Project 2

Robot Localization and Navigation

Arjun Raja – ar6841

1. Project overview

This project entails vision-based estimation of pose, velocity and angular velocity of a Micro Aerial Vehicle. The data for this 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.

2. AprilTag corners

The tags are arranged in a 12 x 9 grid. The top left corner of the top left tag has been used as coordinate (0, 0) with the X coordinate going down the mat and the Y coordinate going to the right. Each tag is a 0.152 m square with 0.152 m between tags, except for the space between columns 3 and 4, and 6 and 7, which is 0.178 m.

In an ideal case with no offsets (ignore the 0.178 m for now), let d be the length of the square and distance between the squares ($d = 0.152$).

For some $(i, j) \in \boldsymbol{N}$ which denote increments of AprilTags along the positive X and Y axes respectively, the x and y distances for each point of all AprilTags form an arithmetic sequence described in the table and figure below.

Position	p0	p1	p2	p3	p4
x	$(2i-2)d + d/2$	$(2i-1)d$	$(2i-1)d$	$(2i-2)d$	$(2i-2)d$
y	$(2j-2)d + d/2$	$(2j-2)d$	$(2j-1)d$	$(2j-1)d$	$(2j-2)d$

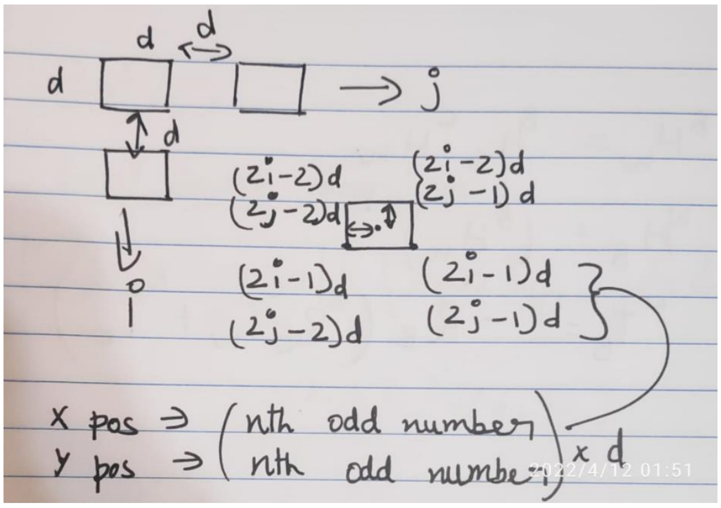
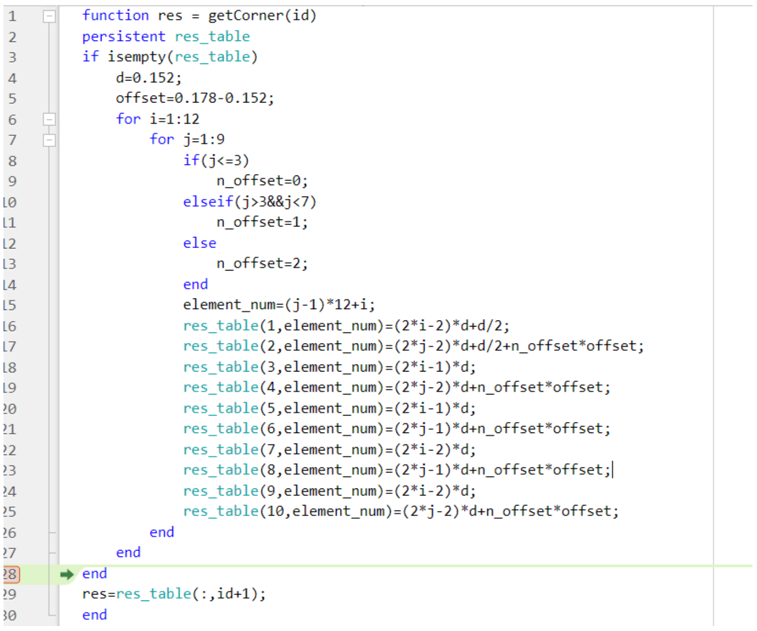


Figure 1 Corner point sequences

The world frame coordinates of points p0, p1, p2, p3 and p4 of all the AprilTags are computed in the getCorner function. Here the offset kicks into the y position when j>3 and j>6. A table of (x,y) positions is created where the columns signify the AprilTag ID and the rows signify the (x,y) coordinates stored in the order p0, p1, p2, p3. This table `'res_table'` is a persistent variable and exists between function calls to improve performance and is only created the first time the function getCorner is used. Subsequently only the ID column is pulled out and given as the output.



```
function res = getCorner(id)
persistent res_table
if isempty(res_table)
    d=0.152;
    offset=0.178-0.152;
    for i=1:12
        for j=1:9
            if(j<=3)
                n_offset=0;
            elseif(j>3&&j<7)
                n_offset=1;
            else
                n_offset=2;
            end
            element_num=(j-1)*12+i;
            res_table(1,element_num)=(2*i-2)*d+d/2;
            res_table(2,element_num)=(2*j-2)*d+d/2+n_offset*offset;
            res_table(3,element_num)=(2*i-1)*d;
            res_table(4,element_num)=(2*j-2)*d+n_offset*offset;
            res_table(5,element_num)=(2*i-1)*d;
            res_table(6,element_num)=(2*j-1)*d+n_offset*offset;
            res_table(7,element_num)=(2*i-2)*d;
            res_table(8,element_num)=(2*j-1)*d+n_offset*offset;
            res_table(9,element_num)=(2*i-2)*d;
            res_table(10,element_num)=(2*j-2)*d+n_offset*offset;
        end
    end
end
res=res_table(:,id+1);
end
```

Figure 2 getCorner code snip

3. Pose estimation

The pose of the MAV describes the position and orientation of the MAV in the world frame. This is computed in the estimatePose() function. The first goal is to use the captured data which contains the AprilTag ID, image coordinates of the points p0, p1, p2, p3, p4 for that tag ID and compute the homography of the camera at time stamp (t). The homography can be computed because we know the world frame coordinates of all the points for all the AprilTags. The homography can be computed by solving the equation

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

$$Ah = 0$$

Where $h = stacked(H)$. To do this we find the A matrix for all the IDs detected and solve the SVD $A = USV^T$ $(h\ will\ be\ the\ 9th\ column\ of\ V)$.

For each AprilTag ID; in the code `p_w(2*j-1)` is the **world x** coordinate of point p(j-1). `data(t).(fns{j+7})(1,i)` is the **image x** coordinate of point p(j-1). Similarly `p_w(2*j) and data(t).(fns{j+7})(2,i)` are the y coordinates of the point p(j-1) in the world and image frames respectively. h is then converted to a (3x3) homography matrix HL whose sign must first be corrected with respect to some point p and its coordinates in the world and image frames, and normalized to find the required homography H.

H contains the data for rotation and translation from the world to the camera frame. To get the transformation matrix we first find $K^{-1}H$ where K is the camera calibration matrix.

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

As orthonormality bust be preserved we need to find a rotation matrix R by finding the SVD.

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

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

The translation $\hat{T}$ must also be scaled by norm($\hat{R}$).

R describes $\;^{c}R_w$ *and* T describes $\;^{c}T_w$. Now we have $\;^{B}R_C =$`rotx(180)*rotz(45)`

and $\;^{c}T_B = [-0.04; 0; -0.03]$ from the parameters and pictures of the MAV provided. Thus we can form the equations

$$^{c}H_B = \begin{bmatrix} ^{c}R_B & ^{c}T_B \\ 0 & 1 \end{bmatrix}$$

$$^{c}H_w = \begin{bmatrix} ^{c}R_w & ^{c}T_w \\ 0 & 1 \end{bmatrix}$$

$$^{W}H_B = (\;^{C}H_W)^{-1} \;^{c}H_B$$

$^{W}H_B$ gives us $\;^{W}R_B$ which is a rotation matrix formed by ZYX Euler angle sequence. It can be converted using rotm2eul() or done manually using:

$$\begin{aligned} \boldsymbol{\beta} &= \tan_2^{-1}\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right) \\ \boldsymbol{\alpha} &= \tan_2^{-1}(r_{21}, r_{11}) \\ \boldsymbol{\gamma} &= \tan_2^{-1}(r_{32}, r_{33}) \end{aligned} \qquad \textit{where } \boldsymbol{\alpha}\, \boldsymbol{\beta}\, \boldsymbol{\gamma} \textit{ are ZYX Euler angles respectively}$$

The fourth column of $\;^{W}H_B$ gives us the translation (`position`) of the MAV in the world frame. The output order of orientation(1:3) is Z, Y and X Euler angles.

The output plots for part1 of the project are given in figures:

4. Corner Extraction and Tracking

Part2 of the project entails the calculation of velocity and angular velocity using the optical flow between frames.

To calculate the velocity and angular velocity of the MAV, features in the image must be identified to calculate optical flow. At some time stamp t, the features from the image at (t-1) were detected using the detectFASTFeatures() method provided in the MATLAB Computer Vision toolbox. The points with a metric greater than some cutoff_metric are then selected to ensure that the strongest points get used. Then the features at the current timestamp t are detected using a vision.PointTracker object from the Computer Vision toolbox, this object utilizes the KLT algorithm to detect and return the pixel coordinates of the matching features.

The pixel coordinates of the matching features of the images at timestamp (t) and (t-1) have now been established.

The pixel coordinates of `matched_points1` and `matched_points2` need to be normalized using the camera calibration matrix K. $(K)^{-1} * matched\_points$ gives the normalized image coordinates.

Note: in the code 'n' represents the timestamp 't'.

5. Optical flow

The optical flow of the image can be calculated by finding the shift in normalized pixel coordinate vector and dividing by change in time between frames. $\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \frac{1}{\Delta t}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}$

The Δt is passed through a lowpass filter to smooth the results. The optical flow $\dot{\mathbf{p}}$ is represented by `p_dot and opt_flow` in the code.

6. Velocity Estimation

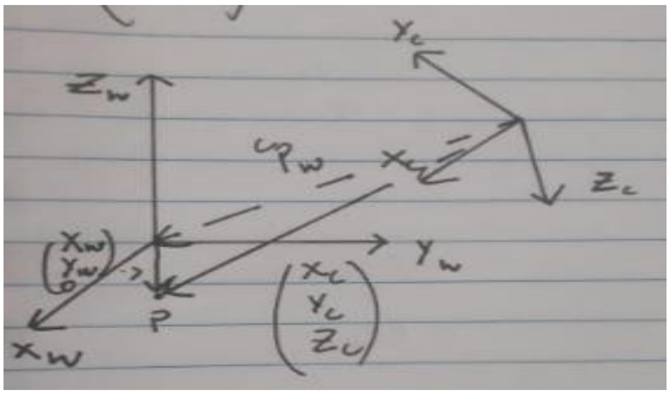
To compute the velocity and angular velocity we must solve the equation

$$\boldsymbol{V} = \begin{pmatrix}\mathbf{V}\\ \boldsymbol{\Omega}\end{pmatrix} = (\boldsymbol{H}^T\mathbf{H})^{-1}\boldsymbol{H}^T\dot{\mathbf{p}}$$

$$\mathbf{H} = \begin{pmatrix}\frac{1}{Z_1}\boldsymbol{A}(\boldsymbol{p}_1) & \boldsymbol{B}(\boldsymbol{p}_1)\\ \vdots & \\ \frac{1}{Z_n}\boldsymbol{A}(\boldsymbol{p}_n) & \boldsymbol{B}(\boldsymbol{p}_n)\end{pmatrix} \quad \dot{\mathbf{p}} = \begin{pmatrix}\dot{\boldsymbol{p}}_1\\ \vdots\\ \dot{\boldsymbol{p}}_n\end{pmatrix}$$

$$\boldsymbol{A}(\boldsymbol{p}_1) = \begin{pmatrix}-1 & 0 & x\\ 0 & -1 & y\end{pmatrix} \quad and \quad \boldsymbol{B}(\boldsymbol{p}_1) = \begin{pmatrix}xy & -(1+x^2) & y\\ 1+y^2 & -xy & -x\end{pmatrix}$$

Where $\boldsymbol{V}$ is a (6,1) vector containing the linear and angular velocities, $\dot{\mathbf{p}}$ is a (2x(number of corners used),1) vector and H is a (2x(number of corners used), 6) matrix. Z_n is the depth of the corner detected.



Y_c
Z_w
${}^cP_w$
X_c
Z_c
$\begin{pmatrix} X_w \\ Y_w \\ 0 \end{pmatrix}$
Y_w
$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$
P
X_w

Figure 3 Vector sum in camera frame

If the corner detected is at $\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix}$ in the camera frame, then we can form the vector sum:

$$ {}^{C}P_W + \quad {}^{C}R_W \begin{pmatrix} x_w \\ y_w \\ 0 \end{pmatrix} = \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \text{ where} $$

$$ {}^{C}\boldsymbol{P}_W \; is\ the\ position\ of\ the\ origin\ of\ the\ world\ frame\ written\ in\ the\ camera\ frame $$

$$ \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \text{ where } \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \; are\ the\ normalized\ image\ coordinates $$

Note: in the code $\;{}^{C}P_W$ is represented by `position_cam`. When we expand $\;{}^{C}R_W$ and simplify as a system of linear equations we get.

$$ \begin{pmatrix} r_{11} & r_{12} & -x \\ r_{21} & r_{22} & -y \\ r_{31} & r_{32} & -1 \end{pmatrix} \begin{pmatrix} x_w \\ y_w \\ \lambda \end{pmatrix} = -{}^{C}\boldsymbol{P}_W $$

$$ \begin{pmatrix} x_w \\ y_w \\ \lambda \end{pmatrix} = -G^{-1} \quad {}^{C}\boldsymbol{P}_W \quad where\ G = \begin{pmatrix} r_{11} & r_{12} & -x \\ r_{21} & r_{22} & -y \\ r_{31} & r_{32} & -1 \end{pmatrix} $$

As λ is Z_n of the corner we can now compute the H matrix for all the corners in the (t-1) frame, vertically concatenate them, and using the optical flow $\dot{\mathbf{p}}$, the velocity $\boldsymbol{V}$ in the equation $\begin{pmatrix} \mathbf{V} \\ \mathbf{\Omega} \end{pmatrix} = pinv(\boldsymbol{H})\ \dot{\mathbf{p}}$ can be calculated. Where pinv() is the pseudoinverse matrix function.

In the code, a for loop from 1 to length(matched points) is used to implement the steps above. This gives us the linear and angular velocity $\boldsymbol{V}$ of the camera expressed in the camera frame. To find the linear and angular velocity expressed in the world frame we can use:

$$ {}^{W}\boldsymbol{V} \quad = \begin{bmatrix} {}^{W}\boldsymbol{R}_C & 0 \\ 0 & {}^{W}\boldsymbol{R}_C \end{bmatrix} \quad {}^{C}\boldsymbol{V} $$

Thus, we have calculated the linear and angular velocity of the MAV between two frames. The steps must be repeated between all the images collected in the dataset.

The final velocity estimatedVel has a bit of a jitter so it was passed through a lowpass() filter with sampling rate 1000 and fs_pass 1 to smooth out the graph.

7. RANSAC Based Outlier Rejection

To improve the results from the previous phase, a flag in the parameter section, ransac_flag can be changed to true. This implements the RANSAC based 3 point outlier rejection in the code. When the flag is set to true, the function velocityRANSAC is called.

The inputs to this function are the optical flow of the matched corners optV, the normalized image coordinates of the matched corners in frame (t-1) optPos, the depth of the corners in the camera frame Z, ${}^{C}\boldsymbol{R}_W$ and the RANSAC hyper parameter ϵ. The output of the function is the (6,1) velocity vector $\boldsymbol{V}$ in the camera frame.

First the number of iterations of finding inliers must be determined using the equation:

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

$$assume\ p_{\text{success}} = 0.99$$

$$M\ is\ the\ minumun\ number\ of\ points\ required\ which\ is\ 3$$

The outermost loop (for i=1 to k) is to create a table of inlier indexes `inlier_table` of the size`(k,length(optPos)+1)`. To do this, three random integers between 1 to length optPos are generated. Using these three integers three corners are extracted from optPos and their corresponding optical flow are extracted from optV. The velocity vector $\begin{pmatrix}\mathbf{V}\\ \mathbf{\Omega}\end{pmatrix}$ is then calculated using the same algorithm as section 6 for these three corners.

Then we check if for some point $\mathbf{p}_j\ in\ optPos\ and\ optV, abs()\ of\ \left(\frac{1}{Z_i}A(\mathbf{p}_j)B(\mathbf{p}_j)\right)\begin{pmatrix}\mathbf{V}\\ \mathbf{\Omega}\end{pmatrix} - \dot{\mathbf{p}}_j$ is lesser than some threshold value `threshold_trail`. This is done for all the points in optPos.

If the condition is met, then the index j is stored in the row i of the inlier table. The last column of the inlier table stores the number of inliers found in each iteration.

The row with the largest inliers is chosen for recomputing the (6,1) velocity vector $\boldsymbol{V}$**.** The algorithm for computing $\boldsymbol{V}$ is the same as discussed in section 6 but we only use the indexes of the inliers in optPos and optV.

The output of velocityRANSAC must also be expressed in the world frame as discussed in section 6.

8. Discussions

The pose graphs follow the ground truth well for data set 1, but for data set 4 as we have lesser data points, the graph is a bit inaccurate.

The velocity and angular velocity track better with RANSAC on, as the outlier rejection algorithm removes errors and jitters in estimatedVel, and again as data set 4 has lesser datapoints than data set 1 the estimate pose gives errors which causes errors in estimatedVel thus data set 4 tracks worse than data set 1. The timestamp of error in pose estimation almost match those in velocity and angular velocity estimation.

9. Output Graphs

Part 1

Figure 4 Data set 1

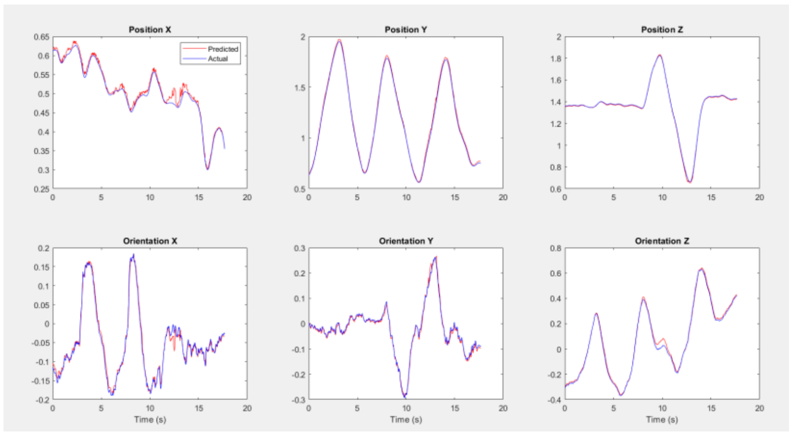
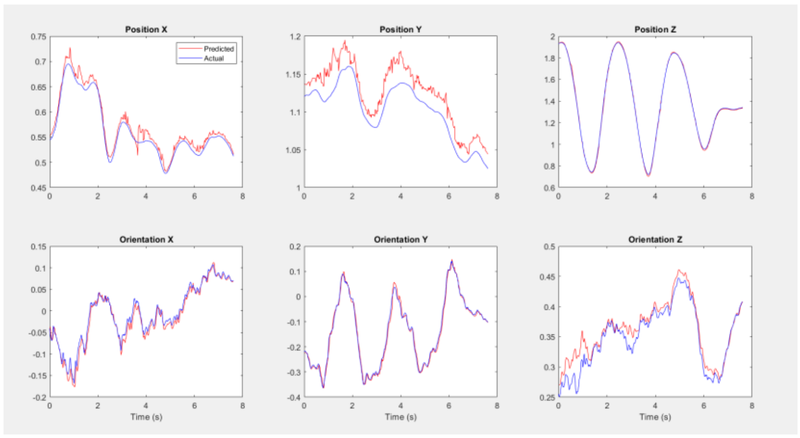


Figure 5 Data set 4



Part 2 Without RANSAC

Figure 6 Data set 1

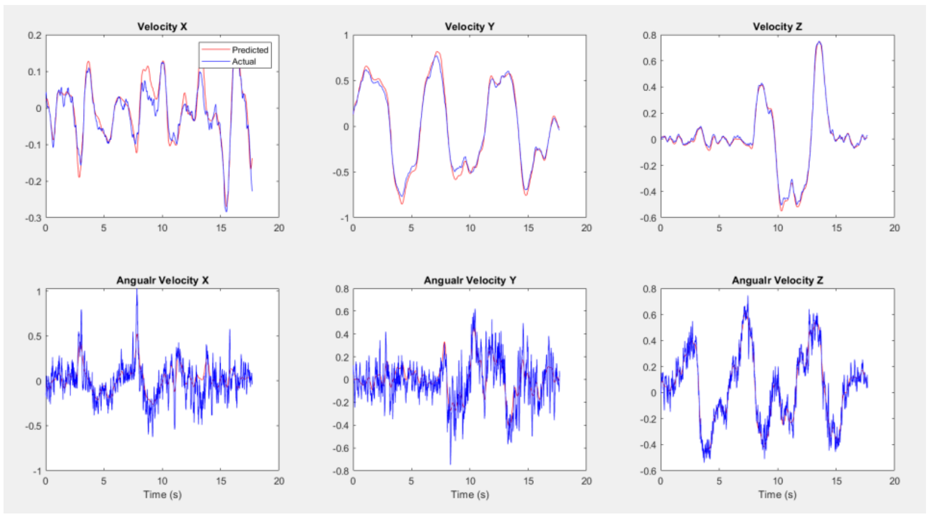
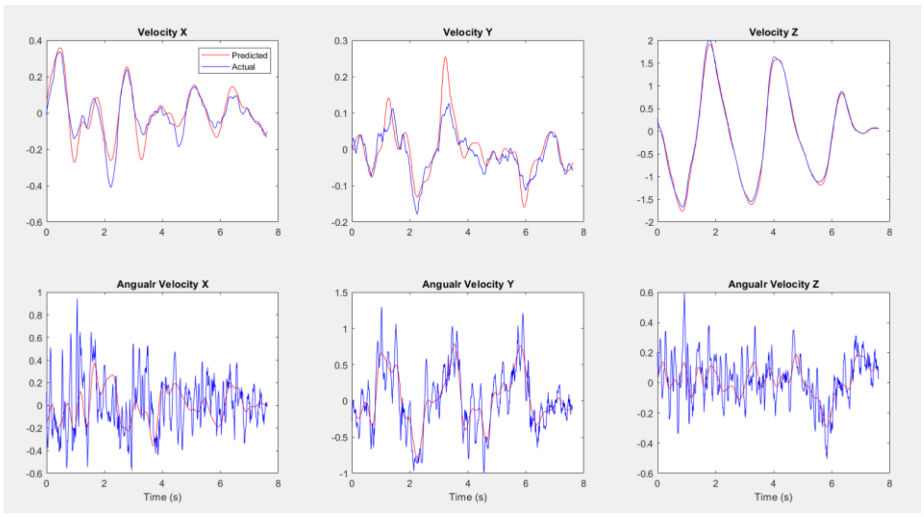


Figure 7 Data set 4



With RANSAC

Figure 8 Data set 1

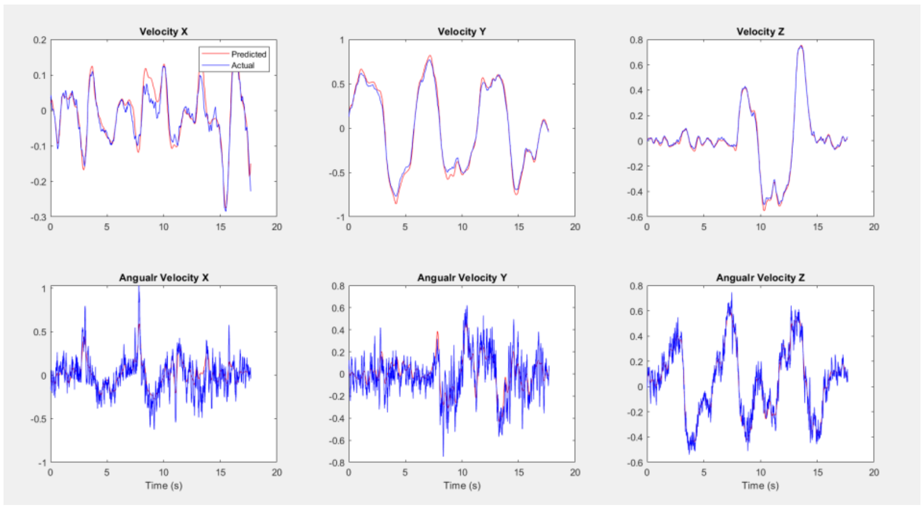
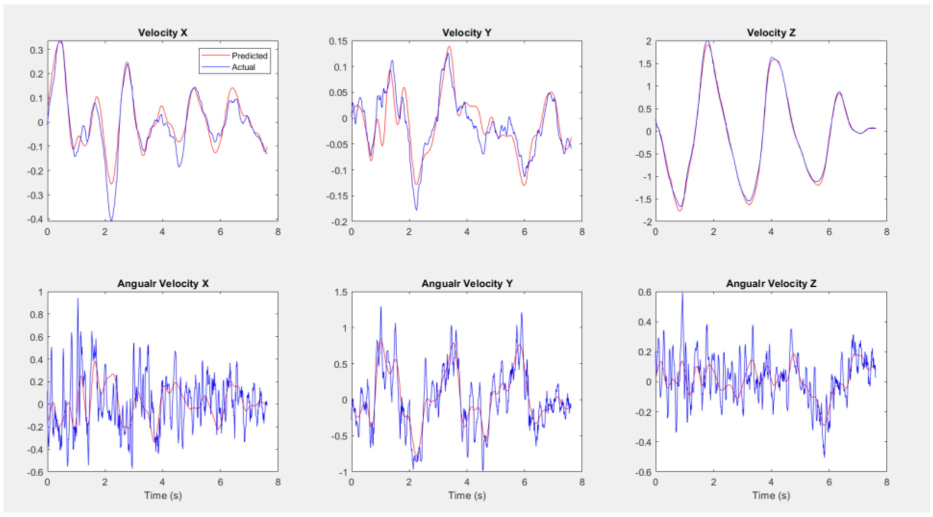


Figure 9 Data set 4


Velocity X
Velocity Y
Velocity Z
Predicted
Actual
Angualr Velocity X
Angualr Velocity Y
Angualr Velocity Z
Time (s)