Pedestrian Path Prediction using GPS Data

Author: Divyendu Narayan email: dnarayan@umich.edu
Graduate Student in Electrical Engineering, University of Michigan-Dearborn

This paper presents an approach using Neural Networks to predict pedestrian position using of past movement. Pedestrian movement is recorded using the GPS data. This paper compares the path prediction using Dead Reckoning against path prediction using Neural Network. Since dead reckoning is based on assumption that the pedestrian speed remains constant during the prediction period, prediction error using dead reckoning increases as the prediction time increases. This paper approaches pedestrian path prediction as a curve fitting problem, where by using the previous movement coordinates it predicts the future position. For this Curve Fitting utility of Neural Network toolbox from Matlab is being used [1]. This paper converts the latitude-longitude to cartesian coordinates, predicts path using Neural Network (curve fitting) and then converts predicted cartesian coordinates back to latitude-longitude. Neural Network based prediction works better when a smaller history of previous coordinates is used.

1. Introduction

Numerous approaches have been used to predict pedestrian path. Vasquez and Fraichard divide pedestrian path into segments, then represent these segments using mean and variance. Using clustering techniques these path segments are grouped into clusters. Using Maximum Likelihood Estimator, in runtime a path segment is identified to belong to a cluster and then trajectory is predicted [2]. Choi and Herbert modeled the pedestrian trajectory as concatenation of segments. The transitions between these segments are assumed to follow a Markov Model. Using Hidden Markov Model the future trajectory of pedestrian is predicted [3]. While work of Vasquez et al[2] worked in a structured environment(such as office), Choi et al[3] have demonstrated the working in both structured as well as unstructured environment(inside university campus).

This paper shows Neural Network based pedestrian path prediction trained by using GPS latitude-longitude (recorded in one geographical location) transformed to local x-y coordinates. It can be used for prediction in any geographical location. Section 2 describes the data acquisition setup. It discusses the signals being recorded. Section 3 shows the working of Dead Reckoning method for prediction of pedestrian path. Section 4 describes the method for training Neural Network for pedestrian path prediction and presents the system architecture for path prediction system. Section 5 shows various experiments to verify the working of Neural Network based path prediction algorithm and discussed influence of parameters such as number of neurons in hidden layer, prediction time and length of history of previous coordinates. Section 6 presents the conclusions.

2. System Configuration

This section discusses the data acquisition setup and nature of signals received. For data acquisition Android based smart phone with application AndroSensor from Google Play store is being used [4]. AndroSensor application provides pedestrian speed and yaw angle calculated from the latitude and longitude data provided by GPS.

M	N	O	P	Q	R	S
LOCATIONLatitude	LOCATIONLongitude	...	L...	LOCATIONSpeedKmh	...	LOCATIONORIENTATION
NUMBER	NUMBER			NUMBER		TEXT
42.322514	-83.47907	...	...	6.12	...	24.6
42.322525	-83.47906	...	...	6.41	...	31.7
42.322525	-83.47906	...	...	6.41	...	31.7
42.322525	-83.47906	...	...	6.41	...	31.7
42.322525	-83.47906	...	...	6.41	...	31.7

Fig 1: Data (latitude, longitude, speed and orientation (yaw angle)) recorded by AndroSensor

The data sampling rate is set to 10 Hz. To verify the correctness of signals two measurements were repeated in the same path in continuous manner (Fig 2(i)). It was found that the path plotted using latitude-longitude approximately coincided, yaw angle changed from approx. 0 rad to 2*pi radian in both loops. Since the first loop was completed in 112.5 s and second loop in 120.7 s (second loop takes longer time), the average speed in second loop was lesser (as seen in graph). Thus, the measurements taken by AndroSensor were verified for their correctness.

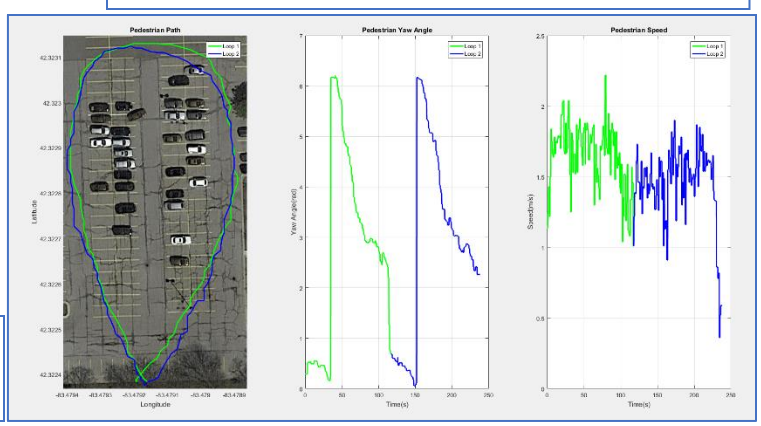


Fig 2: (i) Latitude longitude plot of pedestrian path, (ii) yaw angle of the pedestrian with time (iii) speed of pedestrian with time, loop 1: green, loop 2 : blue

3. Path Prediction Using Dead Reckoning

3.1 Dead Reckoning

Using pedestrian speed and yaw angle the pedestrian path is predicted using Dead Reckoning [5]. During the prediction time the speed of the pedestrian is assumed to be constant and equal to speed at the point of prediction.

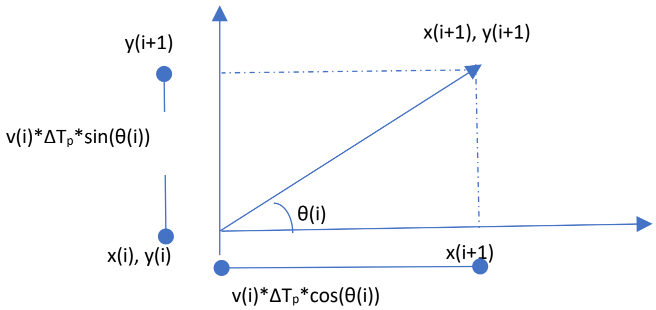


Fig 3: Dead Reckoning

$$x(i+1) = x(i) + v(i)*\Delta T_p*\cos(\theta(i)) \qquad \text{Eq(1)}$$

$$y(i+1) = y(i) + v(i)*\Delta T_p*\sin(\theta(i)) \qquad \text{Eq(2)}$$

where, x(i+1) is the predicted x coordinate, y(i+1) is the predicted y coordinate, x(i) is current x- coordinate, y(i) is the y-coordinate, v(i) is the current speed, ΔT_p is the prediction time and θ(i) is the current yaw angle

3.2 Conversion from Latitude Longitude to Universal Transverse Mercator to Cartesian Coordinates

Universal Transverse Mercator [6] represents the earth in a form of grid with 60 zones. Each zone is 6° longitude in width and ranges from 80° S latitude to 84° N latitude. Matlab library functions [7] are used to convert from latitude-longitude to UTM coordinates. The initial UTM coordinates are subtracted from all other UTM coordinates to obtain pedestrian position in terms of cartesian coordinates. In this way, the starting position becomes origin.

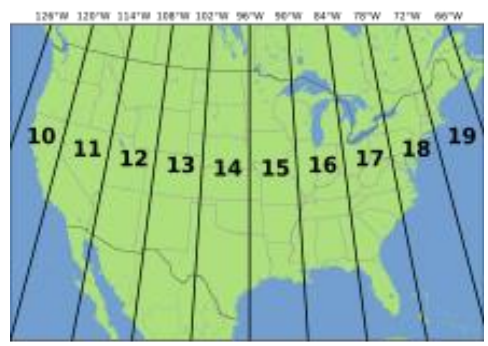


Fig 4: Universal Transverse Mercator zones [6]

3.3 Conversion from Cartesian to Universal Transverse Mercator To Latitude-Longitude

The initial position UTM value is added to the cartesian coordinates to obtain UTM coordinates and then using Matlab Library functions [7] the UTM coordinates are converted to Latitude-Longitude.

3.4 Path Prediction using Dead Reckoning

As described in sections 3.2 and 3.3 the latitude and longitude are converted to obtain cartesian coordinates (x,y) and then by making use of dead reckoning method as described in section 3.1 the future position is predicted(YouTube link: https://youtu.be/MZaqEY1149I)

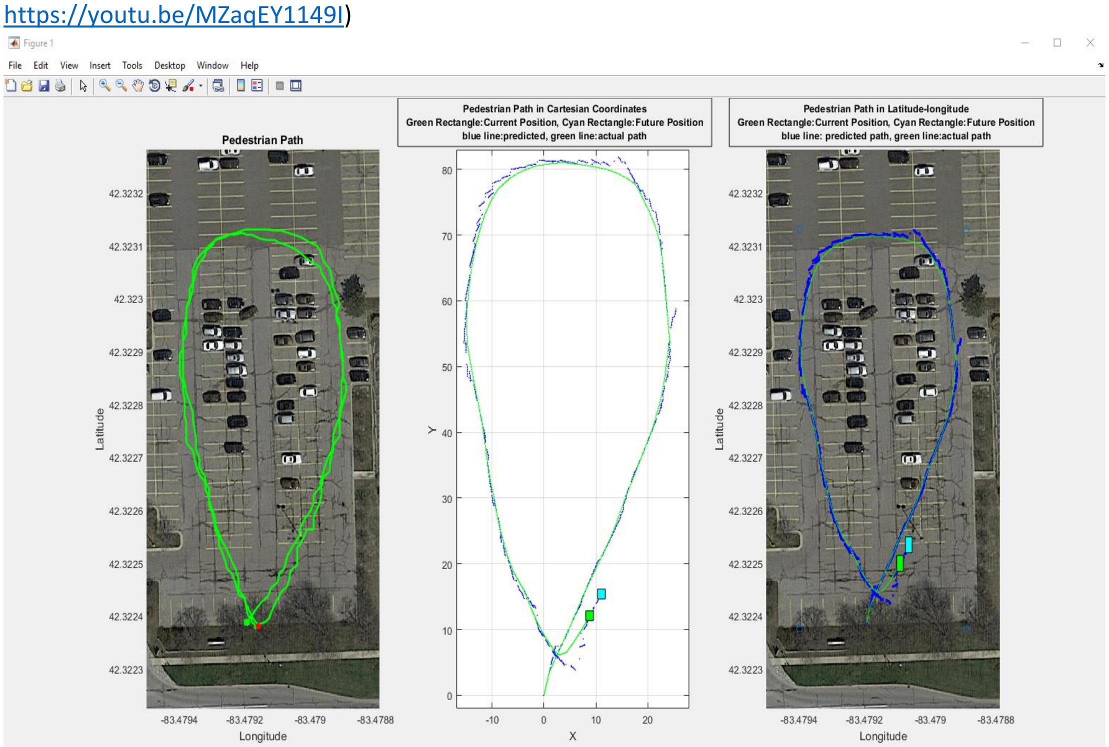


Fig 5: Simulation of the trajectory prediction by Dead Reckoning (i) shows the actual path traversed by the pedestrian on Google Map in terms of latitude and longitude (ii) shows the actual and predicted path on Cartesian Coordinates. Green line shows the actual path, blue line shows the predicted path. Green Rectangle shows the current position, Cyan Rectangle shows the future position (iii) shows the actual and predicted position of pedestrian on Google Map.

4. Path Prediction Using Neural Network

4.1 Creating Training Data

Paths as shown in Fig 6 have been used to obtain trajectory training data. The trajectory data in obtained in terms of longitude and latitude were converted in to cartesian coordinates using procedure as described in section

Fig 6: Pedestrian Paths for collecting data for Neural Network

3.2.

Step 1: Generate trajectory segments using moving window trajectory generation

The trajectory is divided into segments. For example, consider first segment. This segment has 30 pairs of (x,y) values. Since the sampling rate is 10 Hz. This segment represents 3s data.

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24	x25	x26	x27	x28	x29	x30
y1	y2	y3	y4	y5	y6	y7	y8	y9	y10	y11	y12	y13	y14	y15	y16	y17	y18	y19	y20	y21	y22	y23	y24	y25	y26	y27	y28	y29	y30
Past Location																			Current Location	Future Location									Target Location
Segment 1 : Target Location 1s ahead of Current Location, Training Data of 2s																													

First 2s data from (x1,y1) to (x20,y20) represents the training data. For 1s prediction time, pair (x30,y30) represents the target data. This segment window is moved by one sample to generate next set of training and target data as shown below:

x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24	x25	x26	x27	x28	x29	x30	x31
y2	y3	y4	y5	y6	y7	y8	y9	y10	y11	y12	y13	y14	y15	y16	y17	y18	y19	y20	y21	y22	y23	y24	y25	y26	y27	y28	y29	y30	y31
Past Location																			Current Location	Future Location									Target Location
Segment 2 : Target Location 1s ahead of Current Location, Training Data of 2s																													

This process is continued until segments are generated from all the three paths.

step 2: For each segment subtract the current location from all the coordinates to make current location as origin. For example, first segment is transformed as shown below:

x2 - x21	x3 - x21	x4 - x21	x5 - x21	x6 - x21	x7 - x21	x8 - x21	x9 - x21	x10 - x21	x11 - x21	x12 - x21	x13 - x21	x14 - x21	x15 - x21	x16 - x21	x17 - x21	x18 - x21	x19 - x21	x20 - x21	0	x22 - x21	x23 - x21	x24 - x21	x25 - x21	x26 - x21	x27 - x21	x28 - x21	x29 - x21	x30 - x21	x31 - x21
y2 - y21	y3 - y21	y4 - y21	y5 - y21	y6 - y21	y7 - y21	y8 - y21	y9 - y21	y10 - y21	y11 - y21	y12 - y21	y13 - y21	y14 - y21	y15 - y21	y16 - y21	y17 - y21	y18 - y21	y19 - y21	y20 - y21	0	y22 - y21	y23 - y21	y24 - y21	y25 - y21	y26 - y21	y27 - y21	y28 - y21	y29 - y21	y30 - y21	y31 - y21
Past Location																			Current Location	Future Location									Target Location
Segment 2 : subtract current location from all coordinates, so that current location becomes (0,0)																													

Similarly, for second segment,

x1-x20	x2-x20	x3-x20	x4-x20	x5-x20	x6-x20	x7-x20	x8-x20	x9-x20	x10-x20	x11-x20	x12-x20	x13-x20	x14-x20	x15-x20	x16-x20	x17-x20	x18-x20	x19-x20	0	x21 - x20	x22 - x20	x23 - x20	x24 - x20	x25 - x20	x26 - x20	x27 - x20	x28 - x20	x29 - x20	x30 - x20
y1-y20	y2-y20	y3-y20	y4-y20	y5-y20	y6-y20	y7-y20	y8-y20	y9-y20	y10-y20	y11-y20	y12-y20	y13-y20	y14-y20	y15-y20	y16-y20	y17-y20	y18-y20	y19-y20	0	y21 - y20	y22 - y20	y23 - y20	y24 - y20	y25 - y20	y26 - y20	y27 - y20	y28 - y20	y29 - y20	y30 - y20
Past Location																			Current Location	Future Location									Target Location
Segment 1 : subtract current location from all coordinates, so that current location becomes (0,0)																													

This process is carried out until for all segments the current location becomes origin.

Step 3 : Arranging all training data sets

All the segments are collected together so that each row represents a training set as shown below:

Set Num	Training Data																																								Target Data	
1	0	x19-x20	x18-x20	x17-x20	x16-x20	x15-x20	x14-x20	x13-x20	x12-x20	x11-x20	x10-x20	x9-x20	x8-x20	x7-x20	x6-x20	x5-x20	x4-x20	x3-x20	x2-x20	x1-x20	0	y19-y20	y18-y20	y17-y20	y16-y20	y15-y20	y14-y20	y13-y20	y12-y20	y11-y20	y10-y20	y9-y20	y8-y20	y7-y20	y6-y20	y5-y20	y4-y20	y3-y20	y2-y20	y1-y20	x30 - x20	y30 - y20
2	0	x20 - x21	x19 - x21	x18 - x21	x17 - x21	x16 - x21	x15 - x21	x14 - x21	x13 - x21	x12 - x21	x11 - x21	x10 - x21	x9 - x21	x8 - x21	x7 - x21	x6 - x21	x5 - x21	x4 - x21	x3 - x21	x2 - x21	0	y20 - y21	y19 - y21	y18 - y21	y17 - y21	y16 - y21	y15 - y21	y14 - y21	y13 - y21	y12 - y21	y11 - y21	y10 - y21	y9 - y21	y8 - y21	y7 - y21	y6 - y21	y5 - y21	y4 - y21	y3 - y21	y2 - y21	x31-x21	y31-y20
n-19	0	x(n-1)-x(n)	x(n-2)-x(n)	x(n-3)-x(n)	x(n-4)-x(n)	x(n-5)-x(n)	x(n-6)-x(n)	x(n-7)-x(n)	x(n-8)-x(n)	x(n-9)-x(n)	x(n-10)-x(n)	x(n-11)-x(n)	x(n-12)-x(n)	x(n-13)-x(n)	x(n-14)-x(n)	x(n-15)-x(n)	x(n-16)-x(n)	x(n-17)-x(n)	x(n-18)-x(n)	x(n-19)-x(n)	0	y(n-1)-y(n)	y(n-2)-y(n)	y(n-3)-y(n)	y(n-4)-y(n)	y(n-5)-y(n)	y(n-6)-y(n)	y(n-7)-y(n)	y(n-8)-y(n)	y(n-9)-y(n)	y(n-10)-y(n)	y(n-11)-y(n)	y(n-12)-y(n)	y(n-13)-y(n)	y(n-14)-y(n)	y(n-15)-y(n)	y(n-16)-y(n)	y(n-17)-y(n)	y(n-18)-y(n)	y(n-19)-y(n)	x(n+10)-x(n)	y(n+10)-y(n)

4.2 Neural Network Training:

Fitting utility of Neural Network Toolbox from Matlab is used for training. The training and target data as obtained in section 4.1 is used for training the neural network [1]. For training data as shown above the number of inputs are 40 and number of outputs are 2 for each training set. The training data is divided into 70% training, 15% validation (to stop training when the network generalization stops improving) and 15% independent test data. Backpropagation method is used for training the Neural Network.

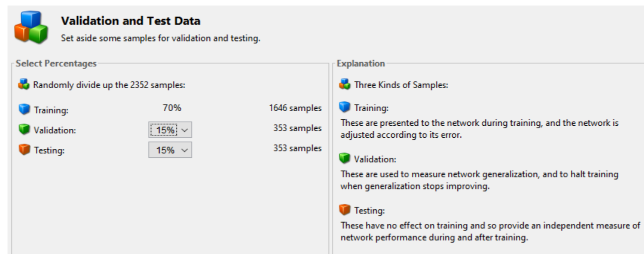
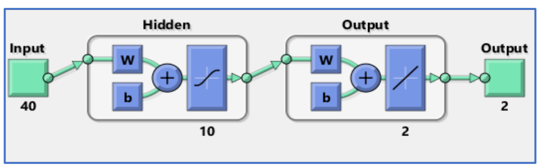


Fig 7: Training, Validation, Test data settings



The architecture is as show in fig 8.
Neural Network Architecture for 40 inputs and 2 outputs. It consists of 10 hidden neurons and 2 output neurons.

Fig 8: Trained Neural Network Architecture

4.3 System Architecture for Neural Network Based Pedestrian Path Prediction

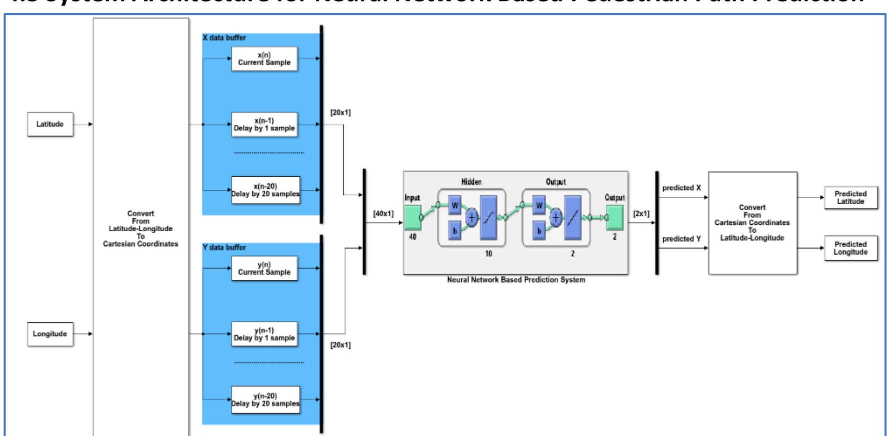


Fig 9: System Architecture for Neural Network based Pedestrian Path Prediction

Incoming sample of latitude and longitude is converted into corresponding (x,y) values. x-data buffer contains current x sample and previous 19 x values. Similarly, y-data buffer contains current y sample and previous 19 y values. These values are passed to the "Neural Network Based Prediction System" which provides predicted (x,y) values. These values are converted back to predicted latitude-longitude.
(YouTube link: https://youtu.be/gRSPtz-GQTE)

5.1. Experiments

For testing a Neural Network trained to use 50 past coordinates to predict the future location has been used.
Performance is compared by using Root Mean Square Error(RMS). It makes use of Matlab function for computing RMS error [8]. The first matrix being passed to the Matlab function contains the actual values of latitude and longitude as column elements, the second second matrix contains the values of latitude and longitude predicted by prediction algorithm.

5.1 Comparison of Neural Network and Dead Reckoning Path Prediction:

To compare the performance of Dead Reckoning and Neural Network based prediction system, a new path is chosen which is different from the training data set and at a new geographical location (as shown in fig 11). It can be observed that for both Dead Reckon as well as Neural Network predicted path, Root Mean Square Error increases with prediction time. Neural Network has lower error for prediction time less than or equal to 3s.

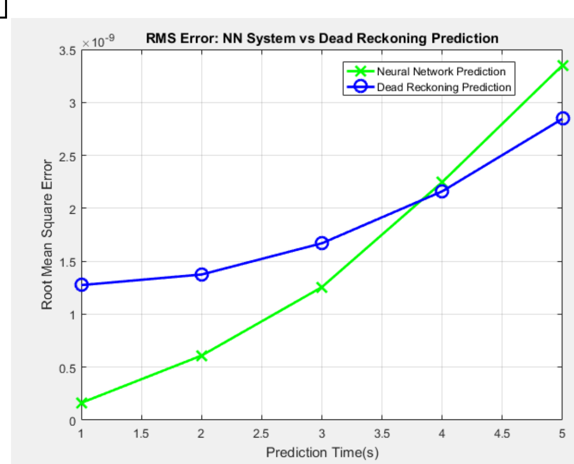


fig 10: RMS Error for Dead Reckoning and Neural Network Prediction System with Prediction time

Fig 11 : Plot for 3s and 5s prediction for Dead Reckon and Neural Network predicted path against actual path

5.2 Performance comparison of neural network system with change in number of past data samples for prediction:

The prediction time is kept at 2s and the number of past data segment is changed from 2s to 5s in steps of 1s. It is observed that RMS Error increases as the number of past data sample for prediction increases.

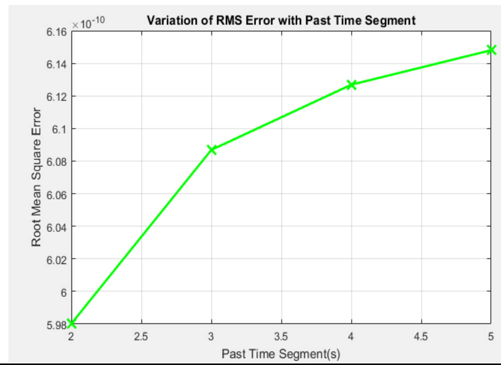


Fig 12: Variation of RMS error for Neural Network Prediction with Past Time Segment

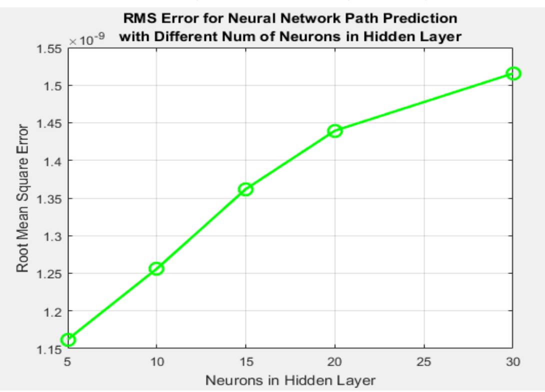


Fig 13: RMS error with number of neurons in hidden layer

5.3 Performance comparison of neural network system with increase in number of neurons in hidden layer

The prediction time is kept at 3s with past data of 5s, one network has 10 neurons in hidden layer, second has 15 and third has 20. As the number of neurons in hidden layer reduced the RMS error reduced.

6. Conclusion

This paper has presented and compared the pedestrian path prediction using Dead Reckoning and Neural Network Curve Fitting approach. By segmenting the pedestrian path recorded in latitude and longitude using moving window, making current location as origin in all segments, data recorded at different locations can be joined together to form a training data set. By transforming geographical coordinates to cartesian coordinates the path prediction algorithm can be made independent of geographical locations. For small prediction times (less than 3s) neural network approach results into lower RMS error compared to dead reckoning. Prediction using short past segments performed better compared to long segment. This is found in congruence with the basic nature of pedestrian movement in unstructured environment which tends to be more short term compared to long term. Also, using lesser number of neurons in hidden layer resulted in better prediction performance which is in congruence with the overfitting problem encountered by neural networks with higher number of neurons in hidden layer.

References:

(1) https://www.mathworks.com/help/nnet/gs/fit-data-with-a-neural-network.html

(2) D. Vasquez and T. Fraichard, "Motion prediction for moving objects: a statistical approach," Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on, 2004, pp. 3931-3936 Vol.4.

(3) P.P. Choi, M. Herbert, “Learning and predicting object trajectory, a piecewise trajectory segment approach”, Robotics Institute, p 337, 2006

(4) https://play.google.com/store/apps/details?id=com.fivasim.androsensor&hl=en

(5) https://en.wikipedia.org/wiki/Dead_reckoning

(6) https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system

(7) https://www.mathworks.com/matlabcentral/fileexchange/45699-ll2utm-and-utm2ll

(8) https://www.mathworks.com/help/images/ref/immse.html