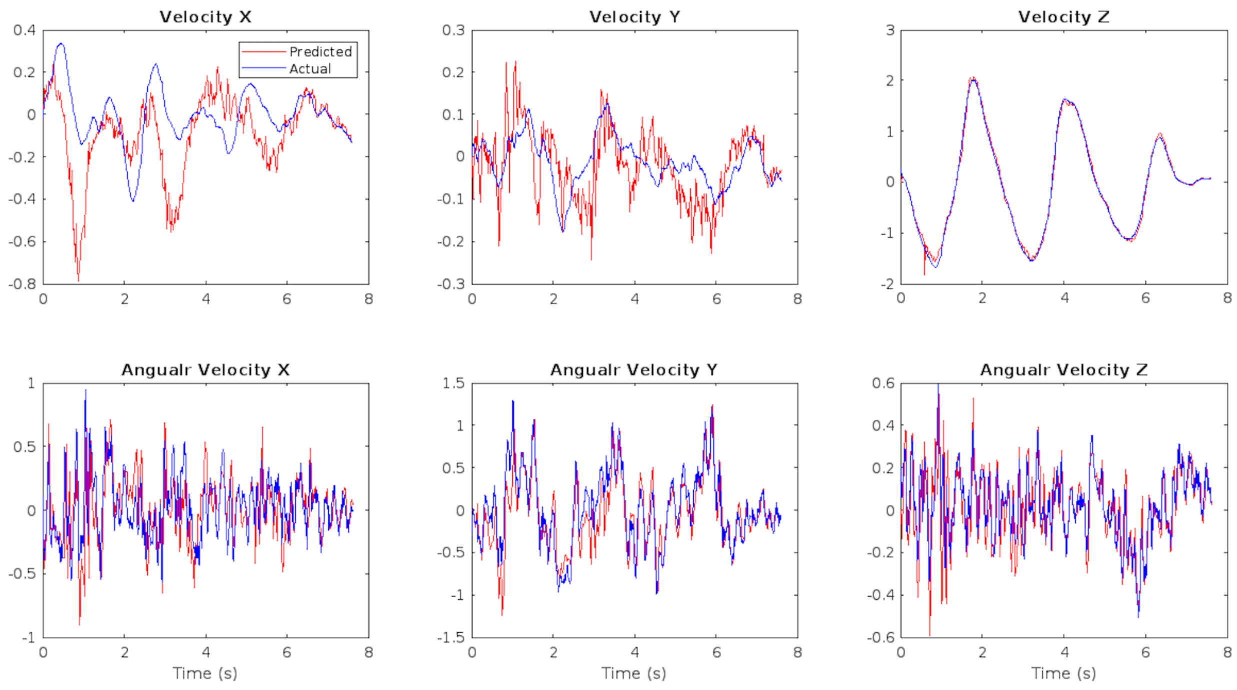
The plots for both velocity and orientation synchronize well, In x velocity there is a slight offset in horizontal direction. But the fluctuations are captured very well.

Dataset 1 Ransac On:



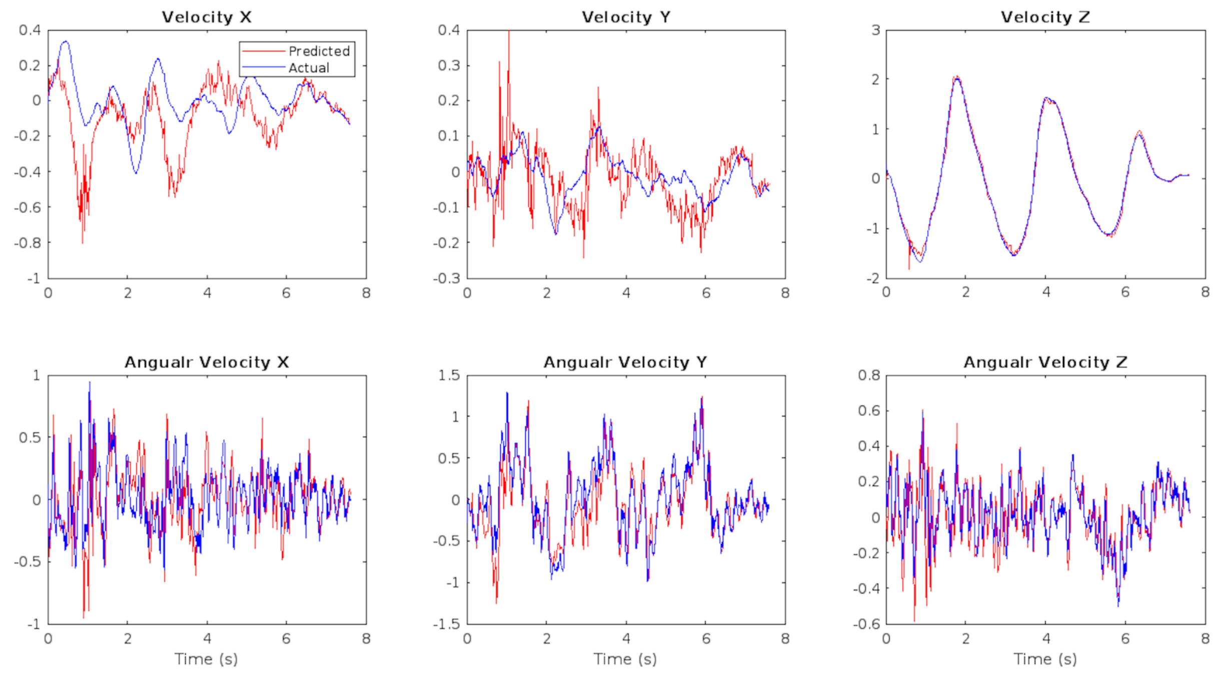
With RANSAC on, there is not much variation as compared to the plot without RANSAC. This can suggest that the dataset was already good to begin with, containing very less number of outliers or that the corner points chosen were good. This also suggest that the probabilistic approach to selecting a number of iterations works.

Dataset 4 Ransac off:



Compared to dataset 1, Dataset 4 has more offset between actual and predicted values.

Dataset 4 Ransac on:



In this case, with RANSAC on the deviations between actual and predicted values has decreased as can be noticed in velocity x for example. The negative peaks in velocity x deviate a lot less with RANSAC on, thus signifying the approach of including inliers only.

Conclusion:

In conclusion, this project successfully implemented a vision-based pose and velocity estimation system for a quadrotor using AprilTags. The pose estimation pipeline was done based on AprilTag detection and homography estimation which was then mathematically modeled to extract the pose of the quadarator and imu in the world frame after relevant transformations between frames were applied. Corner detection and tracking measuring the optical flow, depth estimation of corner points and RANSAC-based outlier rejection, was implemented to effectively estimate the quadrotor's linear and angular velocities. The predicted velocities and pose were very similar when compared with the ground truth vicon data highlighting the potential of vision-based estimation for quadrotor navigation and control, while acknowledging limitations and areas for future improvement. This project lays the foundation for further research and development in autonomous aerial robotics.