LIDAR AND RADAR
TASK 2: A SIMPLE POSITION PREDICTION APPROACH

Aman Dubey(35066)

February 4, 2022

Abstract

In the first task we calculated the dimensions of our Vehicle that is Length, Breadth and Height. In the Second task we implemented an algorithm to calculate resolution of Blickfield and Velodyne LIDAR sensors, After that we concluded on quality of Sensors. So from the last two task we have a clear picture about our dataset[1]. Now we need to advance further and use our dataset more precisely to explore more dimension in our ADAS system. In this task we are going to use Bounding Box data in order to predict the movement of the vehicle, We used two consecutive data to predict upcoming data i.e. predicting the trajectory of our vehicle in 3-D space.

Keywords
1)ADAS(Advanced driver-assistance systems)
2)LIDAR(Light Detection and Ranging)
3)SVD(Single value Decomposition)

1 Introduction

As we are moving in the era of ADAS system. It is very important to deploy a highly reliable system which can predict future environment in the real time system. This helps in avoiding collision beforehand, Whole research going in the industry is to deploy a highly accurate prediction algorithm via machine learning/Neural Networks. In this task, we have carried out research in predicting the movement of vehicle based on past two positions. We used simple linear straight line regression[2] to predict our point.

2 Methods

The approach is simply based on straight line interpolation. We have used two coordinates to predict the third one, considering Velocity of the vehicle to be constant. Here we have calculated displacement between coordinate 1 and coordinate 2. The same displacement is applied on coordinate 2 to predict coordinate 3(predicted coordinate), It is illustrated in fig 1 for 2D. Similarly, it will be done for 3D.

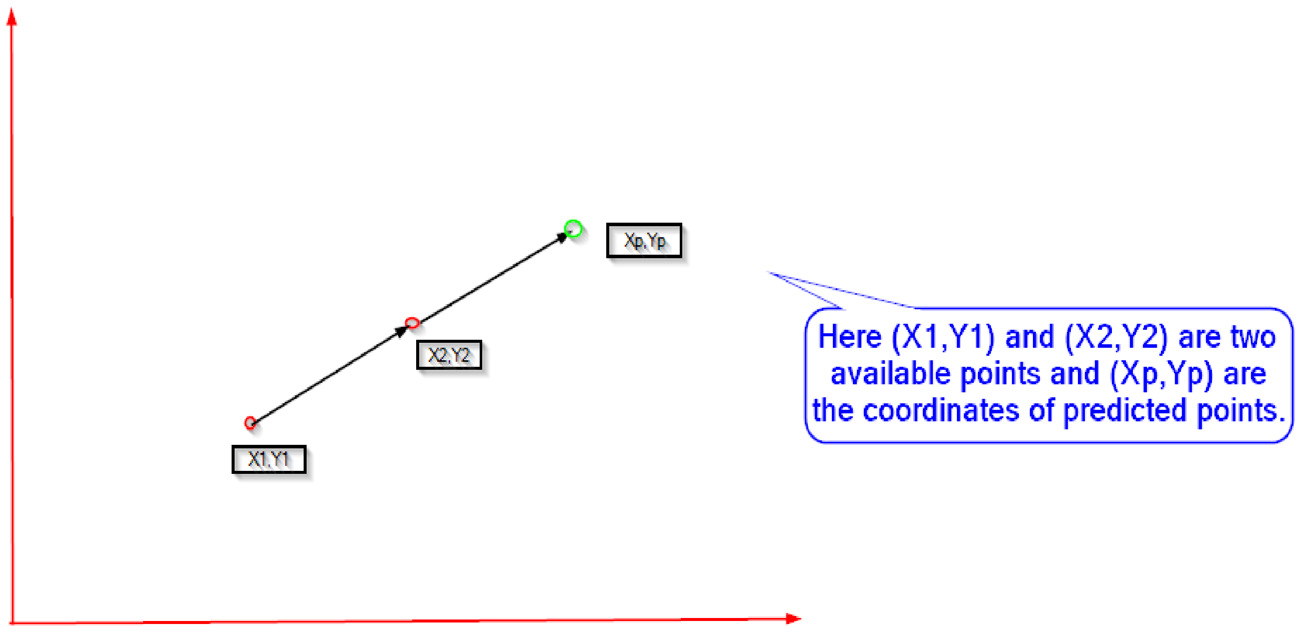

Xp,Yp
X2,Y2
X1,Y1
Here (X1,Y1) and (X2,Y2) are two available points and (Xp,Yp) are the coordinates of predicted points.

Figure 1: Straight line regression in 2D

Now we will try to create a more clear picture by illustrating it via mathematical Equations. Suppose we have two coordinates (x1,y1,z1), (x2,y2,z2) and our predicted point is (xp,yp,zp). Following are the points which explain the method in a lucid way:

1) find direction ratios as follows

$$l = x_2 - x_1, m = y_2 - y_1, n = z_2 - z_1 \tag{1}$$

Here l,m and n represent direction ratio

2) Now take our second point(x2,y2,z2) and write the equation as follows

$$L = \frac{(x - x_2)}{l} = \frac{(y - y_2)}{l} = \frac{(z - z_2)}{l} \tag{2}$$

Here L is a kind of scaling factor, When we consider the Car to be moving with constant velocity this can put to 1. We used L=1 in our solution. Now our new coordinates(xP,yP,zP) would look like

$$x_P = x_2 + (x_2 - x_1) \tag{3}$$

$$y_P = y_2 + (y_2 - y_1) \tag{4}$$

$$z_P = z_2 + (z_2 - z_1) \tag{5}$$

**Please note that we have considered velocity of the object to be constant and our dataset to be uniform in occurrence with regards to time. Time difference between each frame is same

We also need to look at Yaw angle("ϕ") as it is also varying, Similarly we can write for yaw angle as well.

$\phi_p = \phi_2 + (\phi_2 - \phi_1)$

Now we will look into the implementation we did in Python.
Firstly we are going to define 11×11 Matrix as follows

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Now we will create one more matrix which should contain coordinates and dimension of second point and the difference of first and second coordinates. The matrix is as shown below :

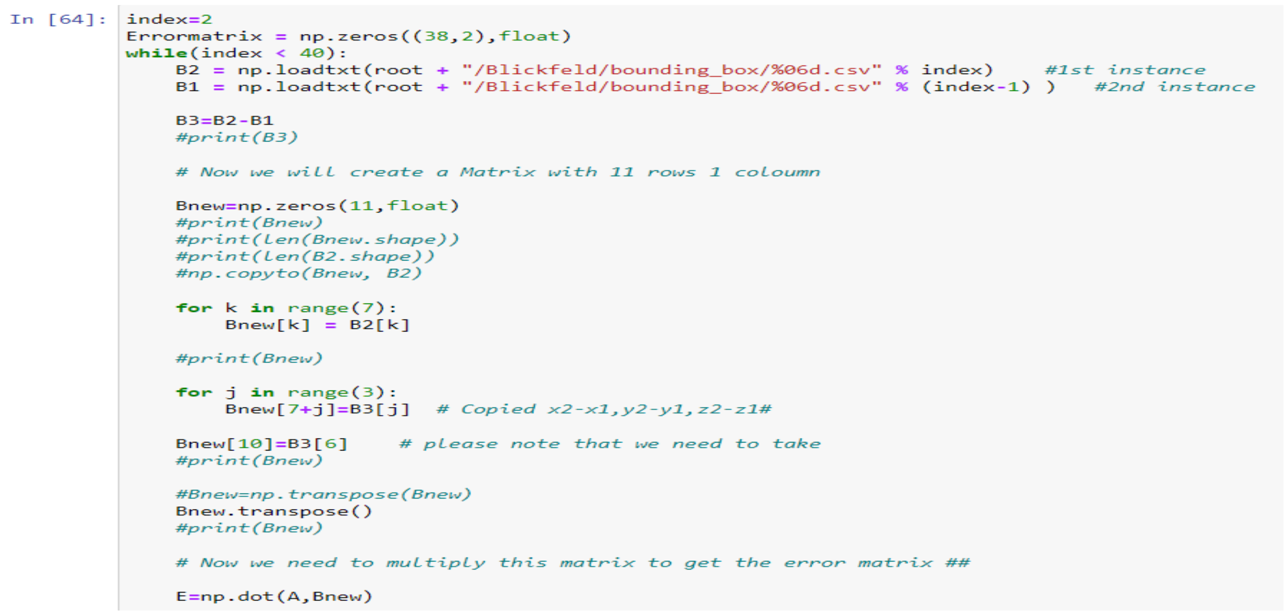
$$B = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ W \\ H \\ L \\ \phi_2 \\ x_2 - X_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ \phi_2 - \phi_1 \end{bmatrix}$$

We will multiply A and B After multiplying the matrix we will get the matrix as follows

$$\text{Pred} = \begin{bmatrix} x_2 + (x_2 - x_1) \\ y_2 + (y_2 - y_1) \\ z_2 + (z_2 - z_1) \\ W \\ H \\ L \\ \phi_2 + (\phi_2 \text{ - } \phi_1) \\ x_2 - X_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ \phi_2 \text{ - } \phi_1 \end{bmatrix}$$

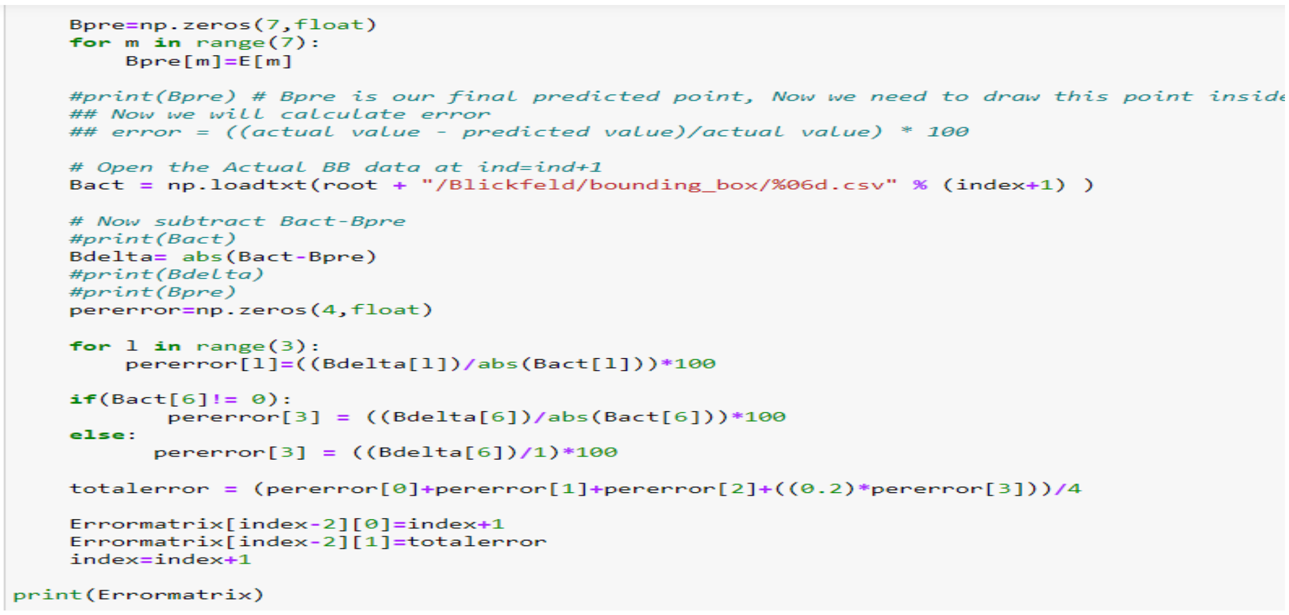
Finally we are going to take out first 7 values of the pre matrix and place them into new matrix, It was done in order to make our predicted system matrix similar to the Bounding Box data(7×1).

$$\text{Final Predicted} = \begin{bmatrix} x_2 + (x_2 - x_1) \\ y_2 + (y_2 - y_1) \\ z_2 + (z_2 - z_1) \\ W \\ H \\ L \\ \phi_2 + (\phi_2 \text{ - } \phi_1) \end{bmatrix}$$



```
In [64]: index=2
         Errormatrix = np.zeros((38,2),float)
         while(index < 40):
             B2 = np.loadtxt(root + "/Blickfeld/bounding_box/%06d.csv" % index)     #1st instance
             B1 = np.loadtxt(root + "/Blickfeld/bounding_box/%06d.csv" % (index-1) )    #2nd instance

             B3=B2-B1
             #print(B3)

             # Now we will create a Matrix with 11 rows 1 coloumn

             Bnew=np.zeros(11,float)
             #print(Bnew)
             #print(len(Bnew.shape))
             #print(len(B2.shape))
             #np.copyto(Bnew, B2)

             for k in range(7):
                 Bnew[k] = B2[k]

             #print(Bnew)

             for j in range(3):
                 Bnew[7+j]=B3[j]  # Copied x2-x1,y2-y1,z2-z1#

             Bnew[10]=B3[6]     # please note that we need to take
             #print(Bnew)

             #Bnew=np.transpose(Bnew)
             Bnew.transpose()
             #print(Bnew)

             # Now we need to multiply this matrix to get the error matrix ##

             E=np.dot(A,Bnew)
```

Figure 2: Full Implementation (1)



```
    Bpre=np.zeros(7,float)
    for m in range(7):
        Bpre[m]=E[m]

    #print(Bpre) # Bpre is our final predicted point, Now we need to draw this point inside
    ## Now we will calculate error
    ## error = ((actual value - predicted value)/actual value) * 100

    # Open the Actual BB data at ind=ind+1
    Bact = np.loadtxt(root + "/Blickfeld/bounding_box/%06d.csv" % (index+1) )

    # Now subtract Bact-Bpre
    #print(Bact)
    Bdelta= abs(Bact-Bpre)
    #print(Bdelta)
    #print(Bpre)
    pererror=np.zeros(4,float)

    for l in range(3):
        pererror[l]=((Bdelta[l])/abs(Bact[l]))*100

    if(Bact[6]!= 0):
          pererror[3] = ((Bdelta[6])/abs(Bact[6]))*100
    else:
         pererror[3] = ((Bdelta[6])/1)*100

    totalerror = (pererror[0]+pererror[1]+pererror[2]+((0.2)*pererror[3]))/4

    Errormatrix[index-2][0]=index+1
    Errormatrix[index-2][1]=totalerror
    index=index+1

print(Errormatrix)
```

Figure 3: Full Implementation (2)

3 Results and Analysis

Now talking about the result it would be more precise to see our result with respect to the error in the prediction algorithm.

ERROR CALCULATION we have actual bounding box and predicted Bounding box data with us.

$$Error = \frac{ActualValue - ComputedValue}{Actualvalue} \times 100 \tag{6}$$

By using above mentioned formulae we have calculated error in x, y, z and ϕ, See Fig ... as shown

* Please note that wherever actual value is zero we have divided our delta(actual value - computed value) by 1.

Now we have error in x, y, z and ϕ coordinates. For calculating final error we have formed a weighted error function[3][4] as shown below

$$WeightedError = \frac{error_x + error_y + error_z + (\frac{error_\phi}{5})}{4} \tag{7}$$

From the above Equation(7) it must be clear that we have receded error contributed by ϕ. Question arises Why we have done this for error in ϕ ?

The answer lies in the values of ϕ, The value of ϕ are so small that a small change inside it causes error percentage to shoot up. This is the reason for dividing error in ϕ by 5.

Now we will look into the graph of error

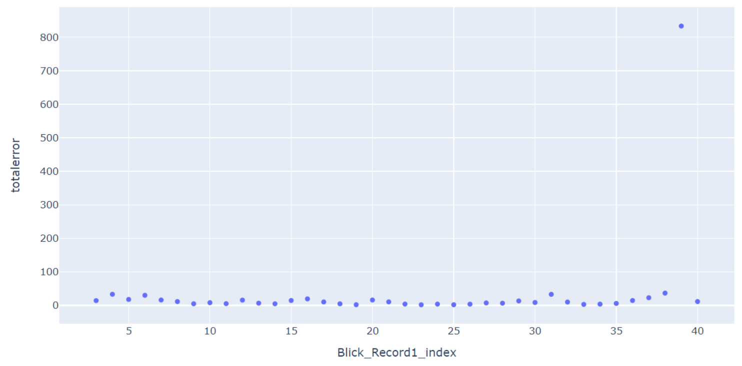


Figure 4: Forward/Backward prediction with Blickfield

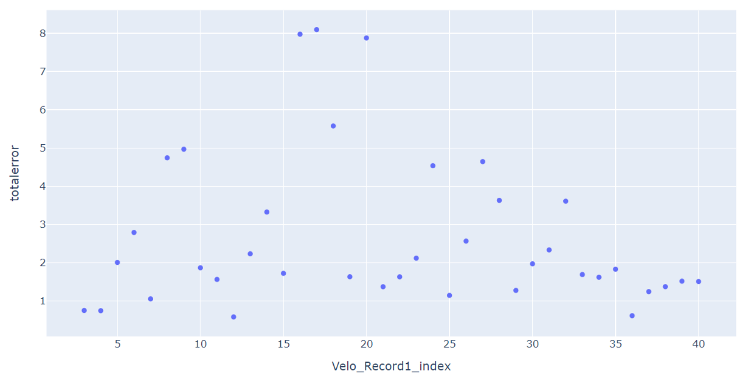


Figure 5: Forward/Backward prediction with Velodyne

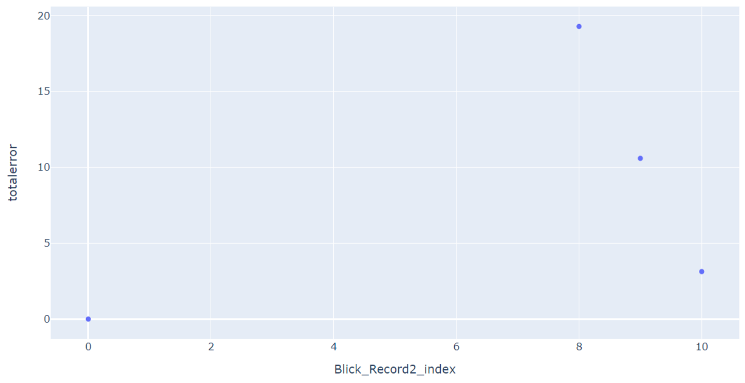


Figure 6: Sideways prediction with Blickfield

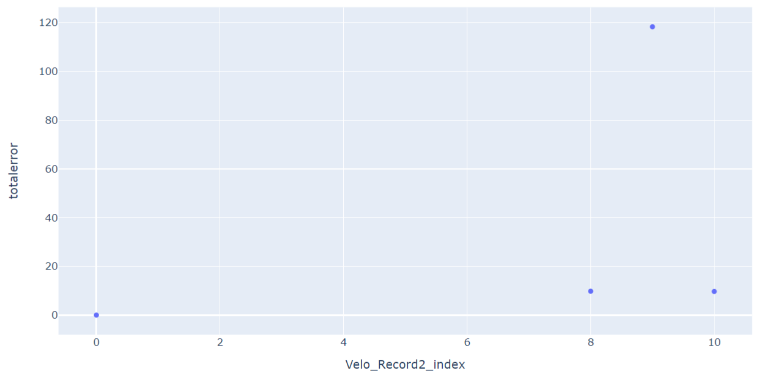


Figure 7: Sideways prediction with Velodyne

From the above graphs we have few observations, We will point out them as follows:

1)In Blickfield sensor for Record1(Forward/Backward), For the majority of indexes we got error less than 10 Percent, which seems very good. But for index 39 our error shoots up to 833 percent Which is not good, But in Reality we should again see the values of coordinates at those points, x coordinate is much lesser than 1 and again for a very small magnitude change we get bigger error(in percent). So although the percent is high, our actual bounding box and predicted one are almost close and this is what we want eventually.

2)In Velodyne sensor for Record1(Forward/Backward), Error seems to be quite well as there are no abrupt changes in the consecutive bounding box data.

3)For Record2 we have only 5 instances where our sensor is able to see the Car, So here we have got three prediction of our vehicle. For both the sensors error seems to be quite well, and we can see very close predicted bounding box to the original one. This was even expected as the object is almost moving in a straight line.

All the Error values are mentioned in below Two tables :

INDEX	ERROR
8	19.27967993
9	10.5915211
10	3.128923589

(a) BLICKFIELD

INDEX	ERROR
8	9.801034504
9	118.364927
10	9.702502212

(b) VELODYNE

Table 1: Error calculation in Sideways

INDEX	ERROR
3	14.34796039
4	33.29854195
5	17.81208185
6	30.17999637
7	16.19219309
8	11.55611395
9	4.929221079
10	8.104922425
11	5.313218743
12	15.84313181
13	6.741580824
14	4.914136469
15	14.66467809
16	19.59669037
17	10.38348792
18	4.759157536
19	2.116987632
20	16.125166
21	10.68027742
22	4.071802771
23	1.976246107
24	3.920374044
25	2.107785736
26	3.732229479
27	7.40705036
28	6.651278363
29	13.33766371
30	8.646412581
31	33.07736908
32	10.08565815
33	3.062309556
34	3.870961158
35	5.920799476
36	14.63854979
37	22.94328304
38	36.71166282
39	833.1652142
40	11.59435198

(a) BLICKFIELD

INDEX	ERROR
3	0.753122595
4	0.747343742
5	2.008441549
6	2.792458168
7	1.05588442
8	4.743066235
9	4.968648865
10	1.869828513
11	1.564409422
12	0.584503886
13	2.234119309
14	3.326087615
15	1.723819365
16	7.974439925
17	8.093976045
18	5.576063359
19	1.63440729
20	7.87671723
21	1.37383488
22	1.632383894
23	2.120759176
24	4.535921274
25	1.146837322
26	2.566101085
27	4.644786579
28	3.631516363
29	1.277578026
30	1.97235103
31	2.336581741
32	3.609364121
33	1.691986446
34	1.622490567
35	1.832259094
36	0.61684411
37	1.246453489
38	1.375401168
39	1.518872564
40	1.510101566

(b) VELODYNE

Table 2: Error calculation in Forward/Backward

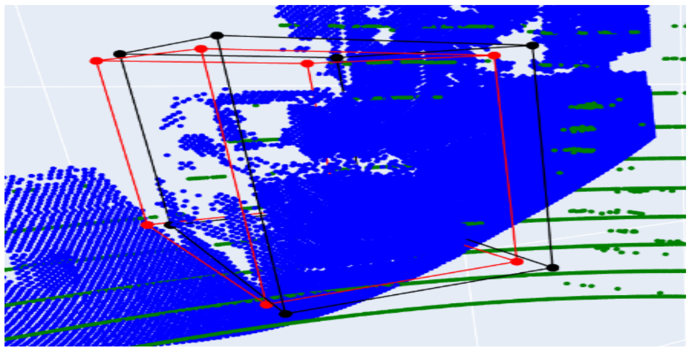


Figure 8: Record 1 Blickfield Bounding Box Predicted(Black) and Actual(Red)

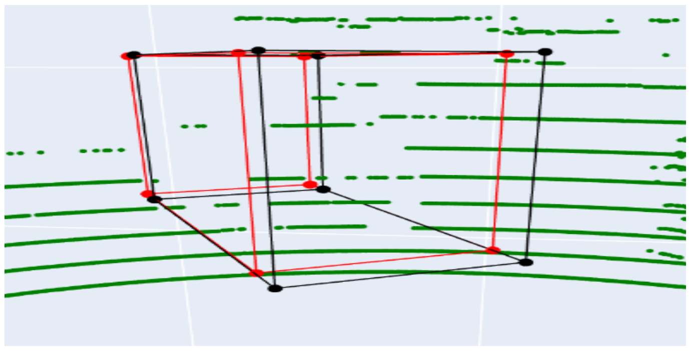


Figure 9: Record 1 Velodyne Bounding Box Predicted(Black) and Actual(Red)

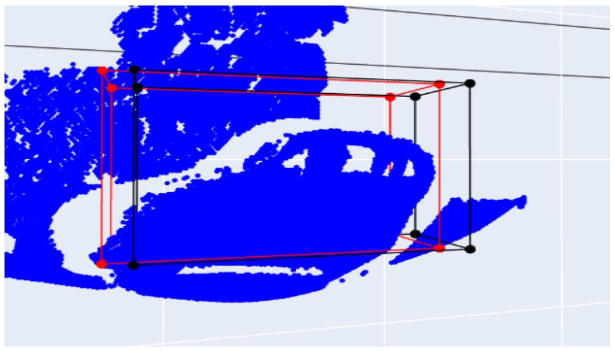


Figure 10: Record 2 Blickfield Bounding Box Predicted(Black) and Actual(Red)

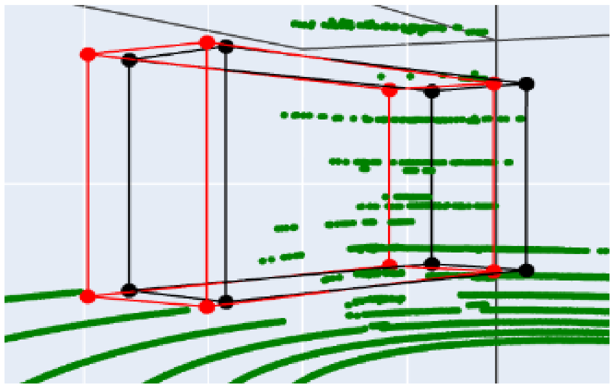


Figure 11: Record 2 Velodyne Bounding Box Predicted(Black) and Actual(Red)

4 Conclusion

This report aimed at predicting the path of the vehicle/Ego by observing past two points. Based on the result observed, We can conclude that this method predict vehicle's movement perfectly for linear movements. The Prediction can be made a little more concrete by observing more than two points, Methods like Spline Interpolation and Single Value Decomposition(SVD). Further research can be done in this approach with a larger dataset by applying various numerical methods.

References

[1] Felix Berens and Saravanan. Lidar dataset at rwu.

[2] John F Kenney and ES Keeping. Linear regression and correlation. *Mathematics of statistics*, 1:252–285, 1962.

[3] Sebastian Türmer. *Car detection in low frame-rate aerial imagery of dense urban areas*. PhD thesis, TU München, Institut für Photogrammetrie und Kartographie, 2014.

[4] Pentti Minkkinen. Weighting error–a potential source of systematic measurement errors in process analysis.