Project2 Part1

To Find: - Position and Orientation (Pose estimate) of the Body with respective World Frame.

Given: -

- The layout of the April Tag mat. The tags are arranged in a 12 x 9 grid. The top left corner of the top left tag should be 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 0.152m square with 0.152m between tags with the exception of the space between columns 3 and 4, and 6 and 7, which is 0.178m.
- The intrinsic camera calibration matrix and the transformation between the camera and the robot Centre. Two photos (top_view.jpg and side_view.jpg) are included to visualize the camera-robot transformation.
- The data for each trial is provided in a mat file. The file contains a struct array of image data called data, which holds all of the data necessary to do pose estimation.

Approach to the results of part1 of the project2

- Firstly, Corners(p0,p1,p2,p3,p4) of all the April Tags are found using the given layout of the April Tag mat that is arranged in a 12 x 9 grid.

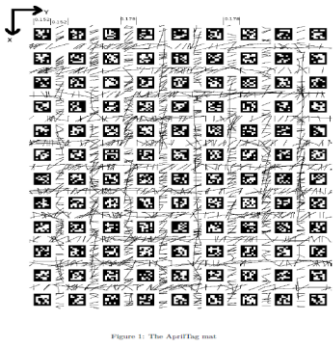


Figure 1: The AprilTag mat

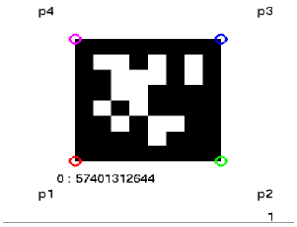


Figure 2: Corners of the AprilTag

(a) April Tag Mat (12 x 9) *(b) Corners of the April Tag*

i. *My MATLAB approach to finding corners of each and every April Tag is, first I found the p4 corners of all the April Tags in the first column.*

```
x=0;
fptlx=zeros(1,12);
for i=0:11
    if i==0
        fptlx(i+1)=x;
    else
        x=x+2*0.152;
        fptlx(i+1)=x;
    end
end
```

© p4 corners of all the April Tags in the first column

ii. *Using the values of p4, I found all the corners of April tags by going in a column-wise fashion.*

```
fptly=zeros(size(fptlx));
fpblx=0.152+fptlx;
fpbly=fptly;
fptrx=fptlx;
fptry=0.152+fptly;
fpbrx=fpblx;
fpbry=0.152+fpbly;
fptl=cat(1,fptlx,fptly);
fptr=cat(1,fptrx,fptry);
fpbl=cat(1,fpblx,fpbly);
fpbr=cat(1,fpbrx,fpbry);
sptl=cat(1,fptlx,2*0.152+fptly);
sptr=cat(1,fptlx,3*0.152+fptly);
spbl=cat(1,fpblx,2*0.152+fpbly);
spbr=cat(1,fpblx,3*0.152+fpbly);
tptl=cat(1,fptlx,4*0.152+fptly);
tptr=cat(1,fptlx,5*0.152+fptly);
tpbl=cat(1,fpblx,4*0.152+fpbly);
tpbr=cat(1,fpblx,5*0.152+fpbly);
frptl=cat(1,fptlx,5*0.152+fptly+0.178);
frptr=cat(1,fptlx,6*0.152+fptly+0.178);
frpbl=cat(1,fpblx,5*0.152+fpbly+0.178);
frpbr=cat(1,fpblx,6*0.152+fpbly+0.178);
fiptl=cat(1,fptlx,7*0.152+fptly+0.178);
fiptr=cat(1,fptrx,8*0.152+fptly+0.178);
fipbl=cat(1,fpblx,7*0.152+fpbly+0.178);
fipbr=cat(1,fpblx,8*0.152+fpbly+0.178);
siptl=cat(1,fptlx,9*0.152+fptly+0.178);
siptr=cat(1,fptlx,10*0.152+fptly+0.178);
sipbl=cat(1,fpblx,9*0.152+fpbly+0.178);
sipbr=cat(1,fpblx,10*0.152+fpbly+0.178);
septl=cat(1,fptlx,10*0.152+fptly+0.178+0.178);
septr=cat(1,fptlx,11*0.152+fptly+0.178+0.178);
sepbl=cat(1,fpblx,10*0.152+fpbly+0.178+0.178);
sepbr=cat(1,fpblx,11*0.152+fpbly+0.178+0.178);
eptl=cat(1,fptlx,12*0.152+fptly+0.178+0.178);
eptr=cat(1,fptlx,13*0.152+fptly+0.178+0.178);
epbl=cat(1,fpblx,12*0.152+fpbly+0.178+0.178);
epbr=cat(1,fpblx,13*0.152+fpbly+0.178+0.178);
nptl=cat(1,fptlx,14*0.152+fptly+0.178+0.178);
nptr=cat(1,fptlx,15*0.152+fptly+0.178+0.178);
npbl=cat(1,fpblx,14*0.152+fpbly+0.178+0.178);
npbr=cat(1,fpblx,15*0.152+fpbly+0.178+0.178);
pfourx=zeros(length(id),1);
pfoury=zeros(length(id),1);
pthreex=zeros(length(id),1);
pthreey=zeros(length(id),1);
ptwox=zeros(length(id),1);
ptwoy=zeros(length(id),1);
ponex=zeros(length(id),1);
poney=zeros(length(id),1);
```

(d) p4, p3, p2, p1, p0 of all the values of April Tags.

iii. After finding out all the corners of the April Tag, I am using a for loop to iterate into a list of id's present inside the data and find corners of each and every id.

```
for j=1:length(id)
    [r,c]=find(Tag==id(j));
    if c==1
        pfourx(j,1)=fptl(1,r);
        pfoury(j,1)=fptl(2,r);
        pthreex(j,1)=fptr(1,r);
        pthreey(j,1)=fptr(2,r);
        ptwox(j,1)=fpbr(1,r);
        ptwoy(j,1)=fpbr(2,r);
        ponex(j,1)=fpbl(1,r);
        poney(j,1)=fpbl(2,r);
    elseif c==2
        pfourx(j,1)=sptl(1,r);
        pfoury(j,1)=sptl(2,r);
        pthreex(j,1)=sptr(1,r);
        pthreey(j,1)=sptr(2,r);
        ptwox(j,1)=spbr(1,r);
        ptwoy(j,1)=spbr(2,r);
        ponex(j,1)=spbl(1,r);
        poney(j,1)=spbl(2,r);
    elseif c==3
        pfourx(j,1)=tptl(1,r);
        pfoury(j,1)=tptl(2,r);
        pthreex(j,1)=tptr(1,r);
        pthreey(j,1)=tptr(2,r);
        ptwox(j,1)=tpbr(1,r);
        ptwoy(j,1)=tpbr(2,r);
        ponex(j,1)=tpbl(1,r);
```

(e) Iterating into the list of IDs present inside the data.

- After finding the corners of all the ids in the getCorners function. The first step in the process of the estimatePose function is to import the image corners present in the data and call the get corners function to get corners of all the detected ids in the AprilTags.

```
function [position, orientation] = estimatePose(data, t)
%% CHANGE THE NAME OF THE FUNCTION TO estimatePose
% Please note that the coordinates for each corner of each AprilTag are
% defined in the world frame, as per the information provided in the
% handout. Ideally a call to the function getCorner with ids of all the
% detected AprilTags should be made. This function should return the X and
% Y coordinate of each corner, or each corner and the centre, of all the
% detected AprilTags in the image. You can implement that anyway you want
% as long as the correct output is received. A call to that function
% should made from this function.
id=data(t).id;
res=getCorner(id);
pimagezero=data(t).p0;
pimageone=data(t).p1;
pimagetwo=data(t).p2;
pimagethree=data(t).p3;
pimagefour=data(t).p4;
```

(f) Importing image corners from data

iv. So I am calculating the A matrix to find the parameters of the projective transformation (h), representing the transformation of the planar object in the image

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

Vector of unknown transformation parameters

(*g) For each point we have two lines of equation*

Since I am taking into account all the four corners and center (p0,p1,p2,p3,p4), so there will be ten lines of equations.

```
A=[];
for f=1:length(id)
    a=[res(f,1),res(f,2),1,0,0,0,-pimagezero(1,f)*res(f,1),-pimagezero(1,f)*res(f,2),-pimagezero(1,f);
        0,0,0,res(f,1),res(f,2),1,-pimagezero(2,f)*res(f,1),-pimagezero(2,f)*res(f,2),-pimagezero(2,f);

        res(length(id)+f,1),res(length(id)+f,2),1,0,0,0,-pimageone(1,f)*res(length(id)+f,1),-pimageone(1,f)*res(length(id)+f,2),-pimageone(1,f);
        0,0,0,res(length(id)+f,1),res(length(id)+f,2),1,-pimageone(2,f)* res(length(id)+f,1),-pimageone(2,f)*res(length(id)+f,2),-pimageone(2,f);

        res(2*length(id)+f,1),res(2*length(id)+f,2),1,0,0,0,-pimagetwo(1,f)*res(2*length(id)+f,1),-pimagetwo(1,f)*res(2*length(id)+f,2),-pimagetwo(1,f);
        0,0,0,res(2*length(id)+f,1),res(2*length(id)+f,2),1,-pimagetwo(2,f)*res(2*length(id)+f,1),-pimagetwo(2,f)*res(2*length(id)+f,2),-pimagetwo(2,f);

        res(3*length(id)+f,1),res(3*length(id)+f,2),1,0,0,0,-pimagethree(1,f)*res(3*length(id)+f,1),-pimagethree(1,f)*res(3*length(id)+f,2),-pimagethree(1,f);
        0,0,0,res(3*length(id)+f,1),res(3*length(id)+f,2),1,-pimagethree(2,f)*res(3*length(id)+f,1),-pimagethree(2,f)*res(3*length(id)+f,2),-pimagethree(2,f);

        res(4*length(id)+f,1),res(4*length(id)+f,2),1,0,0,0,-pimagefour(1,f)*res(4*length(id)+f,1),-pimagefour(1,f)*res(4*length(id)+f,2),-pimagefour(1,f);
        0,0,0,res(4*length(id)+f,1),res(4*length(id)+f,2),1,-pimagefour(2,f)*res(4*length(id)+f,1),-pimagefour(2,f)*res(4*length(id)+f,2),-pimagefour(2,f)];
    A=[A;a];
end
```

(h)For four corner points and the center with a total of ten lines of equation.

v. After computing the A matrix, the H matrix is obtained by taking the Singular-value Decomposition, and transposing the 9th column of V.

```
[u,s,v]=svd(A);
h=reshape(v(:,9),[3,3]);
H=transpose(h)/h(3,3);
```

(i) finding the projective transformation.

vi. Computing the pose(R, T) from the projective transformation

Now we need to extract the rotation and translation from the remainder

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

```
Camera_Matrix=[311.0520,0,201.8724;0,311.3885,113.6210;0,0,1];
rt=pinv(Camera_Matrix)*H;
rone=rt(:,1);
rtwo=rt(:,2);
t=rt(:,3);
```

(j) Obtaining pose upon multiplying the inverse of camera calibration matrix with projective transformation.

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

$$R = U \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(UV^T) \end{pmatrix} V^T \qquad T = \hat{T} / \|\hat{R}_1\|$$

```
rcot=cross(rone,rtwo);
[uo,so,vo]=svd([rone,rtwo,rcot]);
rcw=uo*[1,0,0;0,1,0;0,0,det(uo*transpose(vo))]*transpose(vo);
tcw=t/norm(rone);
Hcw=[rcw,tcw;0,0,0,1];
```

(k) Finding the position and orientation of the World frame with the respective Camera Frame.

vii. Using the given side_view.img, top_view.img and translation from parameters.txt is used to find the rotation and orientation of the camera wthe ith respective body.

```
rcb=[1.414/2,-1.414/2,0;-1.414/2,-1.414/2,0;0,0,-1];
tcb=[0.04;0.0;-0.03];
Hcb=[rcb,tcb;0,0,0,1];
```

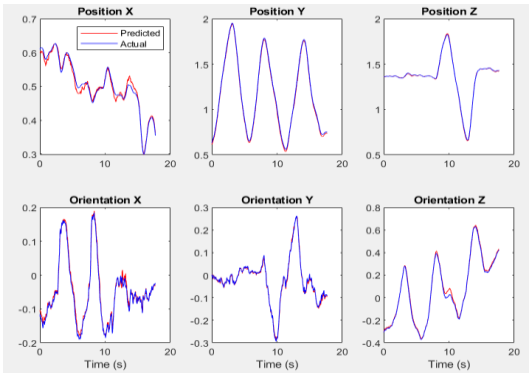
(l) Rotation and Orientation of Camera with the respective body.

viii. Calculating position and orientation of body frame with respective world frame.

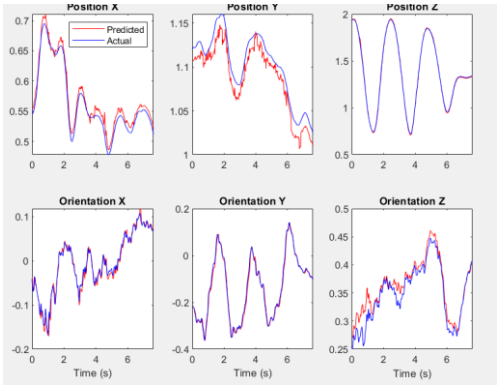
```
hbw=Hcb*Hcw;
Hbw=pinv(hbw);
rbw=Hbw(1:3,1:3);
tbw=Hbw(1:3,4);
oa=rotm2eul(rbw,'ZYX');
position=transpose(tbw);
orientation=oa;
```

(m) Position and Orientation of body frame with respective world frame

Results: -



(n) position and orientation using dataset 1



(o) position and orientation using dataset 4

Project 2 Part 2.1

To Find: - Linear and Angular Velocity of Camera with respective World Frame.

Given: -

- The layout of the April Tag mat. The tags are arranged in a 12 x 9 grid. The top left corner of the top left tag should be 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 0.152m square with 0.152m between tags with the exception of the space between columns 3 and 4, and 6 and 7, which is 0.178m.

- The intrinsic camera calibration matrix and the transformation between the camera and the robot Centre. Two photos (top_view.jpg and side_view.jpg) are included to visualize the camera-robot transformation.

- The data for each trial is provided in a mat file. The file contains a struct array of image data called data, which holds all of the data necessary to do pose estimation.

Approach to the results of part2.1 of the project2

- Similarly, to the previous, after solving for the getCorners function and finding Rc2w in the estimatePose function.
- Initially, the process in the OpticalFlow function begins by loading images into respective variables from sampleData.

```
%% INITIALIZE CAMERA MATRIX AND OTHER NEEDED INFORMATION
Camera_Matrix=[311.0520,0,201.8724;0,311.3885,113.6210;0,0,1];
for n = 2:length(sampledData)
    %% Initalize Loop load images
    prev_images=sampledData(n-1).img;
    curr_images=sampledData(n).img;
```

Initializing images by importing from sampledData

- After loading the images, I detected good points using one of the functions(detectFASTFeatures) from the computer vision toolbox to eliminate the corners that lie outside the image boundary.
- Detected points are used to compare the previous image and current image to generate optimal points.
- The point tracker object tracks a set of points using the Kanade-Lucas-Tomasi (KLT), feature-tracking algorithm. It also works particularly well for tracking objects that do not change shape and for those that exhibit visual texture. That's the primary reason for using vision.PointTracker() with one parameter as MaxBidirectional Error.

```
%% Detect good points
prev=detectFASTFeatures(prev_images);
%[prevFeatures, prevPoints] = extractFeatures(prev_images, prev);
prevPoints=prev.selectStrongest(50);
%% Initalize the tracker to the last frame.
tracker=vision.PointTracker('MaxBidirectionalError',1);
initialize(tracker,prevPoints.Location,prev_images);
```

- After initializing a tracker and finding the location of the next image points using a tracker, the velocity of the camera using motion field equations is found.

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

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

```
%% Calculate velocity
% Use a for loop
CoordPrevpoints=[];
CoordCurrpoints=[];
VELOCITY=[];
U=[];
V=[];

for m=1:length(prevPoints)
    dt=sampledData(n).t-sampledData(n-1).t;
    NormalisedPrevxyz=[prevPoints.Location(m,1);prevPoints.Location(m,2);1];
    FCPrev=pinv(Camera_Matrix)*NormalisedPrevxyz;
    CoordPrevpoints(m,1)=FCPrev(1,1);
    CoordPrevpoints(m,2)=FCPrev(2,1);
    NormalisedCurrxyz=[currPoints(m,1);currPoints(m,2);1];
    FCCurr=pinv(Camera_Matrix)*NormalisedCurrxyz;
    CoordCurrpoints(m,1)=FCCurr(1,1);
    CoordCurrpoints(m,2)=FCCurr(2,1);
    diffx=CoordCurrpoints(m,1)-CoordPrevpoints(m,1);
    diffy=CoordCurrpoints(m,2)-CoordPrevpoints(m,2);
    U(m)=diffx/dt;
    V(m)=diffy/dt;
    VELOCITY=[VELOCITY;U(m);V(m)];
end
```

Motion Field Equations

- The height of the body at different heights is calculated

```
%% Calculate Height
[position, orientation, R_c2w] = estimatePose(sampledData, n);
rbwn=eul2rotm(orientation);
Hbwn=[rbwn,transpose(position);0,0,0,1];
rcbn=[1.414/2,-1.414/2,0;-1.414/2,-1.414/2,0;0,0,-1];
tcbn=[0.04;0.0;-0.03];
Hcbn=[rcbn,tcbn;0,0,0,1];
Hcwn=pinv(Hbwn*Hcbn);
Z=Hcwn(3,4);
functionxyz=[];
for l=1:length(currPoints)
    tempfunction=[-1/Z,0,CoordCurrpoints(l,1)/Z,CoordCurrpoints(l,1)*CoordCurrpoints(l,2),-(1+CoordCurrpoints(l,1)^2),Coo
    0,-1/Z,CoordCurrpoints(l,2)/Z,(1+CoordCurrpoints(l,2)^2),-CoordCurrpoints(l,1)*CoordCurrpoints(l,2),-CoordCurrpoints(
    functionxyz=[functionxyz;tempfunction];
end
```

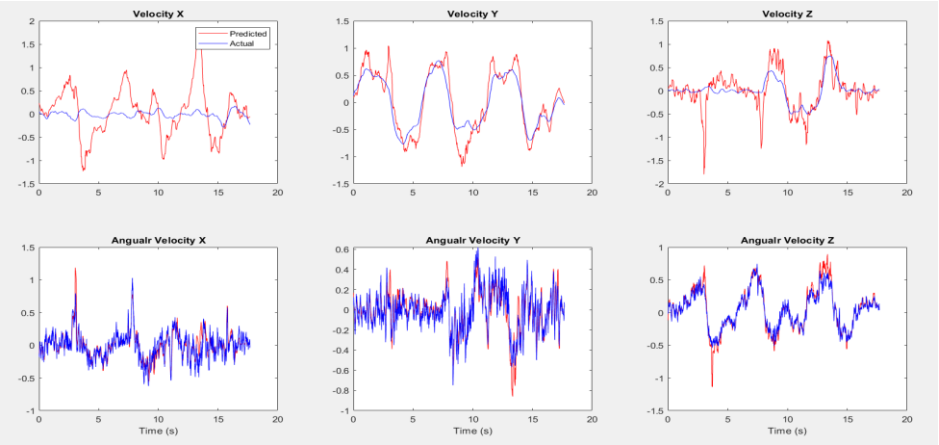
- The results (linear velocity) is obtained by

```
%% Fix the linear velocity
% Change the frame of the computed velocity to world frame
velocity=pinv(functionxyz)*VELOCITY;
Hcwn=pinv(Hcwn);
Vi=[Hcwn(1:3,1:3),-Hcwn(1:3,1:3)*skew(Hcwn(1:3,4));zeros(3),Hcwn(1:3,1:3)]*velocity;
estimatedV(:,n)=Vi;
```

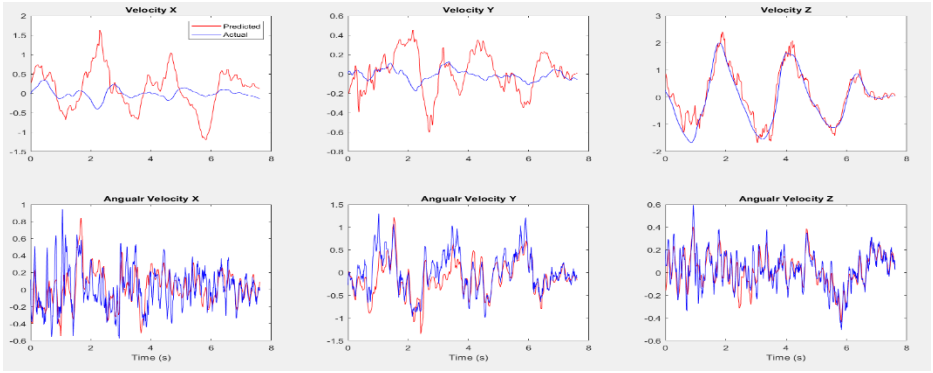
- To reduce a bit of jitter which adds noise to the time measurement, we are using a low pass filter by assuming You can assume that the images were taken approximately at a constant rate and simply use a low-pass filter on the d*t* to get better results.

```
end
    time_vector= zeros(length(sampledData), 1);
        for tv=1:length(time_vector)
            time_vector(tv,1) = sampledData(1,tv).t;
        end
    time_vector = sgolayfilt(time_vector, 1, 101);

    vx = transpose(estimatedV(1,:));
    vy = transpose(estimatedV(2,:));
    vz = transpose(estimatedV(3,:));

    wx = transpose(estimatedV(4,:));
    wy = transpose(estimatedV(5,:));
    wz = transpose(estimatedV(6,:));


    estimatedV(1,:) = sgolayfilt(vx, 1, 17);
    estimatedV(2,:) = sgolayfilt(vy, 1, 17);
    estimatedV(3,:) = sgolayfilt(vz, 1, 7);
    estimatedV(4,:) = sgolayfilt(wx, 1, 5);
    estimatedV(5,:) = sgolayfilt(wy, 1, 5);
    estimatedV(6,:) = sgolayfilt(wz, 1, 5);
```

Results of Part 2.1 project 2



Dataset1 of Part 2.1 project 2



Dataset4 of Part 2.1 project 2

Project 2 Part 2.2

To Find:-

- In this project 2 of part 2.2, 3-point RANSAC is used to reject the outliers. It is also useful to recompute the velocities using inliers.

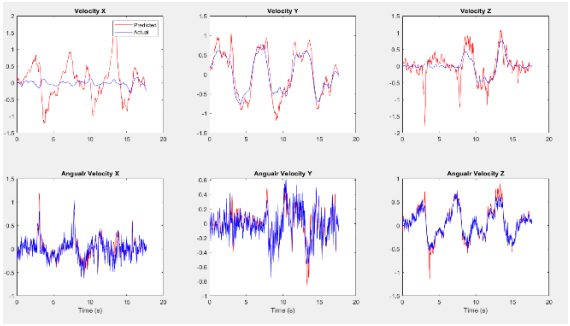
```
p_success=0.99;
Homography_M=4;
e=0.8;
max_no_iterations=log(1-p_success)/log(1-e^Homography_M);
count=0;
for iteration=1:max_no_iterations
    liners=0;
    points=randperm(length(optPos),3);
    point_one=optPos(points(1,1),:);
    point_two=optPos(points(1,2),:);
    point_three=optPos(points(1,3),:);
    h=[-1/Z,0,point_one(1,1)/Z,point_one(1,1)*point_one(1,2),-(1+point_one(1,1)^2),point_one(1,2);
        0,-1/Z,point_one(1,2)/Z,(1+point_one(1,2)^2),-point_one(1,1)*point_one(1,2),-point_one(1,1);
        -1/Z,0,point_two(1,1)/Z,point_two(1,1)*point_two(1,2),-(1+point_two(1,1)^2),point_two(1,2);
        0,-1/Z,point_two(1,2)/Z,(1+point_two(1,2)^2),-point_two(1,1)*point_two(1,2),-point_two(1,1);
        -1/Z,0,point_three(1,1)/Z,point_three(1,1)*point_three(1,2),-(1+point_three(1,1)^2),point_three(1,2);
        0,-1/Z,point_three(1,2)/Z,(1+point_three(1,2)^2),-point_three(1,1)*point_three(1,2),-point_three(1,1)];
    Voptimal=[optV(2*points(1,1)-1);optV(2*points(1,1));optV(2*points(1,2)-1);optV(2*points(1,2));optV(2*points(1,3)-1);op
    Vj=pinv(h)*Voptimal;
    for x=1:length(optPos)
        temp8=[-1/Z,0,optPos(x,1)/Z,optPos(x,1)*optPos(x,2),-(1+optPos(x,1)^2),optPos(x,2);
            0,-1/Z,optPos(x,2)/Z,(1+optPos(x,2)^2),-optPos(x,1)*optPos(x,2),-optPos(x,1)];
        temp9=[optV(2*x-1);optV(2*x)];
        change=norm(temp8*Vj-temp9)^2;
        if(change<=1e-3)
            liners=liners+1;
        end
    end
```

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

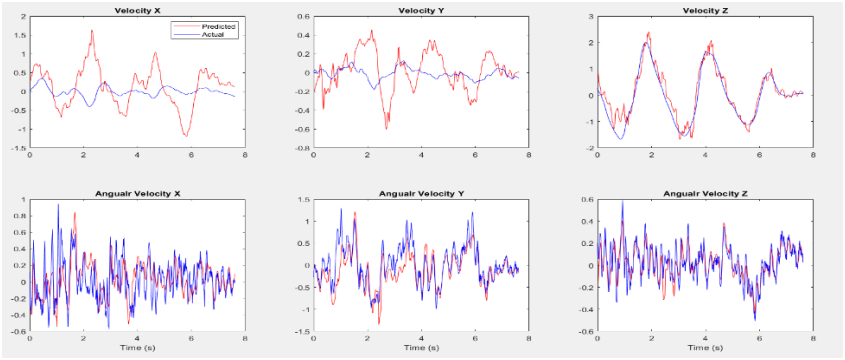
For $p_{success}$ = 0.99 the k needed for various models:

Model	$\epsilon = 0.8$	$\epsilon = 0.5$	$\epsilon = 0.3$	$\epsilon = 0.1$
Line (M=2 points)	5	17	49	459
Homography (M=4 points)	9	72	567	46050
Essential Matrix (M=5 points)	12	146	1893	460515

Results of Part2.2 , project 2



Dataset1, project 2.2, project2



Dataset4, project 2.2, project2