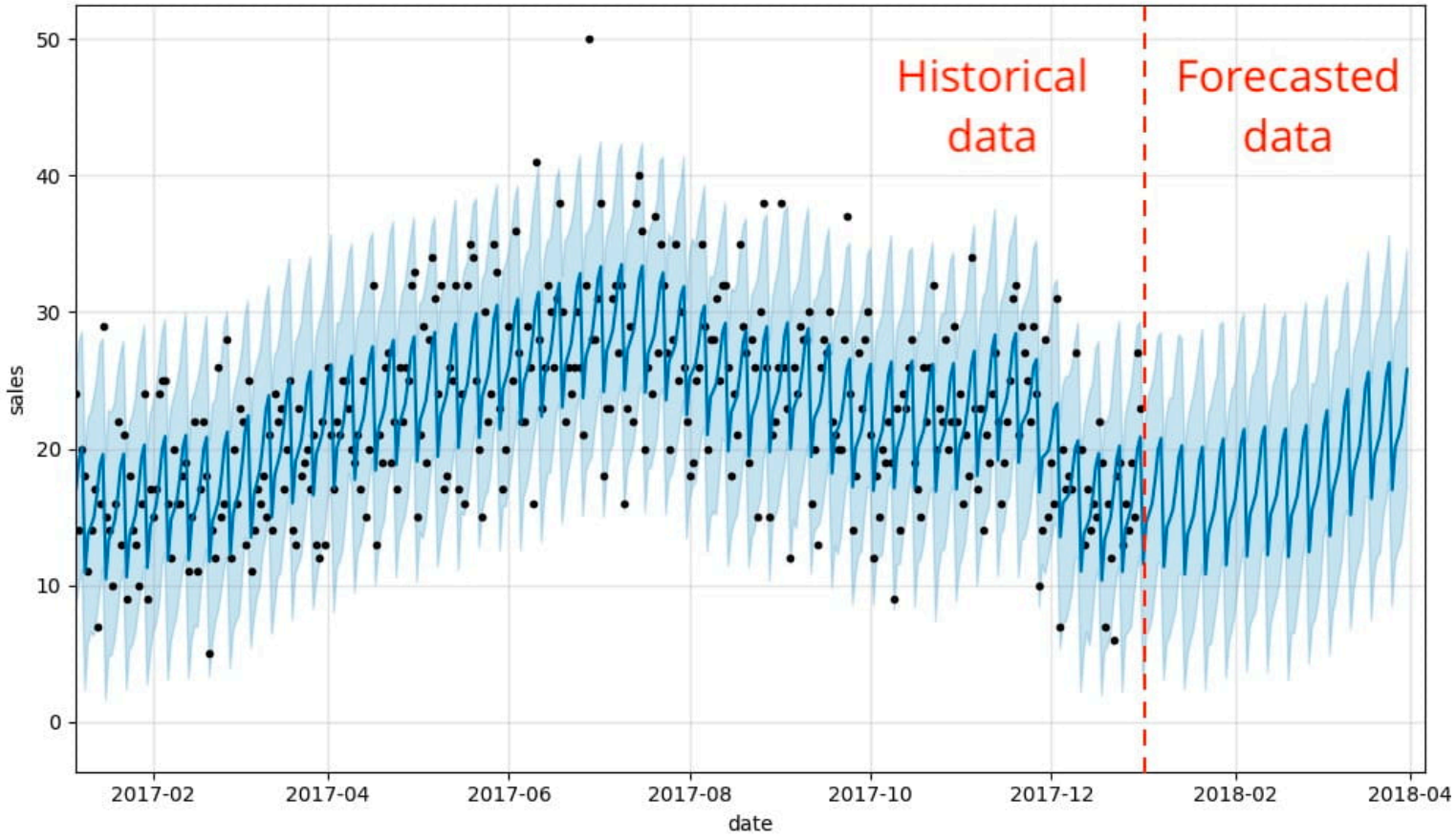
FORECASTING FUTURE POSITION OF MOVING UE

Time Series Forecasting Method



Data preparation stage

Parameters requried

- Timestamp

  "measTimeStampRf": "2021-03-30T09:49:17.306”

- Position of the UE

  x": null, "y": null Note: Unsupported with the current simulator

Additional parameter

- Sinal strength (RSRP)

  "rsrp": 54

- Current distance

  "current_distance": 300

Forcasting model choice

- Auto-regression models and EMA (ARMA, ARIMA, GARCH)
- Neural networks (RNN, LSTM, ConvLSTM)
- Vector autoregression (VAR)

Multivariate lstm model

Pipeline of Machine learning

- Data exporalation (Clean data)
- Data visualization
- Normalize the data
- Model training
- Optimize model with hyperparameters
- Evaluate the model
- Denormalize data back to original values
- Visualizing the predictions
- Get trained model

Method of Evaluation

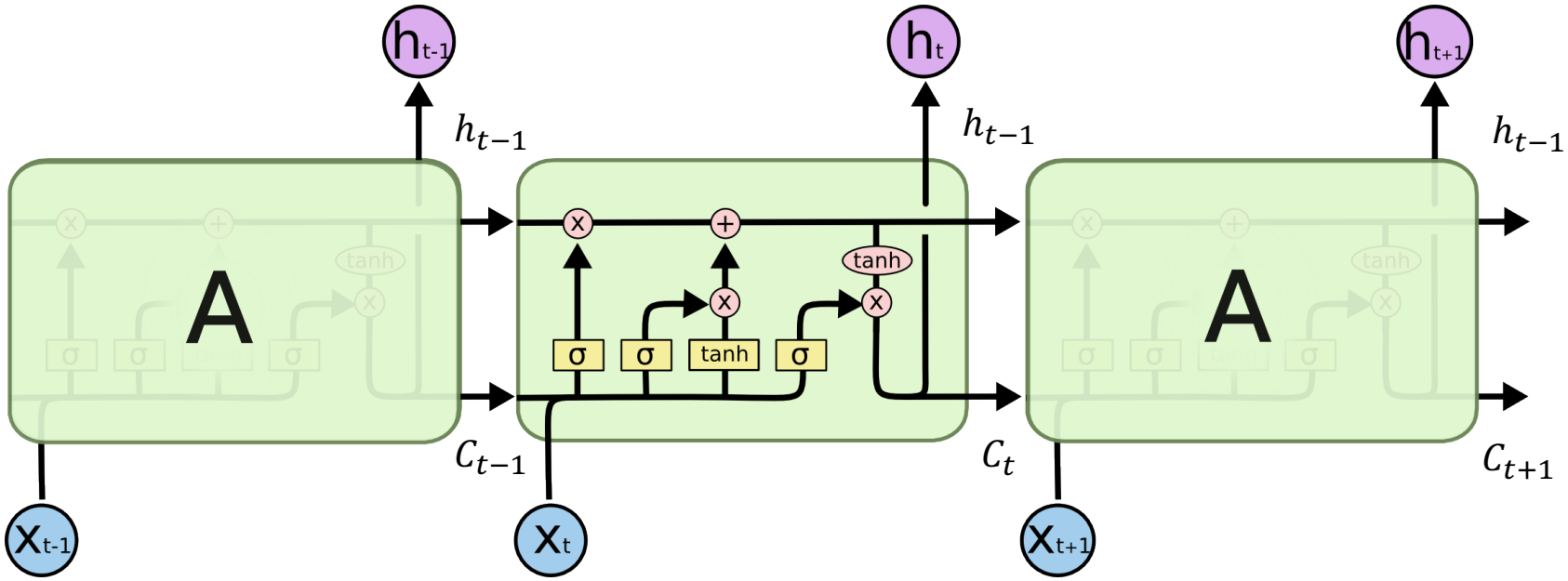
Checking Error

Comparing the actual or expected data with predicting data

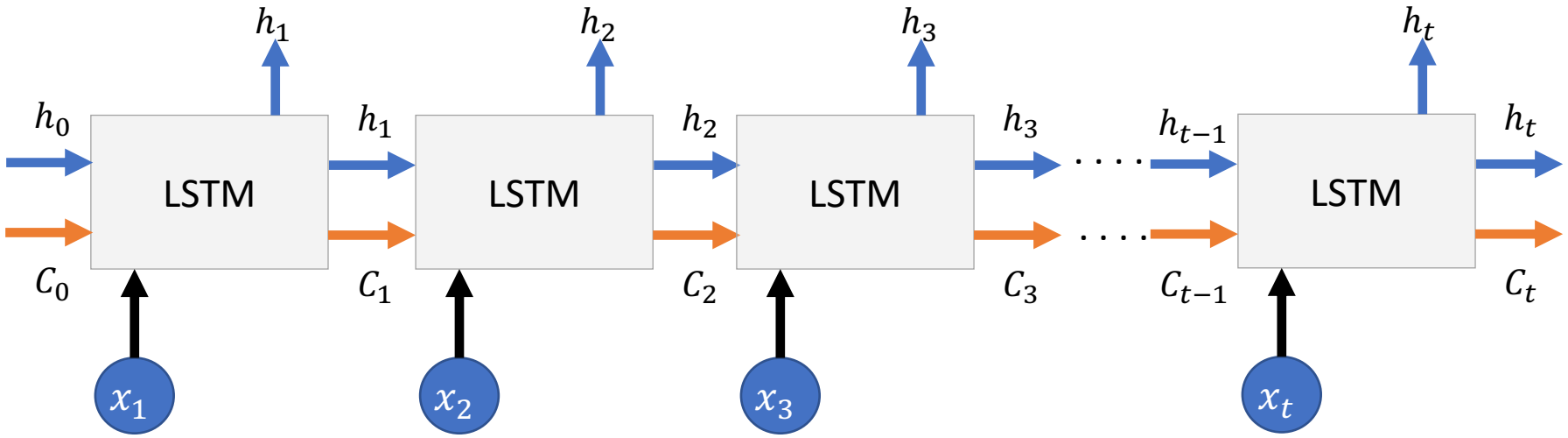
- RMSE (Root Mean Square Error)
- MAE (Mean Absolute Error)

TIME =1, Predicted=351.582196, Expected=339.700000
TIME =2, Predicted=432.169667, Expected=440.400000
TIME =3, Predicted=378.064505, Expected=315.900000
.
.
Test RMSE: 71.721

Long Short-Term Memory

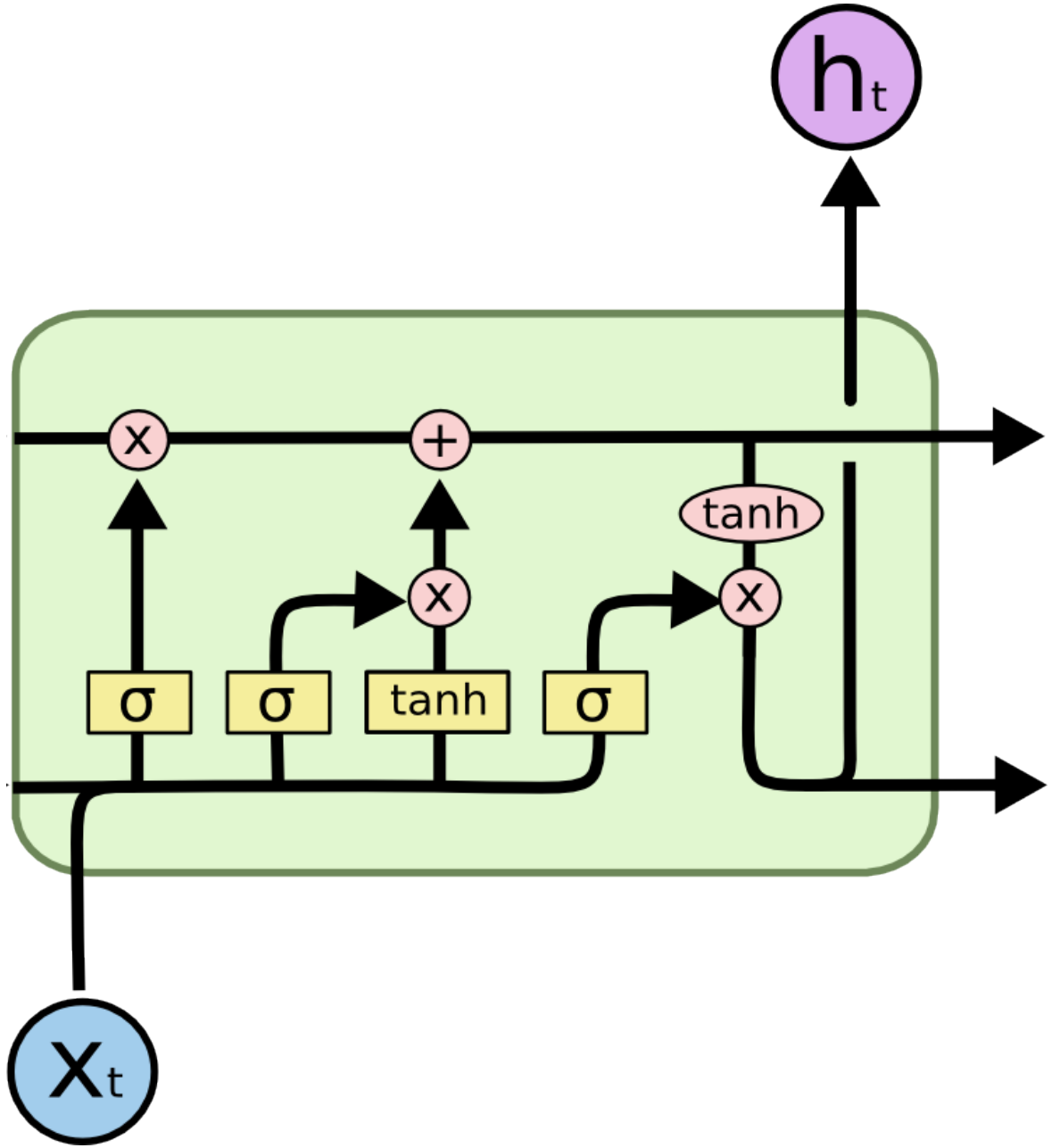


Long Short-Term Memory



LSTM Breakdown

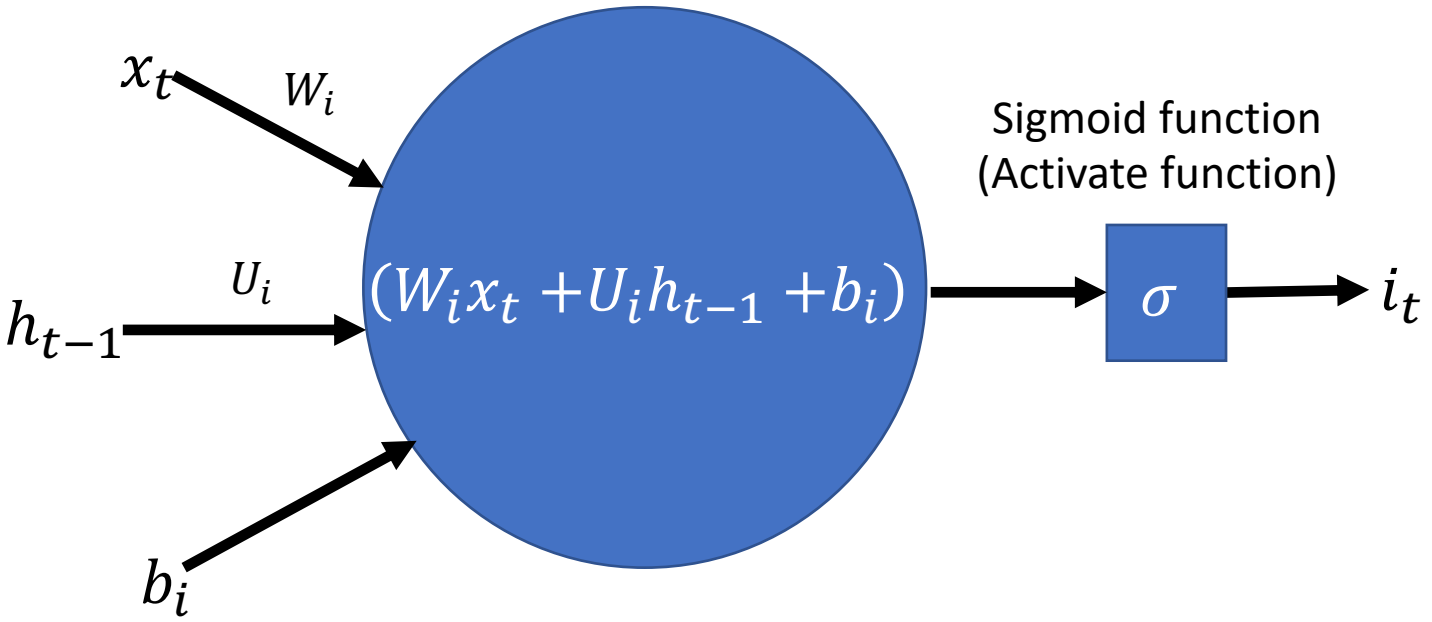
- ❑ Input gate
- ❑ Forget gate
- ❑ Output gate
- ❑ Cell state update



Input Gate

W_i , U_i = Weight vectors
b_i = Bias
h_{t-1} = Previous cell output
x_t = Input vector

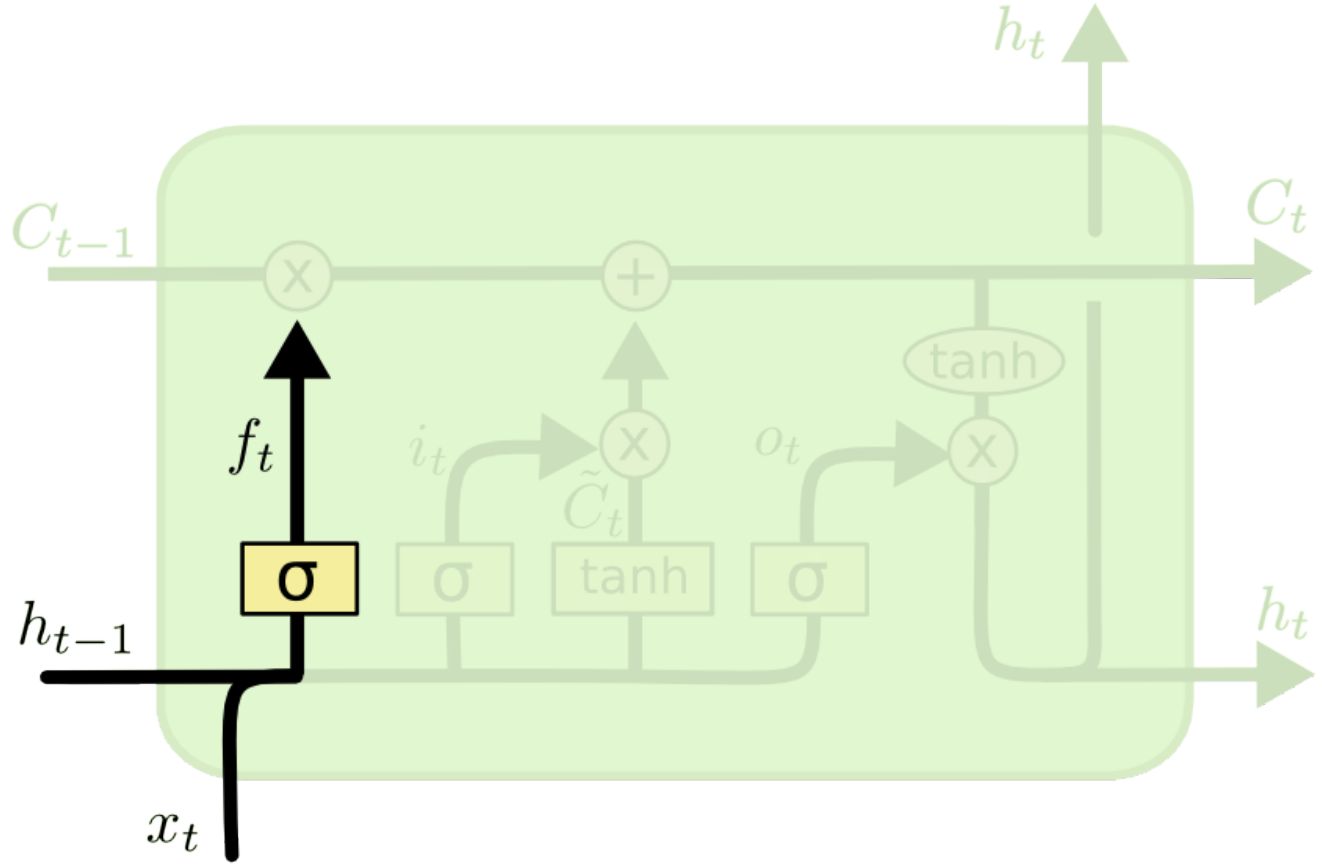
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$



Forget Gate

- DECIDE WHAT INFORMATION IS KEPT FROM PREVIOUS STATES

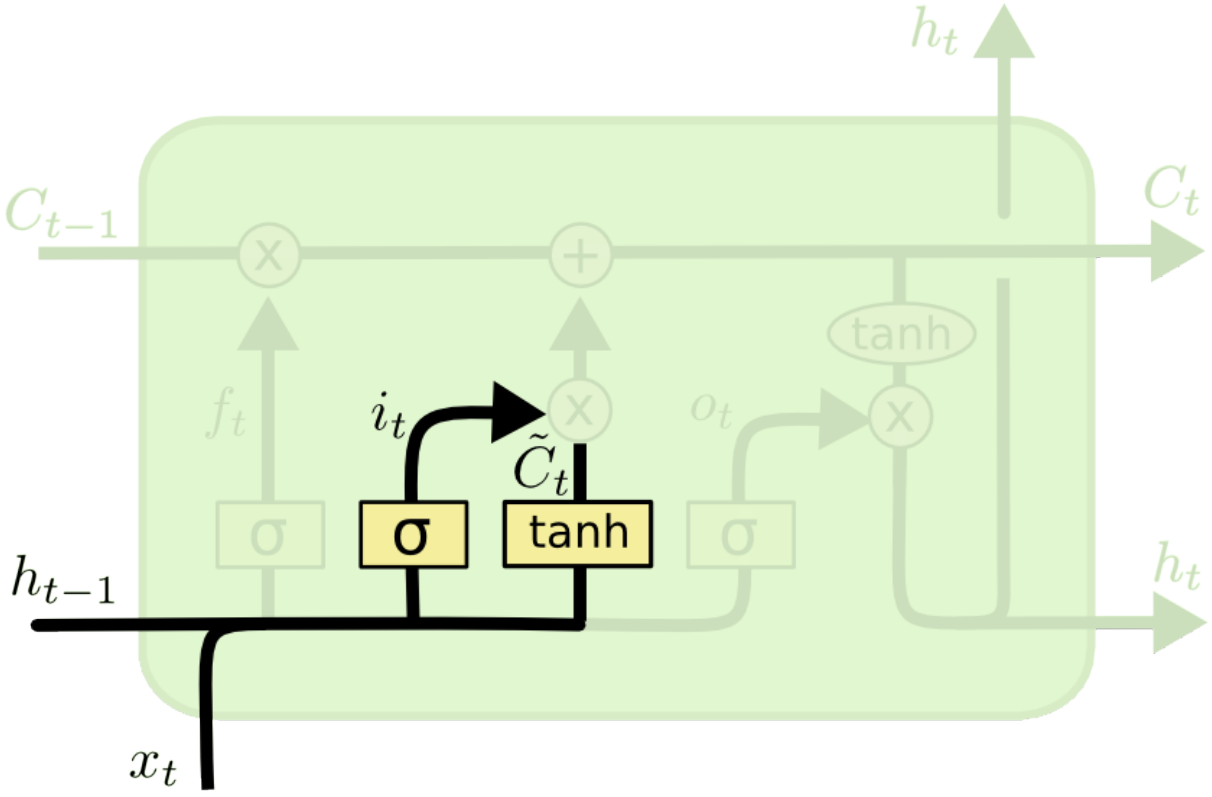
$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right)$$



Input Gate

- DECIDE WHAT INFORMATION IS ADDED TO THE HIDDEN STATE

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$
$$\tilde{C}_t = g_t = \tanh\left(W_g x_t + U_g h_{t-1} + b_g\right)$$

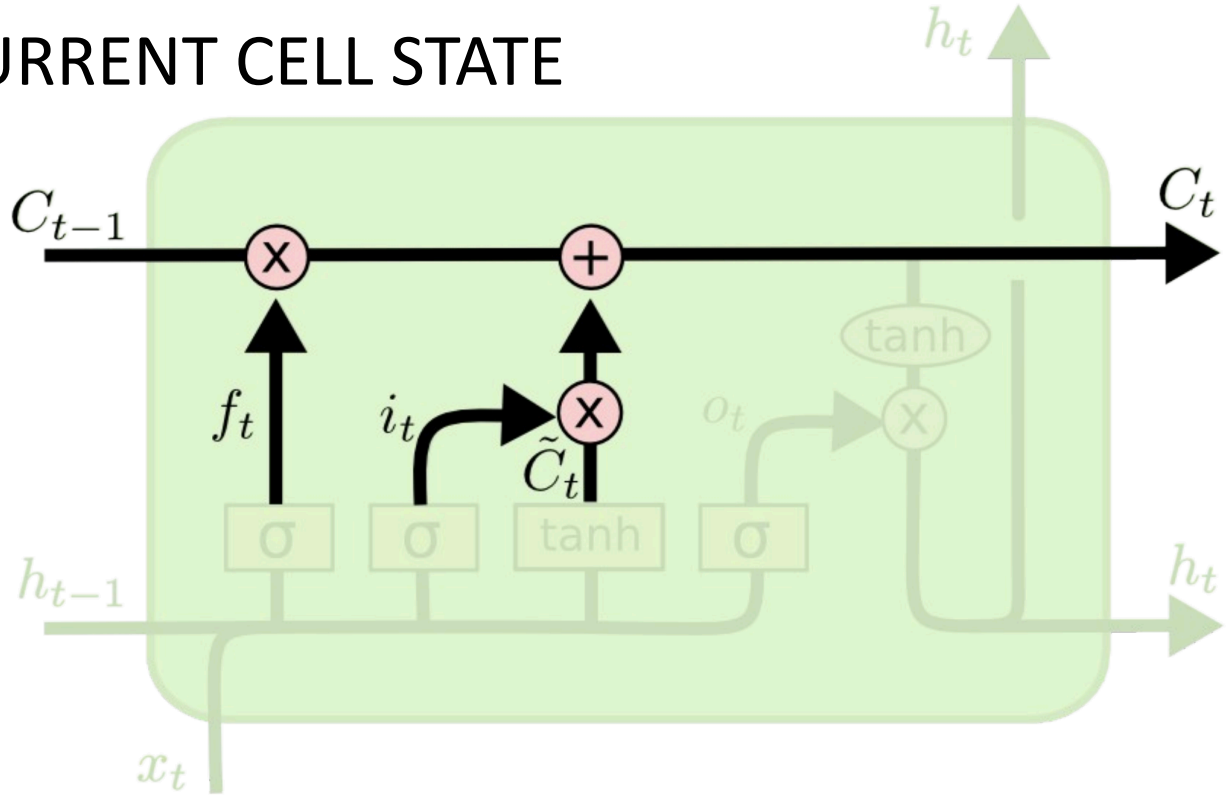


Cell State Update

- DECIDE WHETHER TO FORGET PREVIOUS CELL STATE AND ADD THE NEW VALUE TO THE CURRENT CELL STATE

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

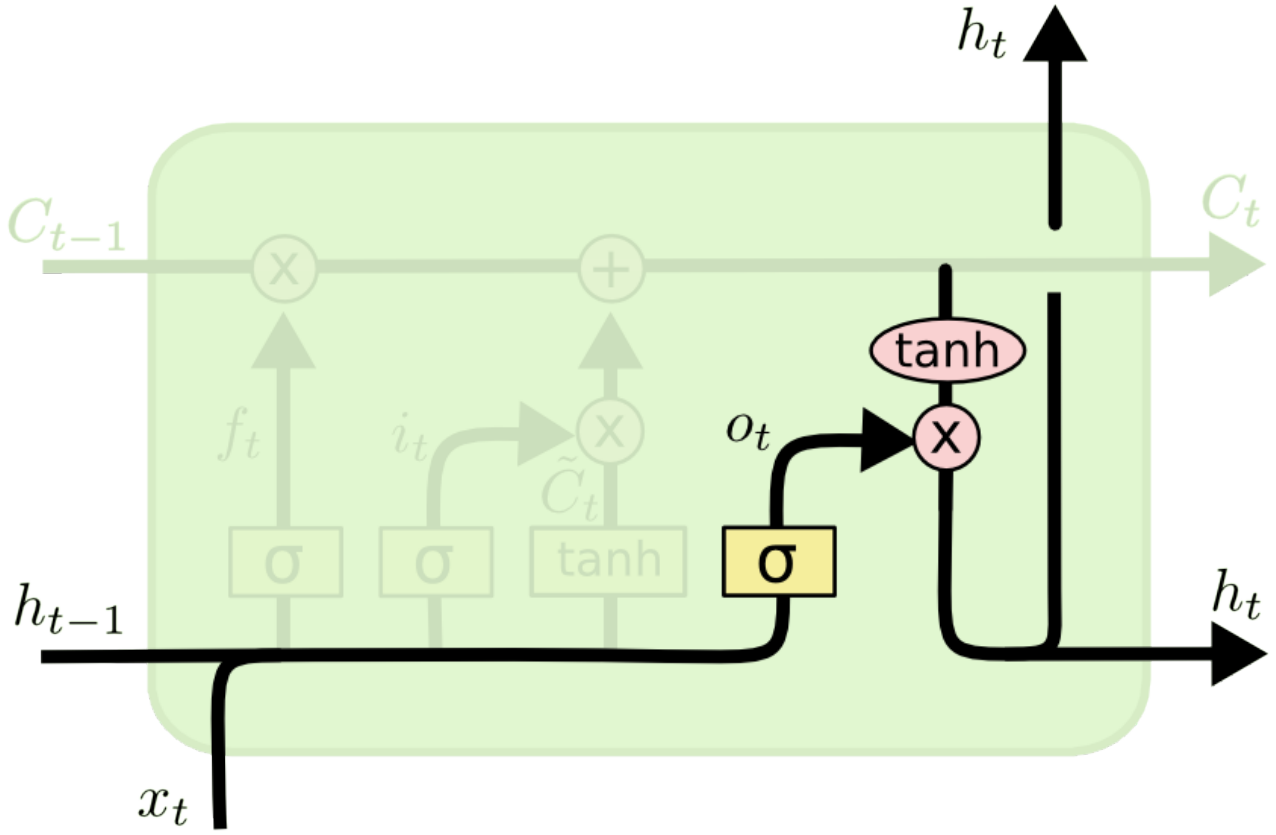
f_t : decide which to forget t
i_t : decide which to update



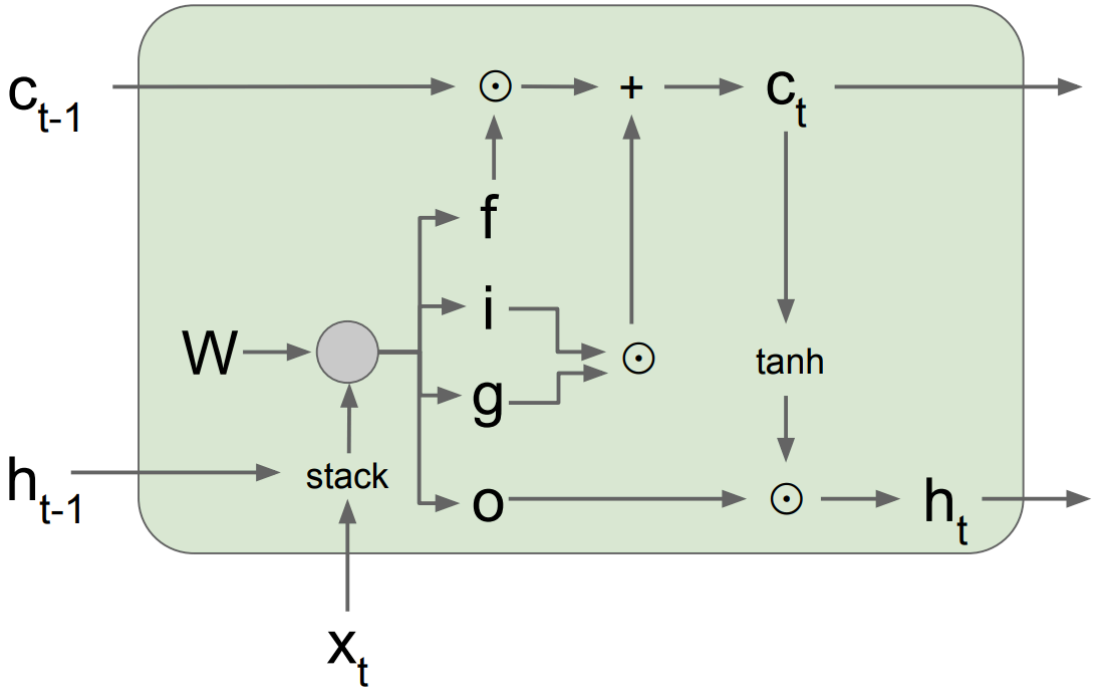
Output Gate

- DECIDE WHAT IS REPORTED AS OUTPUT

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$
$$h_t = o_t \odot \tanh c_t$$

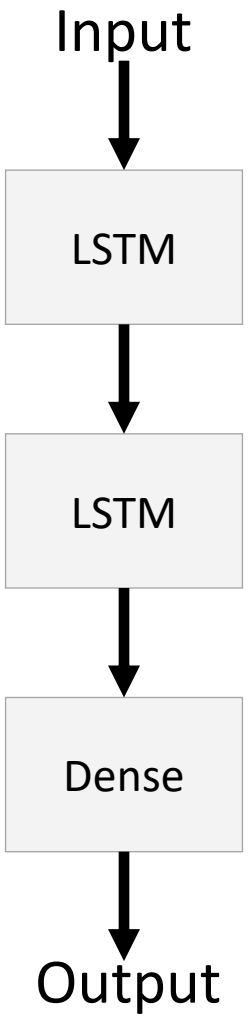


LSTM Summary



$$g_t = \tanh\left(W_g x_t + U_g h_{t-1} + b_g\right)$$
$$i_t = \sigma\left(W_i x_t + U_i h_{t-1} + b_i\right)$$
$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right)$$
$$o_t = \sigma\left(W_o x_t + U_o h_{t-1} + b_o\right)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh c_t$$

Stacked LSTM



References

- C. Olah. (2015). Understanding LSTM Networks [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/ LSTM3-chain.png

- B. Jason. (2017) Stacked Long Short-Term Memory Networks [Online]. Available: https: //machinelearningmastery.com/stacked-long-short-term-memory- networks/.
- A. Benterki, V. Judalet, M. Choubeila and M. Boukhnifer, "Long-Term Prediction of Vehicle Trajectory Using Recurrent Neural Networks," *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, 2019, pp. 3817-3822, doi: 10.1109/IECON.2019.8927604.

  [Online]. Available: https://ieeexplore.ieee.org/document/8927604.

References

- Z. Shi, M. Xu, Q. Pan, B. Yan and H. Zhang, "LSTM-based Flight Trajectory Prediction," 2018 International Joint Conference on Neural Networks (IJCNN), 2018, pp. 1-8, doi: 10.1109/IJCNN.2018.8489734.

  [Online]. Available: https://ieeexplore.ieee.org/document/8489734.

- D. Jeong, M. Baek and S. Lee, "Long-term prediction of vehicle trajectory based on a deep neural network," 2017 International Conference on Information and Communication Technology Convergence (ICTC), 2017, pp. 725-727, doi: 10.1109/ICTC.2017.8190764.

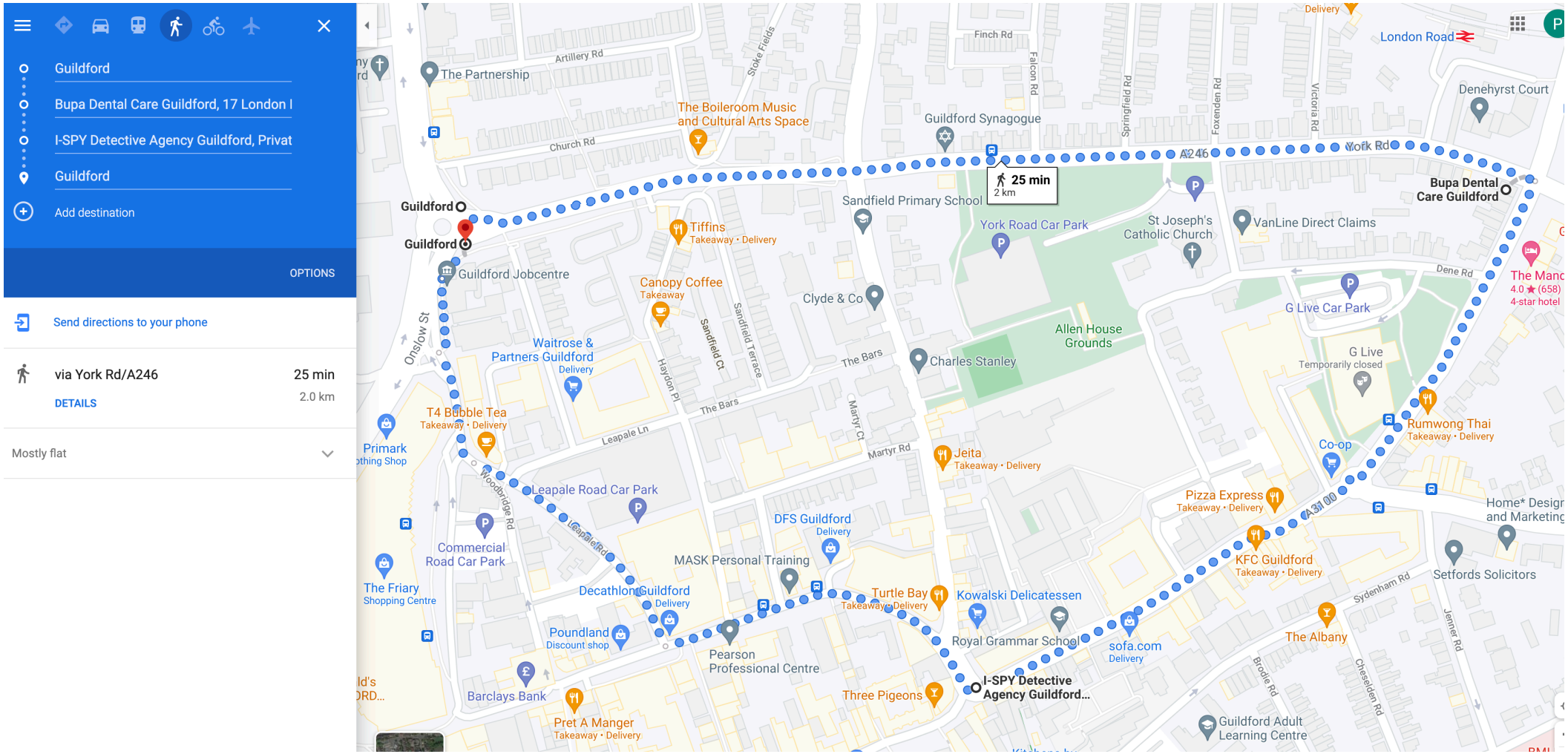
  [Online]. Available: https://ieeexplore.ieee.org/document/8190764.

- C. Wang, L. Ma, R. Li, T. S. Durrani and H. Zhang, "Exploring Trajectory Prediction Through Machine Learning Methods," in IEEE Access, vol. 7, pp. 101441-101452, 2019, doi: 10.1109/ACCESS.2019.2929430.

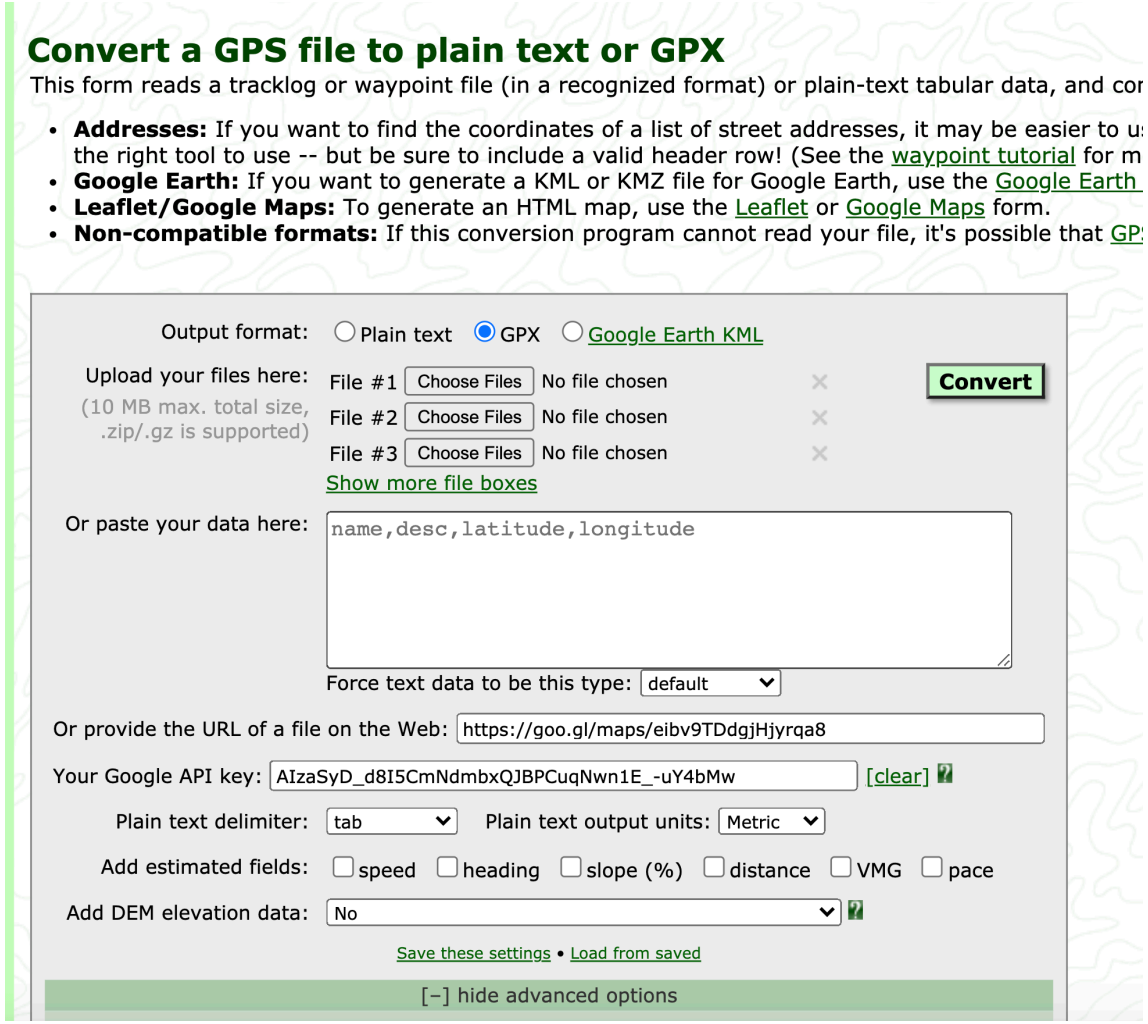
  [Online]. Available: https://ieeexplore.ieee.org/document/8766820

Run LSTM prototype model to forecast the next position of UE

Create a trajectory of UE from Google map



Generate Latitude and longitude with GPX converter



https://www.gpsvisualizer.com/convert_input

In [641]:
```
!gpxinfo d5_2.gpx
```

```
File: d5_2.gpx
    Length 2D: 1.953km
    Length 3D: 1.953km
    Moving time: 00:32:36
    Stopped time: n/a
    Max speed: 1.21m/s = 4.34km/h
    Avg speed: 1.00m/s = 3.59km/h
    Total uphill: 0.00m
    Total downhill: 0.00m
    Started: 2021-07-20 19:11:05+00:00
    Ended: 2021-07-20 19:44:02+00:00
    Points: 160
    Avg distance between points: 12.34m

    Track #0, Segment #0
        Length 2D: 0.774km
        Length 3D: 0.774km
        Moving time: 00:12:55
        Stopped time: n/a
        Max speed: 1.06m/s = 3.81km/h
        Avg speed: 1.00m/s = 3.59km/h
        Total uphill: 0.00m
        Total downhill: 0.00m
        Started: 2021-07-20 19:11:05+00:00
        Ended: 2021-07-20 19:24:00+00:00
        Points: 35
        Avg distance between points: 22.10m
```

In [643]:
```
data = []
segment_length = segment.length_3d()
for point_idx, point in enumerate(segment.points):
    data.append([point.longitude, point.latitude,
                 point.elevation, point.time, segment.get_speed(point_idx)])

from pandas import DataFrame

columns = ['Longitude', 'Latitude', 'Altitude', 'Time', 'Speed']
df = DataFrame(data, columns=columns)
df.head()
```

Out[643]:

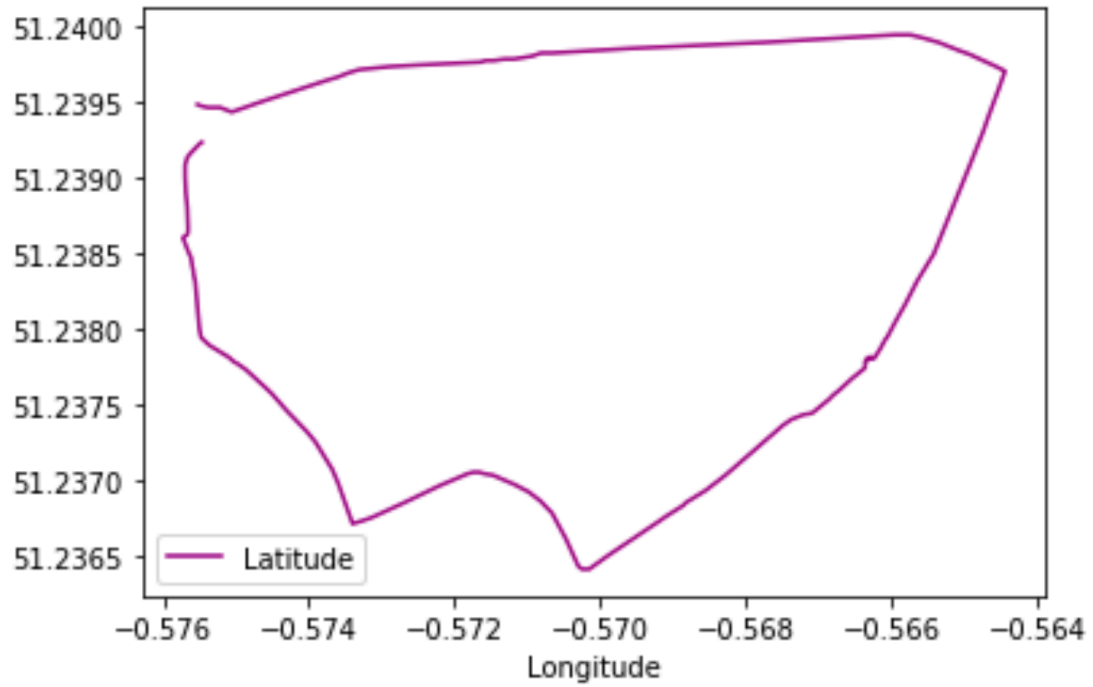
	Longitude	Latitude	Altitude	Time	Speed
0	-0.57553	51.23949	None	2021-07-20 19:11:05+00:00	0.914540
1	-0.57548	51.23948	None	2021-07-20 19:11:09+00:00	0.988139
2	-0.57539	51.23947	None	2021-07-20 19:11:15+00:00	1.024531
3	-0.57522	51.23947	None	2021-07-20 19:11:27+00:00	1.022766
4	-0.57506	51.23944	None	2021-07-20 19:11:38+00:00	1.017665

Plot the Generated Map

```
Name: 55 Church Rd - 17 London Rd - 174 High St - 86 Woodbridge Rd
Description: None
Latitude Bounds: (51.236420, 51.239950)
Longitude Bounds: (-0.575730, -0.564440)
```

: `<matplotlib.axes._subplots.AxesSubplot at 0x7ff202c07050>`

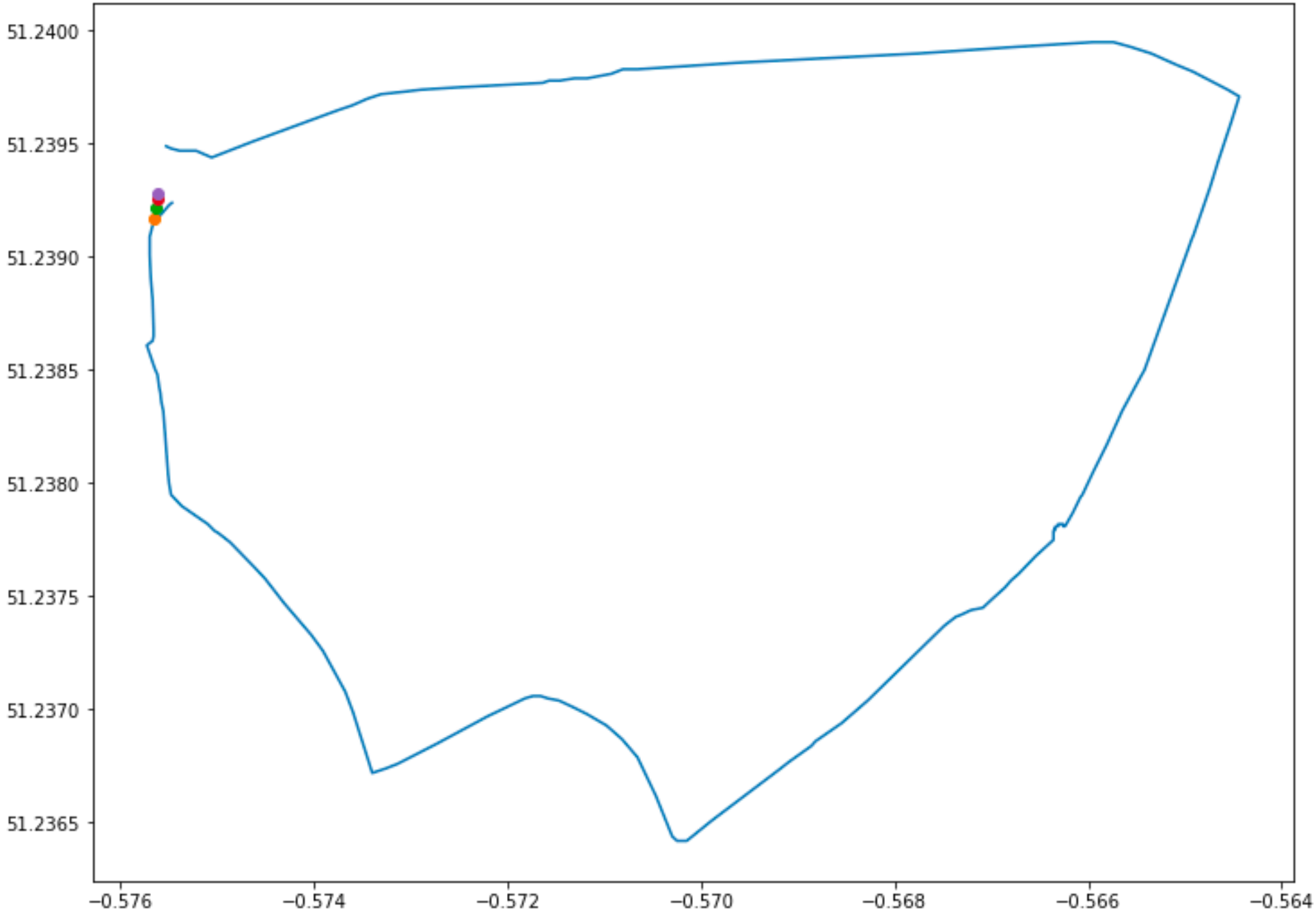
`<Figure size 864x648 with 0 Axes>`



Building the model to forecast next position of the UE

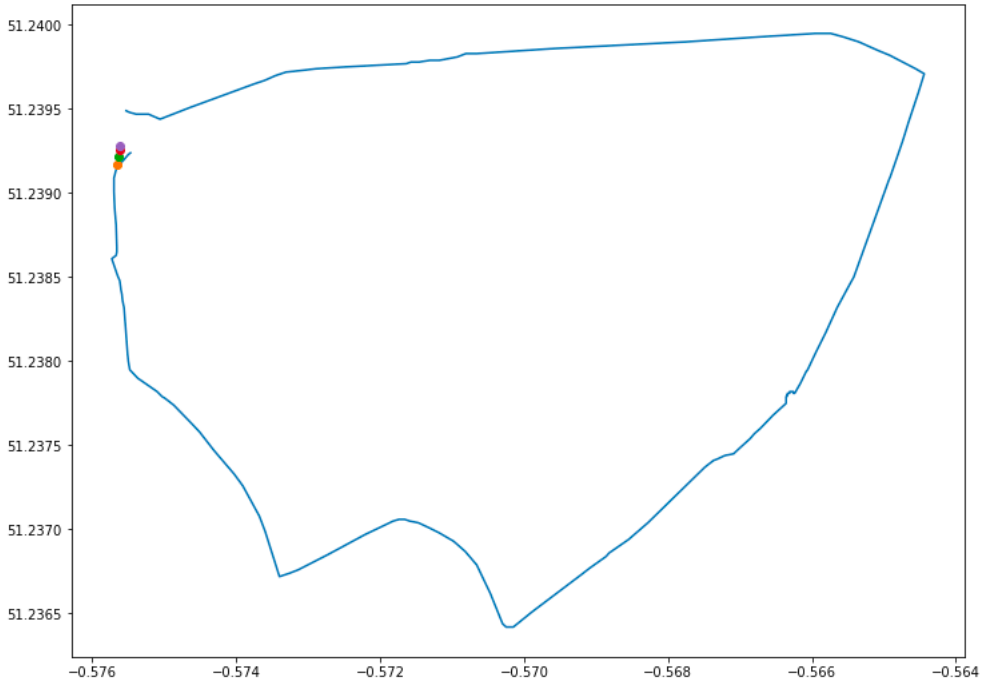
Testing model:

Univariate multi-step vector-output stacked lstm

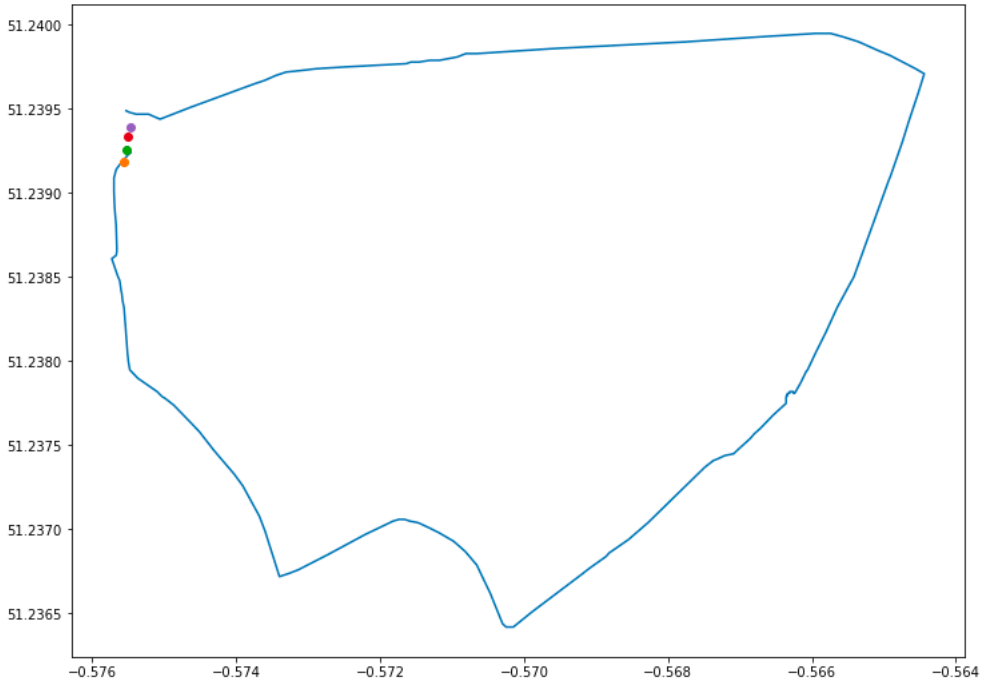

51.2400
51.2395
51.2390
51.2385
51.2380
51.2375
51.2370
51.2365
−0.576
−0.574
−0.572
−0.570
−0.568
−0.566
−0.564

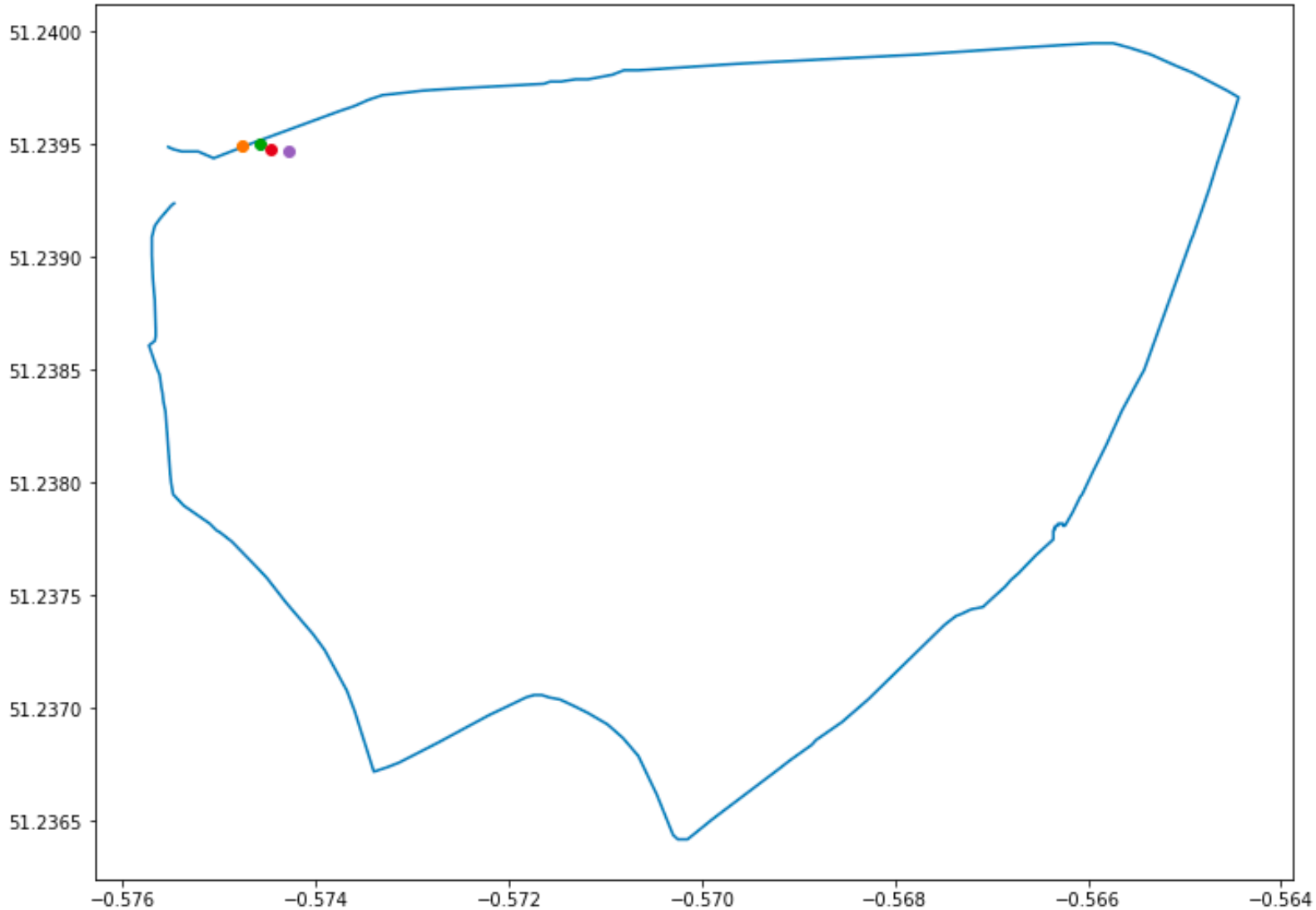
4 forward time steps

1st fitting models



2nd fitting models

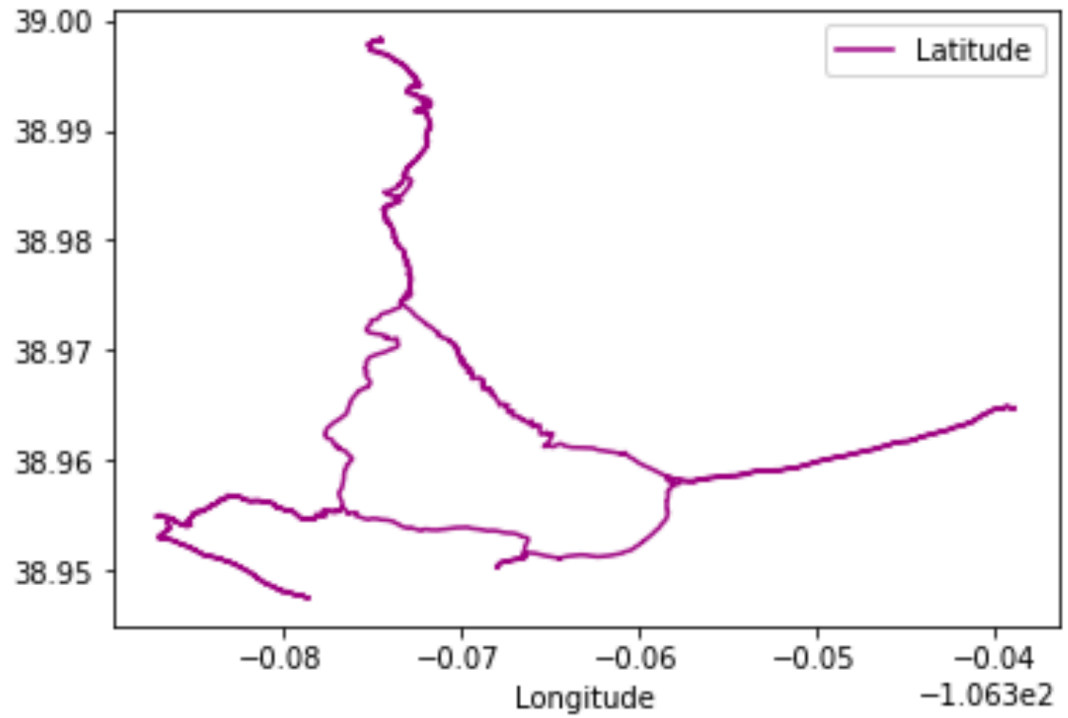



51.2400
51.2395
51.2390
51.2385
51.2380
51.2375
51.2370
51.2365
−0.576
−0.574
−0.572
−0.570
−0.568
−0.566
−0.564

Multivariate LSTM for predicting next UE’s position

```
File: Colorado_Belford_Oxford_and_Missouri_M
    Length 2D: 25.734km
    Length 3D: 26.388km
    Moving time: 09:45:17
    Stopped time: 02:48:40
    Max speed: 1.25m/s = 4.49km/h
    Avg speed: 0.71m/s = 2.56km/h
    Total uphill: 2369.21m
    Total downhill: 2368.91m
    Started: 2017-09-18 11:05:18+00:00
    Ended: 2017-09-18 23:39:15+00:00
    Points: 3121
    Avg distance between points: 8.25m

    Track #0, Segment #0
        Length 2D: 25.734km
        Length 3D: 26.388km
        Moving time: 09:45:17
        Stopped time: 02:48:40
        Max speed: 1.25m/s = 4.49km/h
        Avg speed: 0.71m/s = 2.56km/h
        Total uphill: 2369.21m
        Total downhill: 2368.91m
        Started: 2017-09-18 11:05:18+00:00
        Ended: 2017-09-18 23:39:15+00:00
        Points: 3121
        Avg distance between points: 8.25m
```

```
Name: Belford, Oxford, and Missouri Mountains (Hike)
Description: None
Latitude Bounds: (38.947555, 38.998279)
Longitude Bounds: (-106.387080, -106.338780)

<matplotlib.axes._subplots.AxesSubplot at 0x7ff2af43d110>

<Figure size 864x648 with 0 Axes>
```



Data preparation

	Longitude	Latitude	Time	Speed	Time_diff	Speed2	Haversine_Distance
0	-106.374451	38.998186	2017-09-18 11:05:18+00:00	0.132414	0.0	0.000000	0.000000
1	-106.374427	38.998168	2017-09-18 11:05:40+00:00	0.163309	22.0	0.131013	2.882284
2	-106.374374	38.998203	2017-09-18 11:06:11+00:00	0.143120	31.0	0.193881	6.010296
3	-106.374389	38.998183	2017-09-18 11:06:39+00:00	0.090192	28.0	0.091932	2.574105
4	-106.374400	38.998160	2017-09-18 11:07:10+00:00	0.068097	31.0	0.088014	2.728435
5	-106.374393	38.998177	2017-09-18 11:07:52+00:00	0.050194	42.0	0.047256	1.984748
6	-106.374386	38.998198	2017-09-18 11:08:38+00:00	0.077996	46.0	0.052439	2.412178

Series to supervised

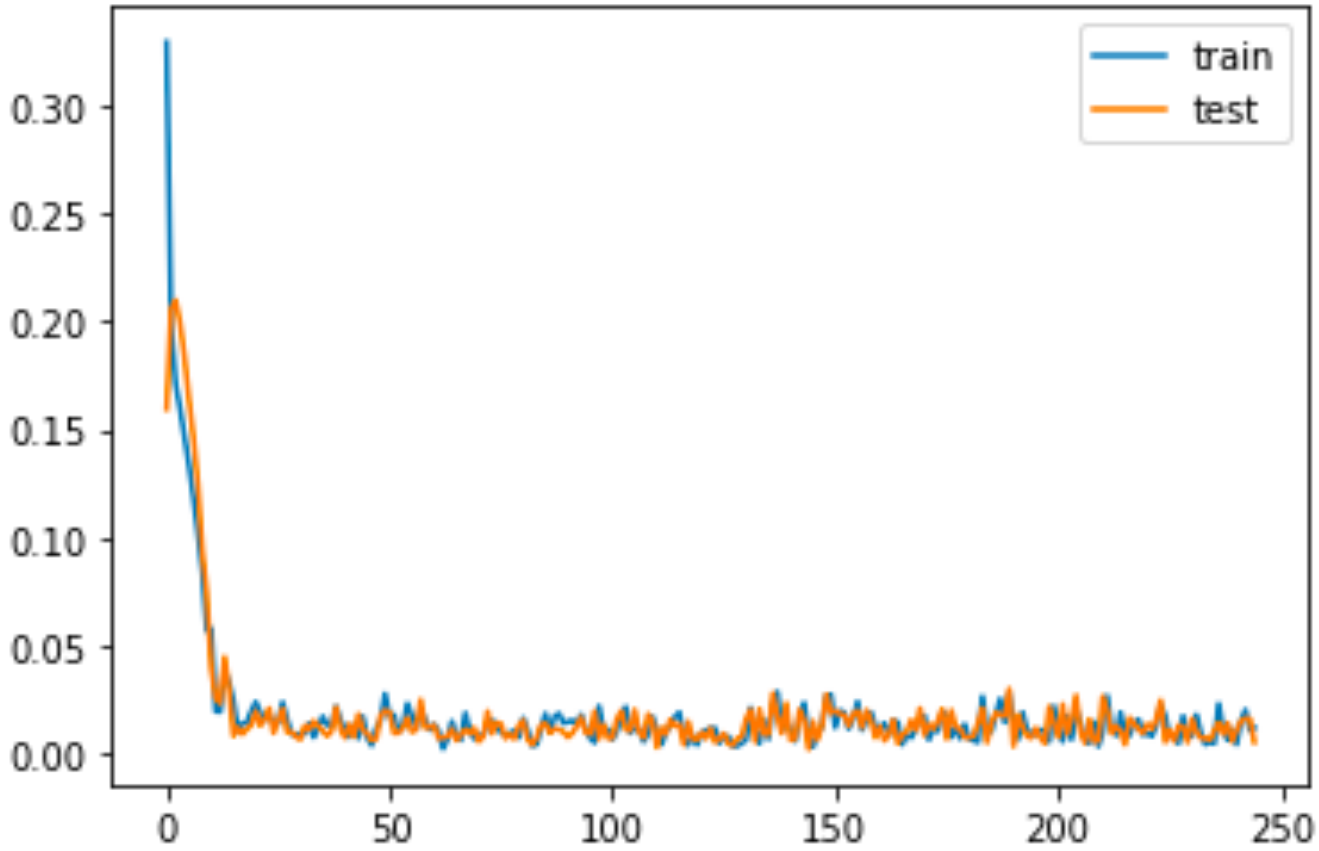
	Input		Output	
	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	-106.374451	38.998184	-106.374428	38.998169
2	-106.374428	38.998169	-106.374374	38.998203
3	-106.374374	38.998203	-106.374390	38.998184
4	-106.374390	38.998184	-106.374397	38.998161
5	-106.374397	38.998161	-106.374390	38.998177

LSTM model

```
# design network
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',mode='auto', patience=100,restore_best_weights=True)
model = Sequential()
model.add(LSTM(150, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(2))
model.compile(loss='mae', optimizer='adam',metrics=['accuracy'])
# fit network
history = model.fit(train_X, train_y, epochs=1000, callbacks=[callback], batch_size=72, validation_data=(test_X, test_y),
verbose=2, shuffle=False)

# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

Loss plot

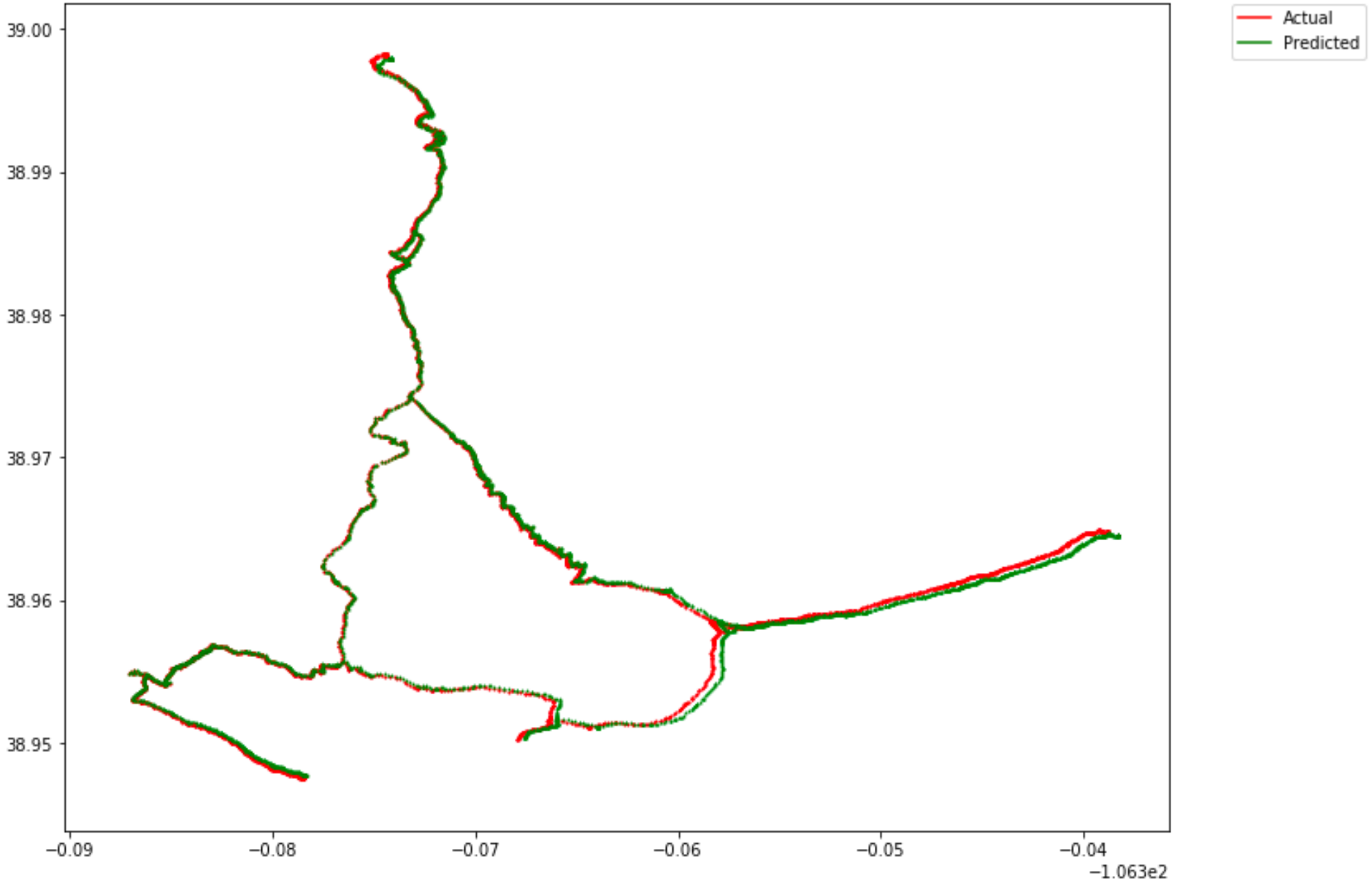
Epoch 245/1000 31/31 - 0s - loss: 0.0118 - accuracy: 0.9863 val_loss: 0.0051 - val_accuracy: 0.9957



```
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y2[:,0], inv_yhat2[:,0]))
print('RMSE for Longitude in Train dataset: %.6f' % rmse)
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y2[:,1], inv_yhat2[:,1]))
print('RMSE for Latitude in Train dataset: %.6f' % rmse)

# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y[:,0], inv_yhat[:,0]))
print('RMSE for Longitude in Test dataset: %.6f' % rmse)
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y[:,1], inv_yhat[:,1]))
print('RMSE for Latitude in Test dataset: %.6f' % rmse)
```

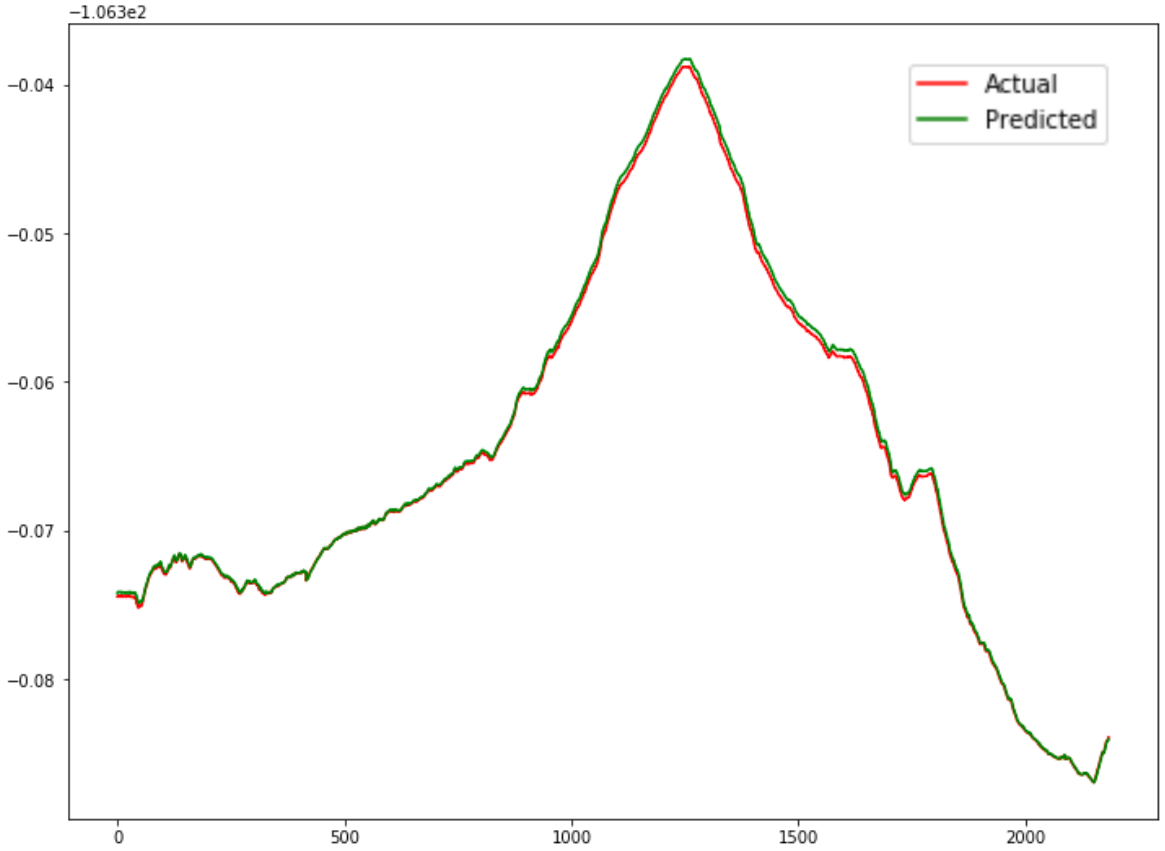
```
RMSE for Longitude in Train dataset: 0.000336
RMSE for Latitude in Train dataset: 0.000124
RMSE for Longitude in Test dataset: 0.000102
RMSE for Latitude in Test dataset: 0.000138
```

Compare the predicted values with actual values

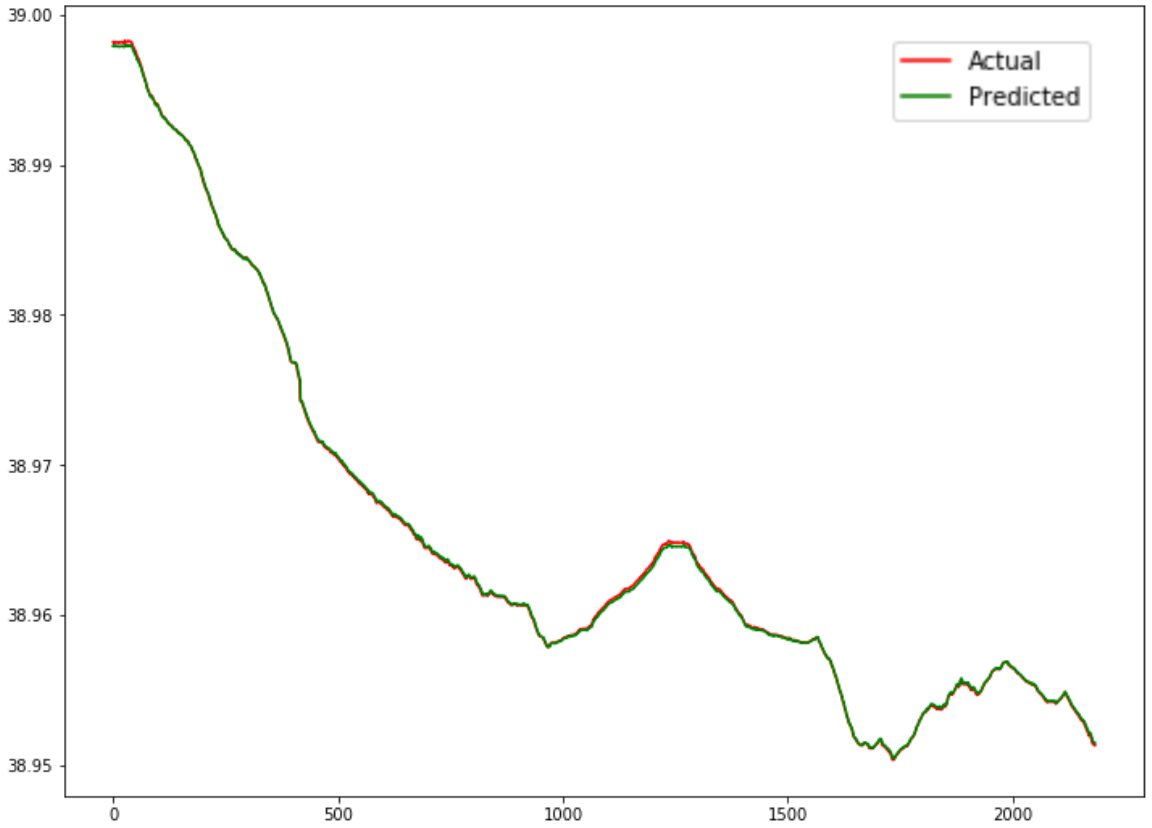


Train dataset

Longitude

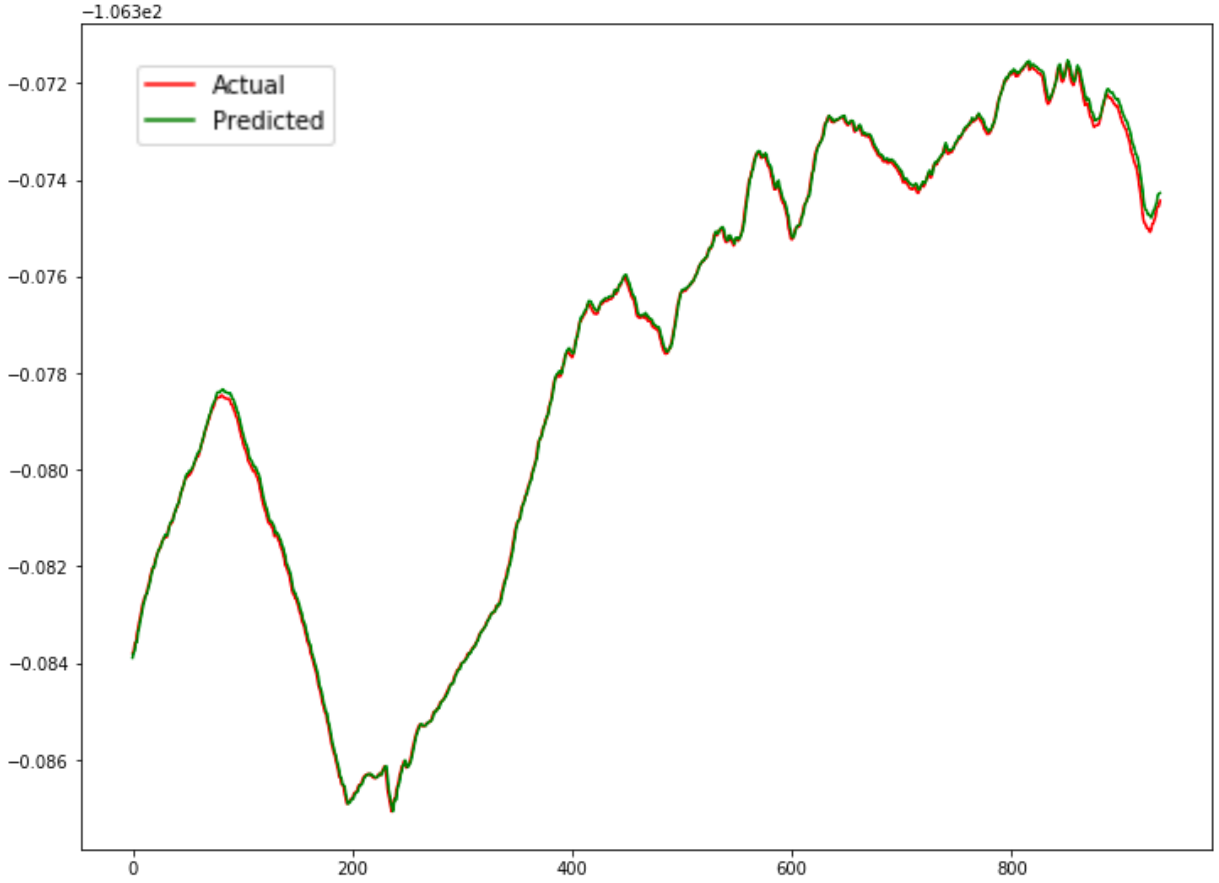


Latitude

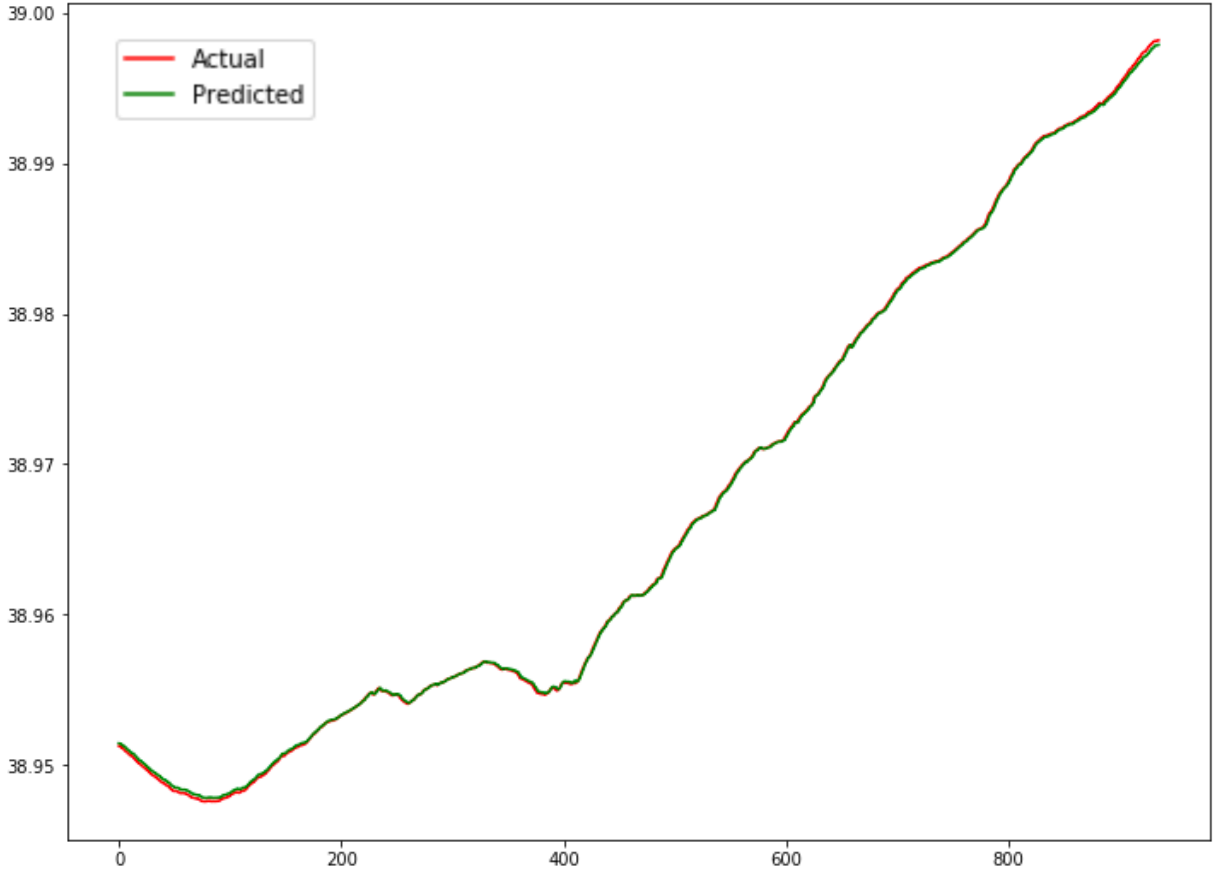


Test dataset

Longitude



Latitude



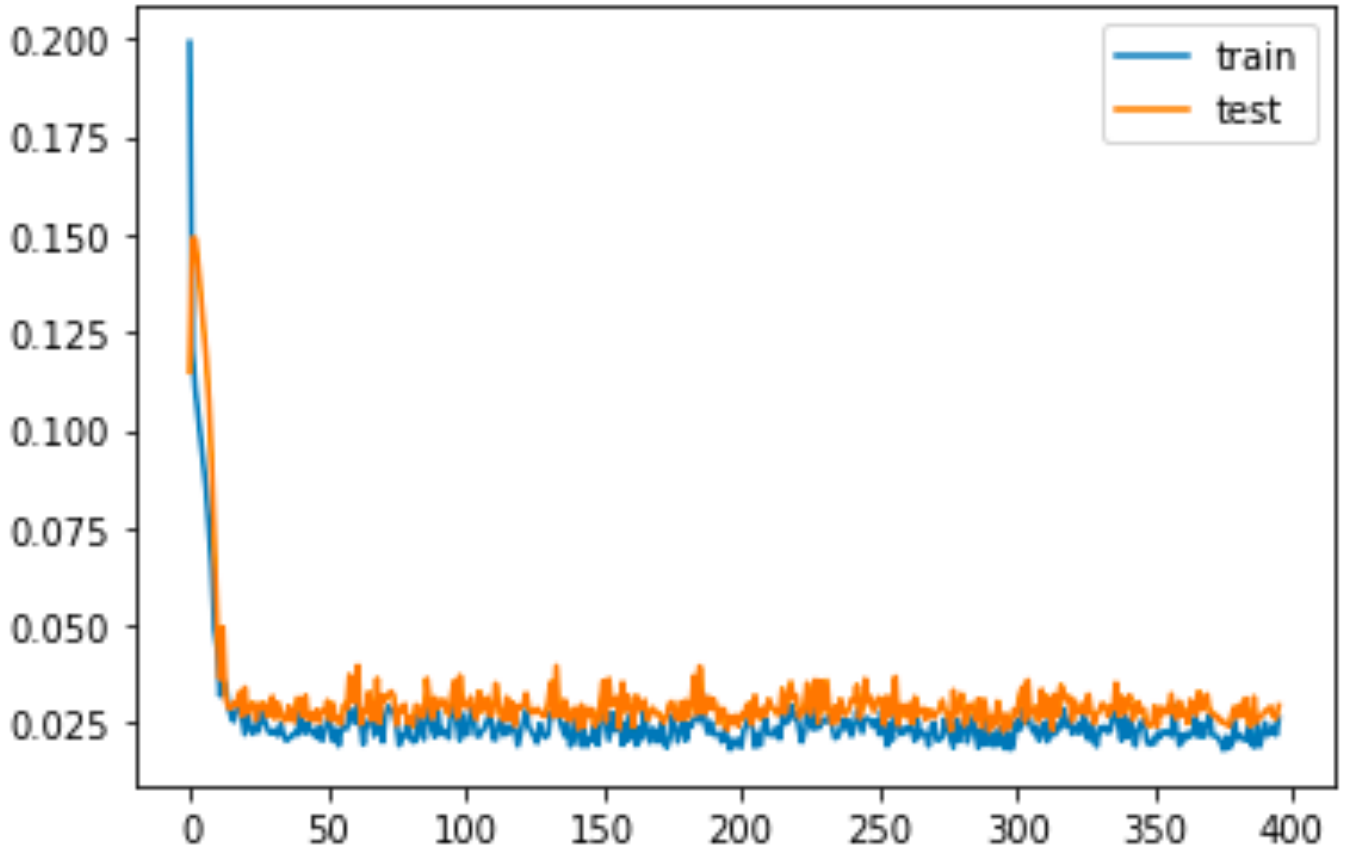
Series to supervised

Longitude	Latitude	Speed	Time_diff

```
    var1(t-1)  var2(t-1)  var3(t-1)  var4(t-1)     var1(t)    var2(t)  \
1 -106.374451  38.998184   0.000000        0.0 -106.374428  38.998169
2 -106.374428  38.998169   0.131013       22.0 -106.374374  38.998203
3 -106.374374  38.998203   0.193881       31.0 -106.374390  38.998184
4 -106.374390  38.998184   0.091932       28.0 -106.374397  38.998161
5 -106.374397  38.998161   0.088014       31.0 -106.374390  38.998177

    var3(t)  var4(t)
1  0.131013     22.0
2  0.193881     31.0
3  0.091932     28.0
4  0.088014     31.0
5  0.047256     42.0
```

Loss plot

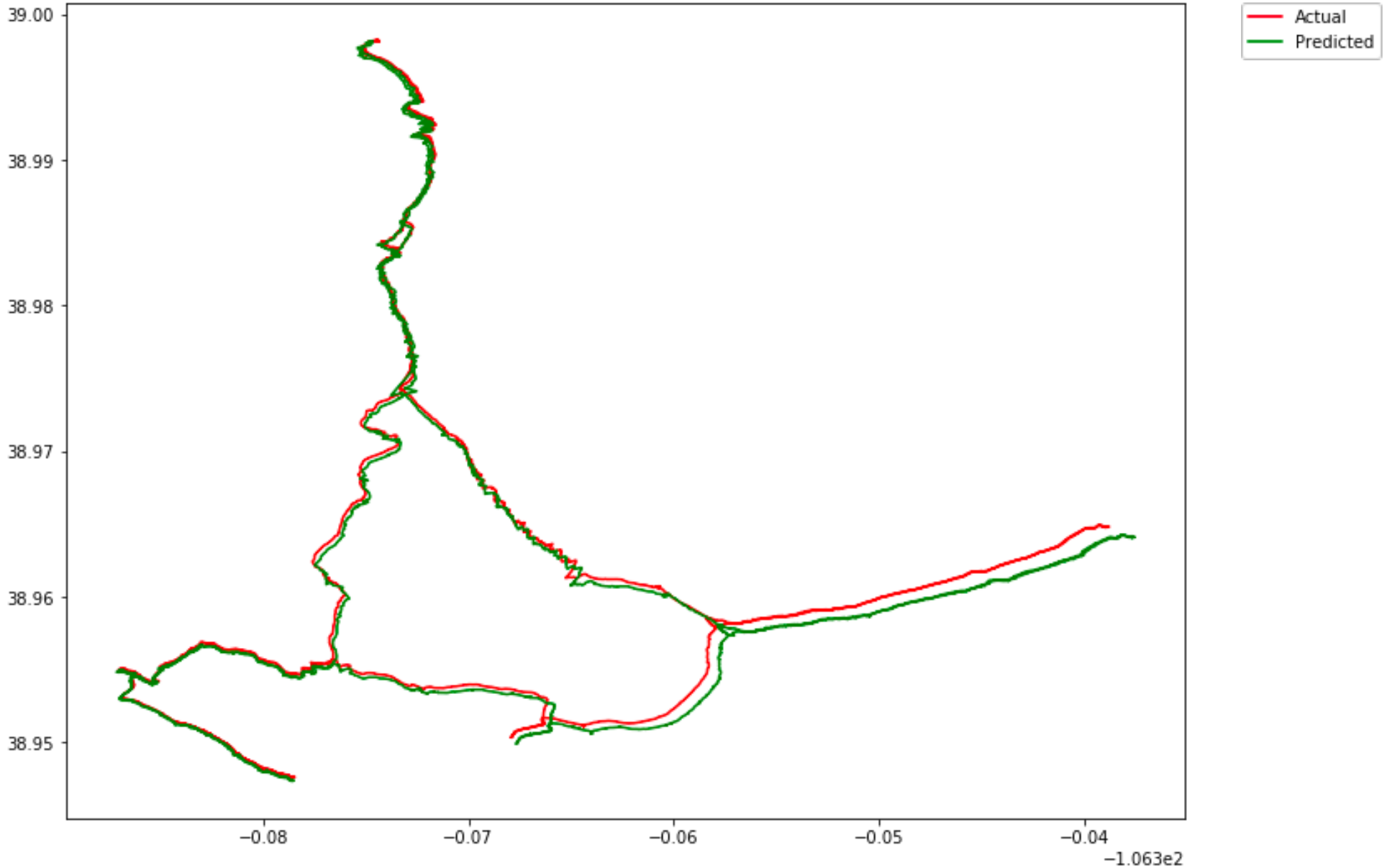
Epoch 396/1000 31/31 - 0s - loss: 0.0263 - accuracy: 0.9203 0.9896 - val_loss: 0.0296 - val_accuracy: 0.8761



RMSE for Train and test dataset

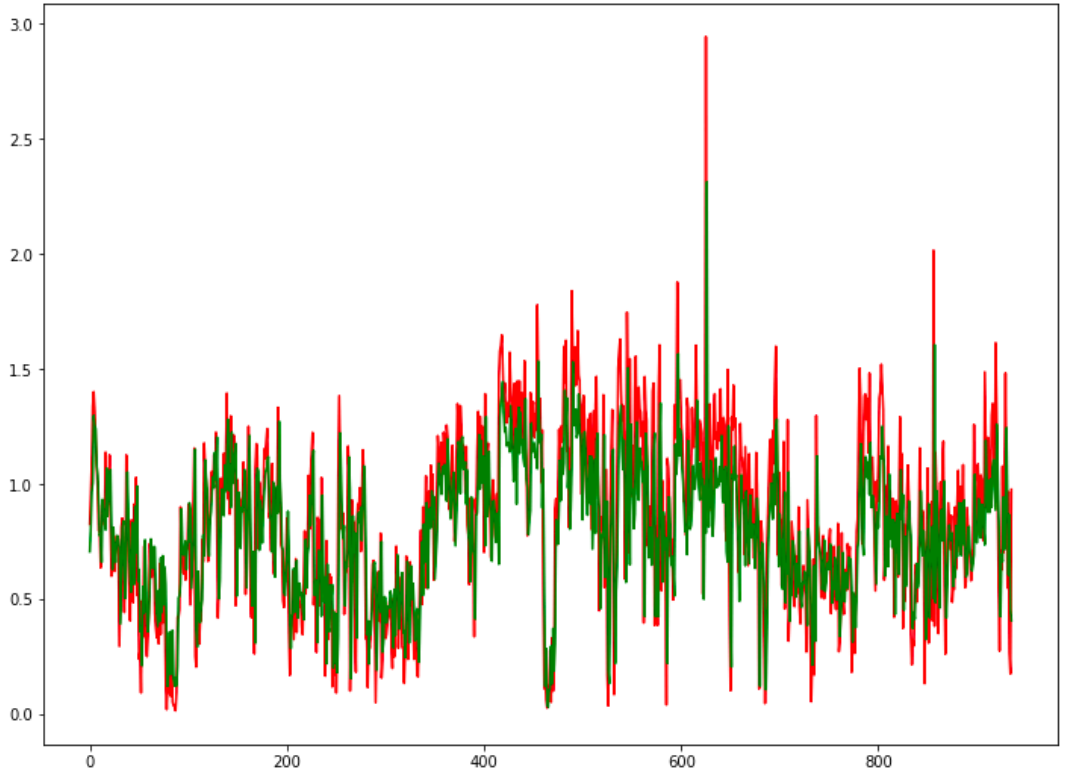
```
RMSE for Longitude in Train dataset: 0.000527
RMSE for Latitude in Train dataset: 0.000421
RMSE for Longitude in Test dataset: 0.000135
RMSE for Latitude in Test dataset: 0.000242
RMSE for speed in Test dataset: 0.297497
RMSE for speed in Train dataset: 0.221189
RMSE for time_diff in Test dataset: 5.016663
RMSE for Time_diff in Train dataset: 8.607296
```

Compare the predicted values with actual values

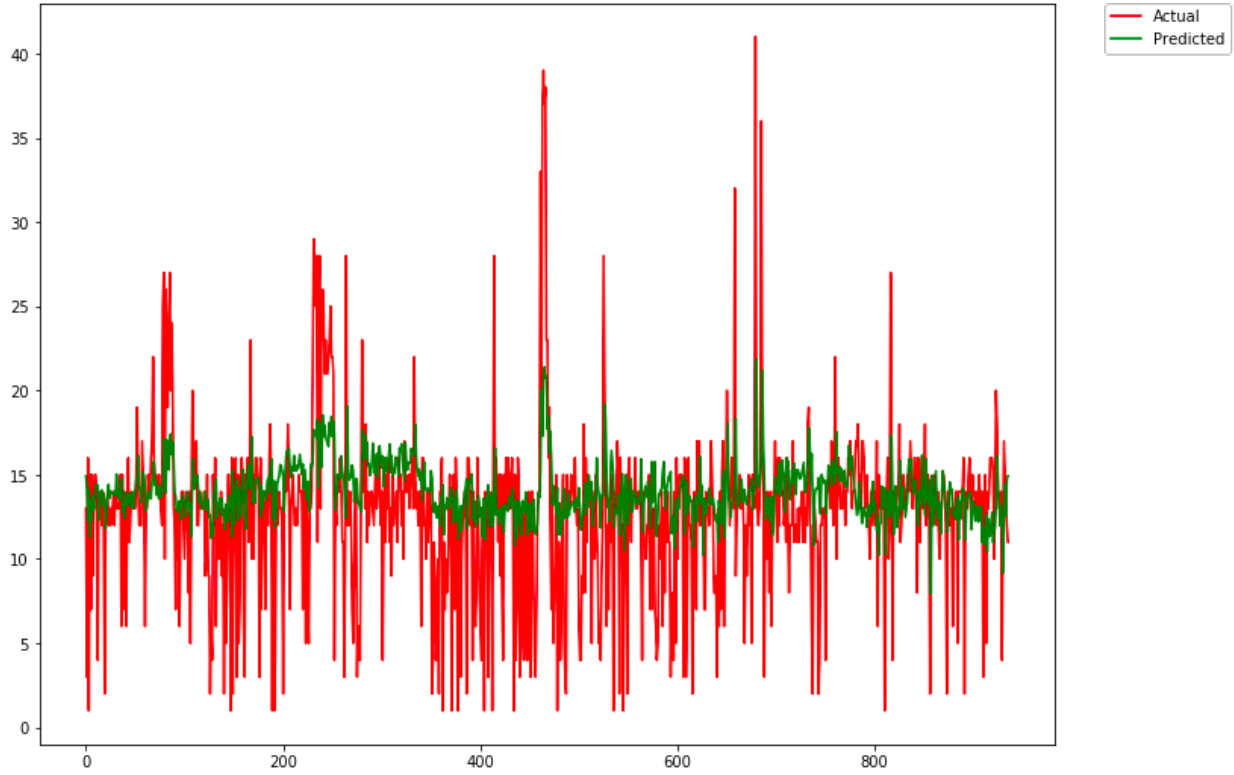


Speed and Time diff

Speed



Time diff



Further research

- Stacked LSTM model
- Bidirectional LSTM
- CNN LSTM
- ConvLSTM
- Multi-Step LSTM

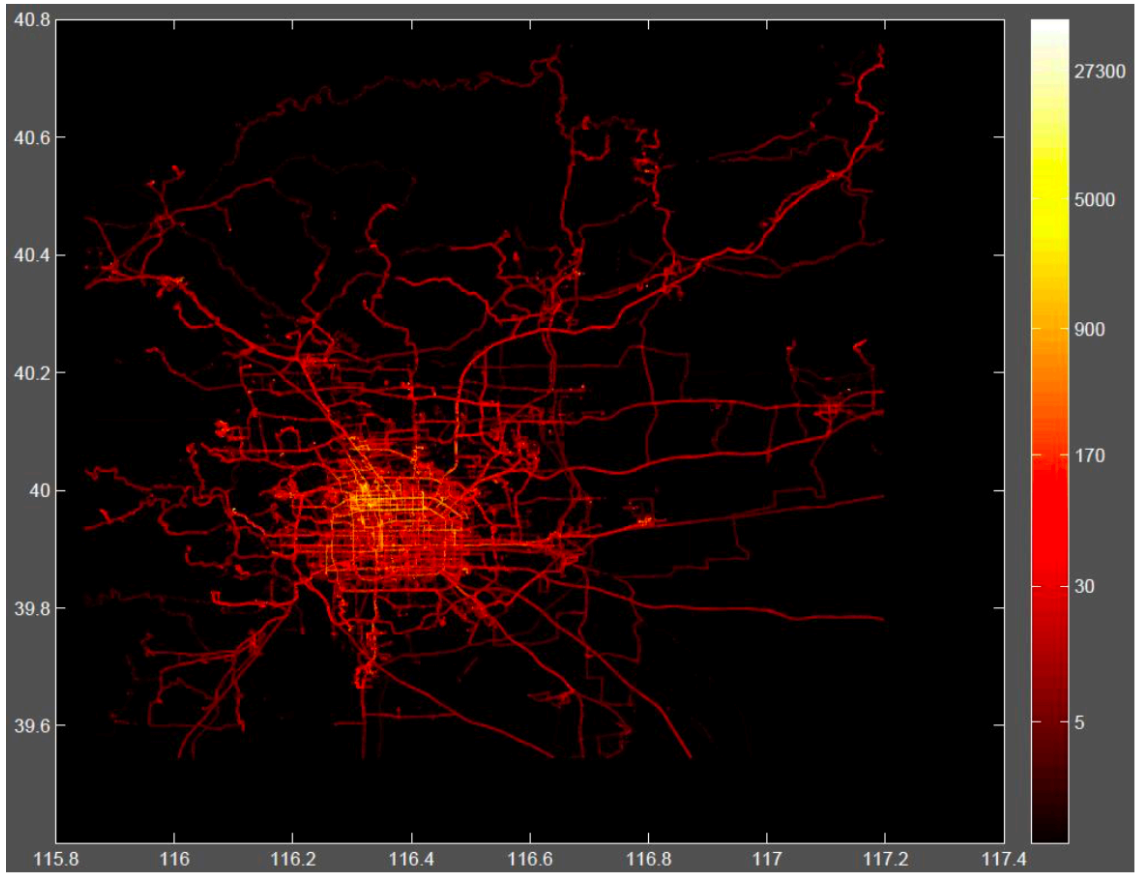
Multivariate LSTM for predicting next UE’s position

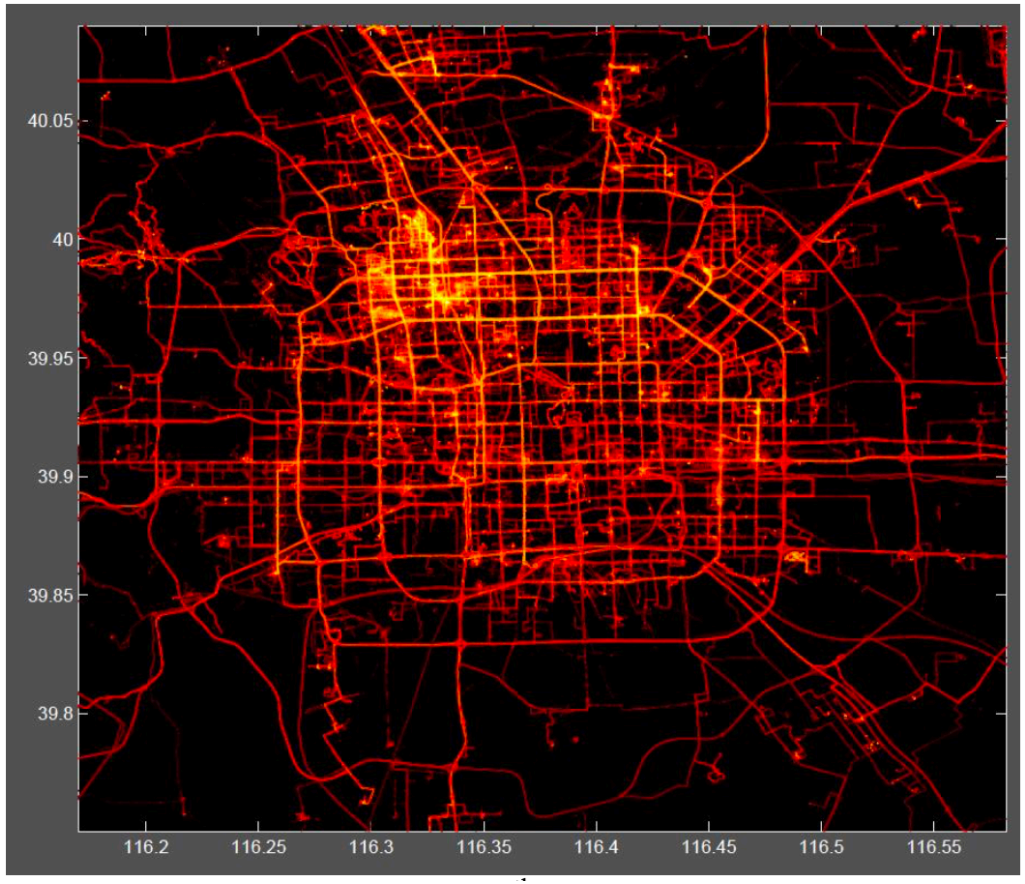
GEOLIFE

Data Description

- GPS trajectory dataset was collected in (Microsoft Research Asia)
- 182 users in a period of over five years (from April 2007 to August 2012).
- Contains the information of latitude, longitude, altitude, timestamp and labels.
- Contains 17,621 trajectories with a total distance of 1,292,951kilometers and a total duration of 50,176 hours.
- Recoded a broad range of users' outdoor movements, including not only life routines like go home and go to work but also some entertainments and sports activities, such as shopping, sightseeing, dining, hiking, and cycling.

Distribution (heat map) of this dataset in Beijing





Transportation mode labels

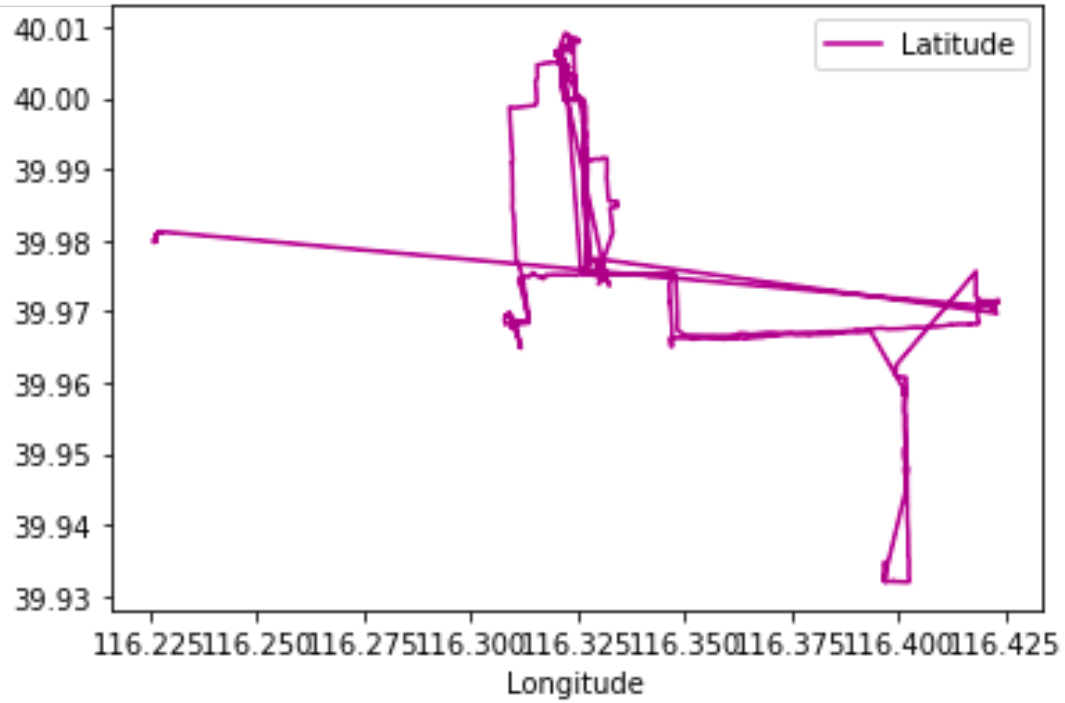
Transportation mode	Distance (km)	Duration (hour)
Walk	10,123	5,460
Bike	6,495	2,410
Bus	20,281	1,507
Car & taxi	32,866	2,384
Train	36,253	745
Airplane	24,789	40
Other	9,493	404
Total	14,0304	12,953

Trajectory of user id: 167 in April

	Latitude	Longitude	Altitude	Date_Time	Id_user	Id_perc	Label	month	day	Time_diff
2975189	39.998423	116.326564	492.0	2008-04-29 00:25:35	167	20080428183304.plt	bike	4	29	NaN
2975190	39.998268	116.326476	490.0	2008-04-29 00:25:37	167	20080428183304.plt	bike	4	29	2.0
2975191	39.998183	116.326481	488.0	2008-04-29 00:25:39	167	20080428183304.plt	bike	4	29	2.0
2975192	39.998149	116.326511	485.0	2008-04-29 00:25:41	167	20080428183304.plt	bike	4	29	2.0
2975193	39.998096	116.326513	483.0	2008-04-29 00:25:43	167	20080428183304.plt	bike	4	29	2.0

```
tr167.Label.value_counts()
```

```
bike     6054
bus      3465
walk     3318
Name: Label, dtype: int64
```



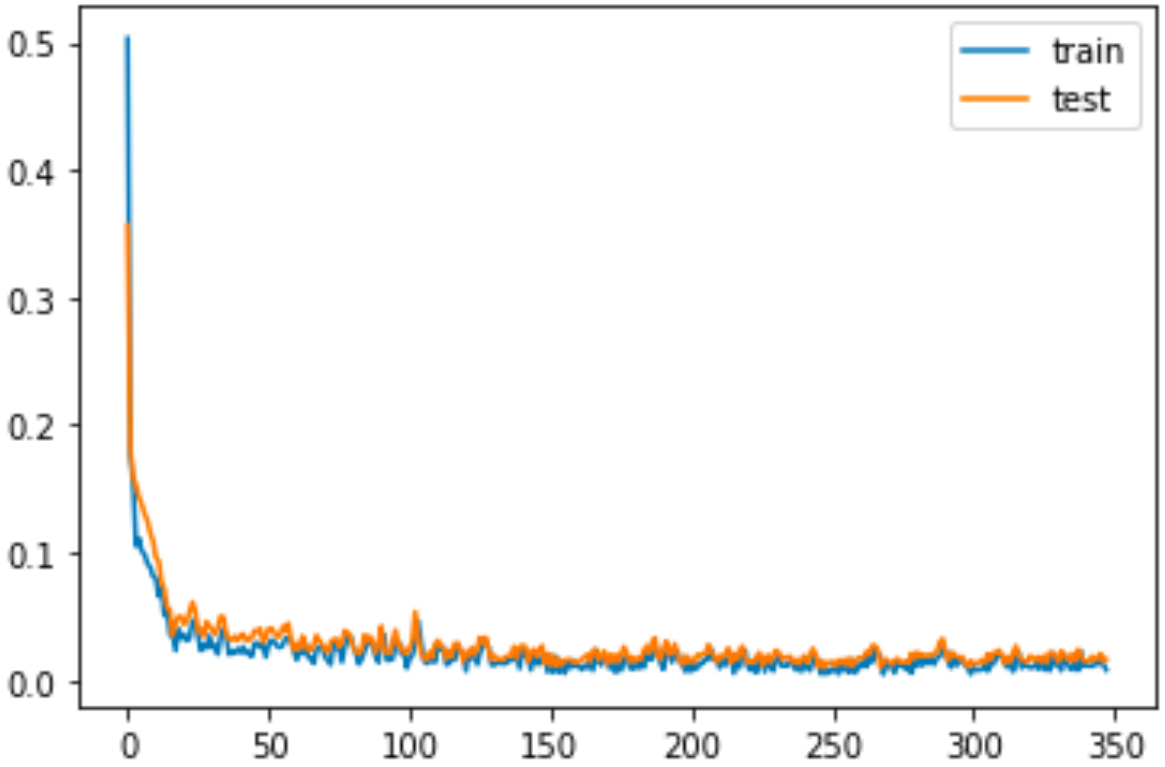
Multivariate LSTM for predicting next UE's position with 1 time step

Transform time series data to supervised problem

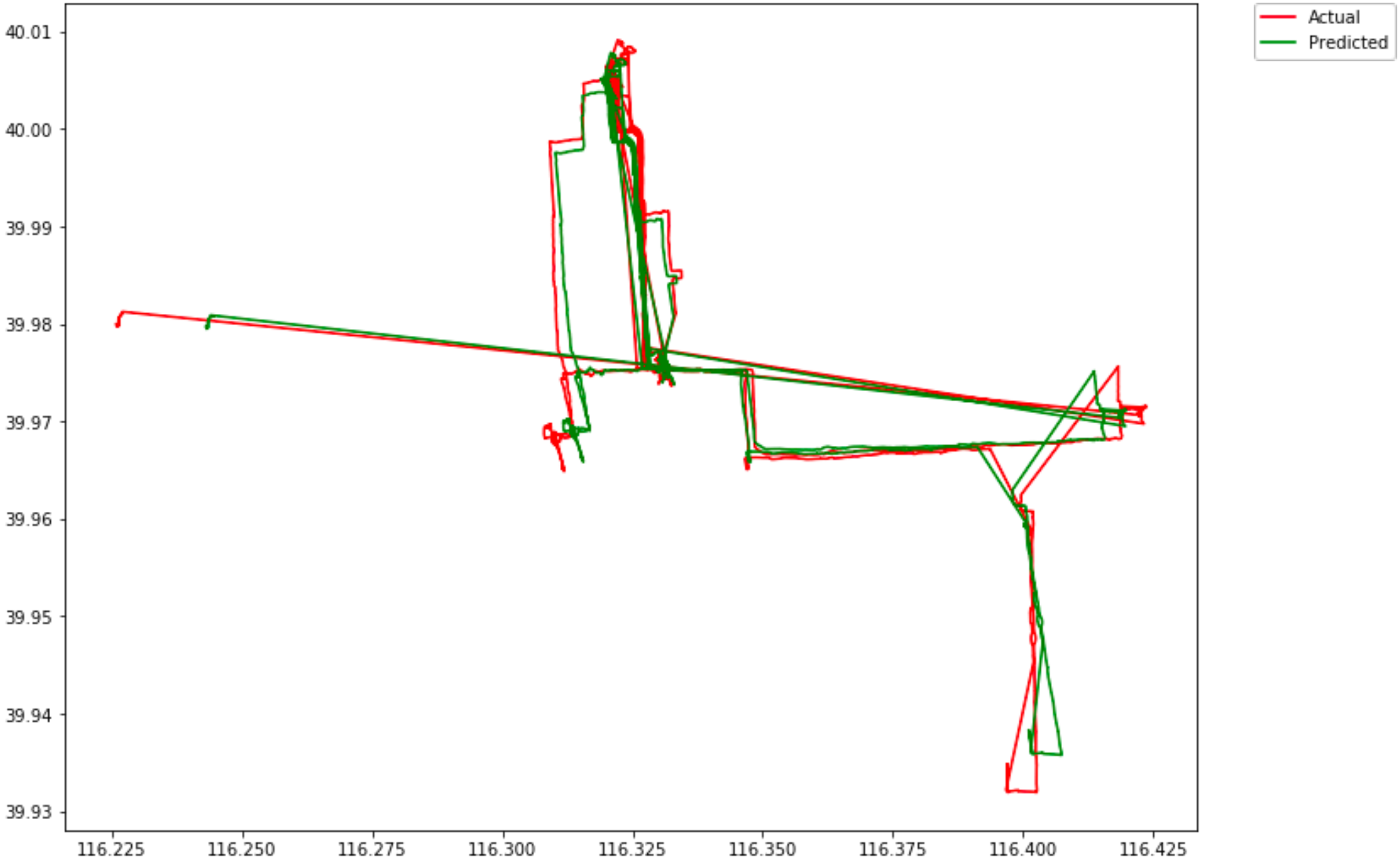
	Longitude	Latitude	Longitude	Latitude
	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	116.227852	39.981152	116.226997	39.980968
2	116.226997	39.980968	116.226501	39.980667
3	116.226501	39.980667	116.226463	39.979832
4	116.226463	39.979832	116.226051	39.979698
5	116.226051	39.979698	116.225868	39.980000

Loss plot and RMSE reports

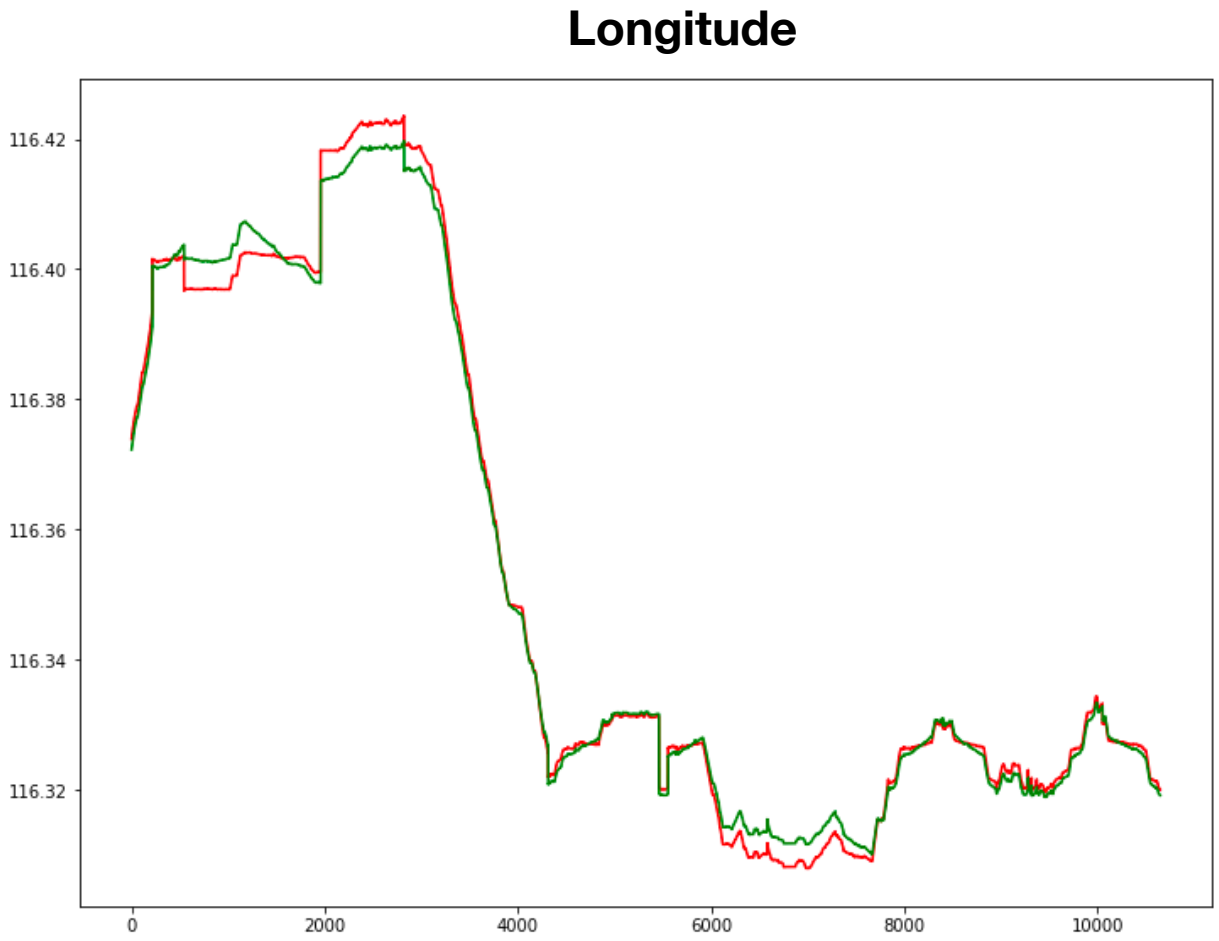
Epoch 348/1000 31/31 - 0s - loss: 0.0081 - accuracy: 0.9945 - val_loss: 0.0166 - val_accuracy: 0.9986

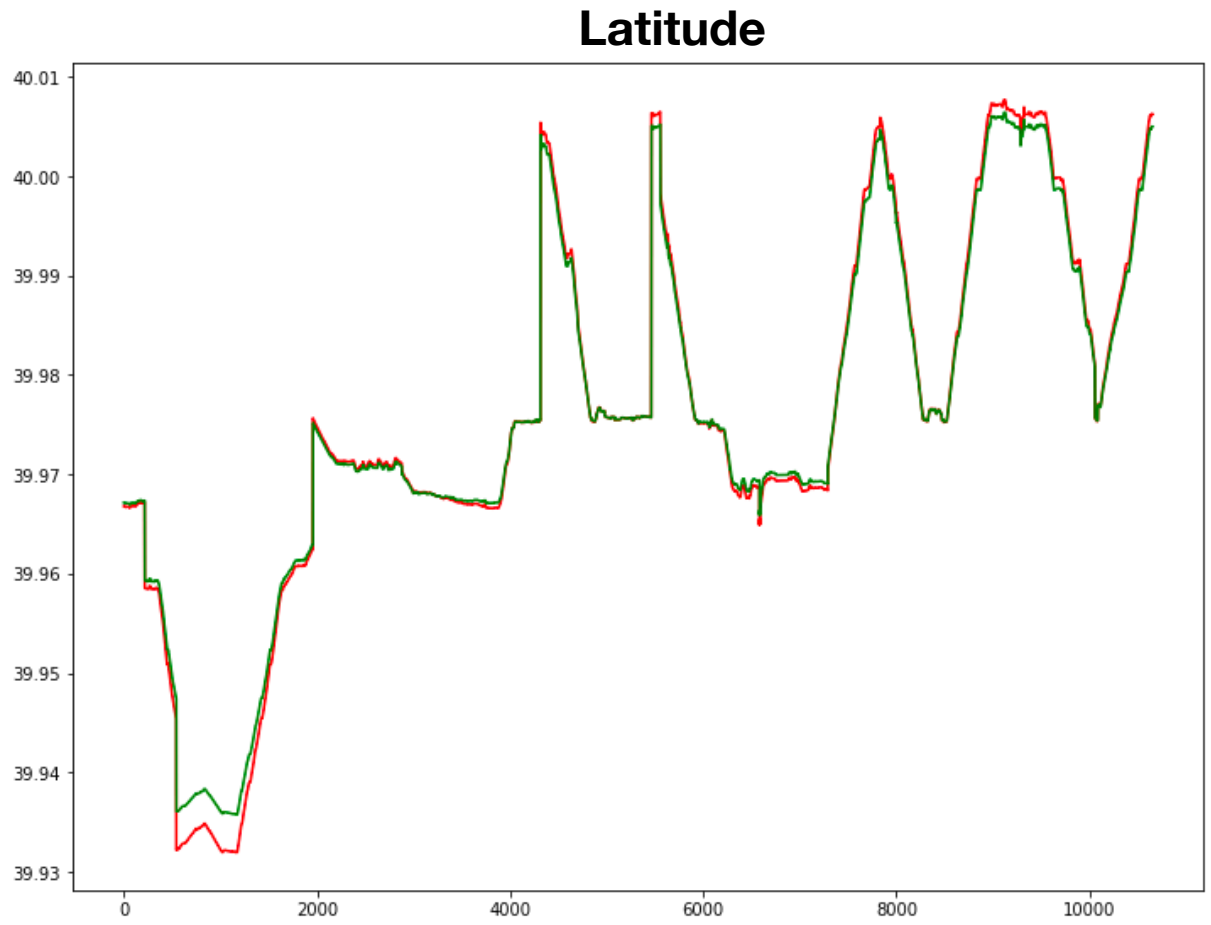


RMSE for Longitude in Train dataset: 0.004516
RMSE for Latitude in Train dataset: 0.000773
RMSE for Longitude in Test dataset: 0.002310
RMSE for Latitude in Test dataset: 0.001311

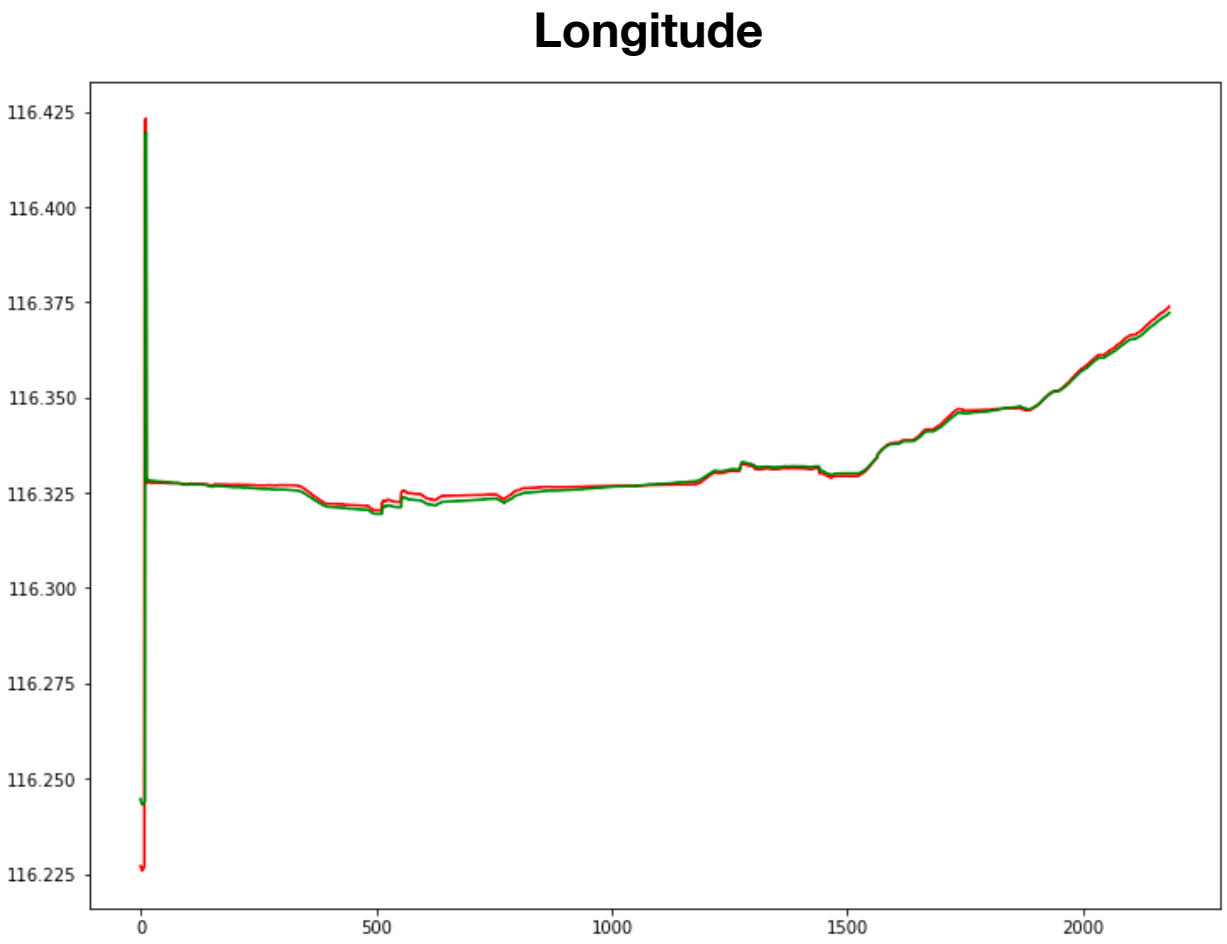

Actual
Predicted
40.01
40.00
39.99
39.98
39.97
39.96
39.95
39.94
39.93
116.225
116.250
116.275
116.300
116.325
116.350
116.375
116.400
116.425

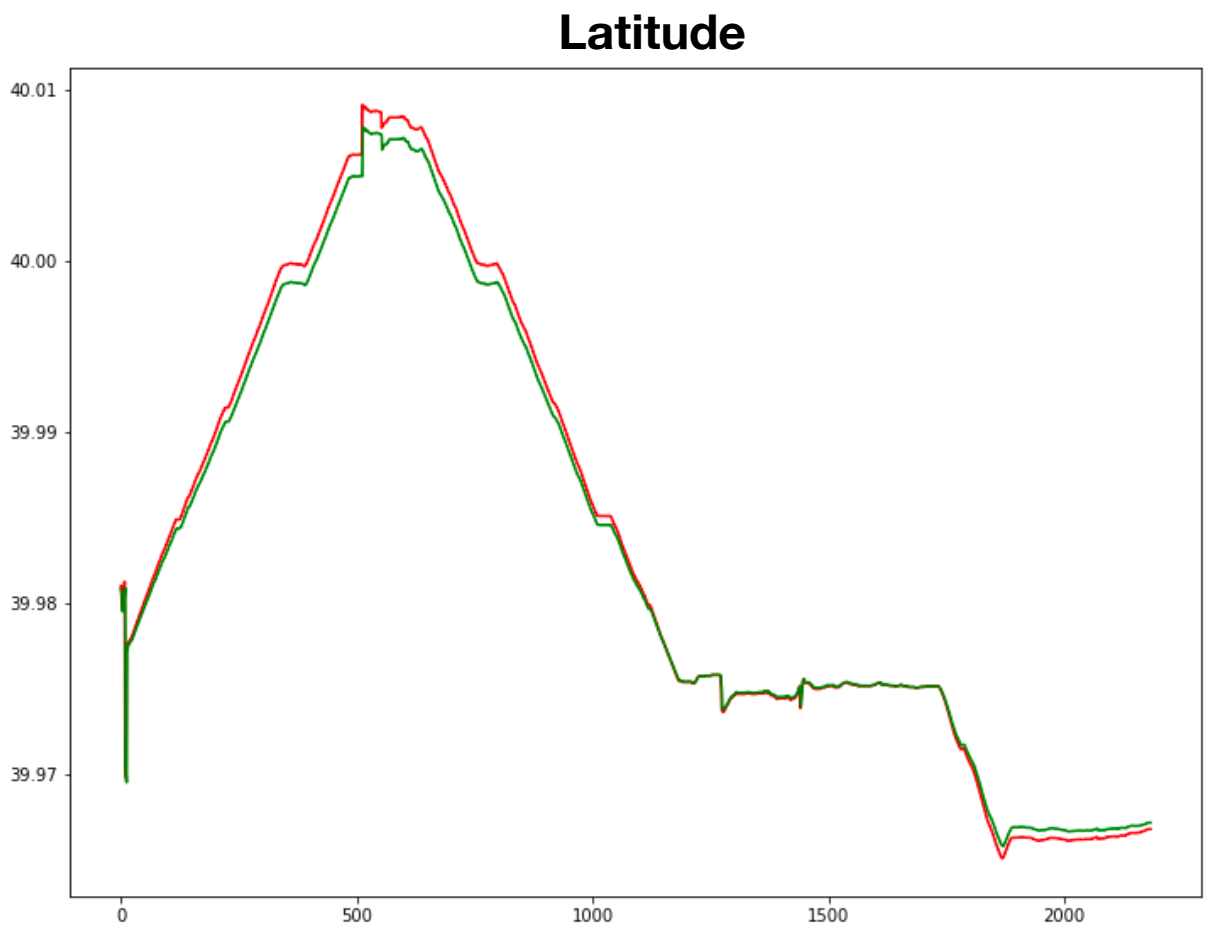
Train dataset





Test dataset

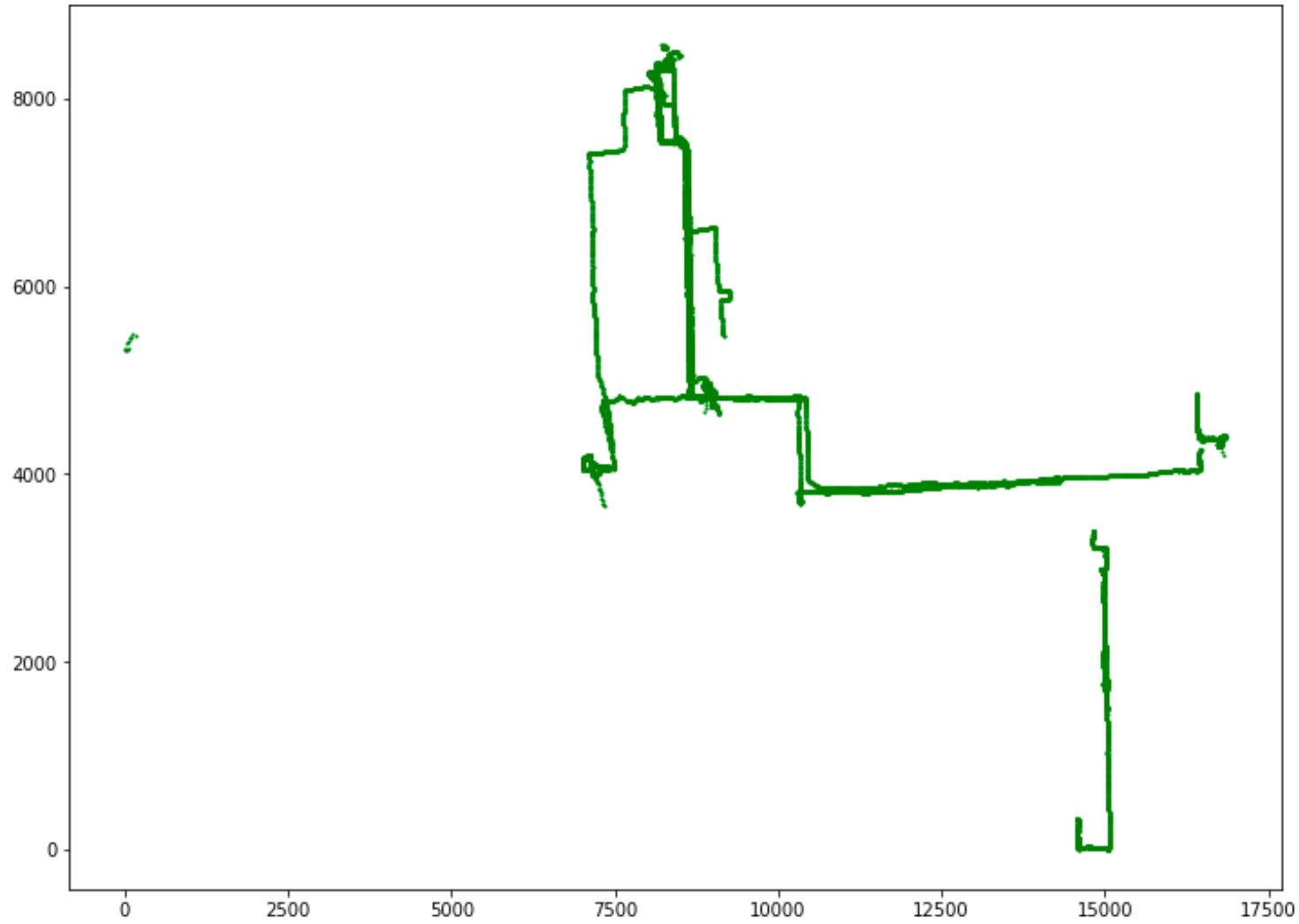




END

Trajectory of user id: 167 in April

	X	Y
0	169.0	5472.5
1	96.0	5452.0
2	54.0	5418.5
3	51.0	5325.5
4	16.0	5310.5
...	...	...
12832	8043.0	8267.0
12833	8042.0	8266.0
12834	8041.0	8265.0
12835	8039.0	8264.5
12836	8037.0	8262.5



Multivariate LSTM for predicting next UE's position with 1 time step

Transform time series data to supervised problem

	X (m)	Y (m)	X (m)	Y (m)
	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	169.0	5472.5	96.0	5452.0
2	96.0	5452.0	54.0	5418.5
3	54.0	5418.5	51.0	5325.5
4	51.0	5325.5	16.0	5310.5
5	16.0	5310.5	0.0	5344.0

Multivariate LSTM for predicting next UE's position with 1 time step

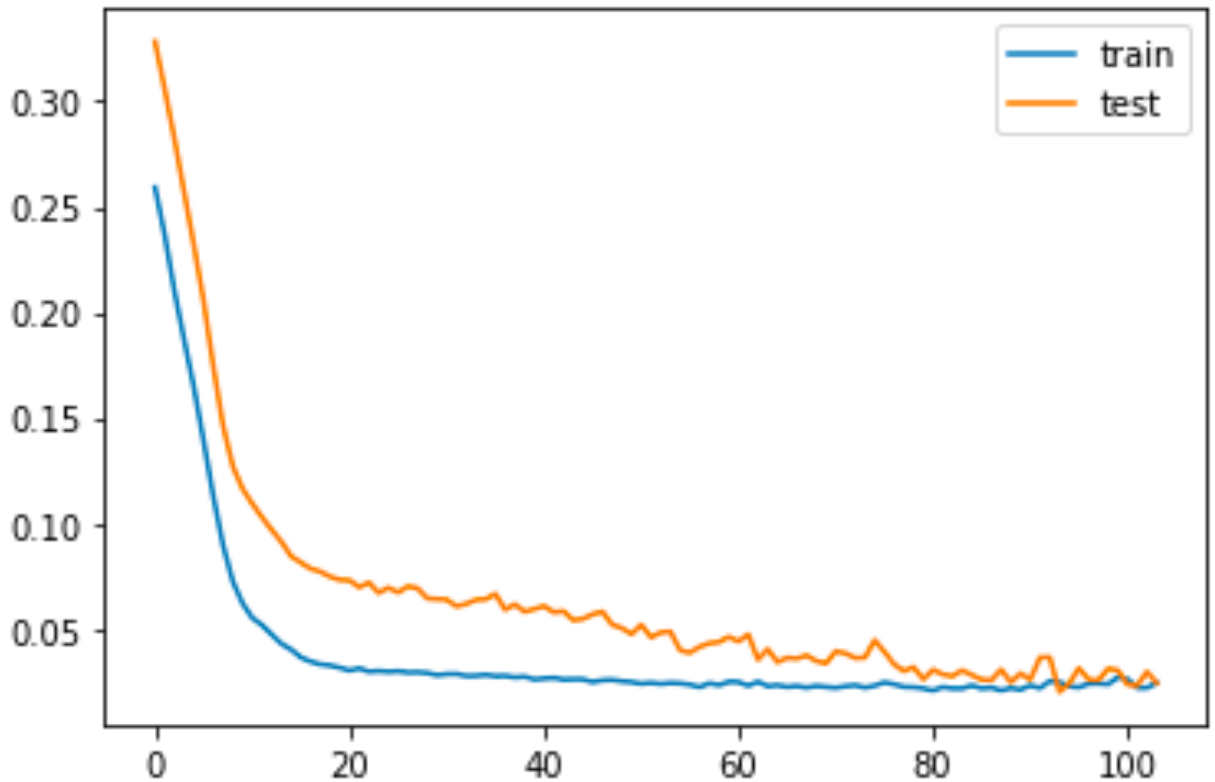
Transform time series data to supervised problem

	Feature Variables		**Target Variables**	
	X (m)	Y (m)	X (m)	Y (m)
	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	-0.979949	0.275715	-0.988610	0.270937
2	-0.988610	0.270937	-0.993593	0.263127
3	-0.993593	0.263127	-0.993949	0.241448
4	-0.993949	0.241448	-0.998102	0.237951
5	-0.998102	0.237951	-1.000000	0.245760

Single LSTM

```
# design network
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',mode='auto',
patience=10,restore_best_weights=True)
model = Sequential()
model.add(LSTM(64,activation='relu' ,input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dropout(0.1))
model.add(Dense(2))
model.compile(loss='mae', optimizer='adam',metrics=['accuracy','mean_squared_error'])
# fit network
history = model.fit(train_X, train_y, epochs=1000, callbacks=[callback], batch_size=72, validation_data=(test_X,
test_y), verbose=2, shuffle=False)
#history = model.fit(train_X, train_y, epochs=743, batch_size=72, validation_data=(test_X, test_y), verbose=2,
shuffle=False)
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

Epoch 104/1000 36/36 - 0s - loss: 0.0243 - accuracy: 0.9910 - val_loss: 0.0243 - val_accuracy: 0.9965



Stacked LSTM

```
# design network
callback = tf.keras.callbacks.EarlyStopping(monitor='loss',mode='auto', patience=10,restore_best_weights=True)

model = Sequential()

# first layer
model.add(LSTM(64, input_shape=(train_X.shape[1], train_X.shape[2]),return_sequences=True))
#model.add(Dropout(0.1))

# second layer
model.add(LSTM(units=128,return_sequences=False))
model.add(Dropout(0.1))

# Third dense layer
model.add(Dense(units=2))

model.compile(loss='mae', optimizer='adam',metrics=['accuracy'])

history = model.fit(train_X, train_y, epochs=600, callbacks=[callback], batch_size=72, validation_data=(test_X, test_y), verbose=2, shuffle=False)

model.summary()
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

Bidirectional LSTM

```
from keras.layers import Bidirectional

# design network
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',mode='auto', patience=10,restore_best_weights=True)
model = Sequential()
model.add(Bidirectional(LSTM(150, input_shape=(train_X.shape[1], train_X.shape[2]))))
model.add(Dropout(0.1))
model.add(Dense(2))
model.compile(loss='mae', optimizer='adam',metrics=['accuracy'])
# fit network
history = model.fit(train_X, train_y, epochs=1000, callbacks=[callback], batch_size=72, validation_data=(test_X, test_y),
verbose=2, shuffle=False)
#history = model.fit(train_X, train_y, epochs=743, batch_size=72, validation_data=(test_X, test_y), verbose=2, shuffle=False)

# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

GRU LSTM

```
from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional

# design network
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',mode='auto', patience=10,restore_best_weights=True)
model = Sequential()
model.add(GRU(64, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dropout(0.1))
model.add(Dense(2))
model.compile(loss='mae', optimizer='adam',metrics=['accuracy'])
# fit network
history = model.fit(train_X, train_y, epochs=1000, callbacks=[callback], batch_size=72, validation_data=(test_X, test_y),
verbose=2, shuffle=False)
#history = model.fit(train_X, train_y, epochs=743, batch_size=72, validation_data=(test_X, test_y), verbose=2,
shuffle=False)

# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

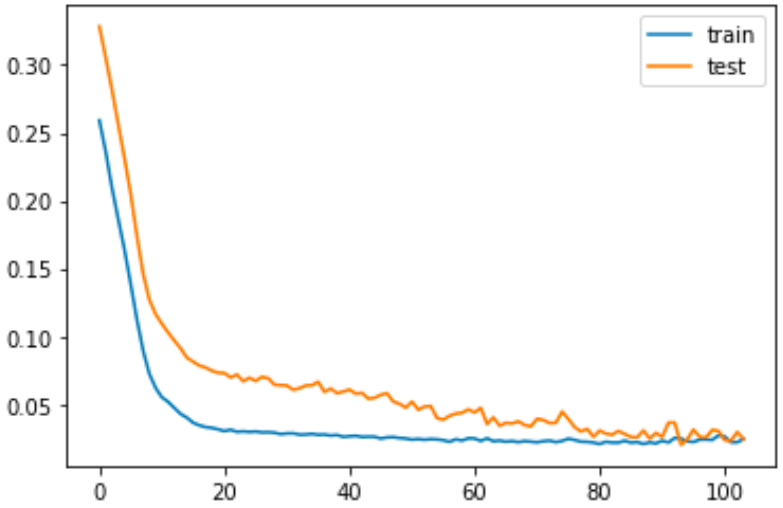
Stacked GRU

```
from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
# design network
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',mode='auto', patience=10,restore_best_weights=True)
model = Sequential()
model.add(GRU(32,return_sequences=True, activation='tanh', input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(GRU(units=64, activation='tanh',return_sequences=False))
model.add(Dropout(0.1))
model.add(Dense(2))
model.compile(loss='mae', optimizer='adam',metrics=['accuracy'])
# fit network
history = model.fit(train_X, train_y, epochs=1000, callbacks=[callback], batch_size=72, validation_data=(test_X,
test_y), verbose=2, shuffle=False)
#history = model.fit(train_X, train_y, epochs=743, batch_size=72, validation_data=(test_X, test_y), verbose=2,
shuffle=False)

# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

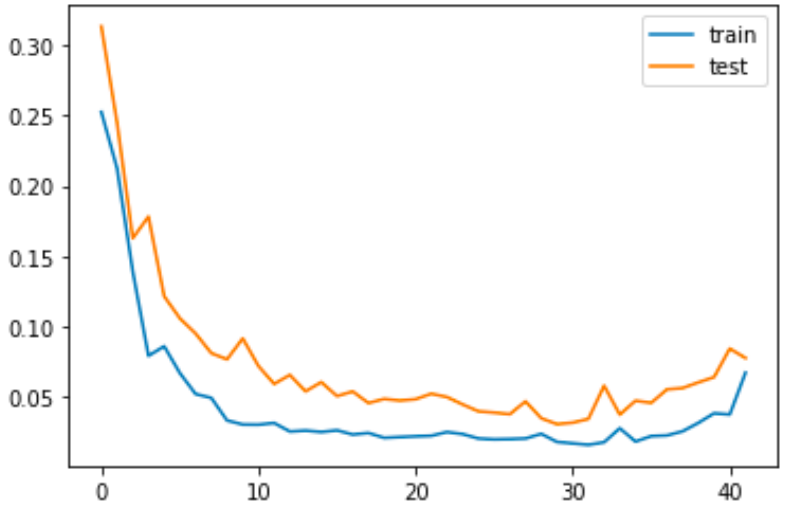
Simple RNN

```
# design network
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',mode='auto', patience=10,restore_best_weights=True)
model = Sequential()
model.add(layers.SimpleRNN(64, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dropout(0.1))
model.add(Dense(2))
model.compile(loss='mae', optimizer='adam',metrics=['accuracy'])
# fit network
history = model.fit(train_X, train_y, epochs=1000, callbacks=[callback], batch_size=72, validation_data=(test_X, test_y), verbose=2, shuffle=False)
#history = model.fit(train_X, train_y, epochs=743, batch_size=72, validation_data=(test_X, test_y), verbose=2, shuffle=False)

# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

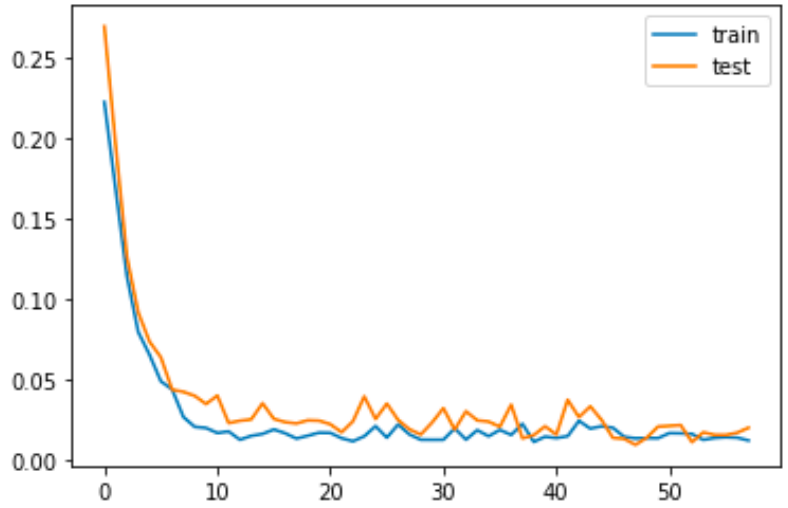
LSTM



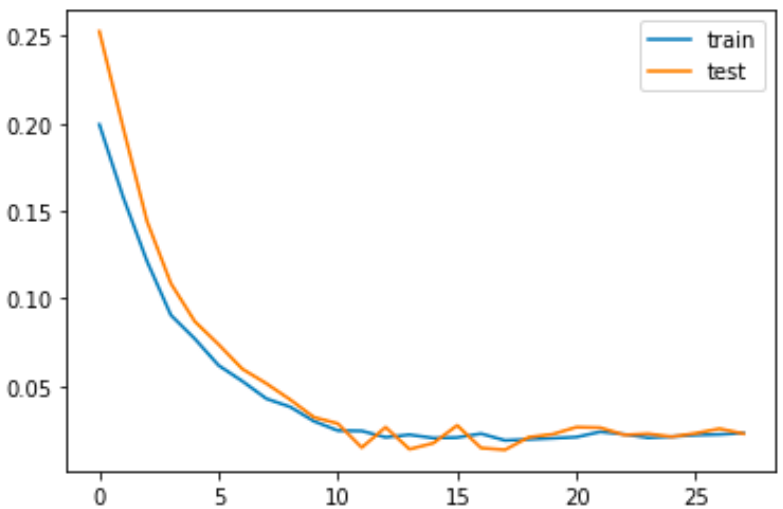
Stacked LSTM



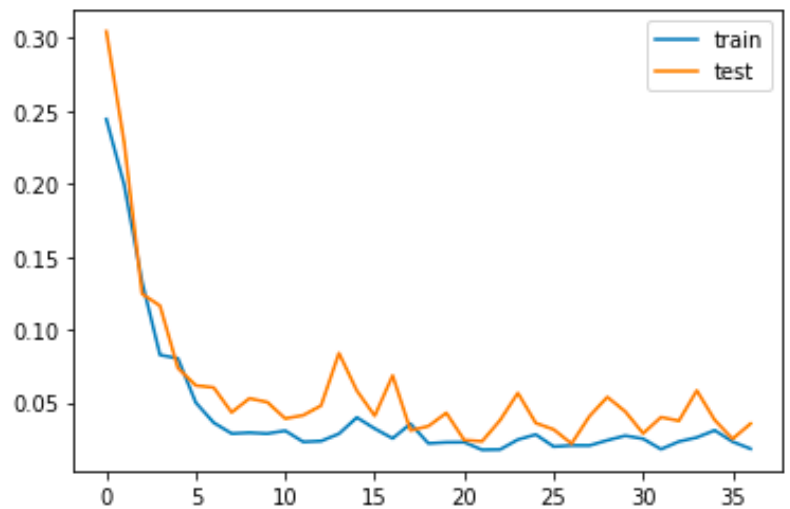
Bidirectional LSTM



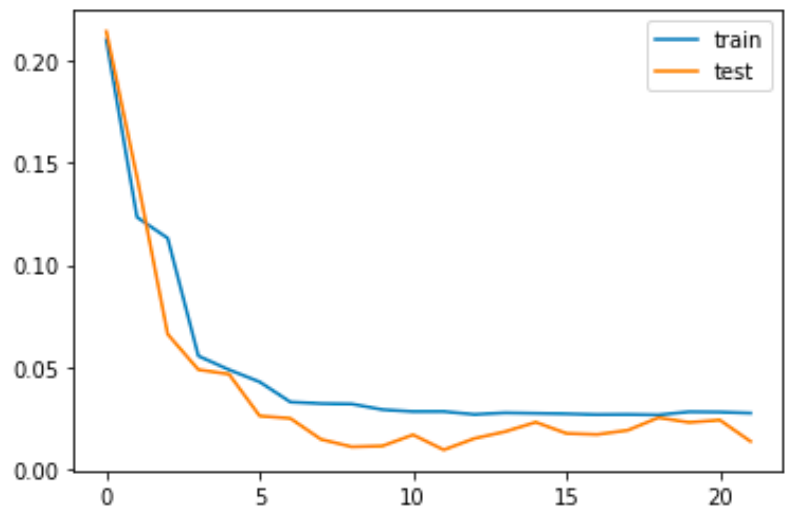
GRU



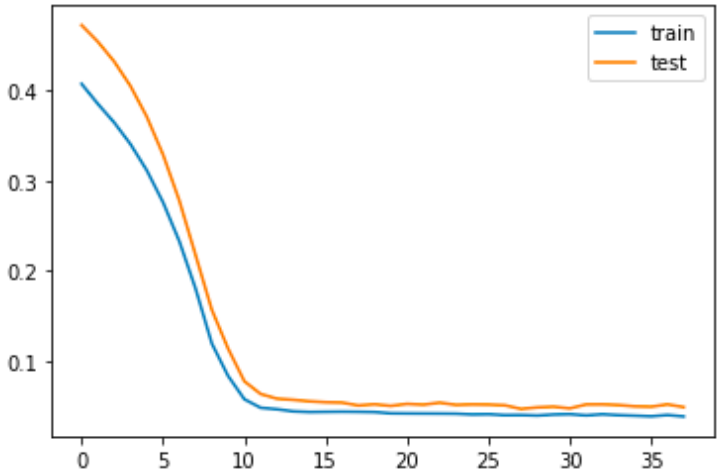
Stacked GRU



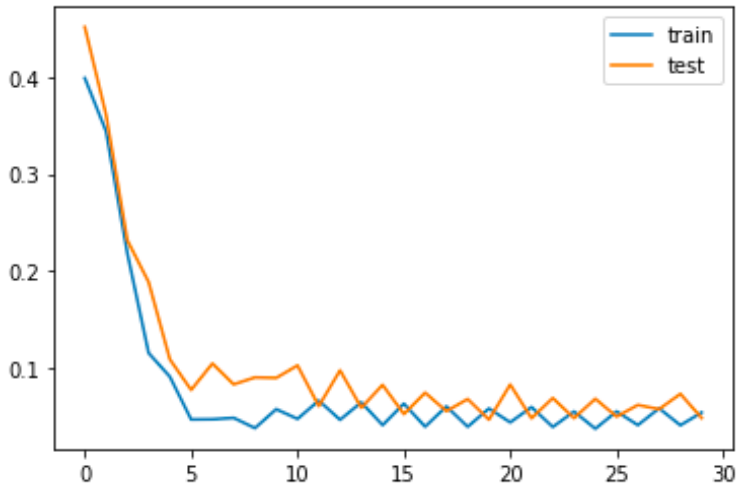
SimpleRNN



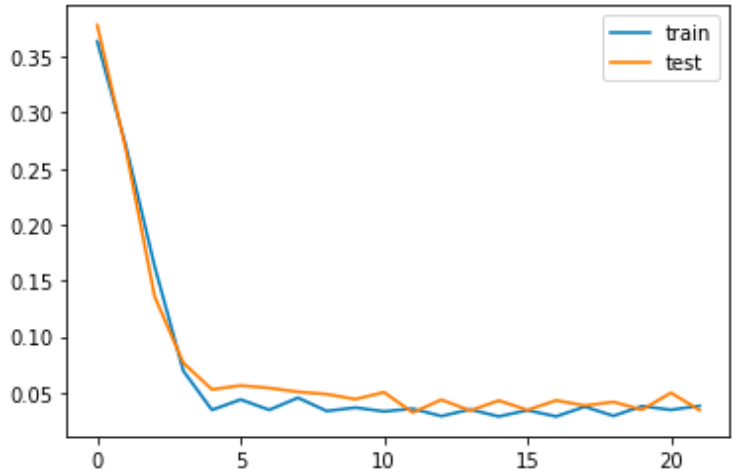
LSTM



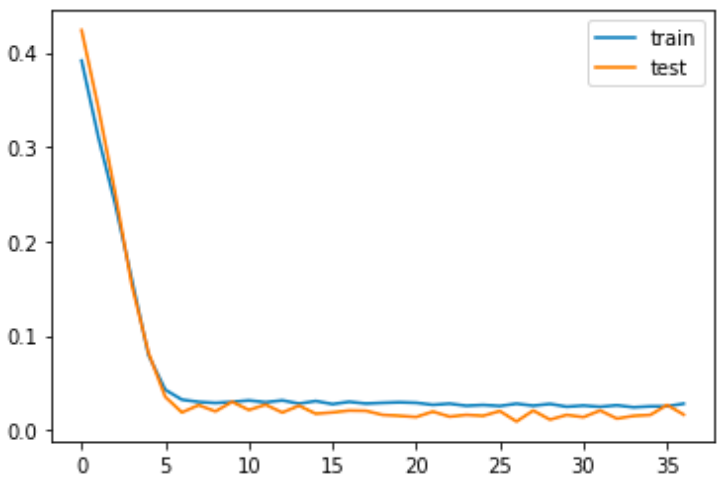
Stacked LSTM



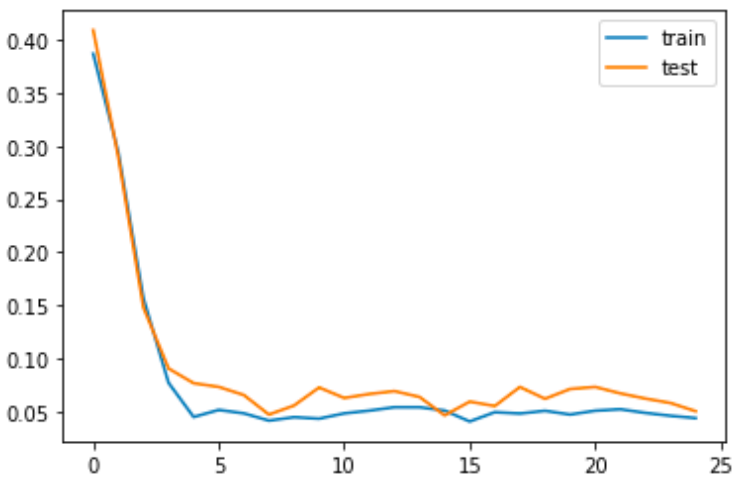
Bidirectional LSTM



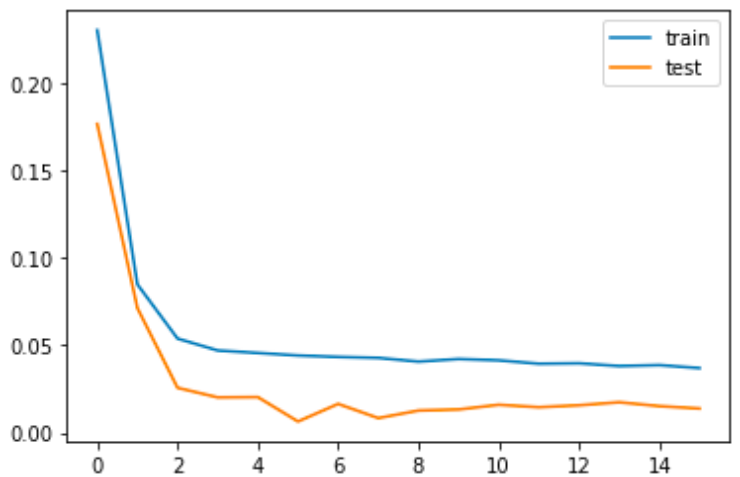
GRU



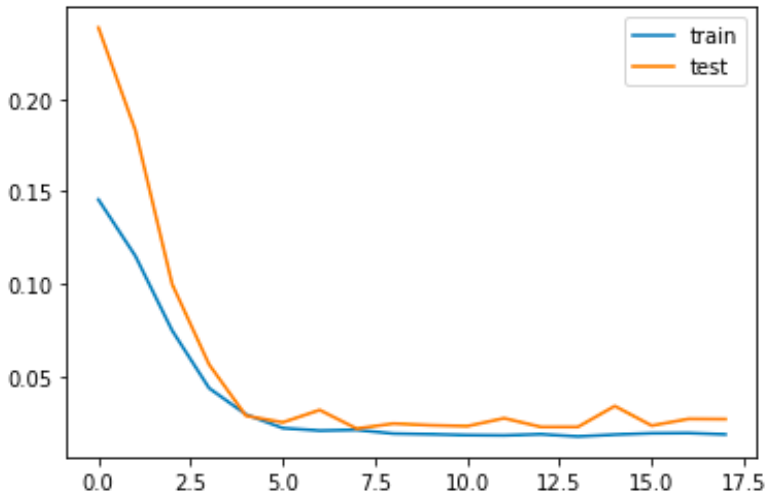
Stacked GRU



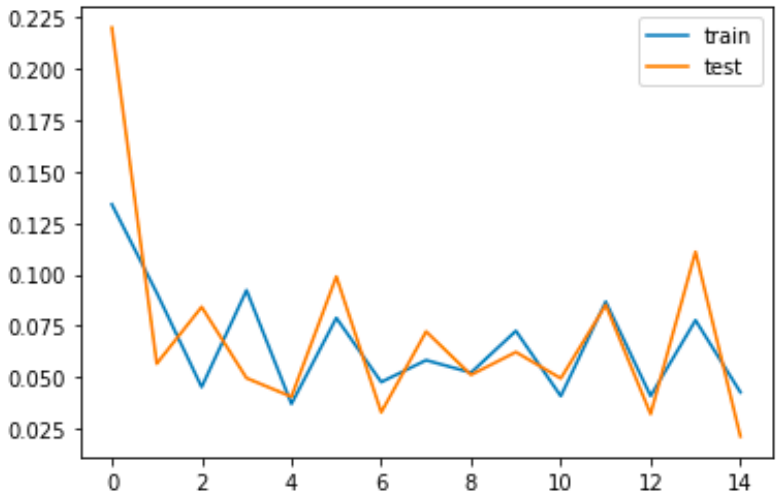
SimpleRNN



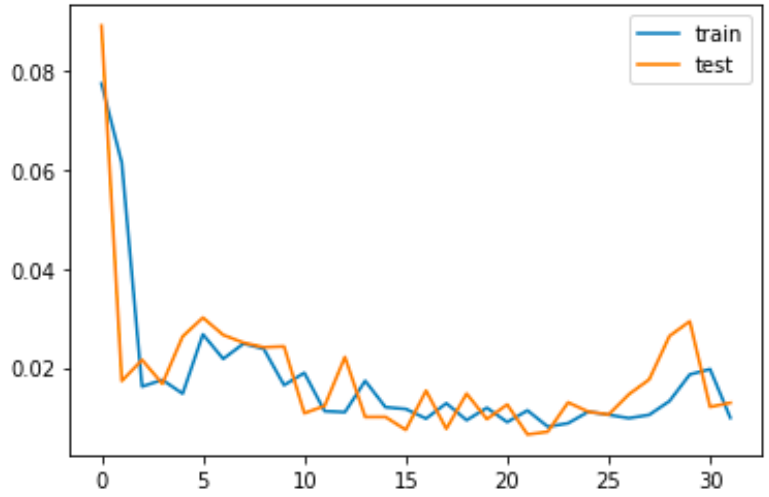
LSTM



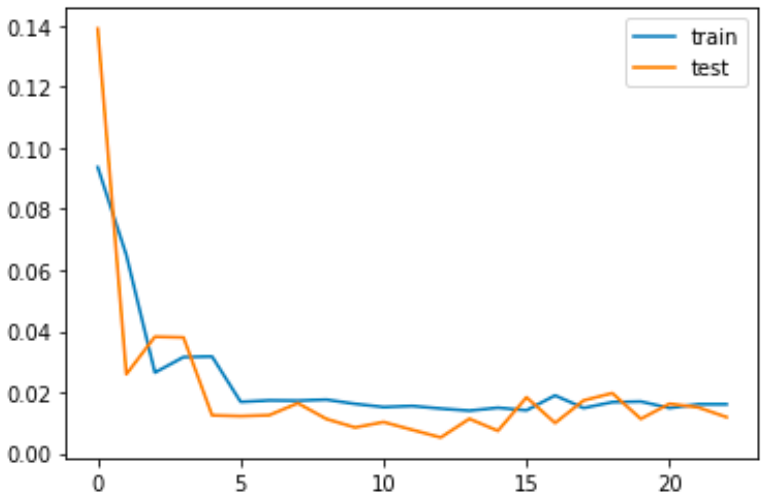
Stacked LSTM



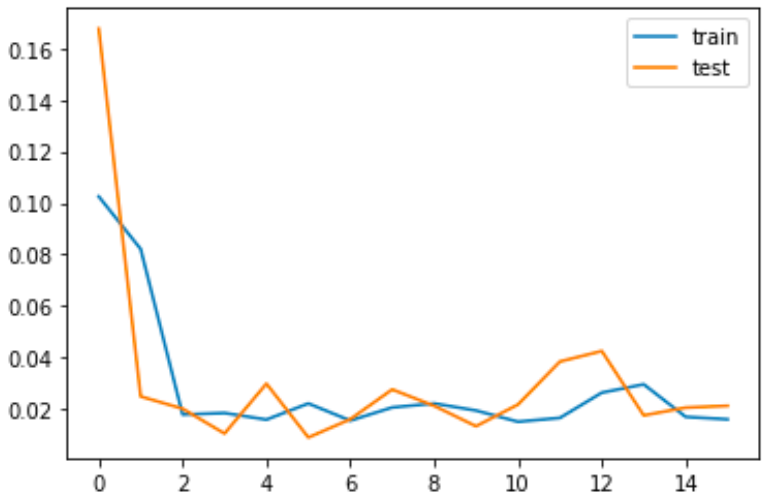
Bidirectional LSTM



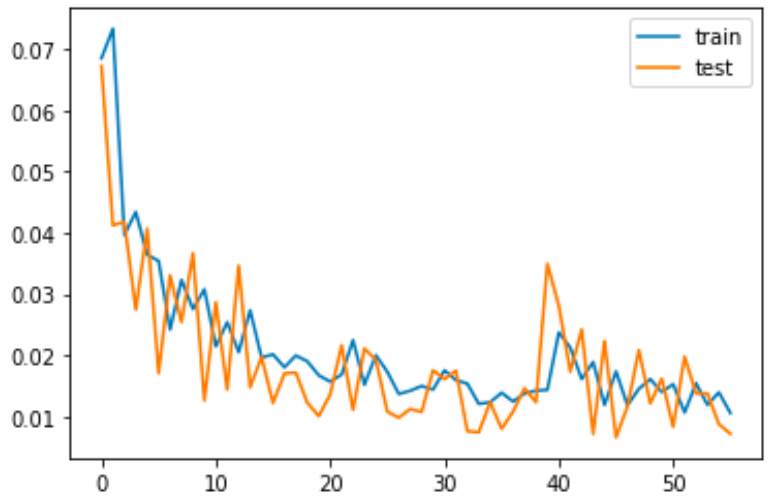
GRU



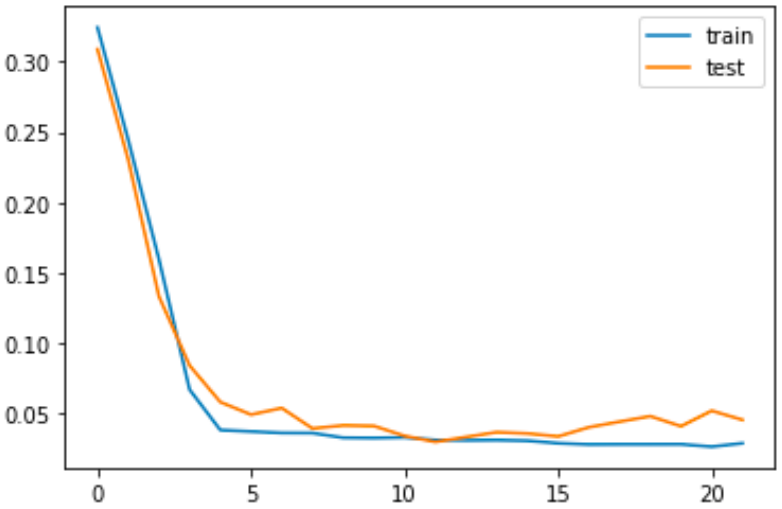
Stacked GRU



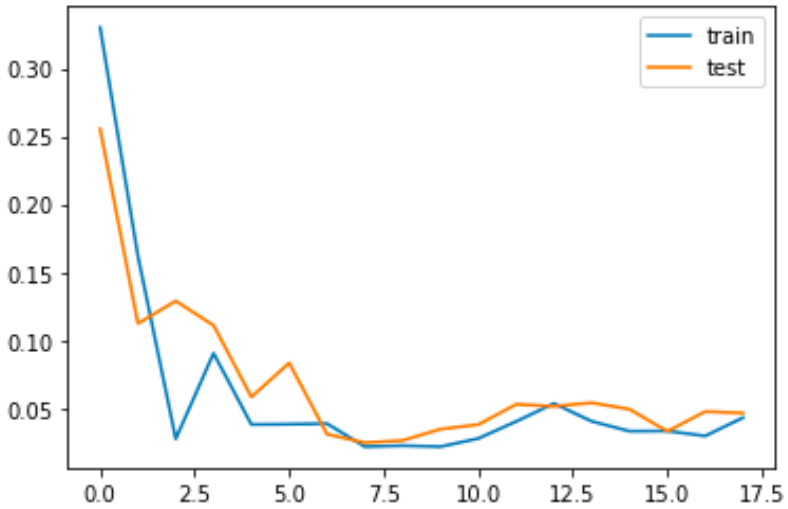
SimpleRNN



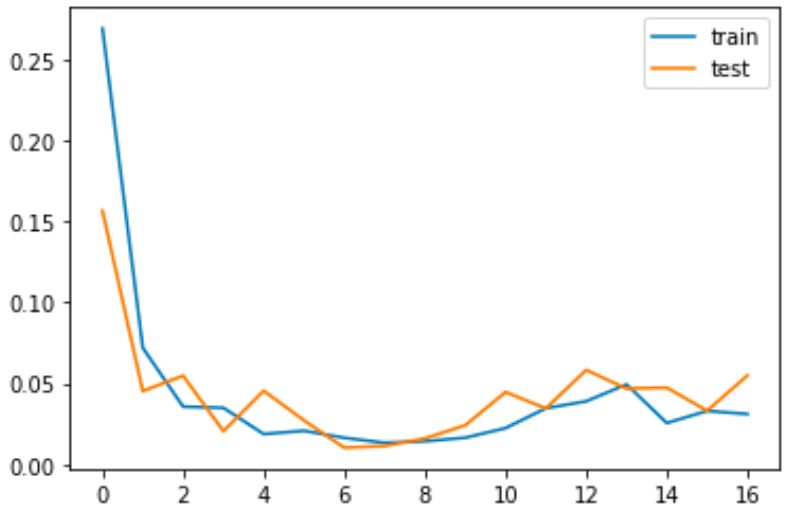
LSTM



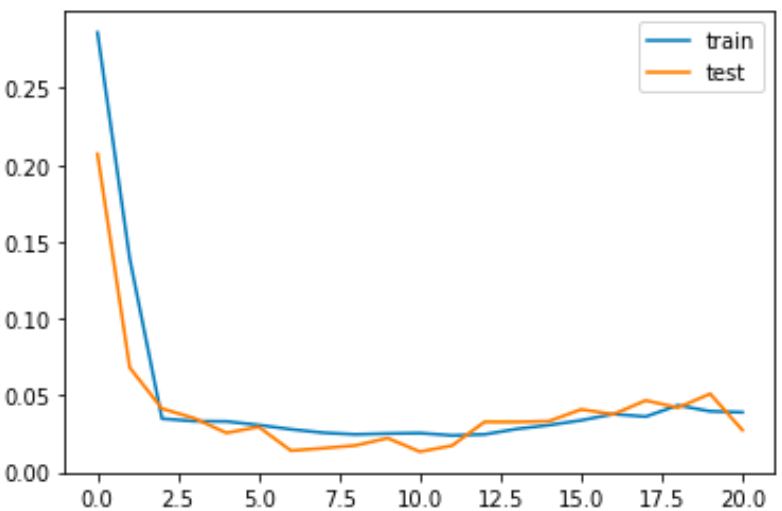
Stacked LSTM



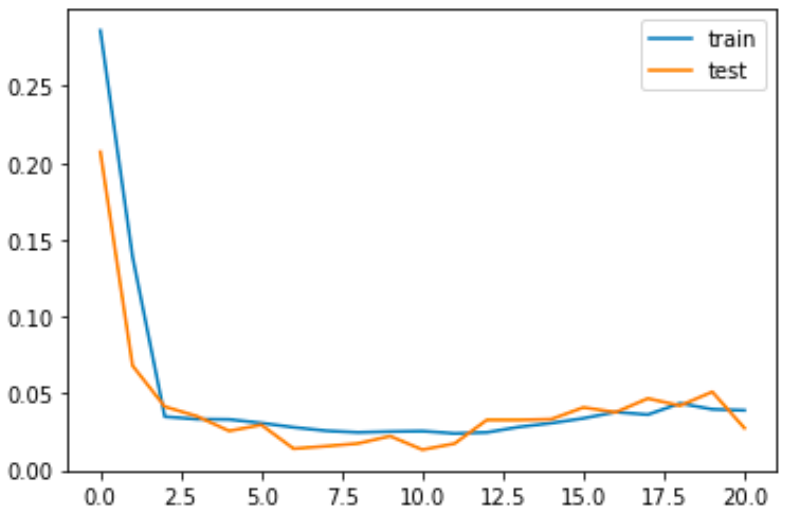
Bidirectional LSTM



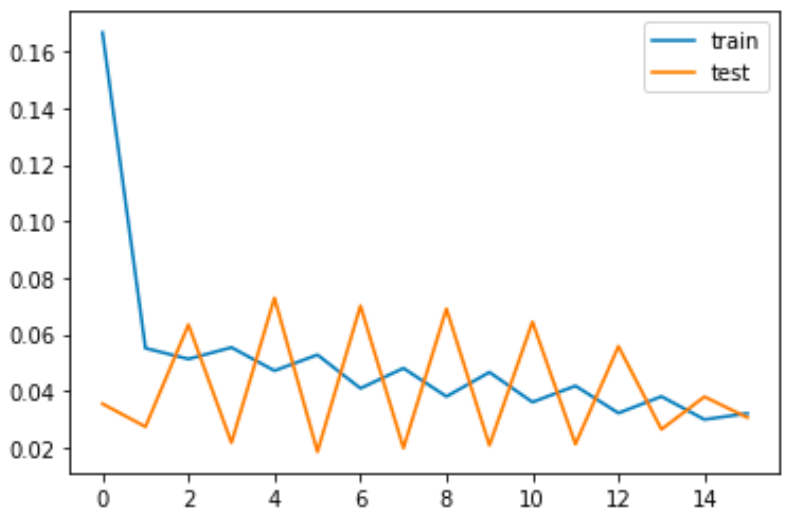
GRU

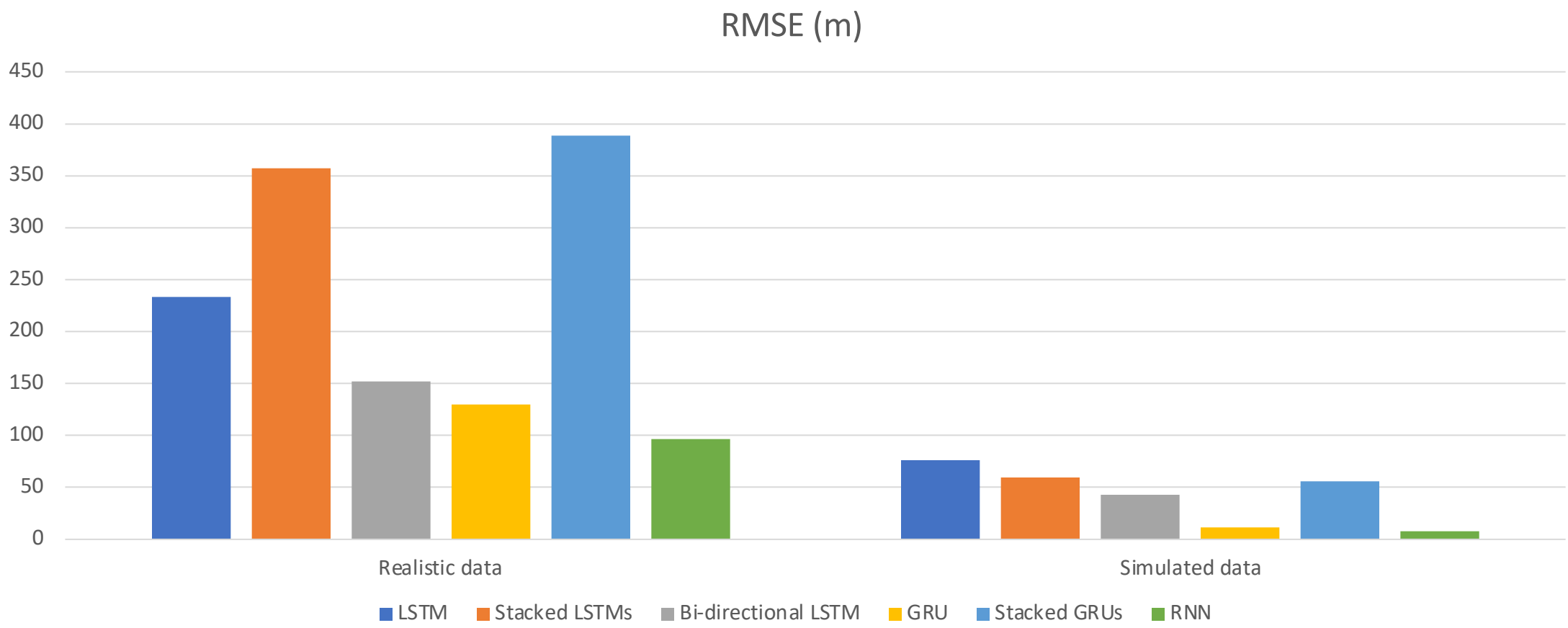


Stacked GRU

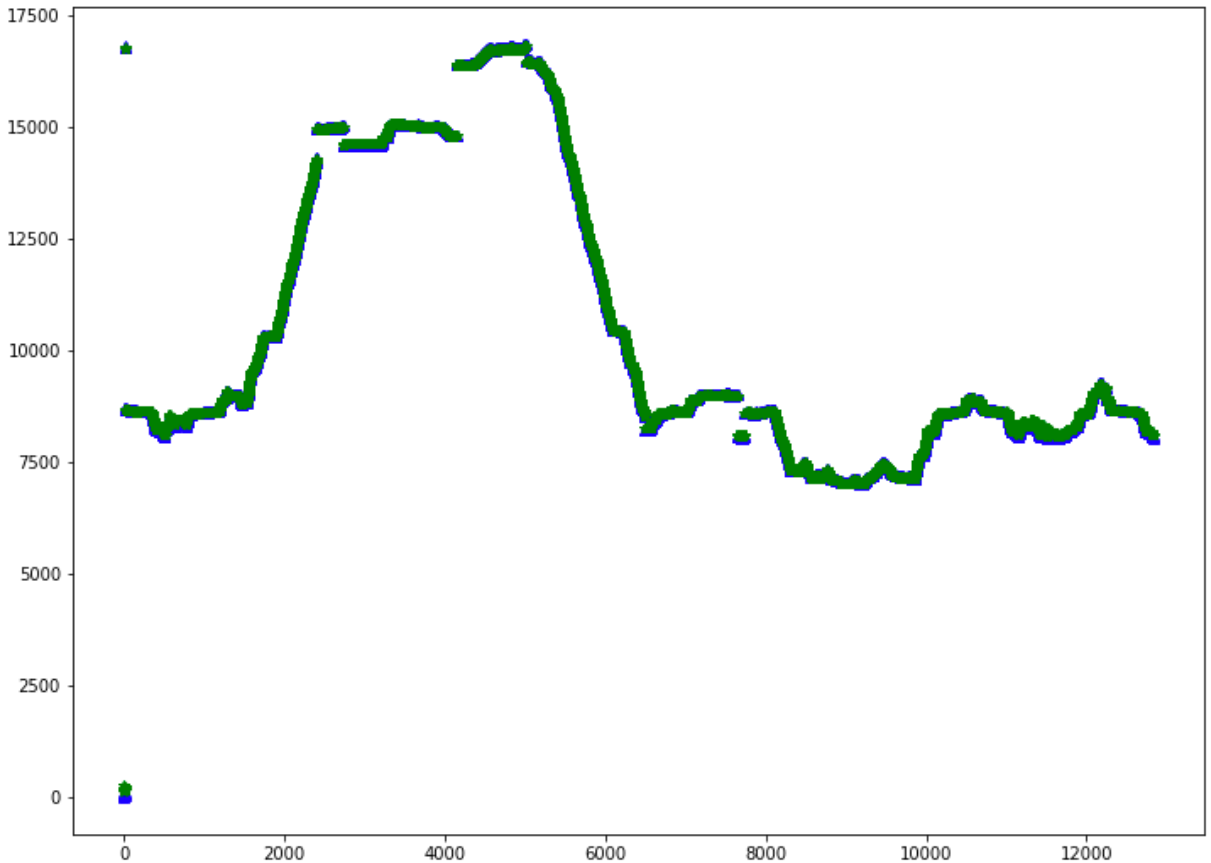


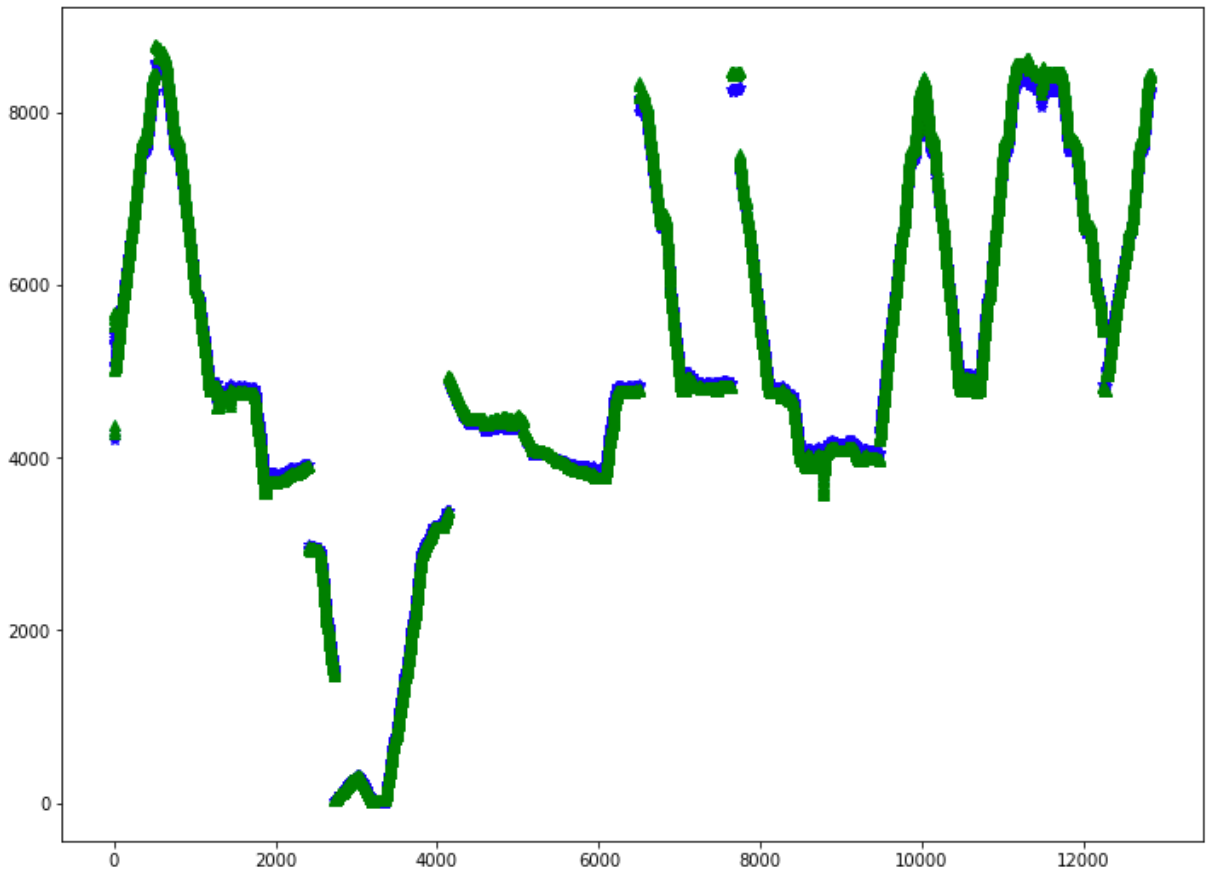
SimpleRNN

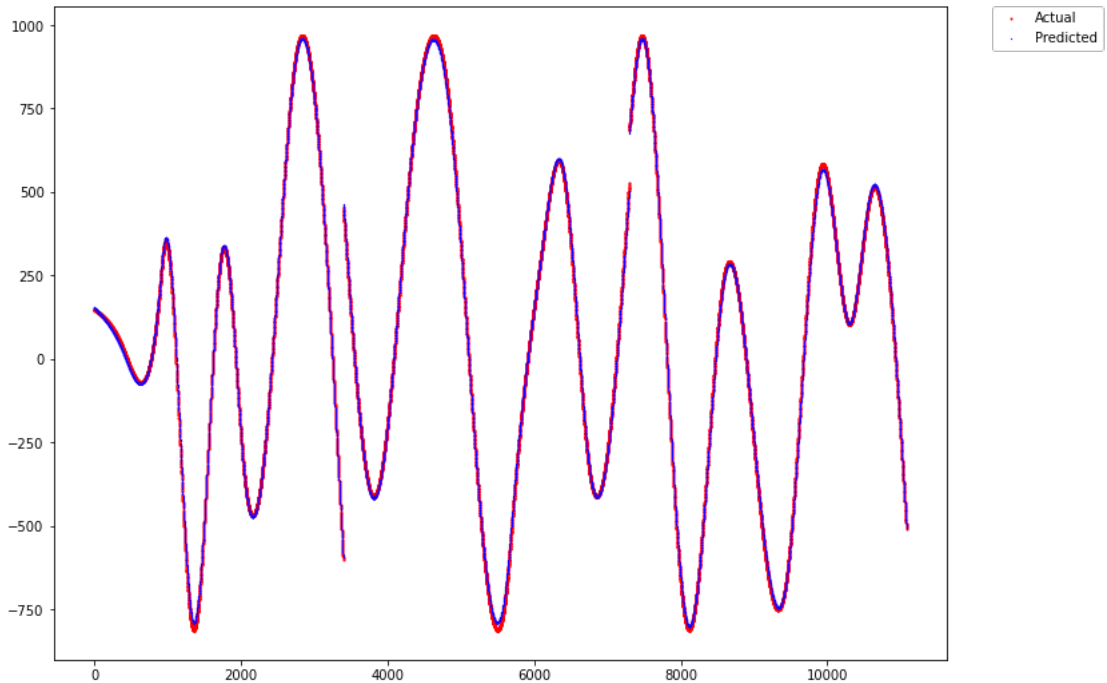


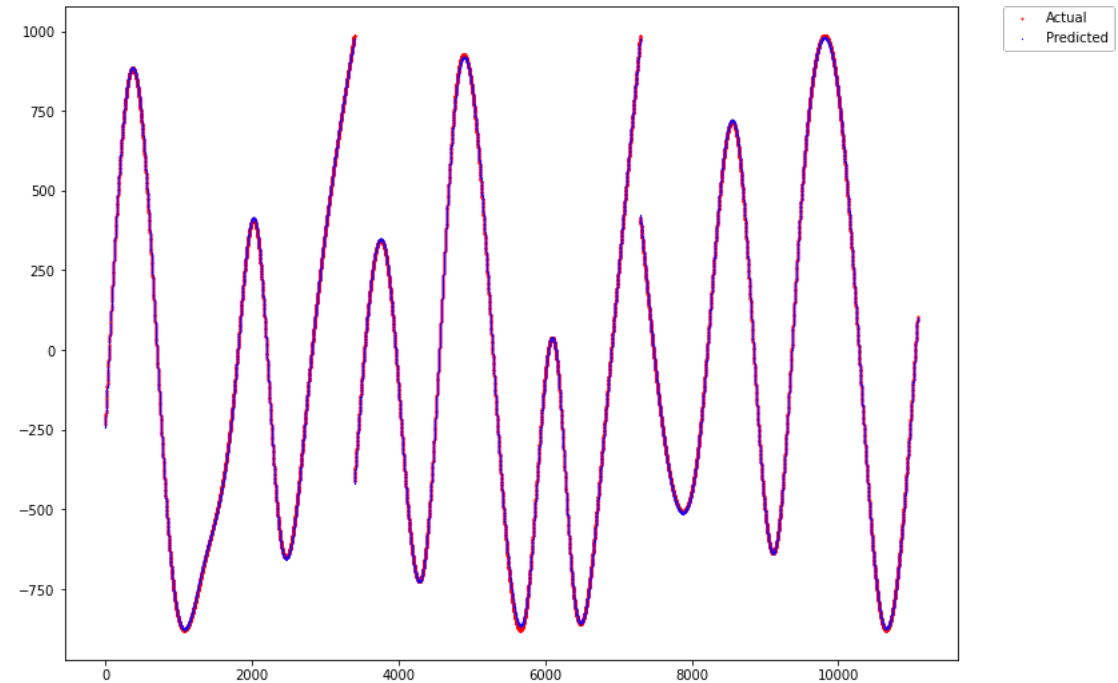

RMSE (m)
450
400
350
300
250
200
150
100
50
0
Realistic data
Simulated data
LSTM
Stacked LSTMs
Bi-directional LSTM
GRU
Stacked GRUs
RNN

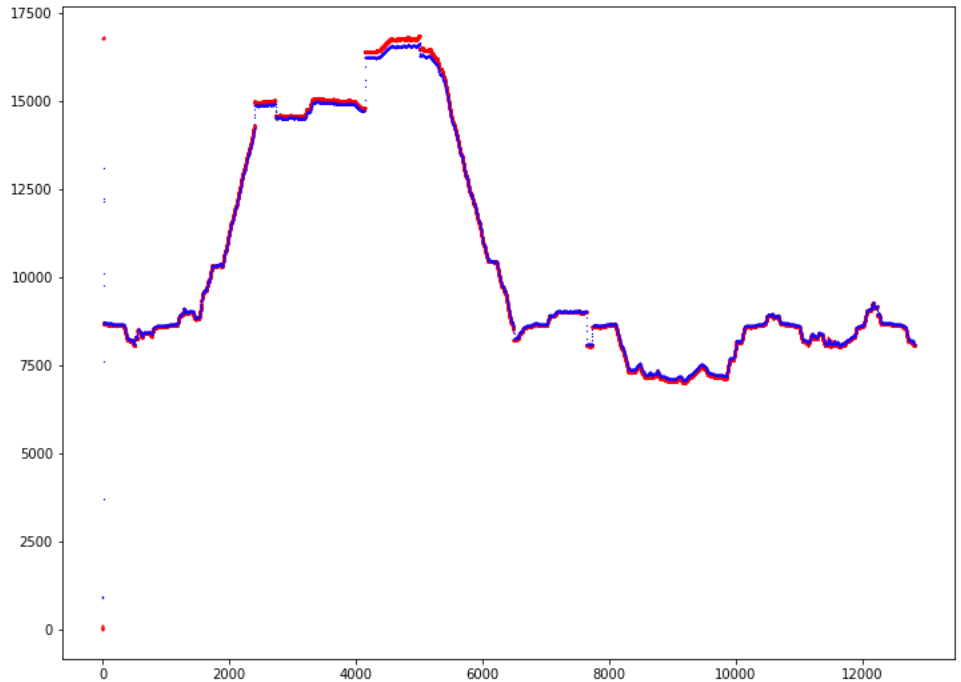
Average RMSE for Y in Train dataset: 365.403394 m
Average RMSE for X in Train dataset: 97.632337 m
Average RMSE for Y in Test dataset: 32.015023 m
Average RMSE for X in Test dataset: 99.740601 m

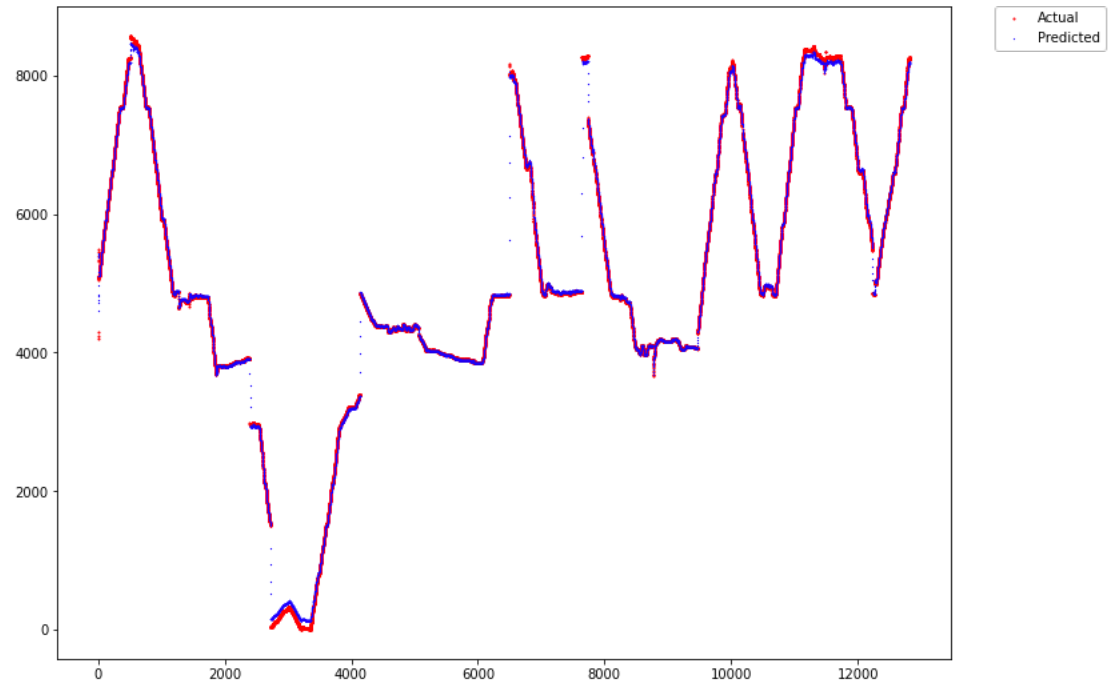


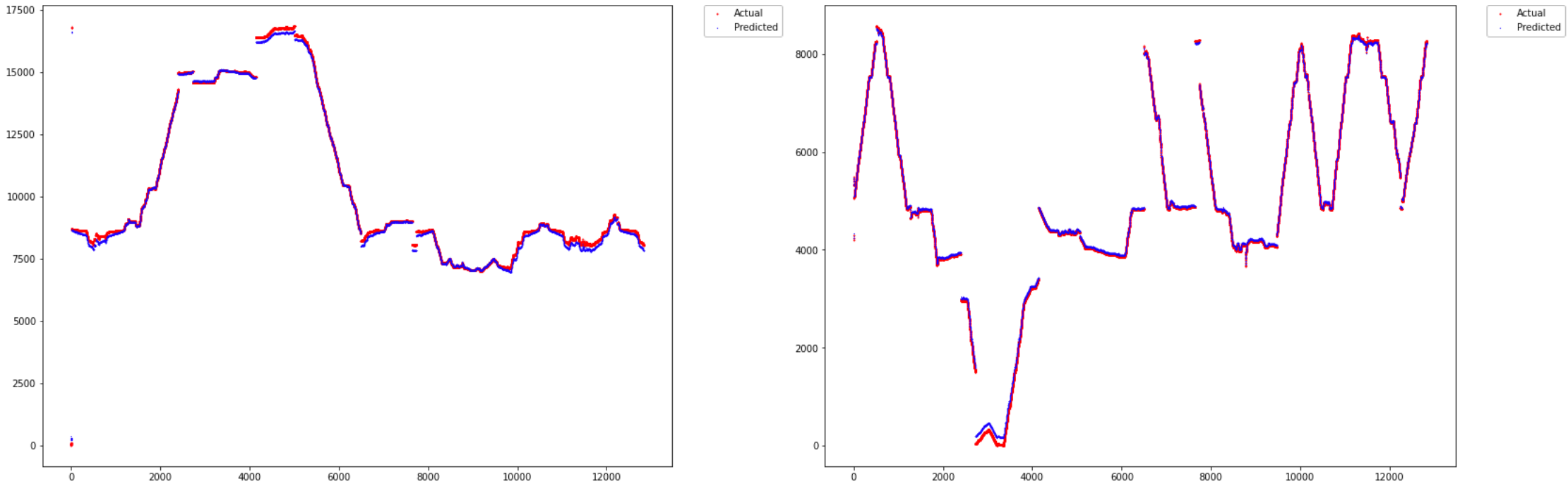


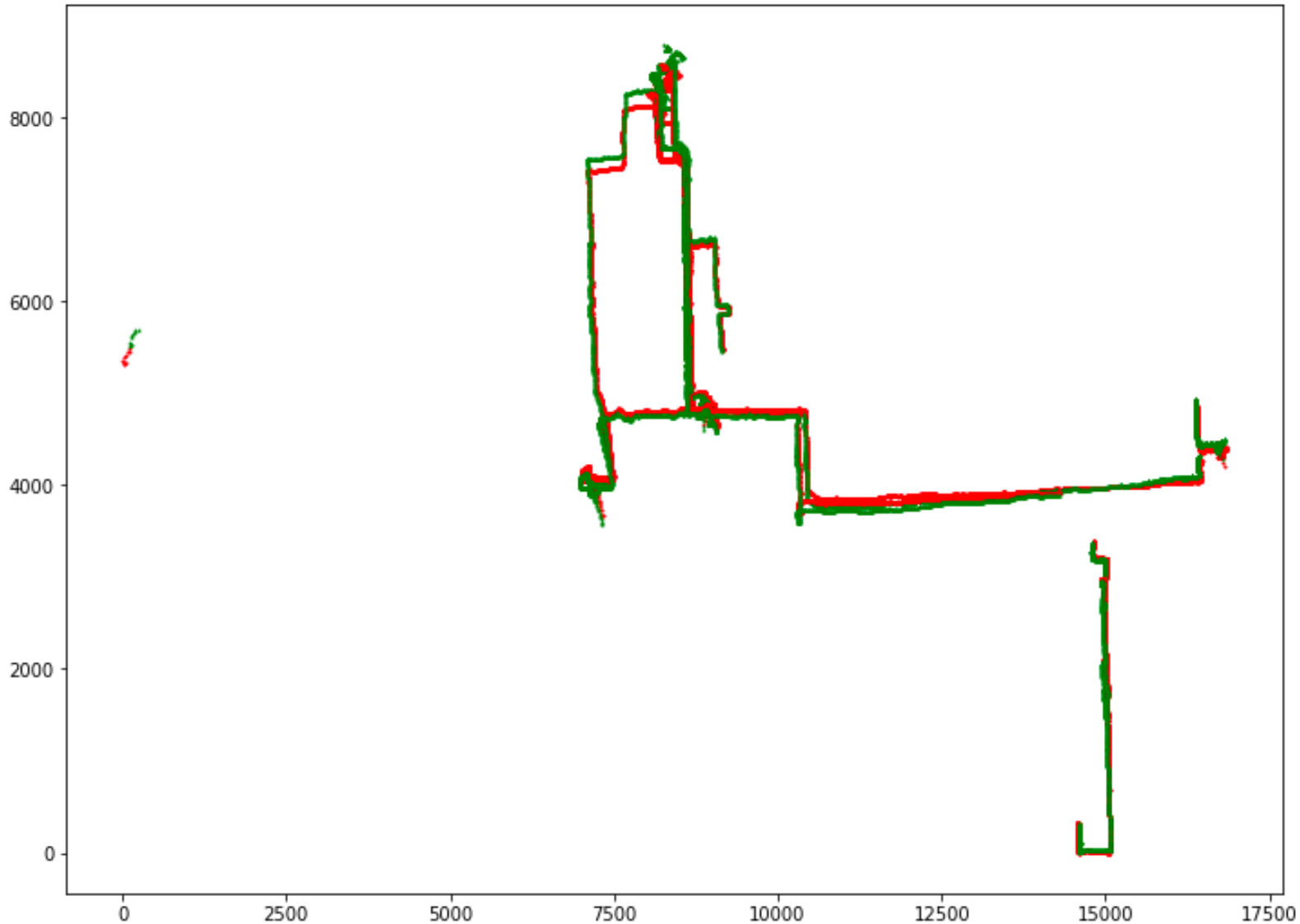

Actual
Predicted
1000
750
500
250
0
−250
−500
−750
0
2000
4000
6000
8000
10000


Actual
Predicted
1000
750
500
250
0
−250
−500
−750
0
2000
4000
6000
8000
10000


17500
15000
12500
10000
7500
5000
2500
0
0
2000
4000
6000
8000
10000
12000
Actual
Predicted


8000
6000
4000
2000
0
0
2000
4000
6000
8000
10000
12000
Actual
Predicted


Actual
Predicted
17500
15000
12500
10000
7500
5000
2500
0
0
2000
4000
6000
8000
10000
12000
Actual
Predicted
8000
6000
4000
2000
0
0
2000
4000
6000
8000
10000
12000


8000
6000
4000
2000
0
0
2500
5000
7500
10000
12500
15000
17500

Multivariate RNN model for predicting next UE's position with multi steps

X dataset

	var1(t-5)	var1(t-4)	var1(t-3)	var1(t-2)	var1(t-1)	var1(t)	var1(t+1)	\
5	-0.979949	-0.988610	-0.993593	-0.993949	-0.998102	-1.000000	-0.995492	
6	-0.988610	-0.993593	-0.993949	-0.998102	-1.000000	-0.995492	-0.995492	
7	-0.993593	-0.993949	-0.998102	-1.000000	-0.995492	-0.995492	-0.990864	
8	-0.993949	-0.998102	-1.000000	-0.995492	-0.995492	-0.990864	-0.987186	
9	-0.998102	-1.000000	-0.995492	-0.995492	-0.990864	-0.987186	0.991339	

	var1(t+2)
5	-0.995492
6	-0.990864
7	-0.987186
8	0.991339
9	0.991220

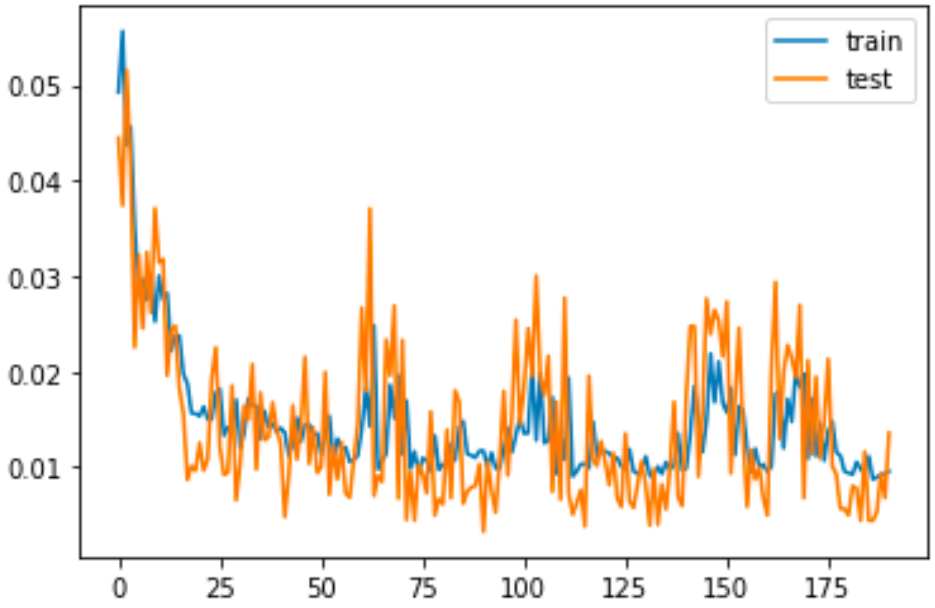
Y dataset

	var1(t-5)	var1(t-4)	var1(t-3)	var1(t-2)	var1(t-1)	var1(t)	var1(t+1)	\
5	0.275715	0.270937	0.263127	0.241448	0.237951	0.245760	0.240632	
6	0.270937	0.263127	0.241448	0.237951	0.245760	0.240632	0.257416	
7	0.263127	0.241448	0.237951	0.245760	0.240632	0.257416	0.270004	
8	0.241448	0.237951	0.245760	0.240632	0.257416	0.270004	0.277697	
9	0.237951	0.245760	0.240632	0.257416	0.270004	0.277697	0.002506	

	var1(t+2)
5	0.257416
6	0.270004
7	0.277697
8	0.002506
9	-0.011248

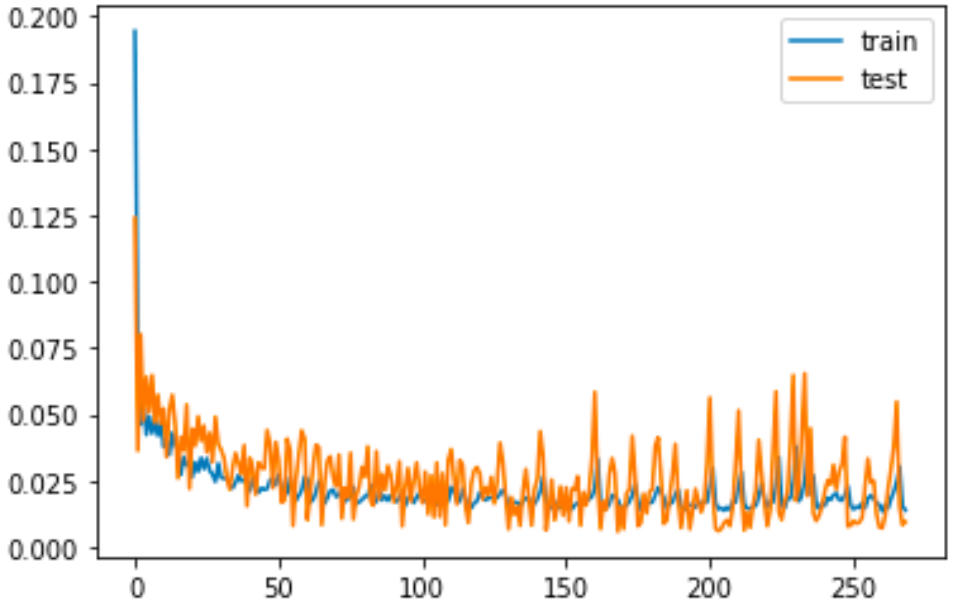
X dataset

Epoch 259/1000
36/36 - 0s - loss: 0.0096 - accuracy: 0.3843 - val_loss: 0.0136 - val_accuracy: 0.4154

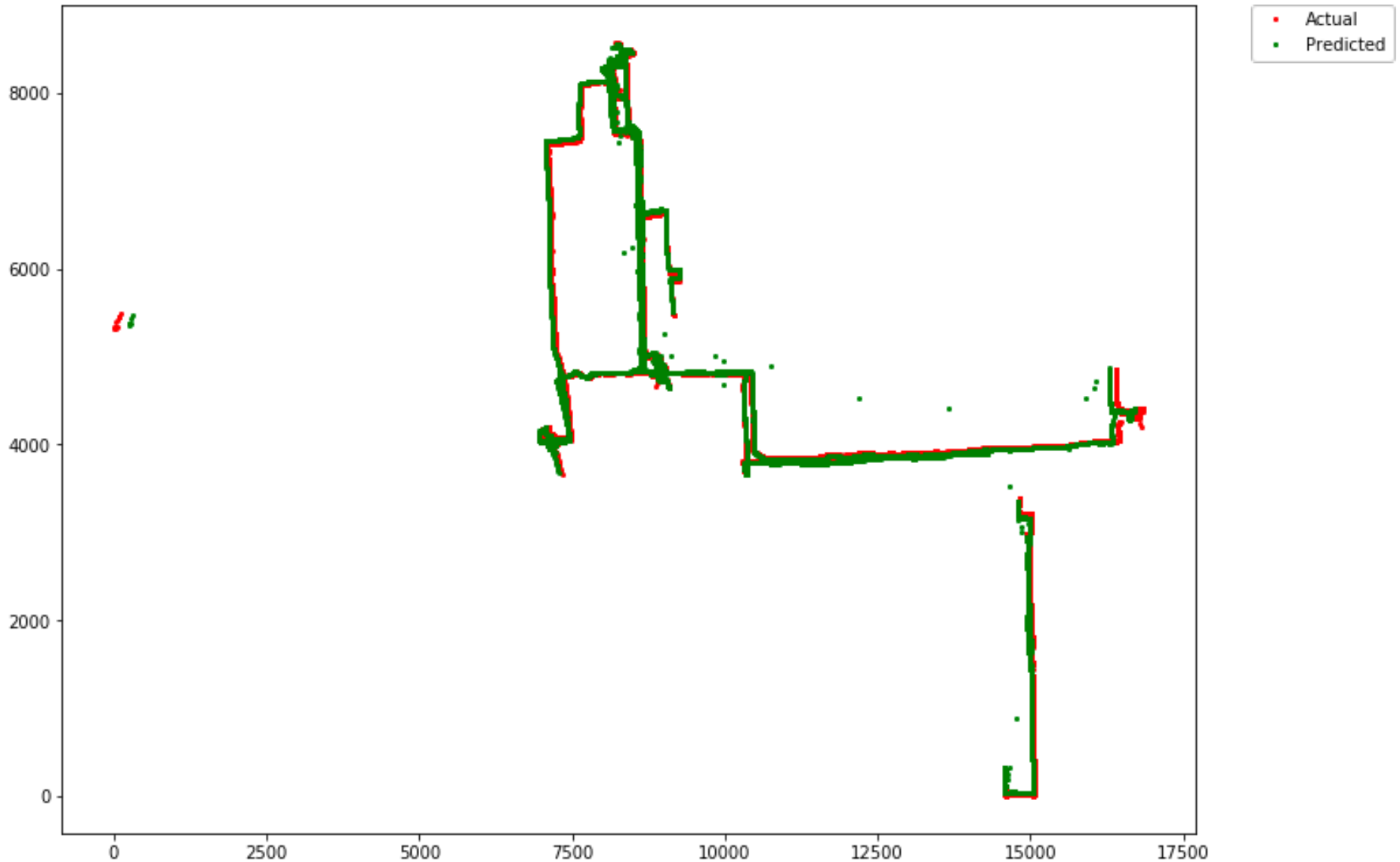


Y dataset

Epoch 185/1000
36/36 - 0s - loss: 0.0140 - accuracy: 0.5421 - val_loss: 0.0095 - val_accuracy: 0.3598



The predicted data at 1 time step forward

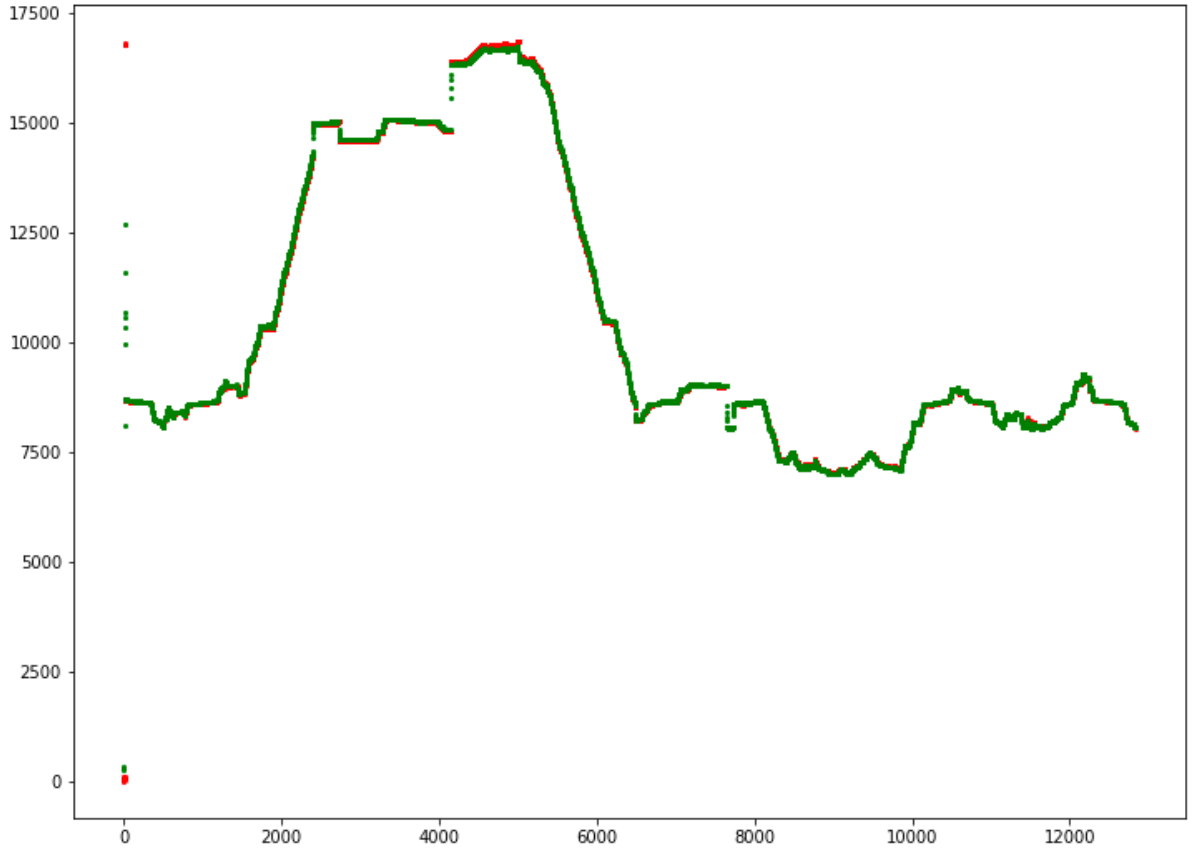


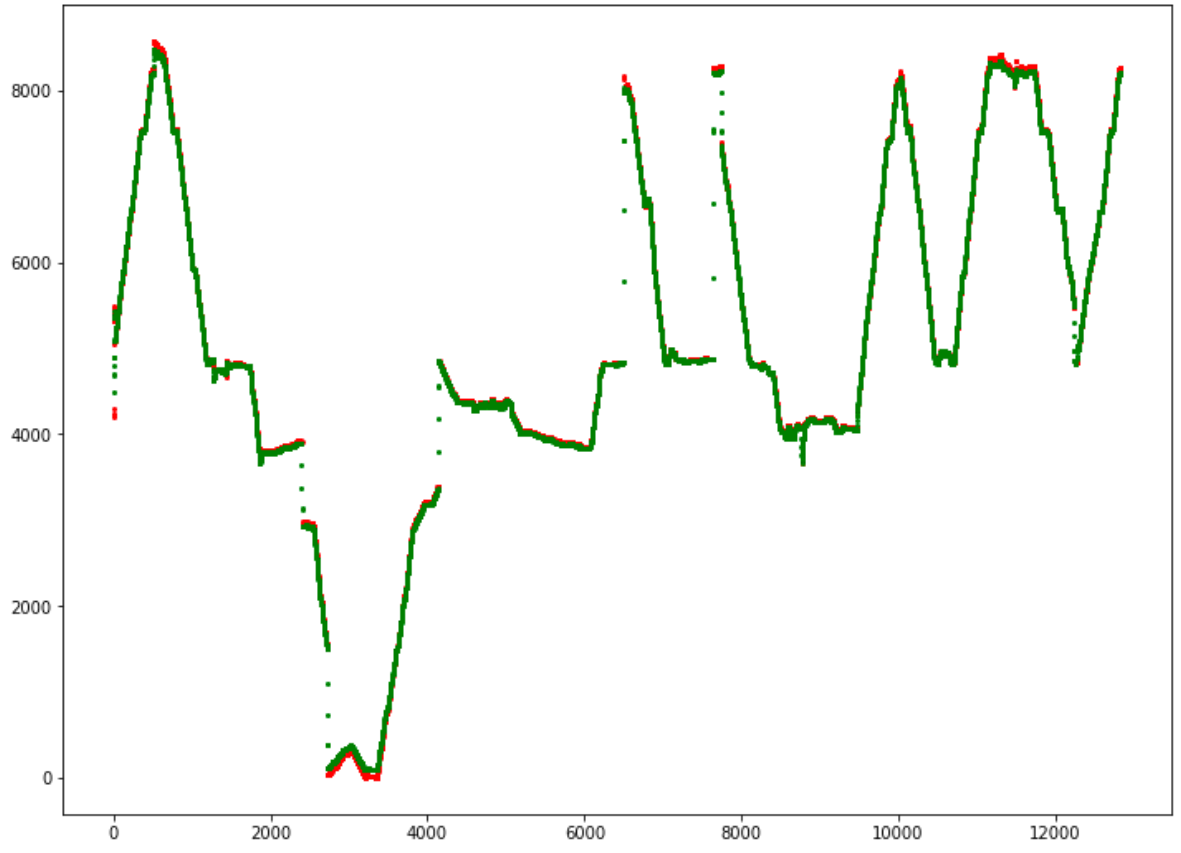
The predicted data at 2 time step forward

The predicted data at 3 time step forward

Average RMSE for Test dataset of X : 44.347912
Average RMSE for Train dataset of X : 83.229446

Average RMSE for Test dataset of Y : 491.423808
Average RMSE for Test dataset of Y : 53.256638





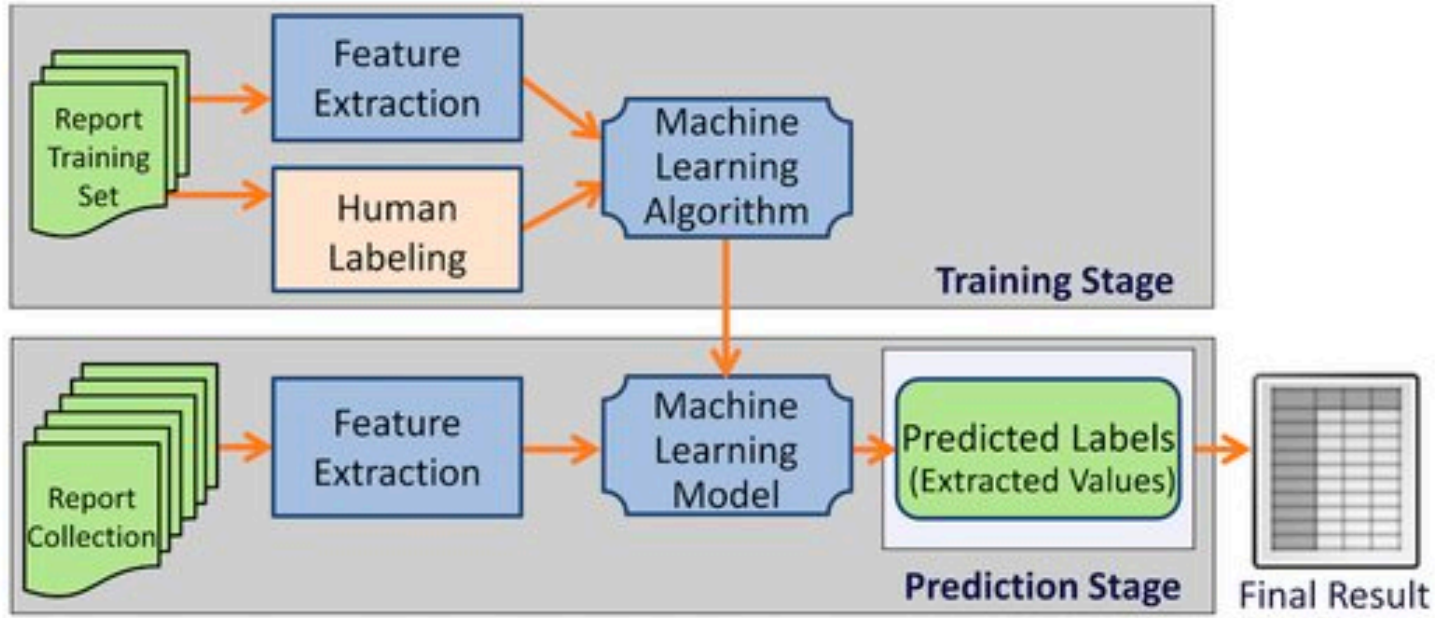
Online learning

- online learning is generally described as doing machine learning in a streaming data setting, i.e. training a model in consecutive rounds.

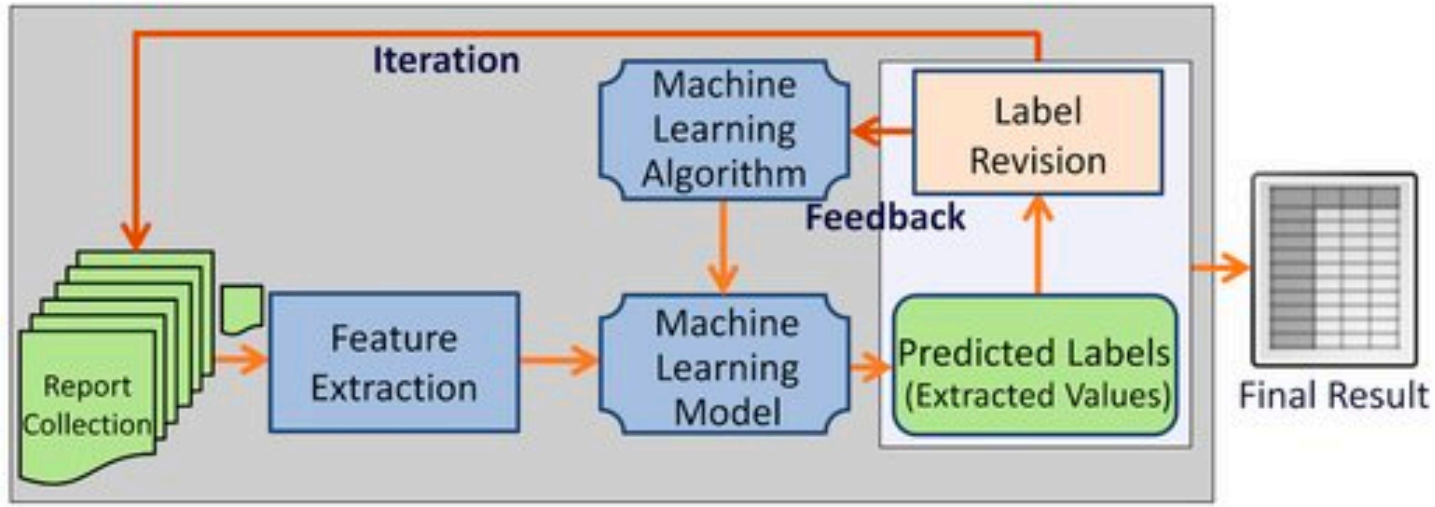
  - At the beginning of each round the ML is presented with an input sample and perform a prediction

  - The ML verifies whether its prediction was correct or incorrent, and feeds this information back into the model for subsequent rounds

- In an offline state, learning have access to whole dataset to train on
- While, in online state, model evolves as ML is trained with incomming new data.



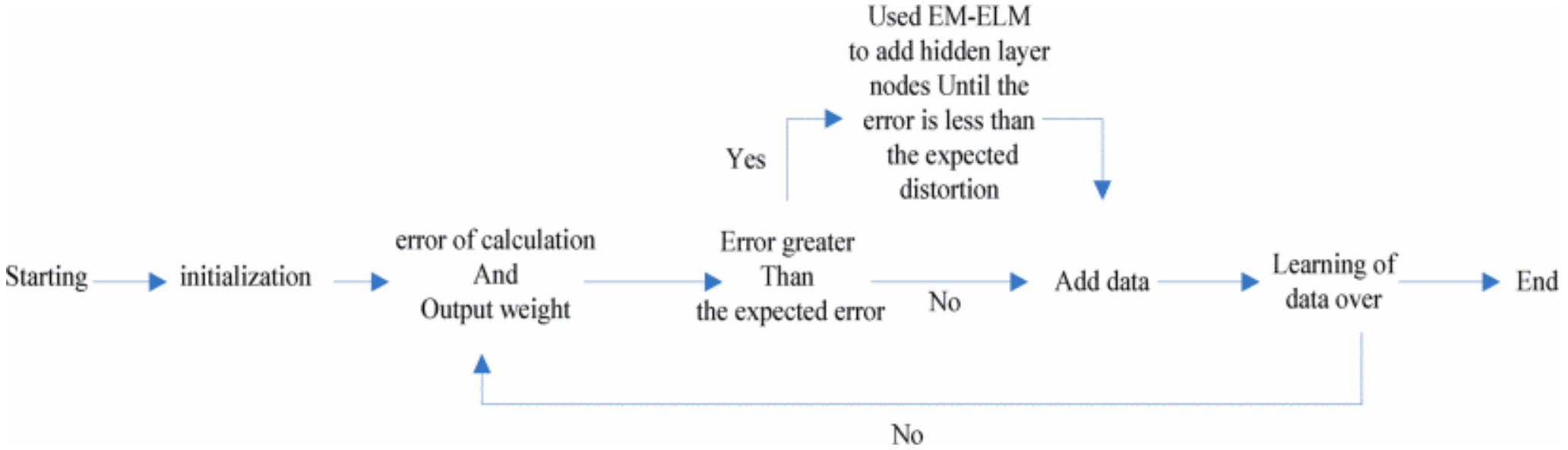
(a) Traditional Batch Machine Learning



(b) Online Machine Learning

Online sequential extreme learning machine (OS-ELM)

- https://ieeexplore.ieee.org/document/8394791



LSTM for line prediction

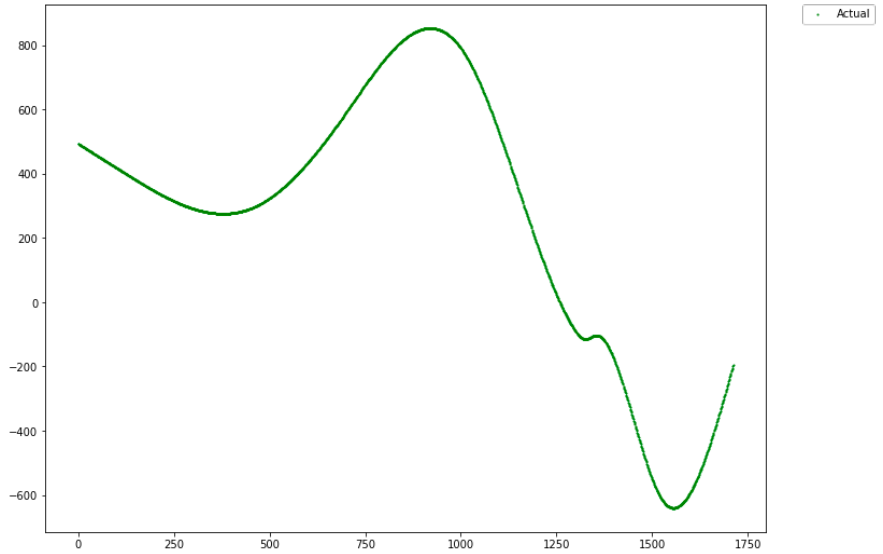
X data was chosen from the 3 datsets to be fed into ML

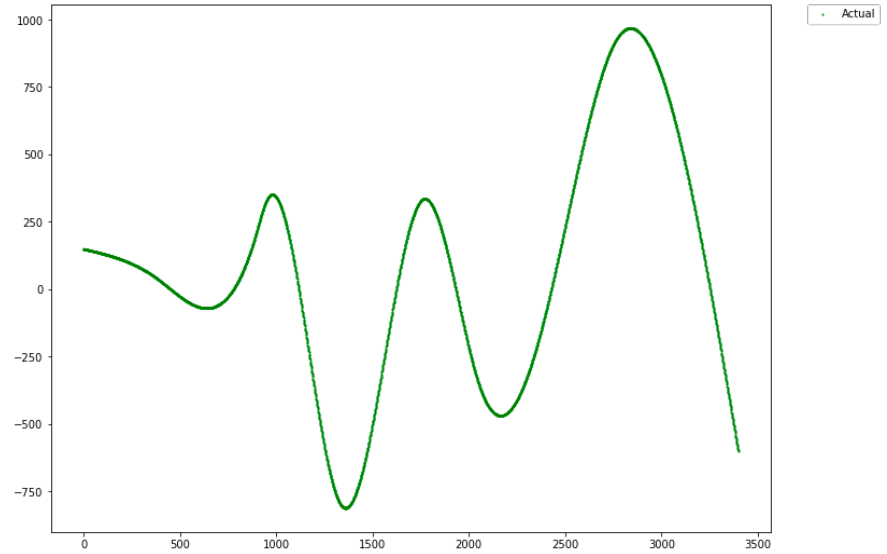
```
]: traj.columns = ["X", "Y", "Speed", "Time"]
   traj2.columns = ["X", "Y", "Speed", "Time"]
   traj3.columns = ["X", "Y", "Speed", "Time"]
   traj,traj2,traj3
```

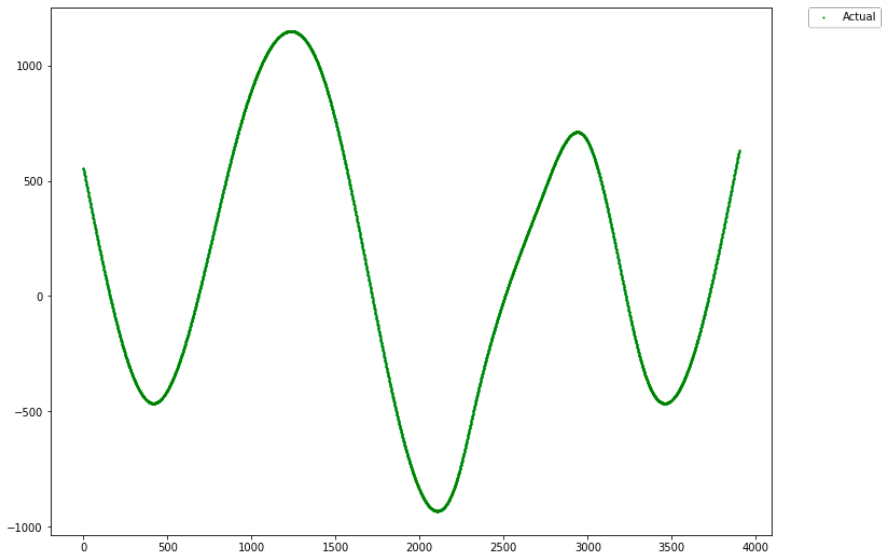
]: (

	X	Y	Speed	Time
0	493.000000	197.000000	0.000000	0
1	492.243628	201.164848	4.232973	1
2	491.487258	205.329223	4.232507	2
3	490.730891	209.492650	4.231574	3
4	489.974530	213.654657	4.230175	4

Plot of of X dataset in 3 lines





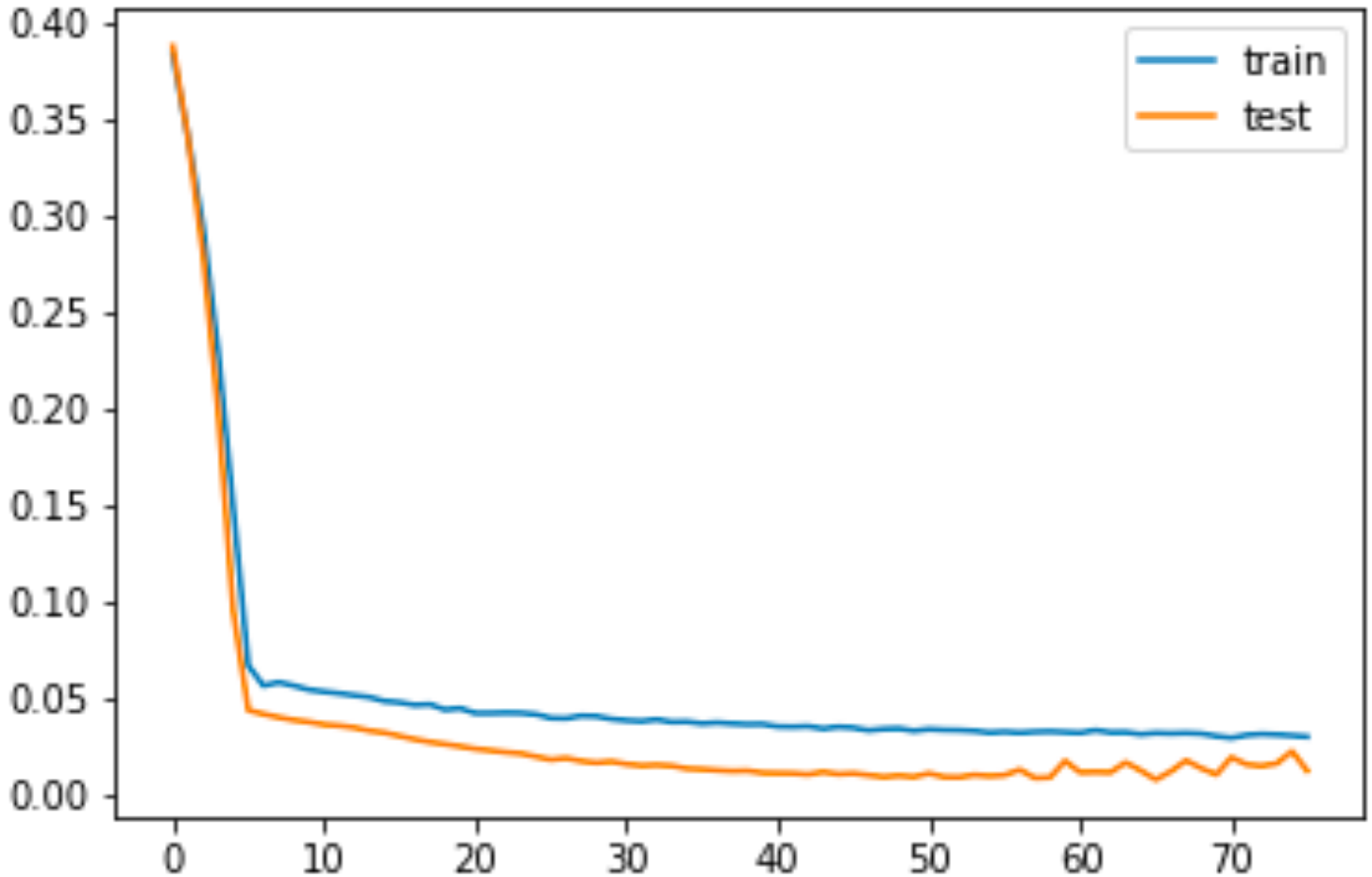


Transform time series data to supervised problem and concatenate all X dataset into 1 dataset

Feature Variable	Target Variable
var1(t-1)	var1(t)
1 0.517702	0.516689
2 0.516689	0.515676
3 0.515676	0.514663
4 0.514663	0.513650
5 0.513650	0.512637
var1(t-1)	var1(t)
1 0.079566	0.079378
2 0.079378	0.079189
3 0.079189	0.079000
4 0.079000	0.078812
5 0.078812	0.078623
var1(t-1)	var1(t)
1 0.429691	0.426156
2 0.426156	0.422621
3 0.422621	0.419085
4 0.419085	0.415550
5 0.415550	0.412015

```
# design single LSTM network
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',mode='auto', patience=10,restore_best_weights=True)
model1 = Sequential()
model1.add(LSTM(64,activation='relu' ,input_shape=(train_X.shape[1], train_X.shape[2])))
model1.add(Dropout(0.1))
model1.add(Dense(test_y.shape[1]))
model1.compile(loss='mae', optimizer='adam',metrics=['accuracy'])
model1.summary()
# fit network
history1 = model1.fit(train_X, train_y, epochs=600, callbacks=[callback], batch_size=72, validation_data=(test_X,
test_y), verbose=2, shuffle=False)

# plot history
pyplot.plot(history1.history['loss'], label='train')
pyplot.plot(history1.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

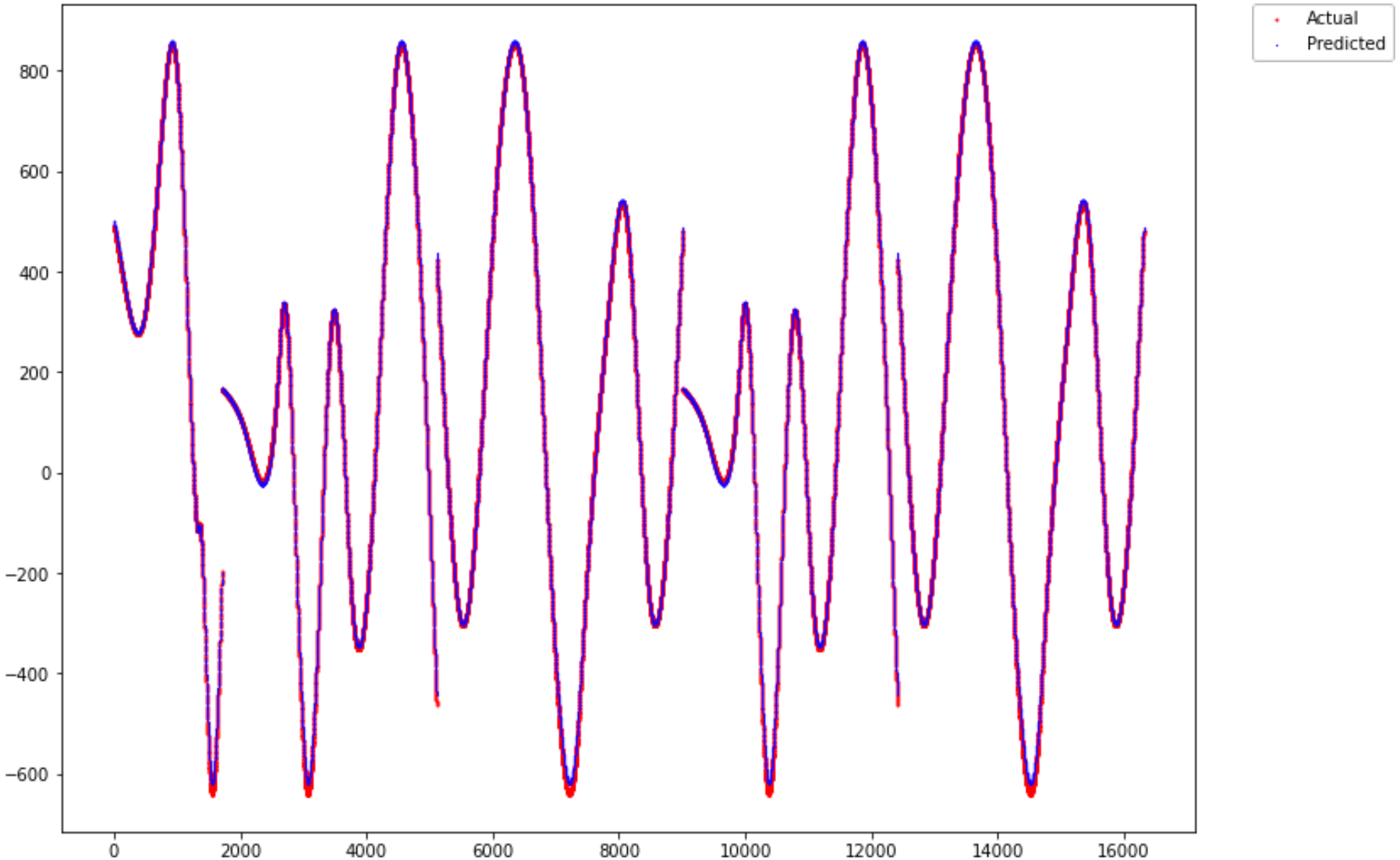
Epoch 76/600 46/46 - 0s - loss: 0.0294 - accuracy: 3.0656e-04 - val_loss: 0.0118 - val_accuracy: 1.5326e-04



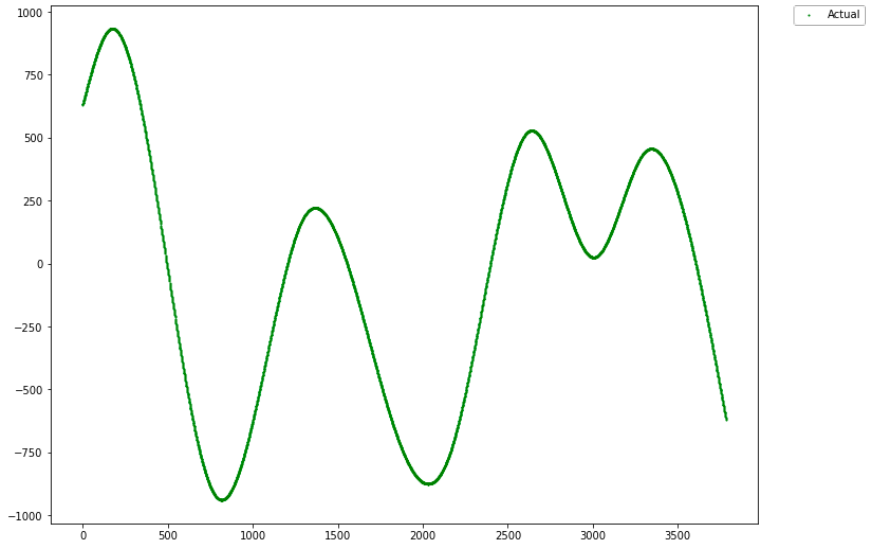
```
yhat = model1.predict(test_X)
Loss, accuracy = model1.evaluate(test_X, test_y)
print('Accuracy: %.4f' % (accuracy*100))

# calculate RMSE
print('Loss: %.6f' % Loss)
```

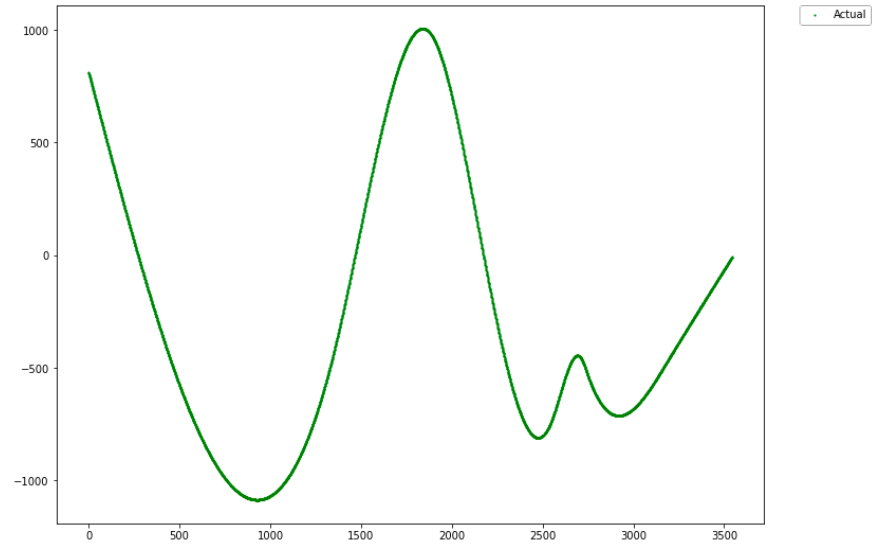
```
408/408 [==============================] - 0s 1ms/step - loss: 0.0071 - accuracy: 1.5326e-04
Accuracy: 0.0153
Loss: 0.007070
```

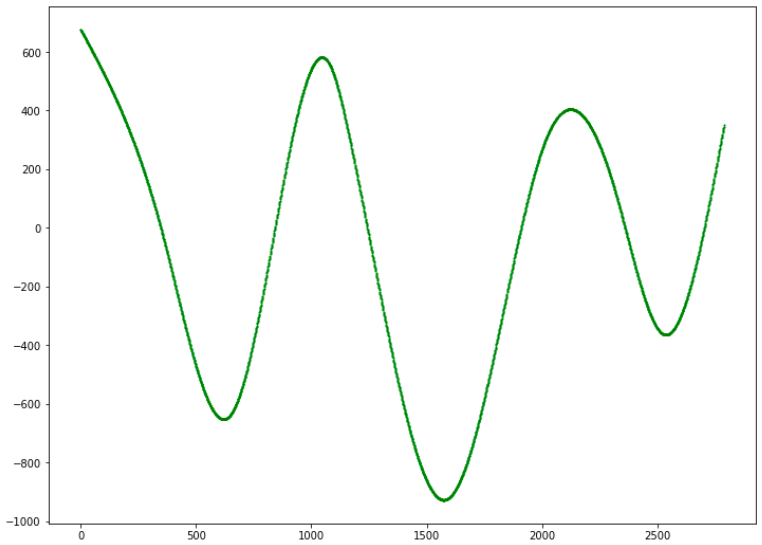
RMSE for X in Train dataset: 9.270551 m
RMSE for X in Test dataset: 7.653299 m



Try feeding 3 different paths and Let ML predict

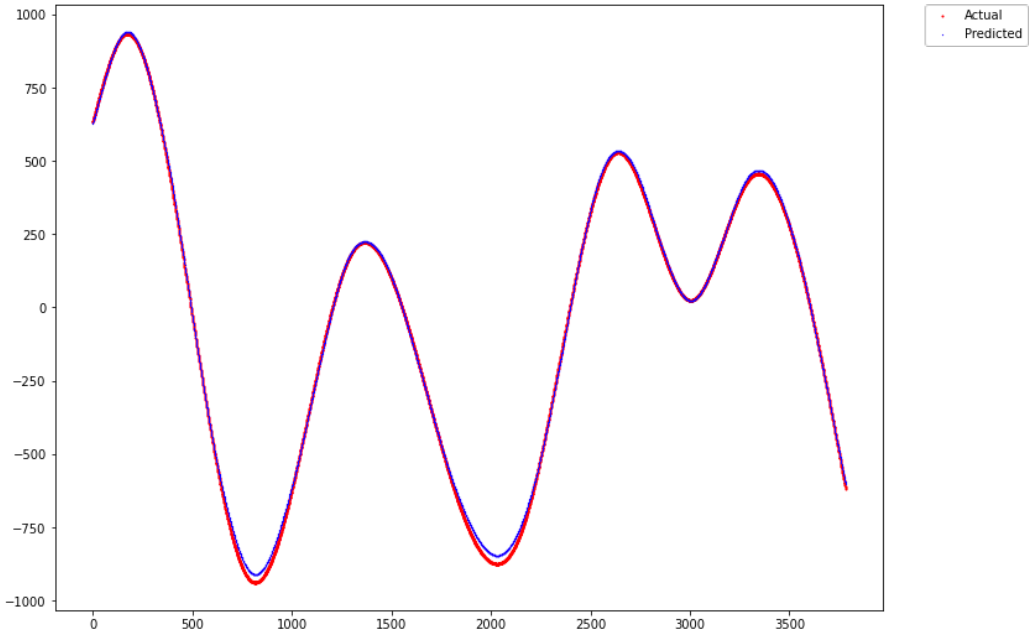




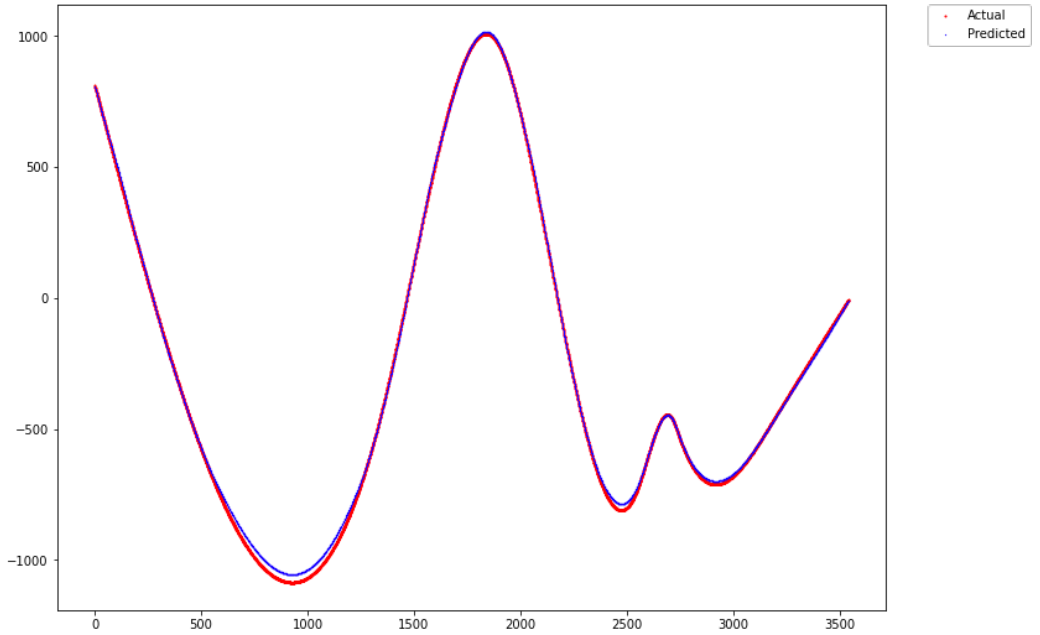


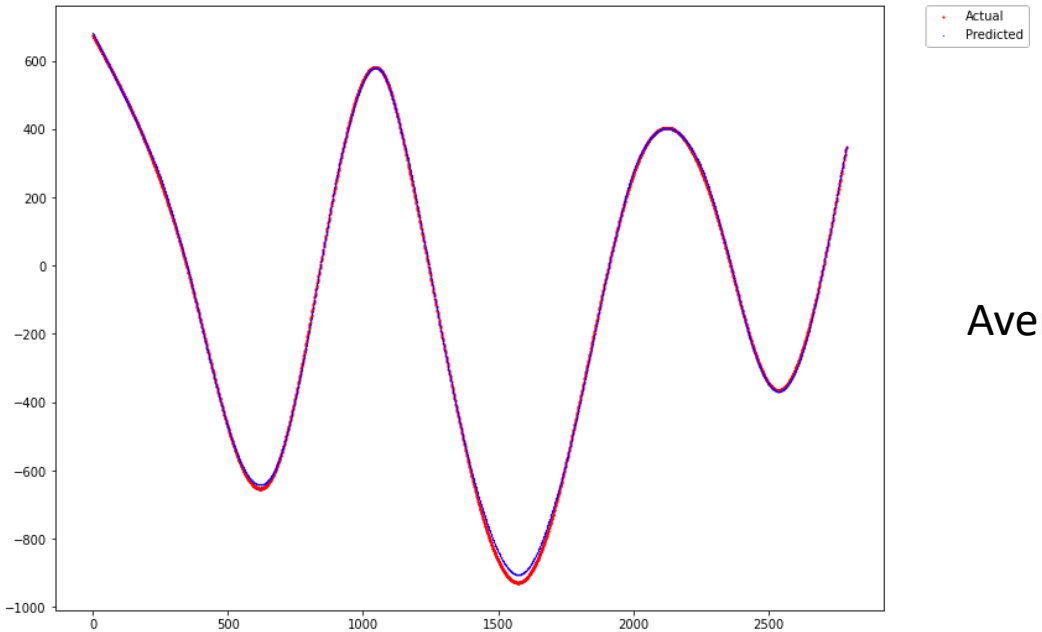
Prediction results

Average RMSE: 13.316234 m



Average RMSE: 15.091153 m





Average RMSE: 9.406013 m

LSTM for X,Y prediction

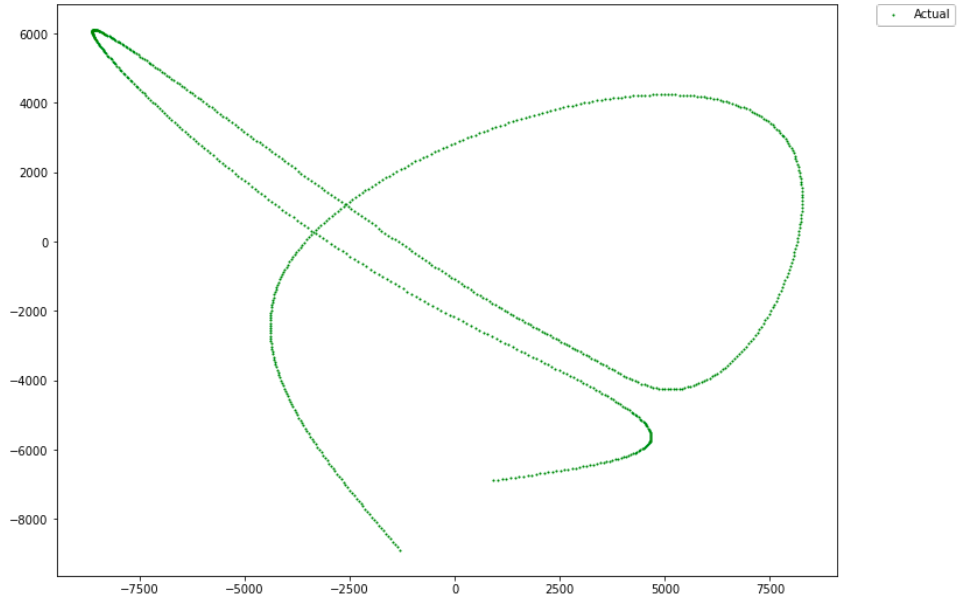
X and Y data was chosen from the 10 datsets to be fed into ML

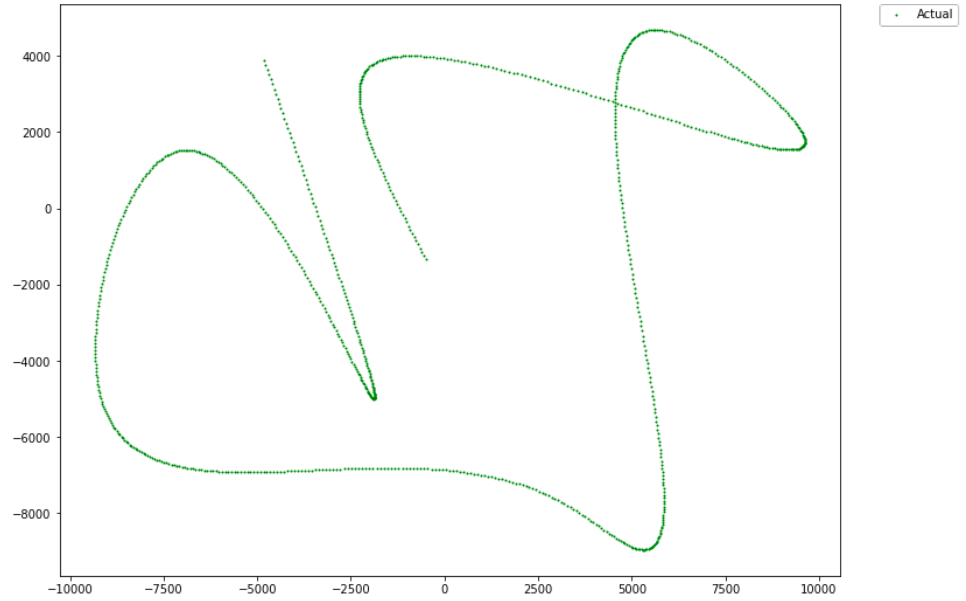
```
traj01.columns = ["X", "Y", "Speed", "Time"]
traj02.columns = ["X", "Y", "Speed", "Time"]
traj03.columns = ["X", "Y", "Speed", "Time"]
traj04.columns = ["X", "Y", "Speed", "Time"]
traj05.columns = ["X", "Y", "Speed", "Time"]
traj06.columns = ["X", "Y", "Speed", "Time"]
traj07.columns = ["X", "Y", "Speed", "Time"]
traj08.columns = ["X", "Y", "Speed", "Time"]
traj09.columns = ["X", "Y", "Speed", "Time"]
traj10.columns = ["X", "Y", "Speed", "Time"]
```

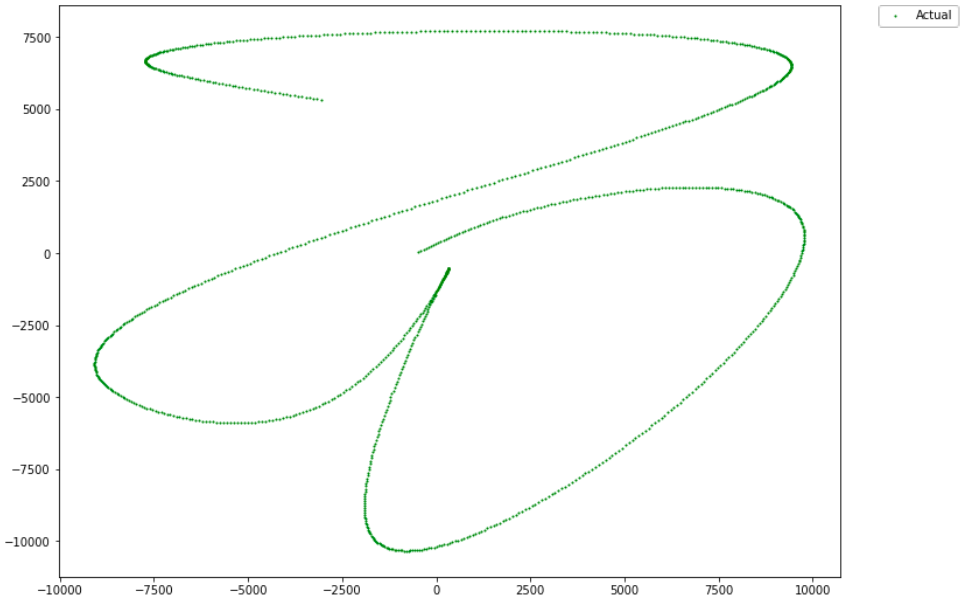
```
traj01,traj10
```

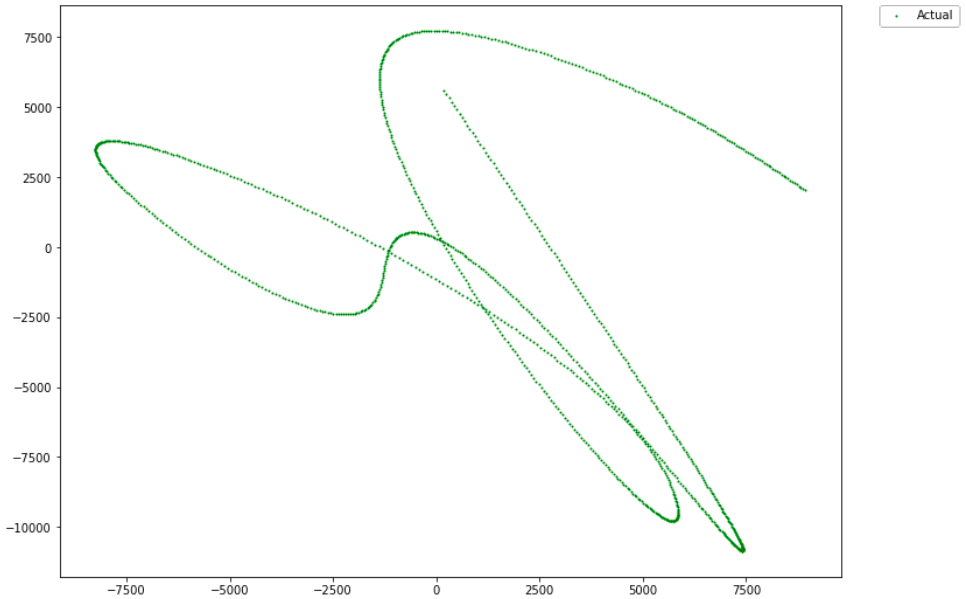
(	X	Y	Speed	Time
0	-1309.000000	-8881.000000	0.000000	0
1	-1364.323797	-8802.824599	95.771164	1
2	-1419.631820	-8724.650237	95.761204	2
3	-1474.908294	-8646.477955	95.741288	3
4	-1530.137444	-8568.308792	95.711426	4
..	...	...	...	...
835	1330.265530	-6807.994029	103.153956	835
836	1228.247784	-6825.039314	103.431921	836
837	1126.008103	-6842.050377	103.645206	837
838	1023.613862	-6859.037608	103.793769	838
839	921.132439	-6876.011395	103.877580	839

Sample plots from all routes









Transform time series data to supervised problem and concatenate all X Y dataset into 1 dataset

Feature Variable

Target Variable

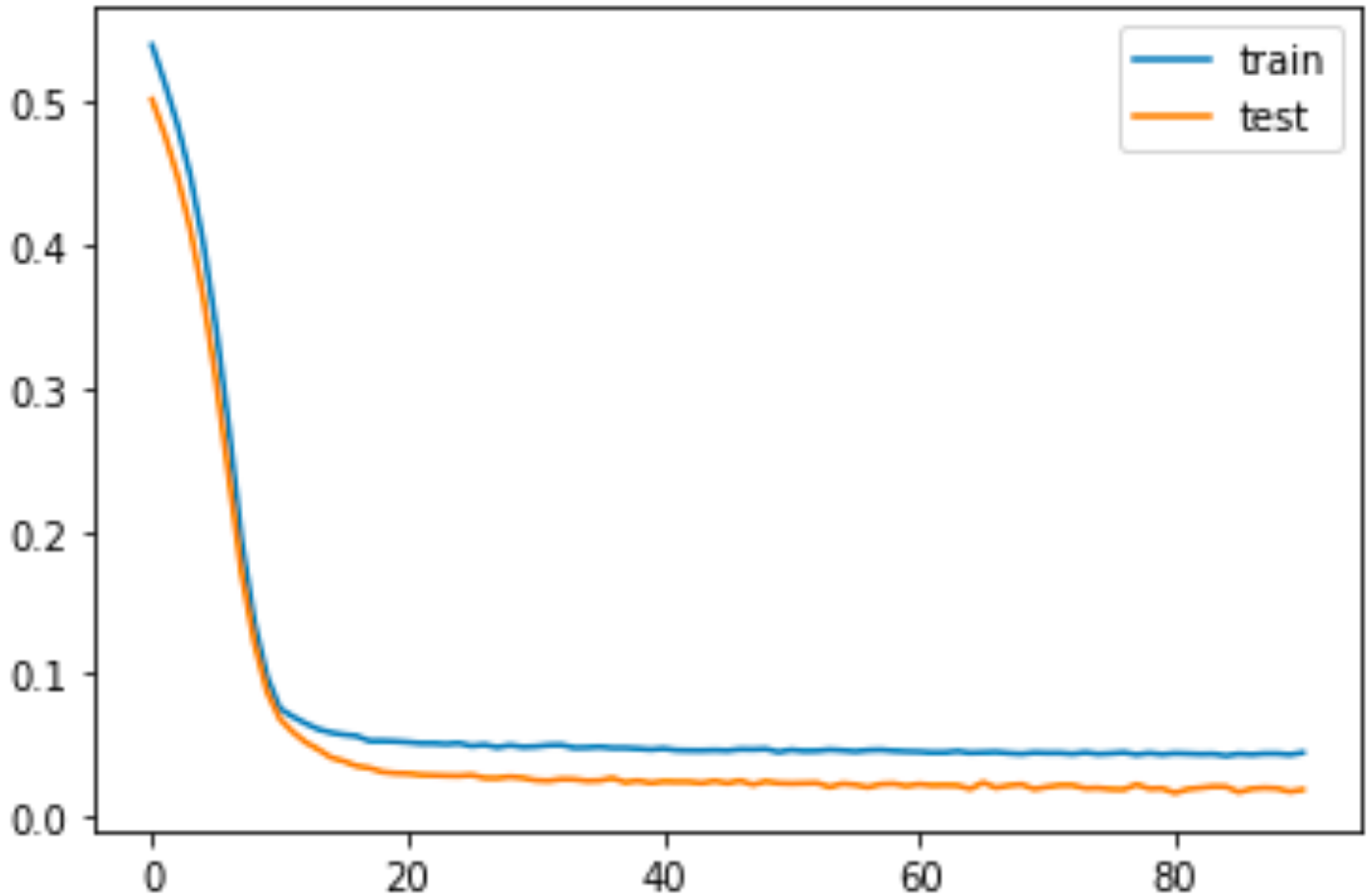
	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	-0.133186	-1.000000	-0.139738	-0.989572
2	-0.139738	-0.989572	-0.146288	-0.979145
3	-0.146288	-0.979145	-0.152834	-0.968717
4	-0.152834	-0.968717	-0.159374	-0.958290
5	-0.159374	-0.958290	-0.165907	-0.947864

	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	0.617546	0.027658	0.625560	0.019306
2	0.625560	0.019306	0.633571	0.010955
3	0.633571	0.010955	0.641574	0.002607
4	0.641574	0.002607	0.649567	-0.005739
5	0.649567	-0.005739	0.657545	-0.014080

Single LSTM model

```
# design single LSTM network
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',mode='auto', patience=10,restore_best_weights=True)
model1 = Sequential()
model1.add(LSTM(64,activation='relu' ,input_shape=(train_X.shape[1], train_X.shape[2])))
model1.add(Dropout(0.1))
model1.add(Dense(test_y.shape[1]))
model1.compile(loss='mae', optimizer='adam',metrics=['accuracy'])
model1.summary()
# fit network
history1 = model1.fit(train_X, train_y, epochs=600, callbacks=[callback], batch_size=72, validation_data=(test_X,
test_y), verbose=2, shuffle=False)

# plot history
pyplot.plot(history1.history['loss'], label='train')
pyplot.plot(history1.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

Epoch 91/600 30/30 - 0s - loss: 0.0451 - accuracy: 0.9890 - val_loss: 0.0192 - val_accuracy: 0.9889

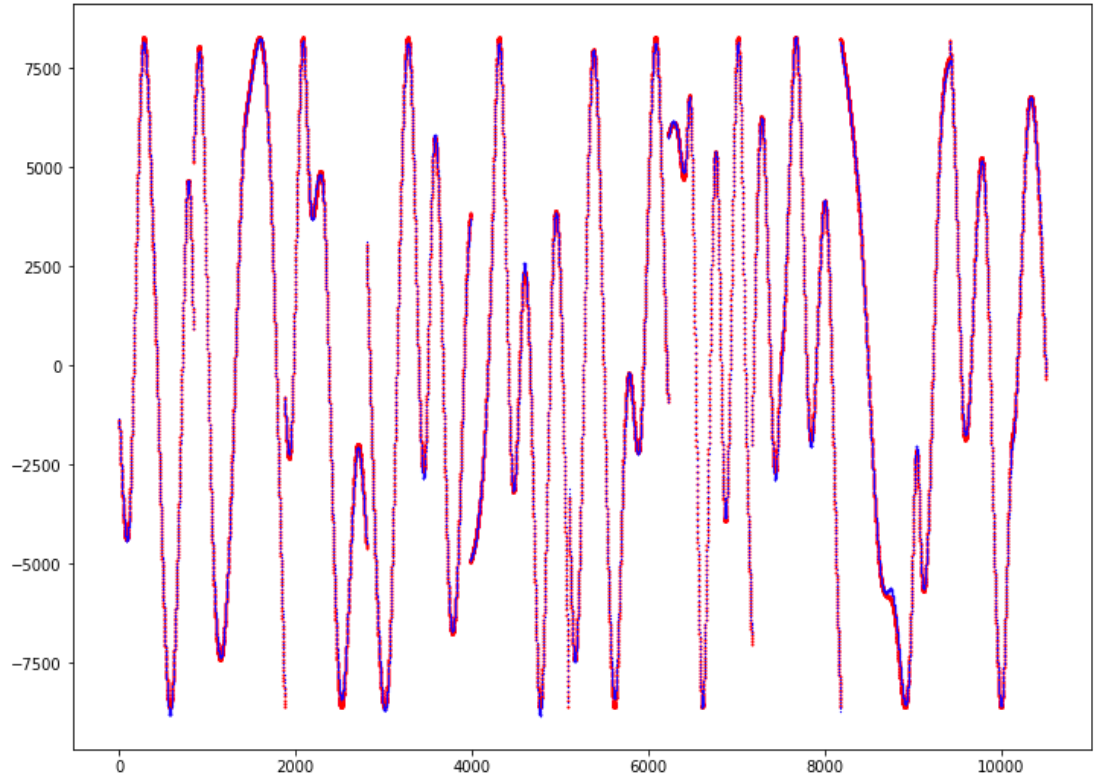


```
yhat = model1.predict(test_X)
Loss, accuracy = model1.evaluate(test_X, test_y)
print('Accuracy: %.4f' % (accuracy*100))

# calculate Loss
print('Loss: %.6f' % Loss)
```

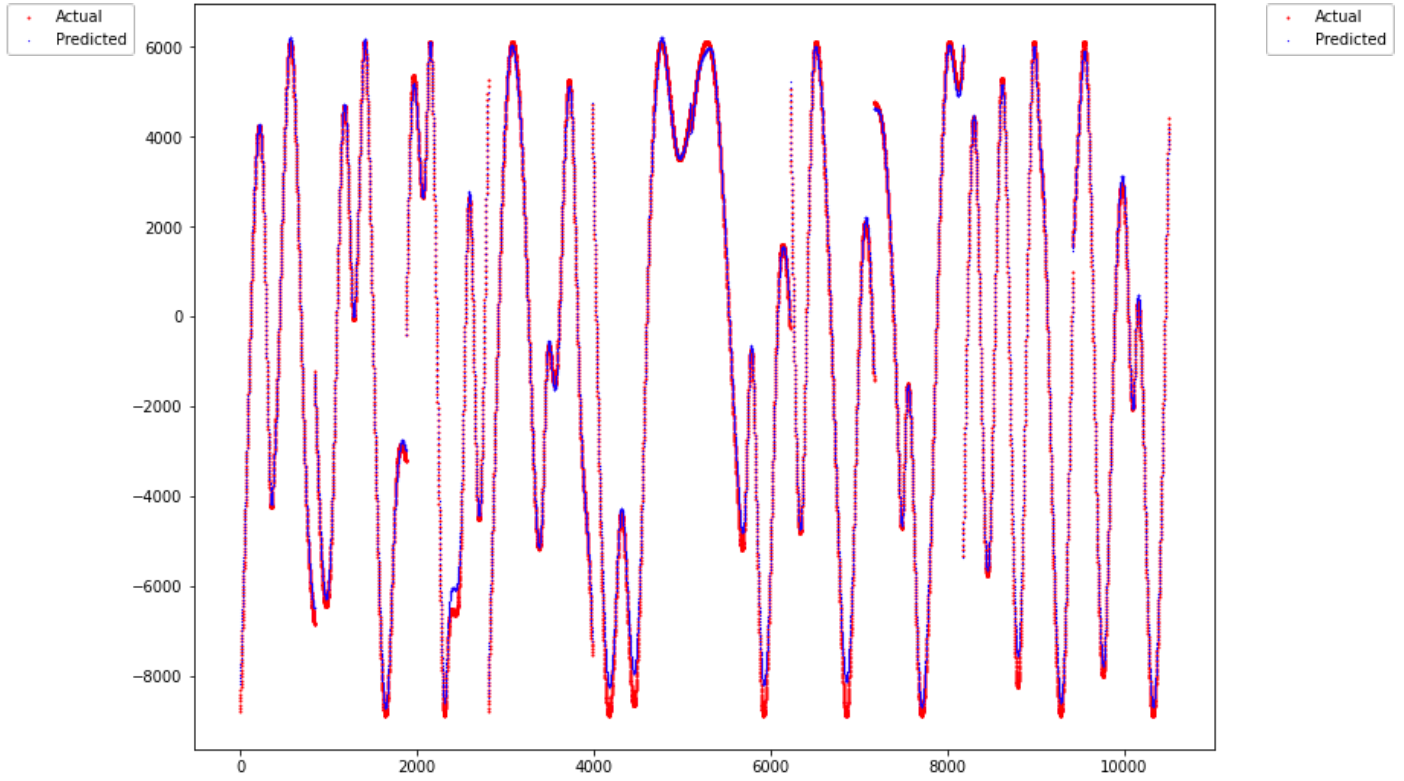
```
263/263 [==============================] - 0s 1ms/step - loss: 0.0168 - accuracy: 0.9925
Accuracy: 99.2502
Loss: 0.016845
```

Average RMSE for test dataset: 183.608555 m
Average RMSE for Train dataset: 123.043080 m

Plot of X positions



Plot of Y positions

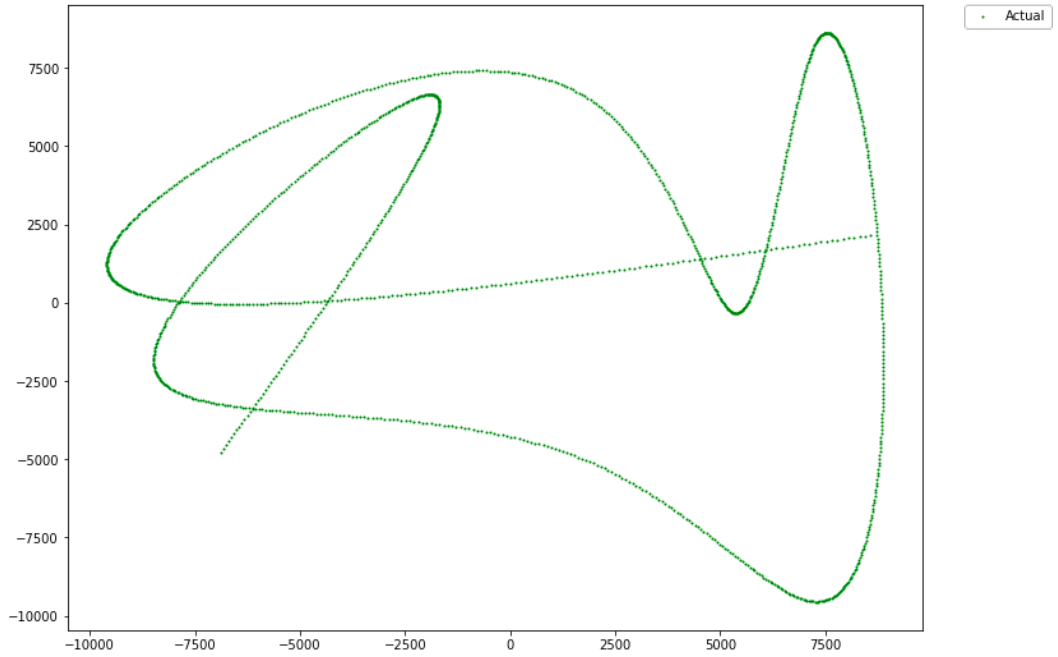


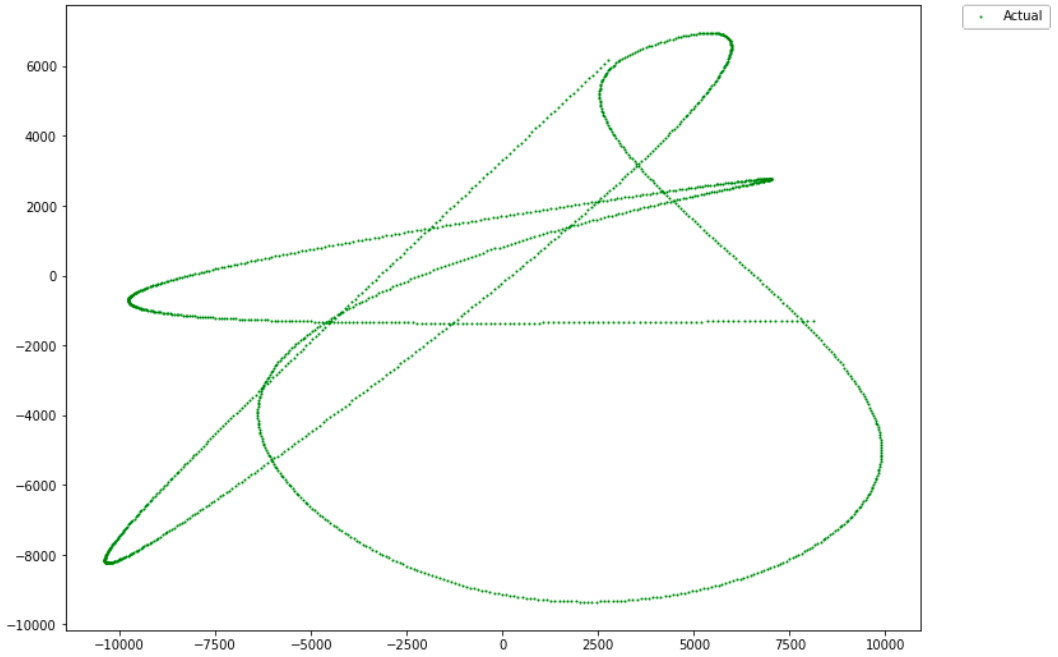
Plot of X and Y positions

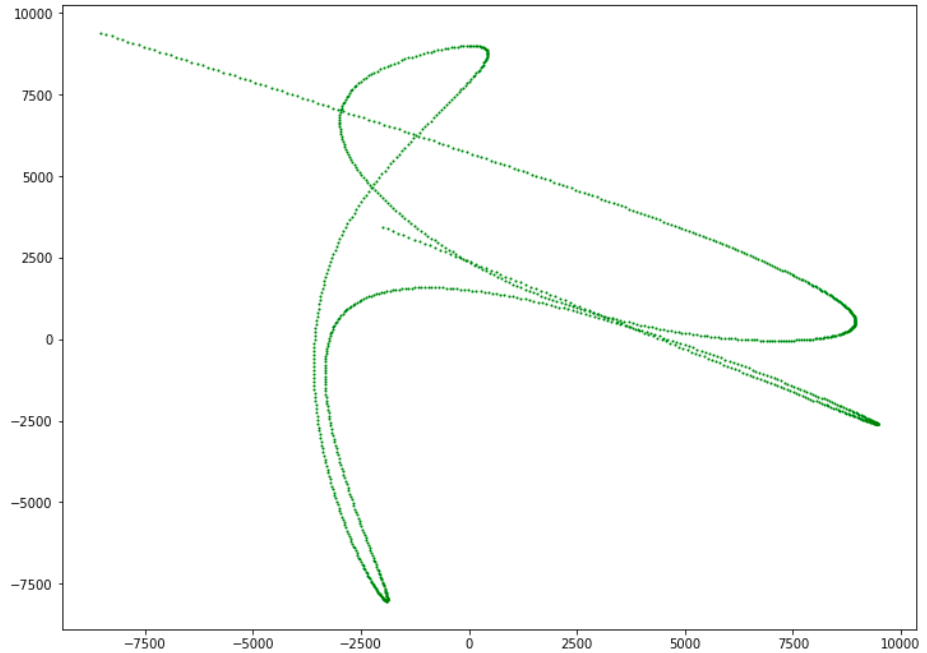
Try feeding 6 different paths and Let ML predicts

```
traj11 = pd.read_csv('UE0_12-11-2021_00-22-41.csv', low_memory=False,  header=None)
traj12 = pd.read_csv('UE1_12-11-2021_00-22-41.csv', low_memory=False,  header=None)
traj13 = pd.read_csv('UE2_12-11-2021_00-22-41.csv', low_memory=False,  header=None)
traj14 = pd.read_csv('UE3_12-11-2021_00-22-41.csv', low_memory=False,  header=None)
traj15 = pd.read_csv('UE4_12-11-2021_00-22-41.csv', low_memory=False,  header=None)
traj16 = pd.read_csv('UE5_12-11-2021_00-22-41.csv', low_memory=False,  header=None)

traj11.columns = ["X", "Y", "Speed", "Time"]
traj12.columns = ["X", "Y", "Speed", "Time"]
traj13.columns = ["X", "Y", "Speed", "Time"]
traj14.columns = ["X", "Y", "Speed", "Time"]
traj15.columns = ["X", "Y", "Speed", "Time"]
traj16.columns = ["X", "Y", "Speed", "Time"]
```

Sample plots



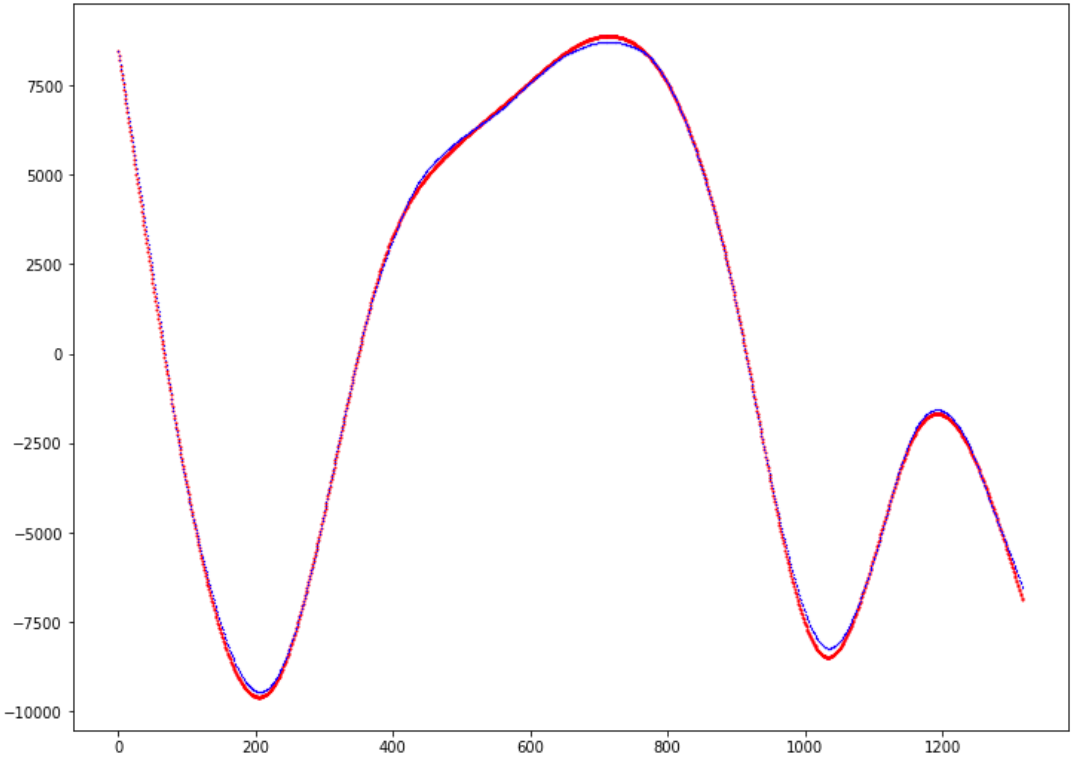




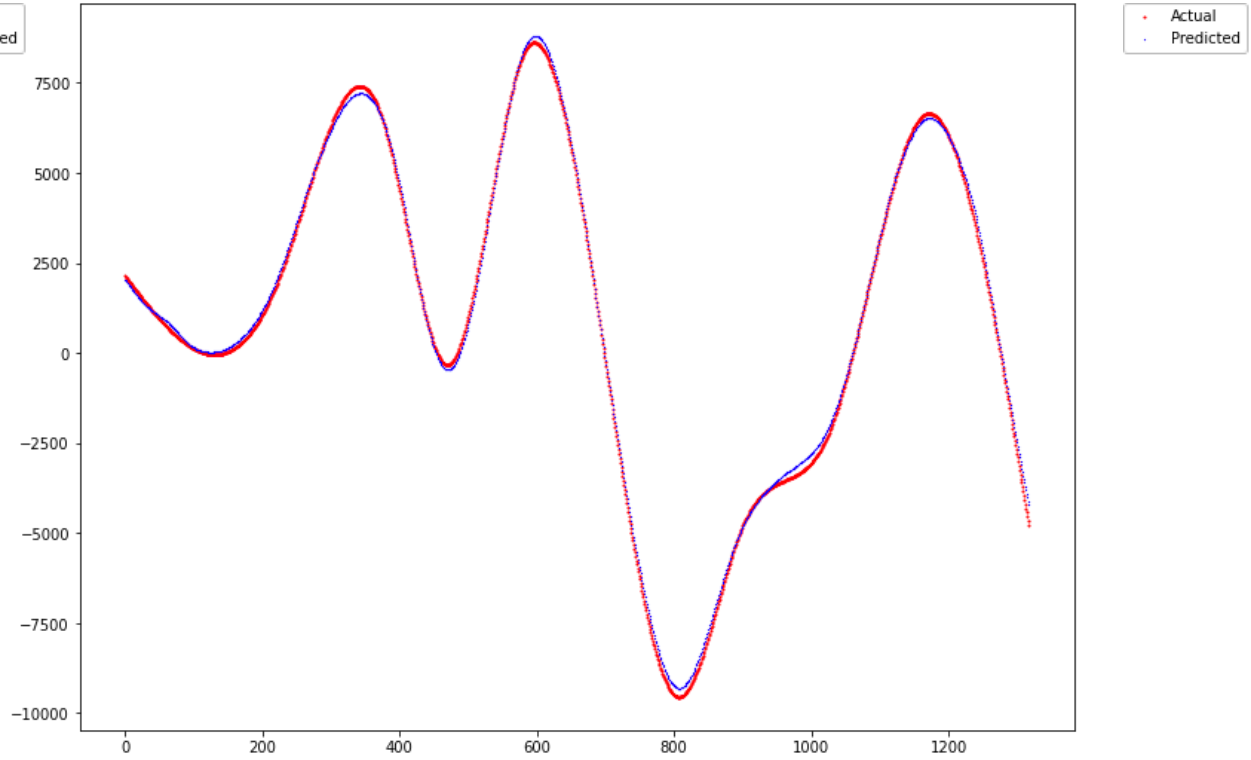
Prediction results

Actual vs Predicted plots for trajectory 11

Plot of X positions

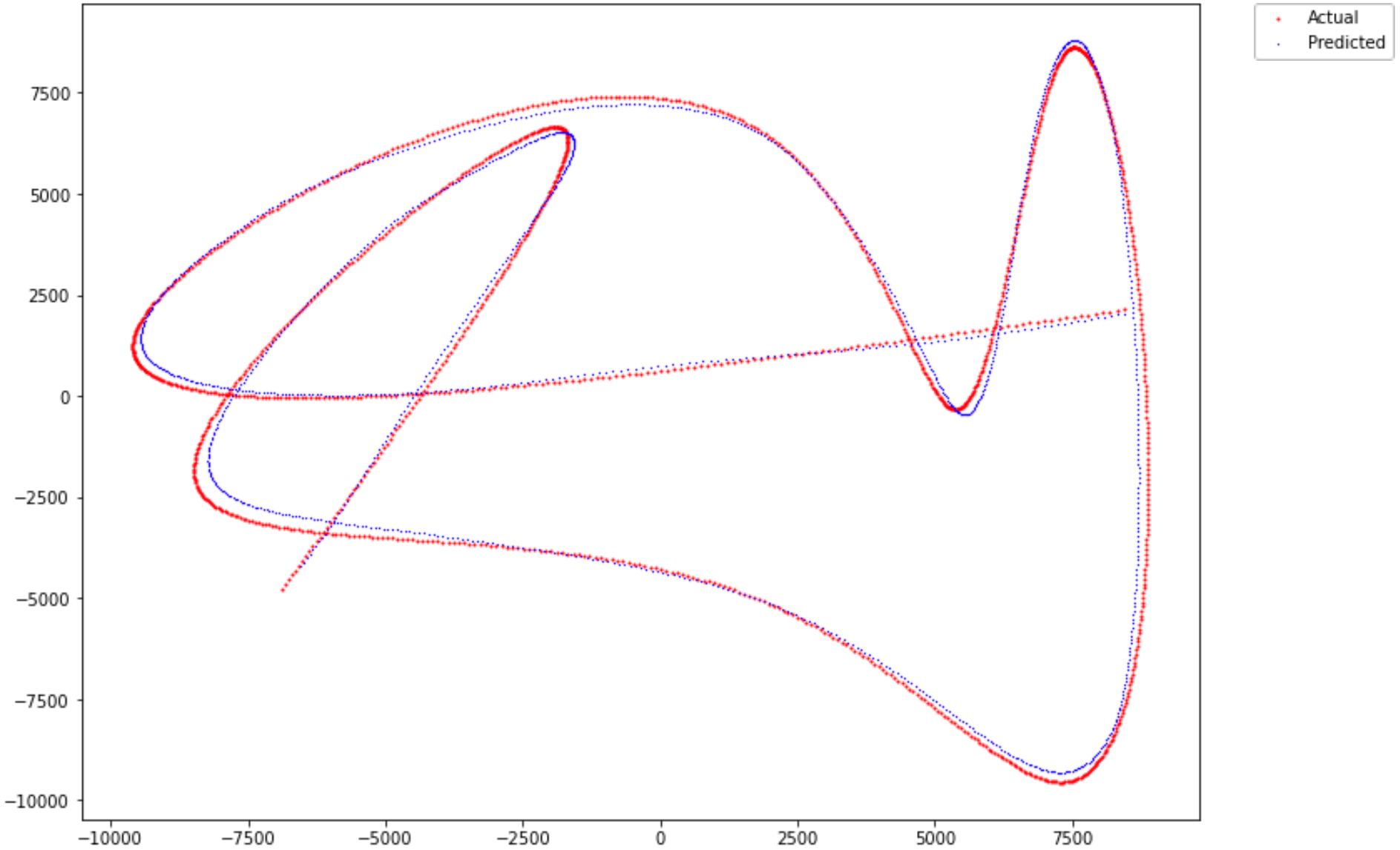


Plot of Y positions



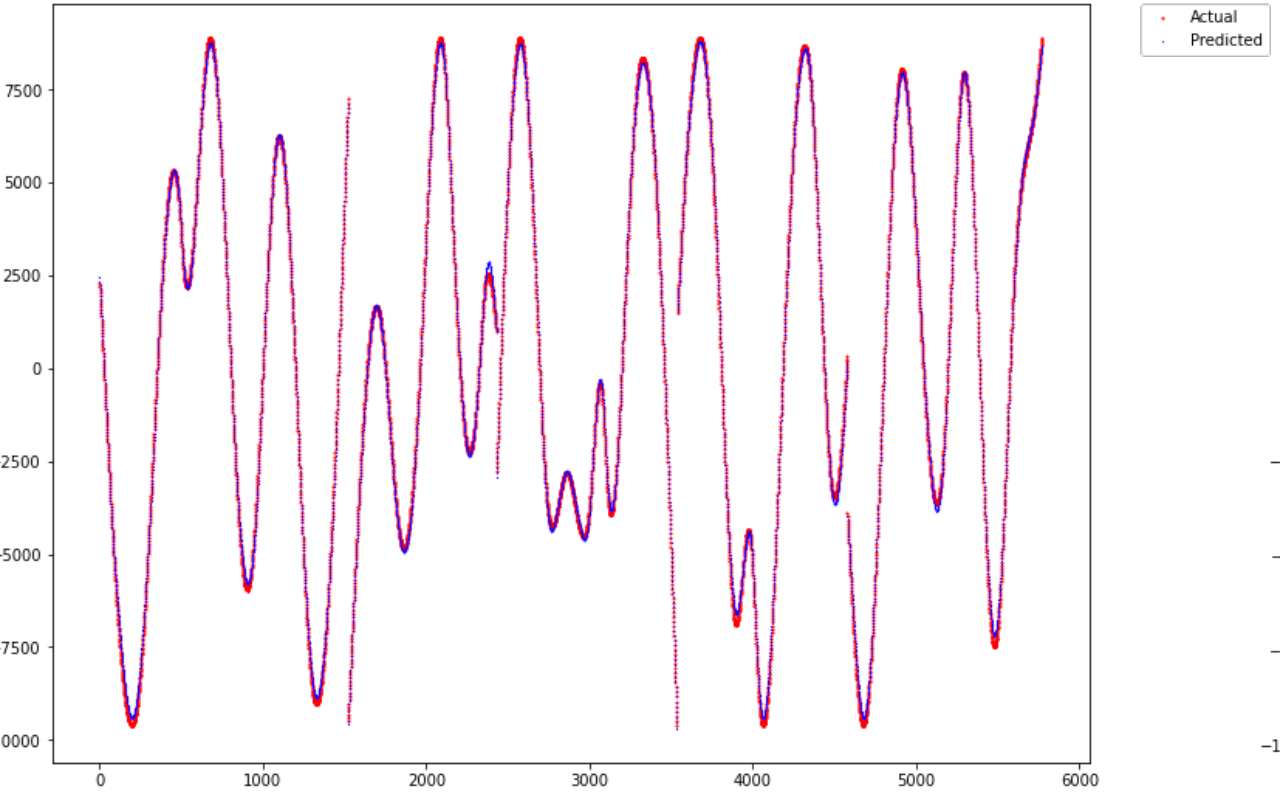
Average RMSE: 149.242959 m

Actual vs Predicted plot for trajectory 11

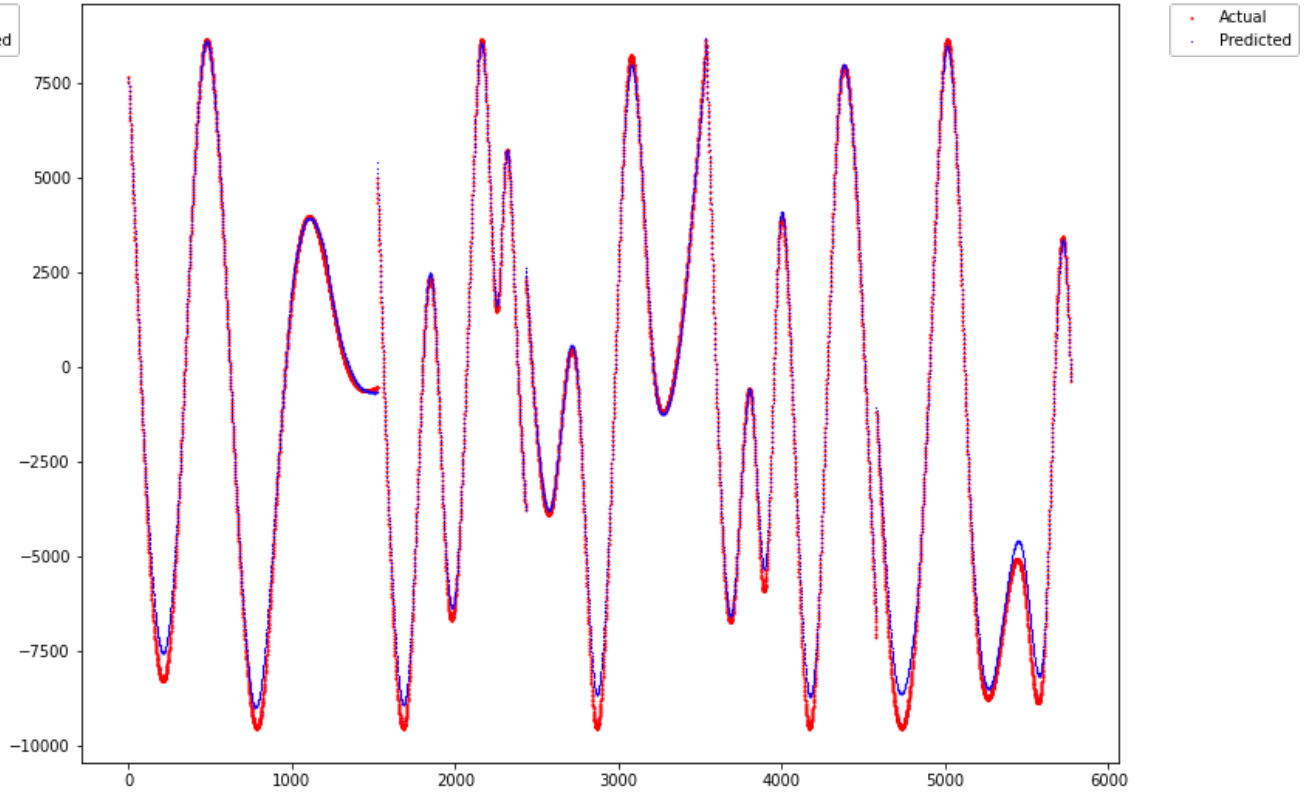


Actual vs Predicted plots for trajectory 12-16

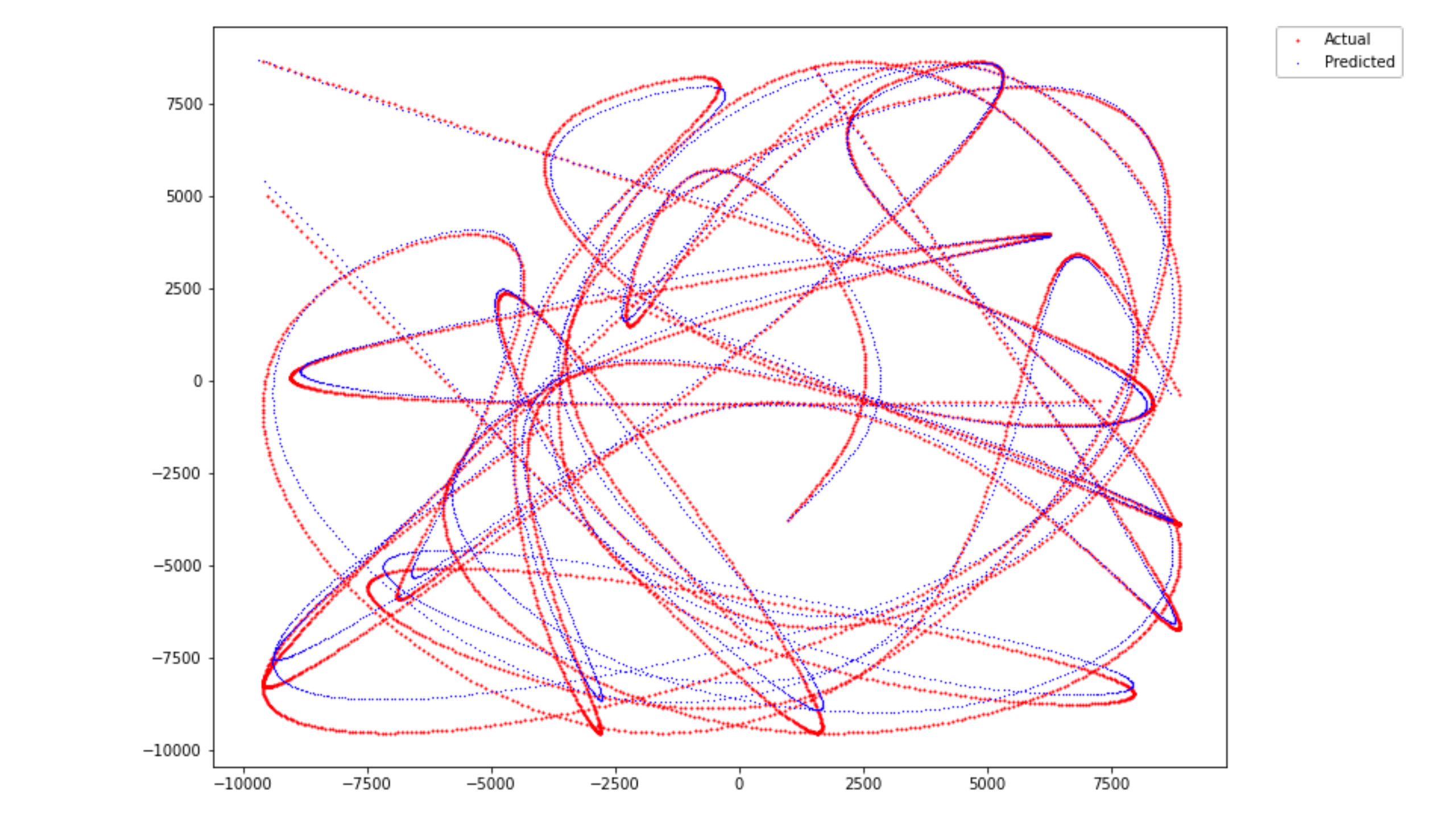
Plot of X positions



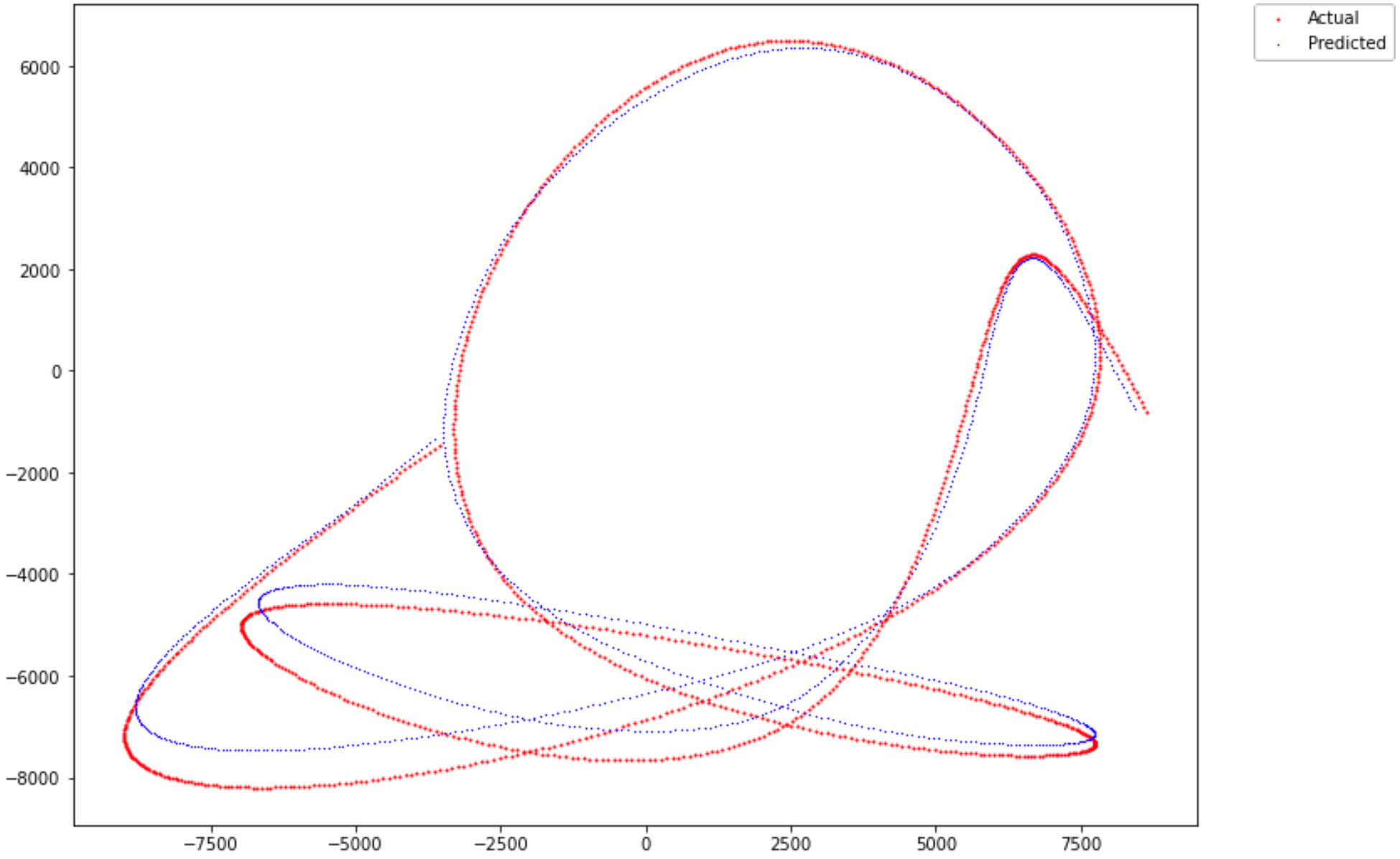
Plot of Y positions

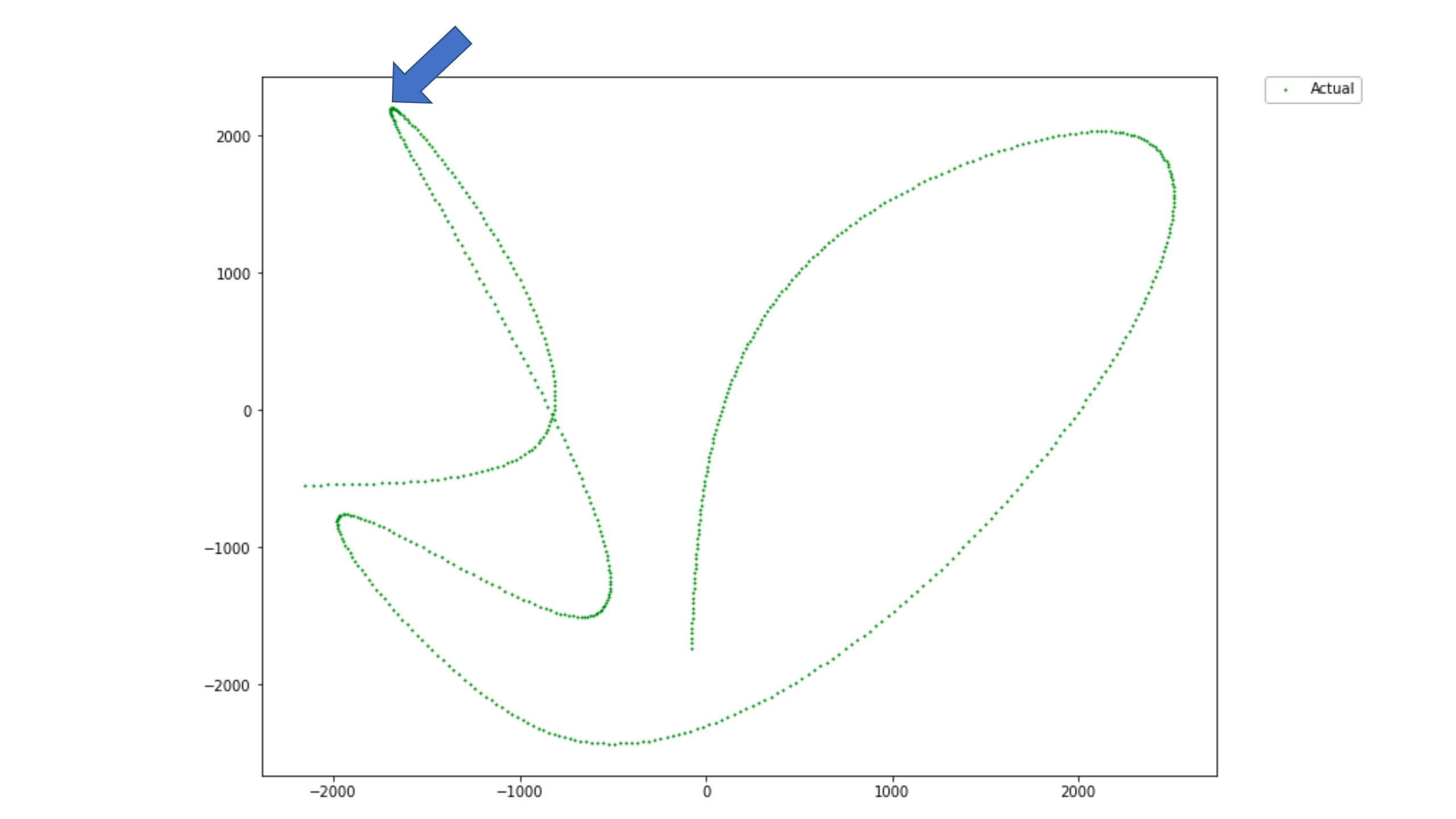


Average RMSE: 266.976796 m


Actual
Predicted
7500
5000
2500
0
−2500
−5000
−7500
−10000
−10000
−7500
−5000
−2500
0
2500
5000
7500

Actual vs Predicted plot for trajectory 16 only




Actual
2000
1000
0
−1000
−2000
−2000
−1000
0
1000
2000

Start points

1	**-2153.0**	**-549.0**	**0**	**0**
2	**-2111.3799640347000**	-548.1999280159790	41.627725243311300	1
3	**-2069.795973647960**	-547.3697803407030	41.59227574498410	2
4	**-2028.2840744183600**	-546.4794812829150	41.521445182710100	3
5	**-1986.8803119244600**	-545.4989551513590	41.41537130276440	4
6	**-1945.6207317448300**	-544.3981262547790	41.27426293537960	5
7	**-1904.5413794580400**	-543.1469189019190	41.09840269575460	6
8	**-1863.6783006426500**	-541.7152574015220	40.88815066647830	7
9	**-1823.0675408772300**	-540.073066062334	40.643949132918500	8
10	**-1782.7451457403400**	-538.1902691930970	40.36632846353280	9
11	**-1742.7471608105600**	-536.0367911025560	40.05591424907590	10
12	**-1703.1096316664600**	-533.582556099455	39.71343583852310	11
13	**-1663.8686038865900**	-530.7974884925370	39.339736435258300	12
14	**-1625.060123049520**	-527.651512590547	38.93578494465800	13
15	**-1586.720234733830**	-524.1145527022280	38.50268979319810	14

Peak points

106	**-1592.1274178120000**	2098.2391922157500	22.69403845319710	105
107	**-1604.5173604351300**	2115.4317130364400	21.191825083614500	106
108	**-1616.2423229383800**	2131.1919448800200	19.64331065441710	107
109	**-1627.2707419116900**	2145.4803272878700	18.049484670861000	108
110	**-1637.5710539450200**	2158.257299801350	16.411808388906800	109
111	**-1647.111695628320**	2169.483301961840	14.732513982226800	110
112	**-1655.8611035515300**	2179.118773310720	13.015162201139900	111
113	**-1663.7877143046200**	2187.1241533893700	11.26575644307060	112
114	**-1670.8599644775300**	2193.4598817391600	9.495165940124850	113
115	**-1677.0462906602200**	2198.086397901460	7.724977892435210	114
116	**-1682.3151294426300**	2200.964141417650	6.0035047980332	115
117	**-1686.6349540608400**	2202.053603243240	4.455088304577130	116
118	**-1689.9875696073000**	2201.333978785630	3.4289780346794200	117
119	**-1692.3878790966700**	2198.8308994832200	3.4679809167385000	118
120	**-1693.8558343381300**	2194.577080179140	4.49998569584034	119
121	**-1694.4113871408400**	2188.605235716450	5.997629965501500	120
122	**-1694.0744893139900**	2180.9480809382700	7.66456257347471	121
123	**-1692.8650926667400**	2171.6383306877000	9.38797581901777	122
124	**-1690.8031490082600**	2160.708699807820	11.12242971751580	123
125	**-1687.9086101477200**	2148.1919031417500	12.847122401343600	124
126	**-1684.2014278943000**	2134.120655532560	14.551398885984700	125

LSTM for X,Y prediction

X and Y data was chosen from the 10 datsets to be fed into ML

```
traj01.columns = ["X", "Y", "Speed", "Time"]
traj02.columns = ["X", "Y", "Speed", "Time"]
traj03.columns = ["X", "Y", "Speed", "Time"]
traj04.columns = ["X", "Y", "Speed", "Time"]
traj05.columns = ["X", "Y", "Speed", "Time"]
traj06.columns = ["X", "Y", "Speed", "Time"]
traj07.columns = ["X", "Y", "Speed", "Time"]
traj08.columns = ["X", "Y", "Speed", "Time"]
traj09.columns = ["X", "Y", "Speed", "Time"]
traj10.columns = ["X", "Y", "Speed", "Time"]
```

```
traj01,traj10
```

(	X	Y	Speed	Time
0	-1309.000000	-8881.000000	0.000000	0
1	-1364.323797	-8802.824599	95.771164	1
2	-1419.631820	-8724.650237	95.761204	2
3	-1474.908294	-8646.477955	95.741288	3
4	-1530.137444	-8568.308792	95.711426	4
..	...	...	...	...
835	1330.265530	-6807.994029	103.153956	835
836	1228.247784	-6825.039314	103.431921	836
837	1126.008103	-6842.050377	103.645206	837
838	1023.613862	-6859.037608	103.793769	838
839	921.132439	-6876.011395	103.877580	839

Transform time series data to supervised problem and concatenate all X Y dataset into 1 dataset

Feature Variable

Target Variable

	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	-0.133186	-1.000000	-0.139738	-0.989572
2	-0.139738	-0.989572	-0.146288	-0.979145
3	-0.146288	-0.979145	-0.152834	-0.968717
4	-0.152834	-0.968717	-0.159374	-0.958290
5	-0.159374	-0.958290	-0.165907	-0.947864

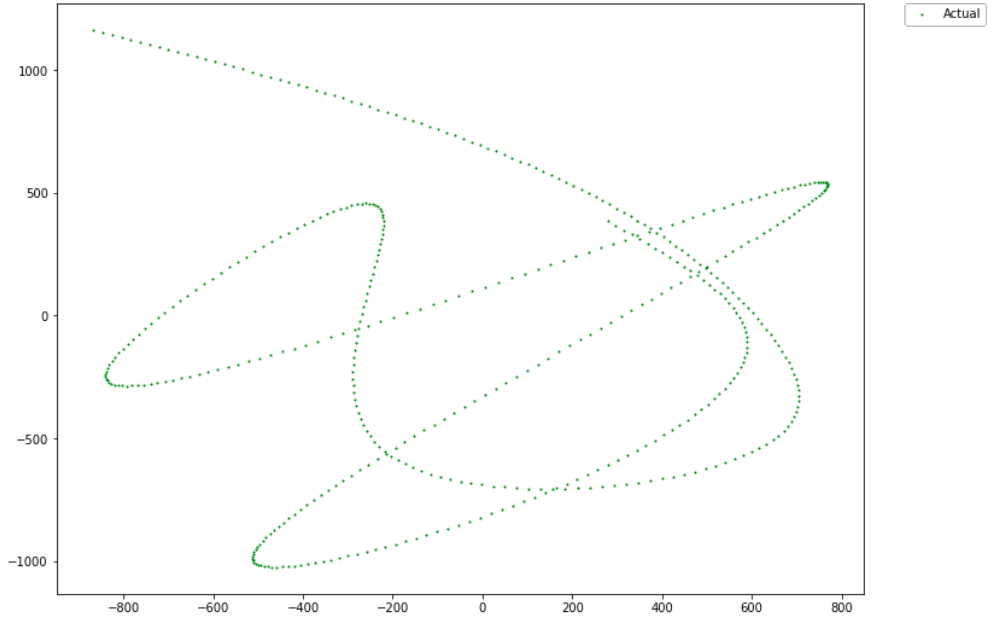
	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	0.617546	0.027658	0.625560	0.019306
2	0.625560	0.019306	0.633571	0.010955
3	0.633571	0.010955	0.641574	0.002607
4	0.641574	0.002607	0.649567	-0.005739
5	0.649567	-0.005739	0.657545	-0.014080

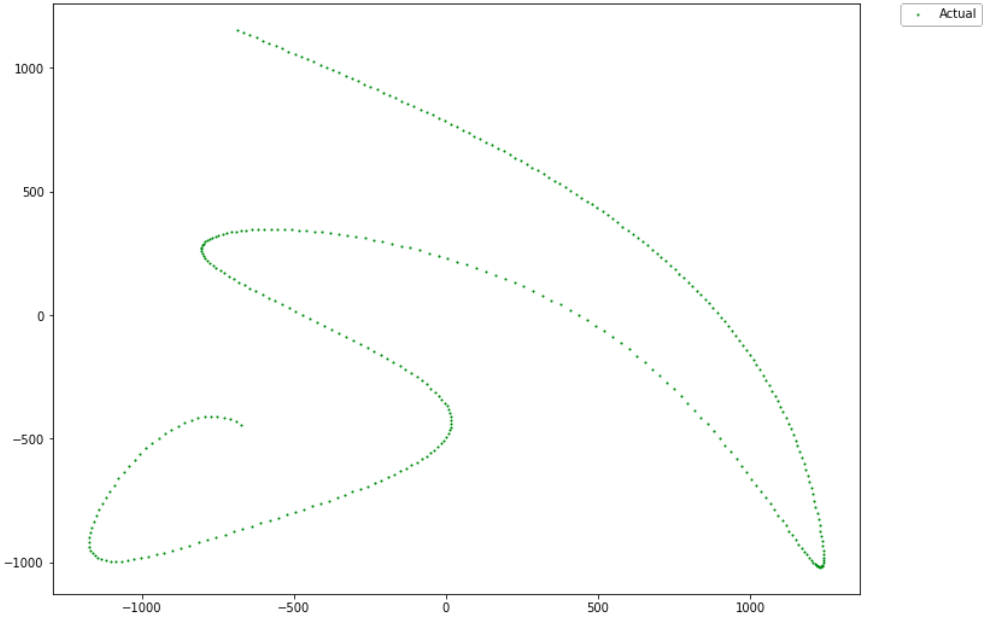
Bidirectional LSTM model

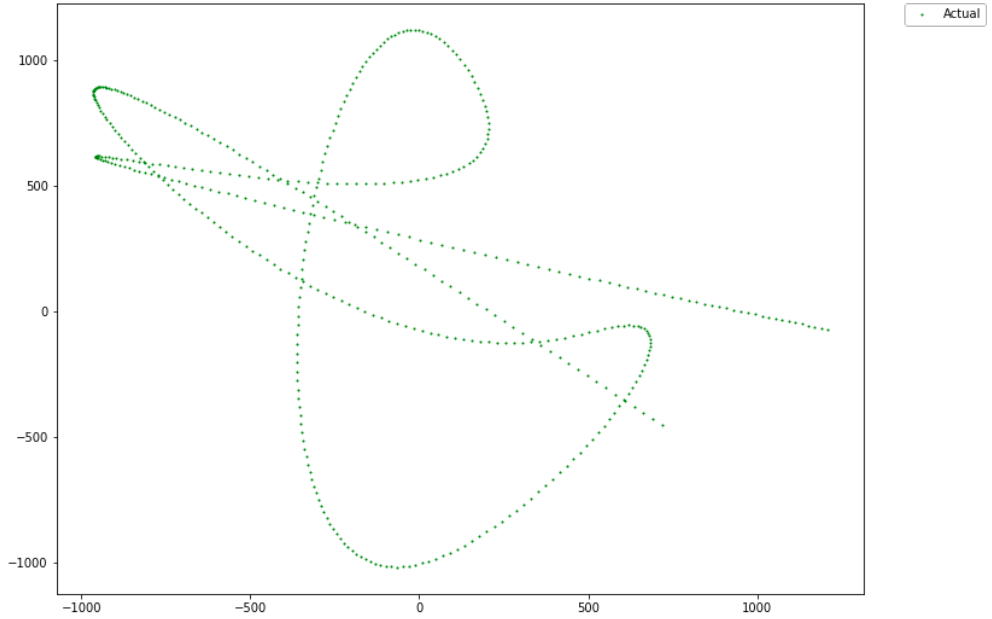
```
from keras.layers import Bidirectional

# design network
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',mode='auto', patience=10,restore_best_weights=True)
model1 = Sequential()
model1.add(Bidirectional(LSTM(150, input_shape=(train_X.shape[1], train_X.shape[2]))))
model1.add(Dropout(0.1))
model1.add(Dense(test_y.shape[1]))
model1.compile(loss='mae', optimizer='adam',metrics=['accuracy'])

# fit network
history1 = model1.fit(train_X, train_y, epochs=600, callbacks=[callback], batch_size=72, validation_data=(test_X,
test_y), verbose=2, shuffle=False)

# plot history
pyplot.plot(history1.history['loss'], label='train')
pyplot.plot(history1.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

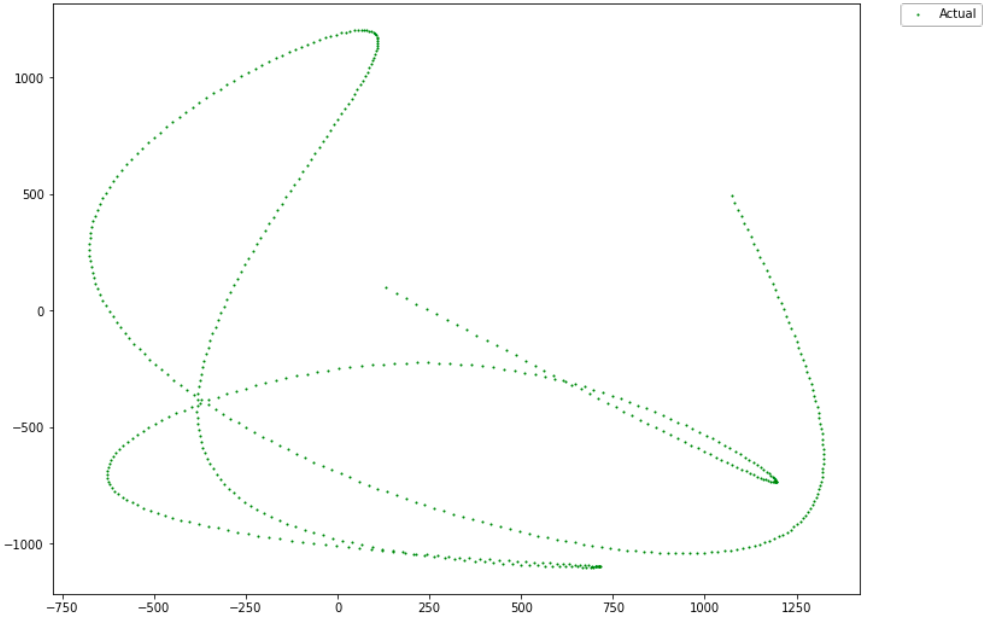
UMA with speed at 22.22 m/s

Sample plots from all routes

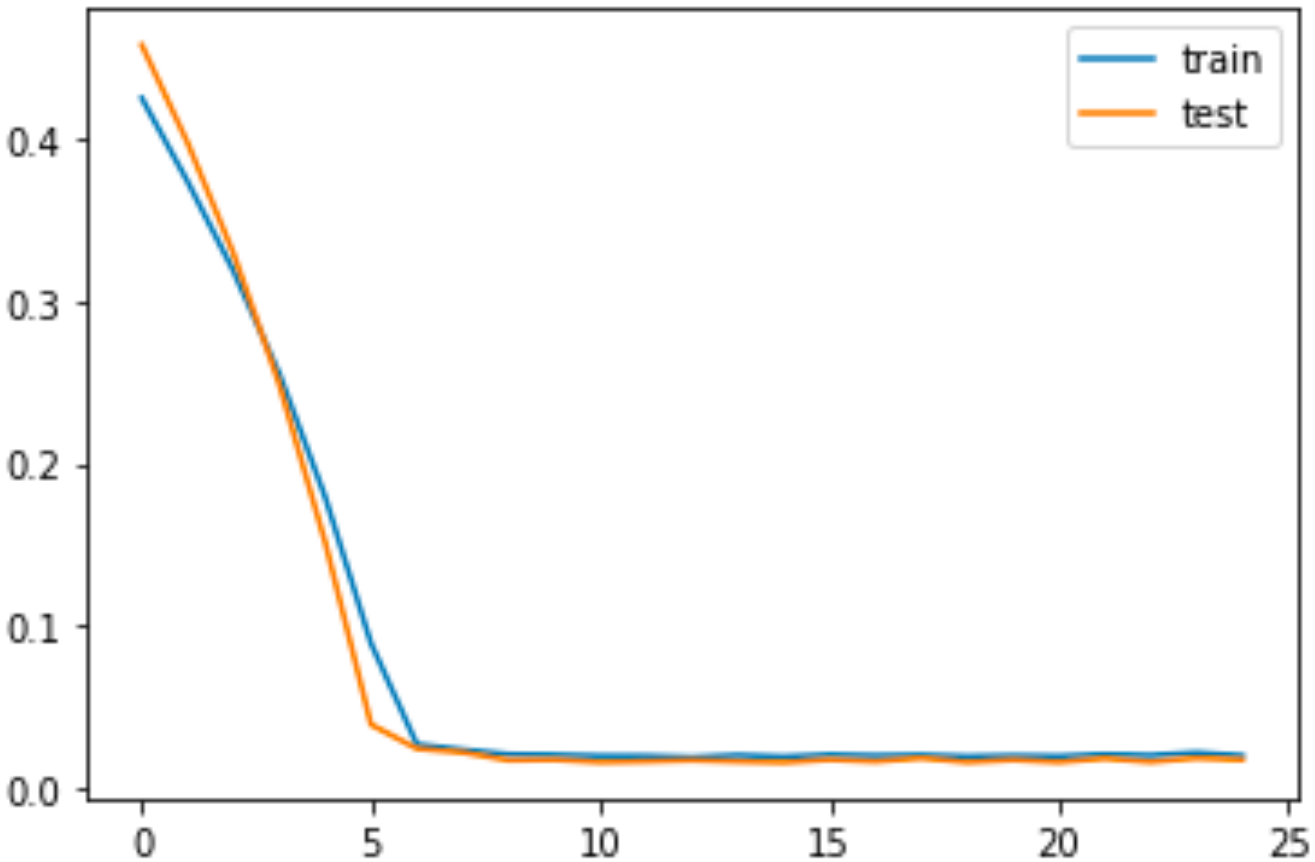








Epoch 25/600 15/15 - 0s - loss: 0.0204 - accuracy: 0.9943 - val_loss: 0.0178 - val_accuracy: 0.9891

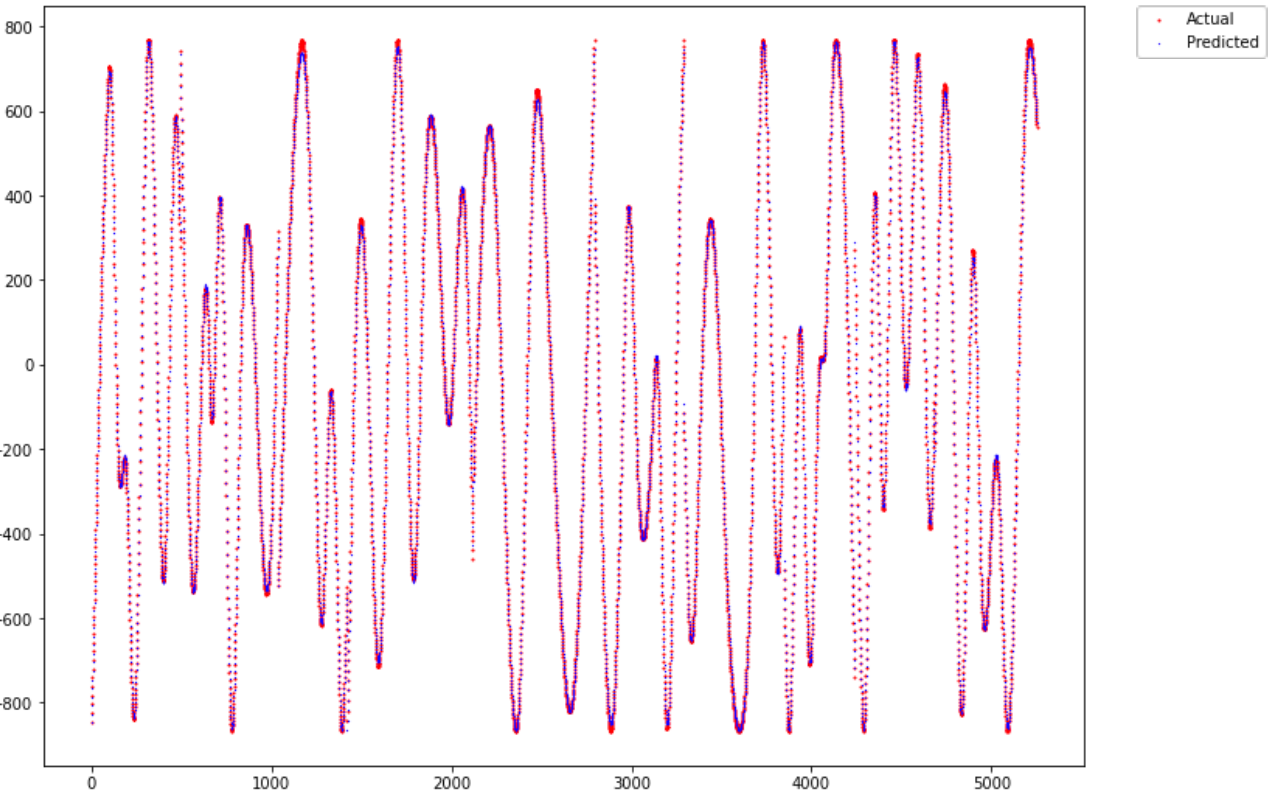


132/132 [==============================] - 0s 1ms/step - loss: 0.0163 - accuracy: 0.9919

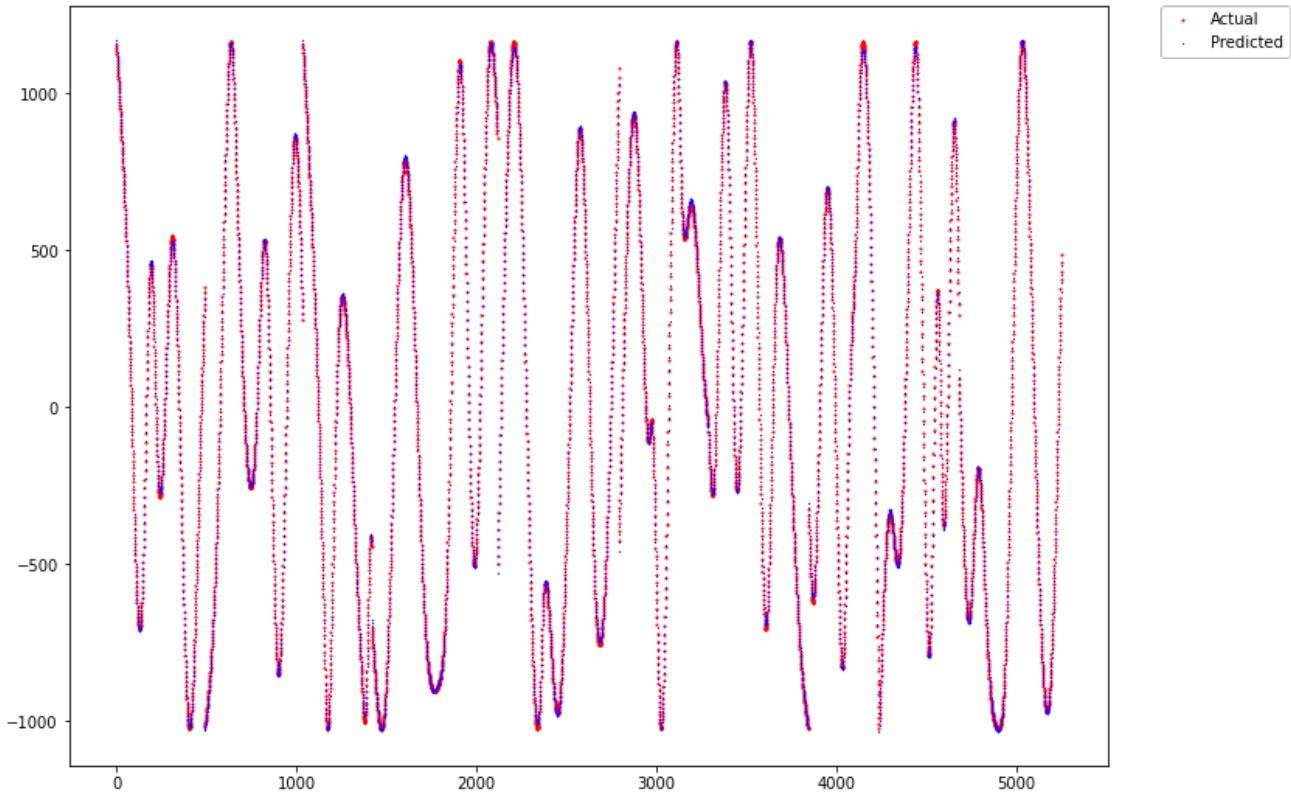
Accuracy: 99.1911
Loss: 0.016284

Average RMSE for Test RMSE: 18.483987 m
Average RMSE for Train RMSE: 15.566598 m

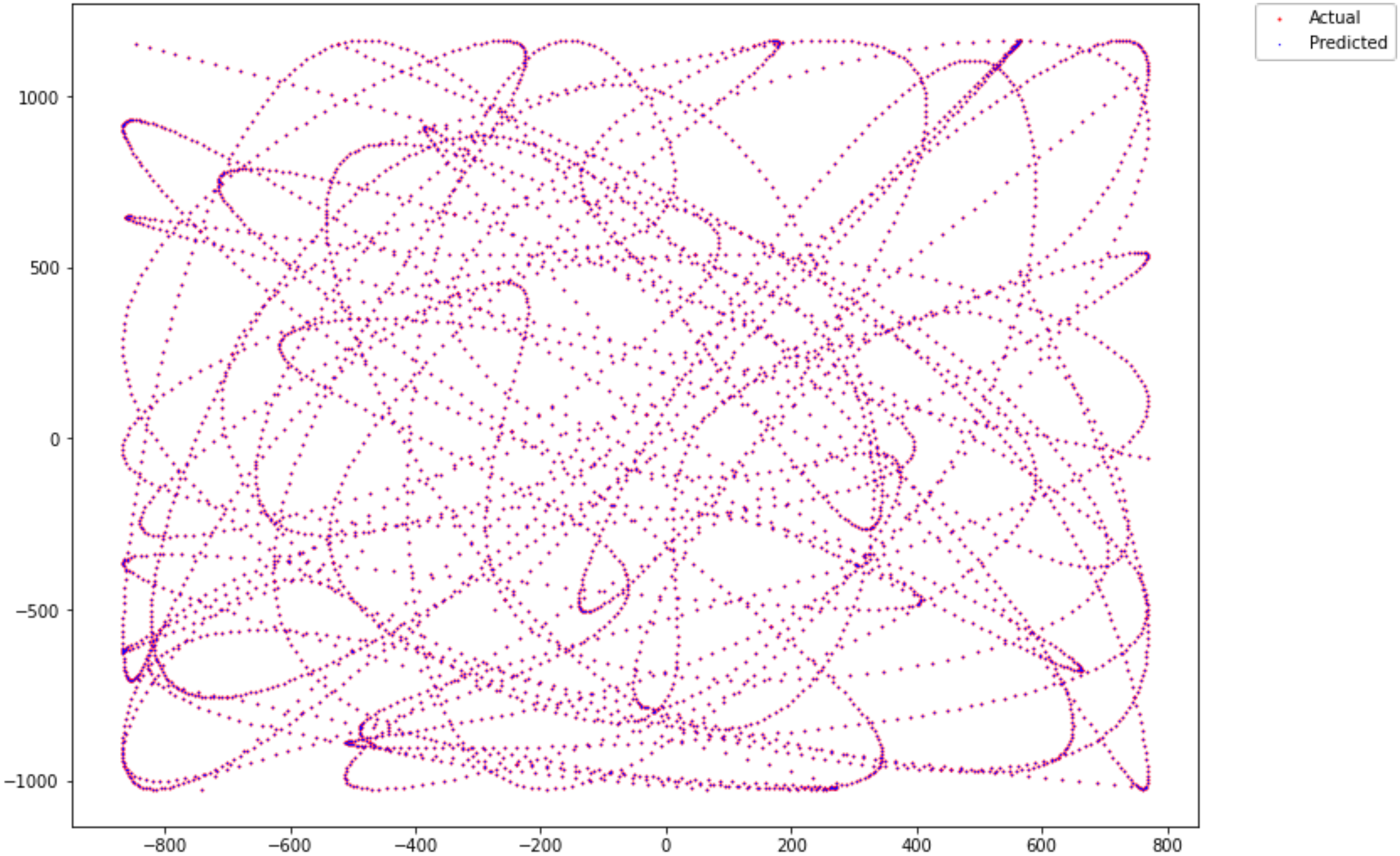
Plot of X positions



Plot of Y positions



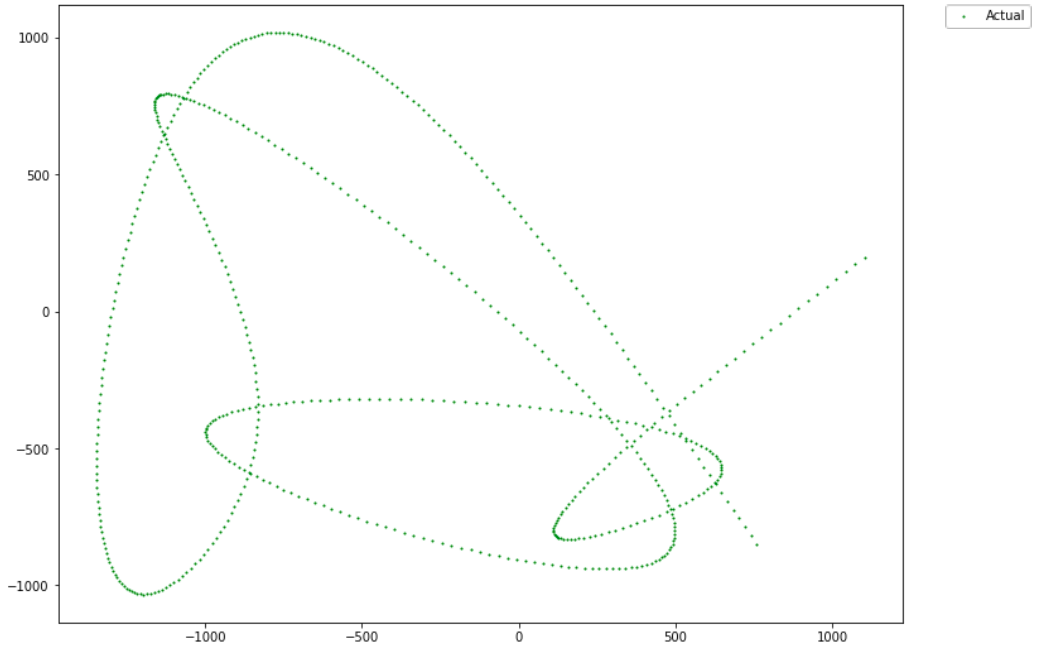
Plot of X and Y positions

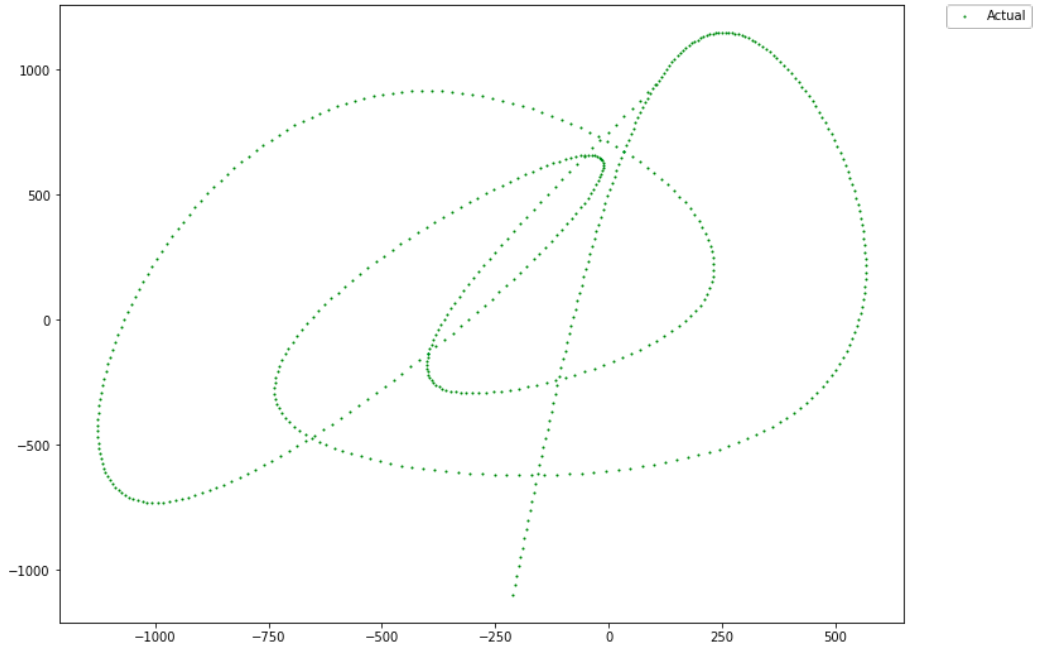


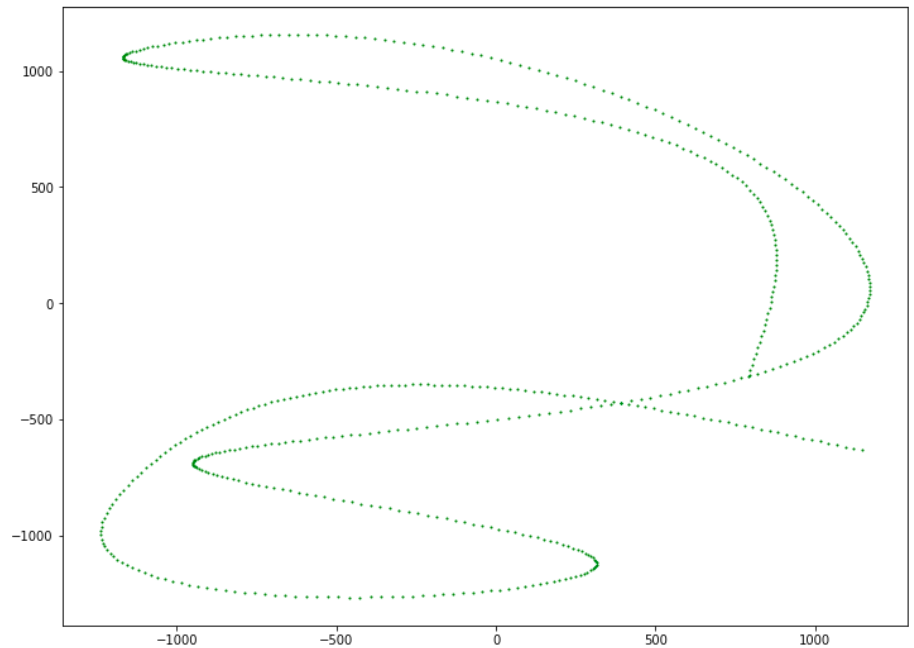
Try feeding 6 different paths and Let ML predicts

```
traj11 = pd.read_csv('UE5_18-11-2021_23-37-17.csv', low_memory=False,  header=None)
traj12 = pd.read_csv('UE5_18-11-2021_23-38-11.csv', low_memory=False,  header=None)
traj13 = pd.read_csv('UE6_18-11-2021_23-37-17.csv', low_memory=False,  header=None)
traj14 = pd.read_csv('UE7_18-11-2021_23-37-17.csv', low_memory=False,  header=None)
traj15 = pd.read_csv('UE8_18-11-2021_23-37-17.csv', low_memory=False,  header=None)
traj16 = pd.read_csv('UE9_18-11-2021_23-37-17.csv', low_memory=False,  header=None)

traj11.columns = ["X", "Y", "Speed", "Time"]
traj12.columns = ["X", "Y", "Speed", "Time"]
traj13.columns = ["X", "Y", "Speed", "Time"]
traj14.columns = ["X", "Y", "Speed", "Time"]
traj15.columns = ["X", "Y", "Speed", "Time"]
traj16.columns = ["X", "Y", "Speed", "Time"]
```

Sample plots



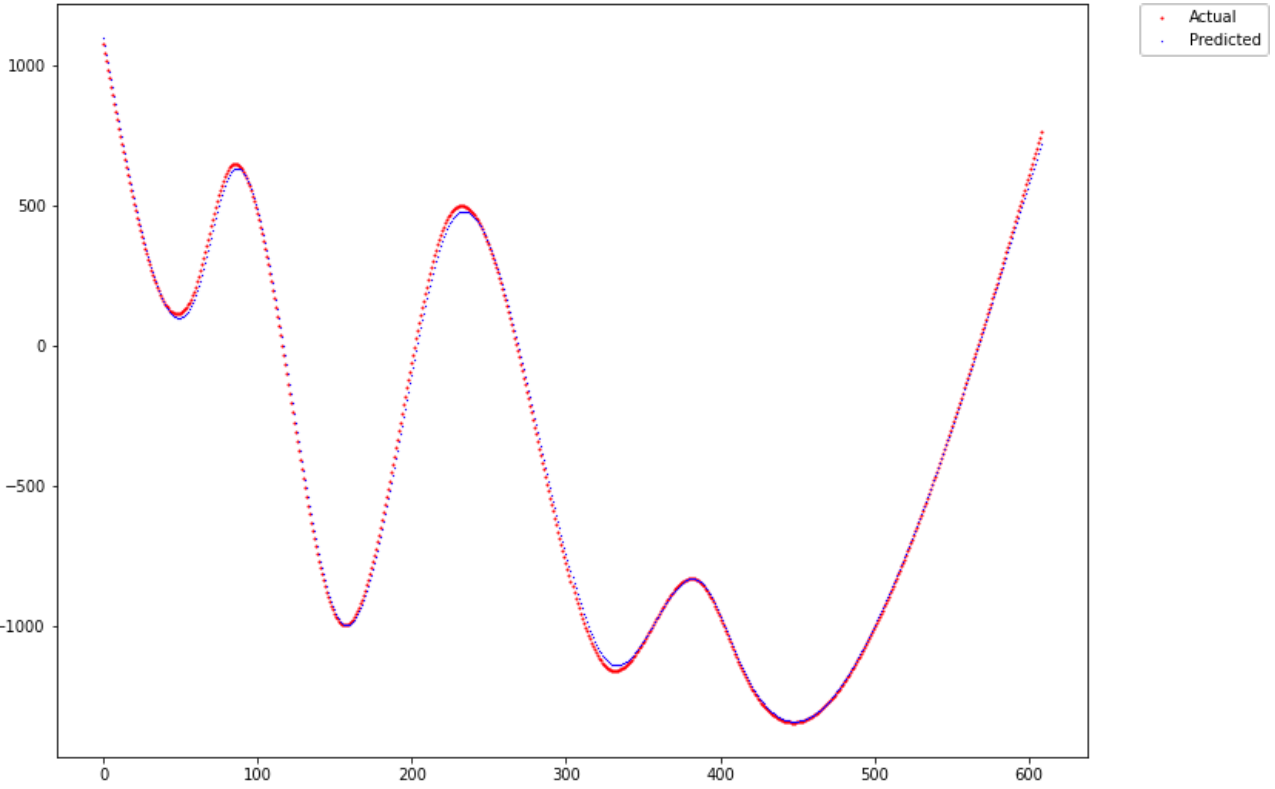




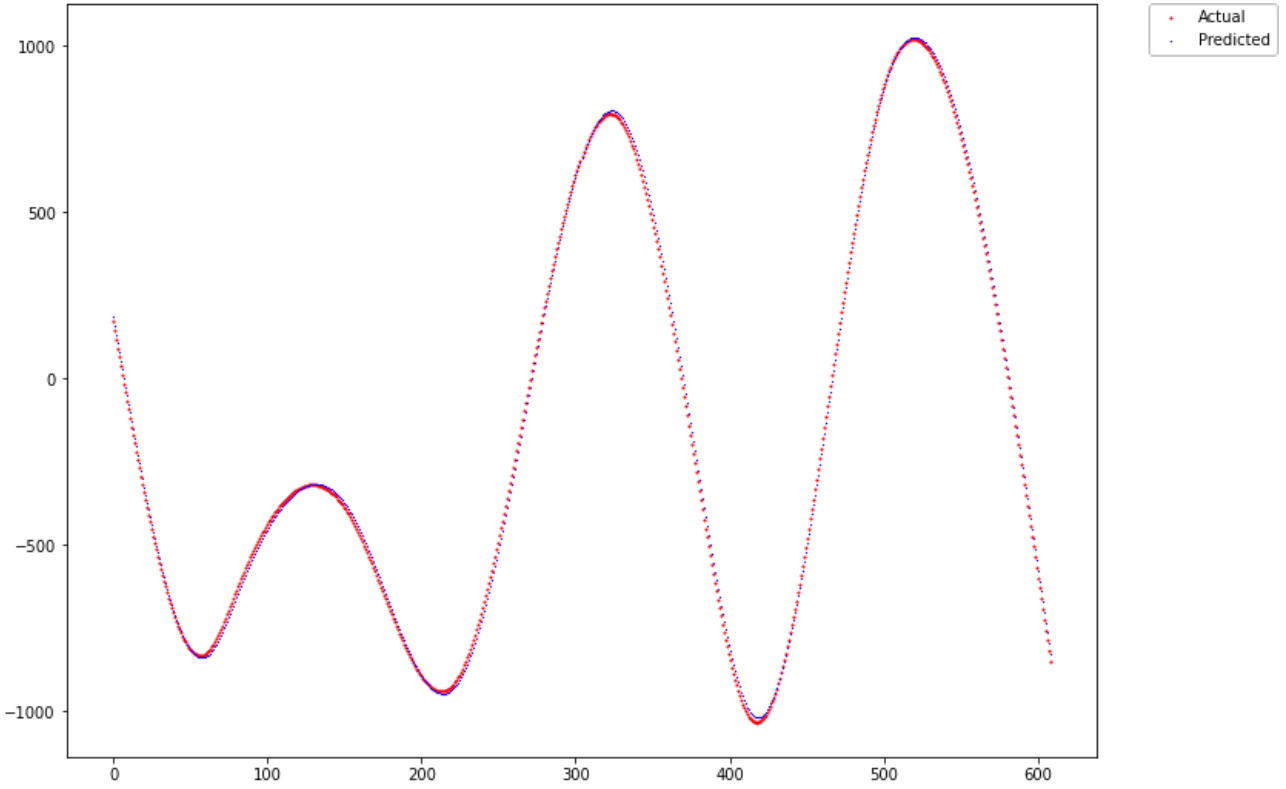
Prediction results

Actual vs Predicted plots for trajectory 11

Plot of X positions

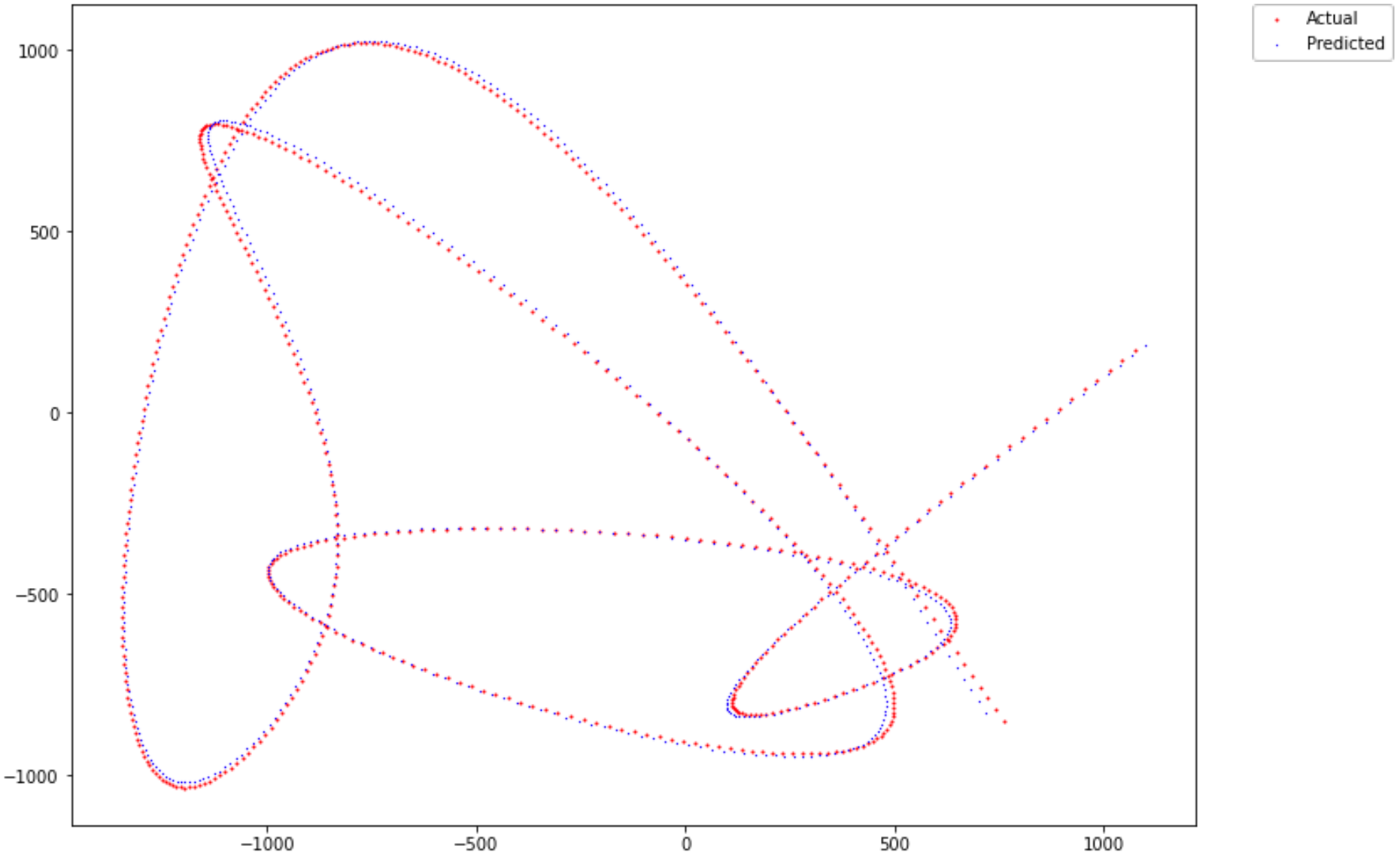


Plot of Y positions



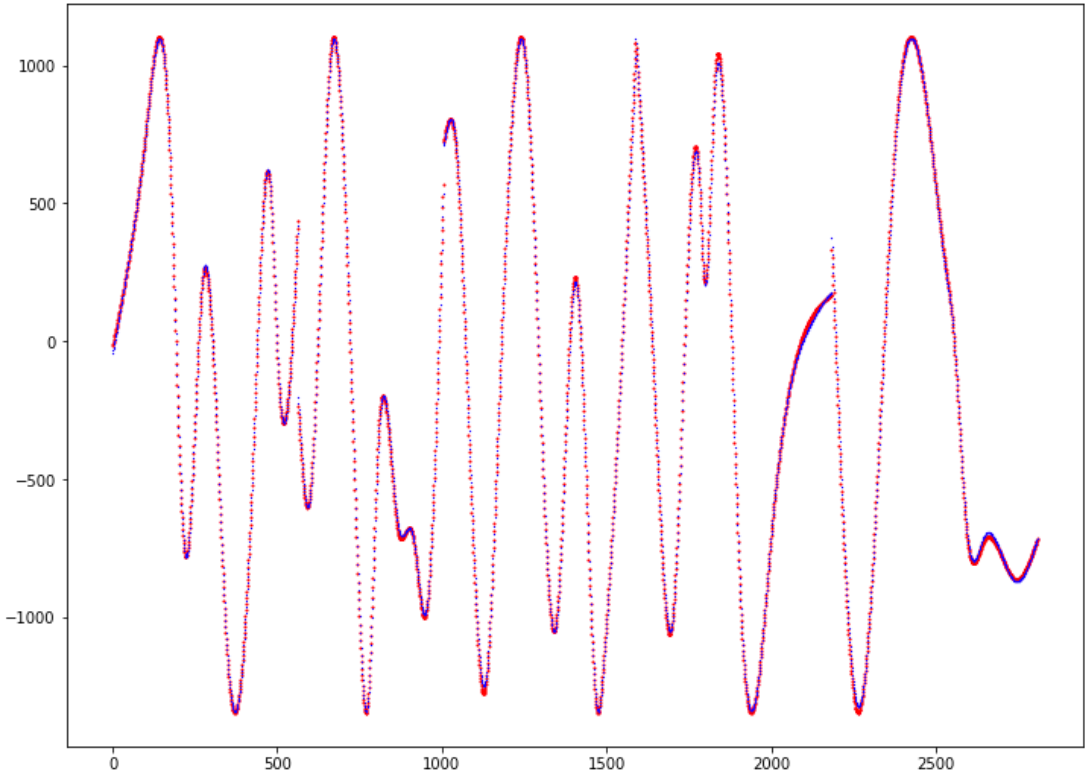
Average RMSE: 20.392048 m

Actual vs Predicted plot for trajectory 11

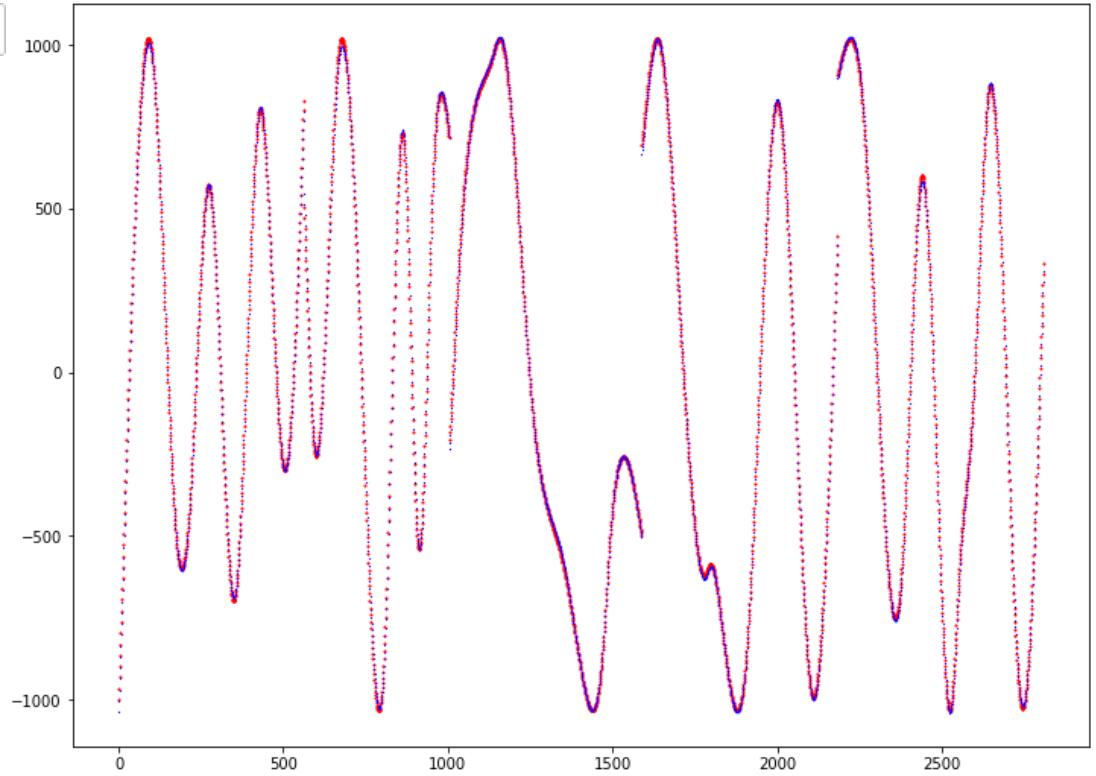


Actual vs Predicted plots for trajectory 12-16

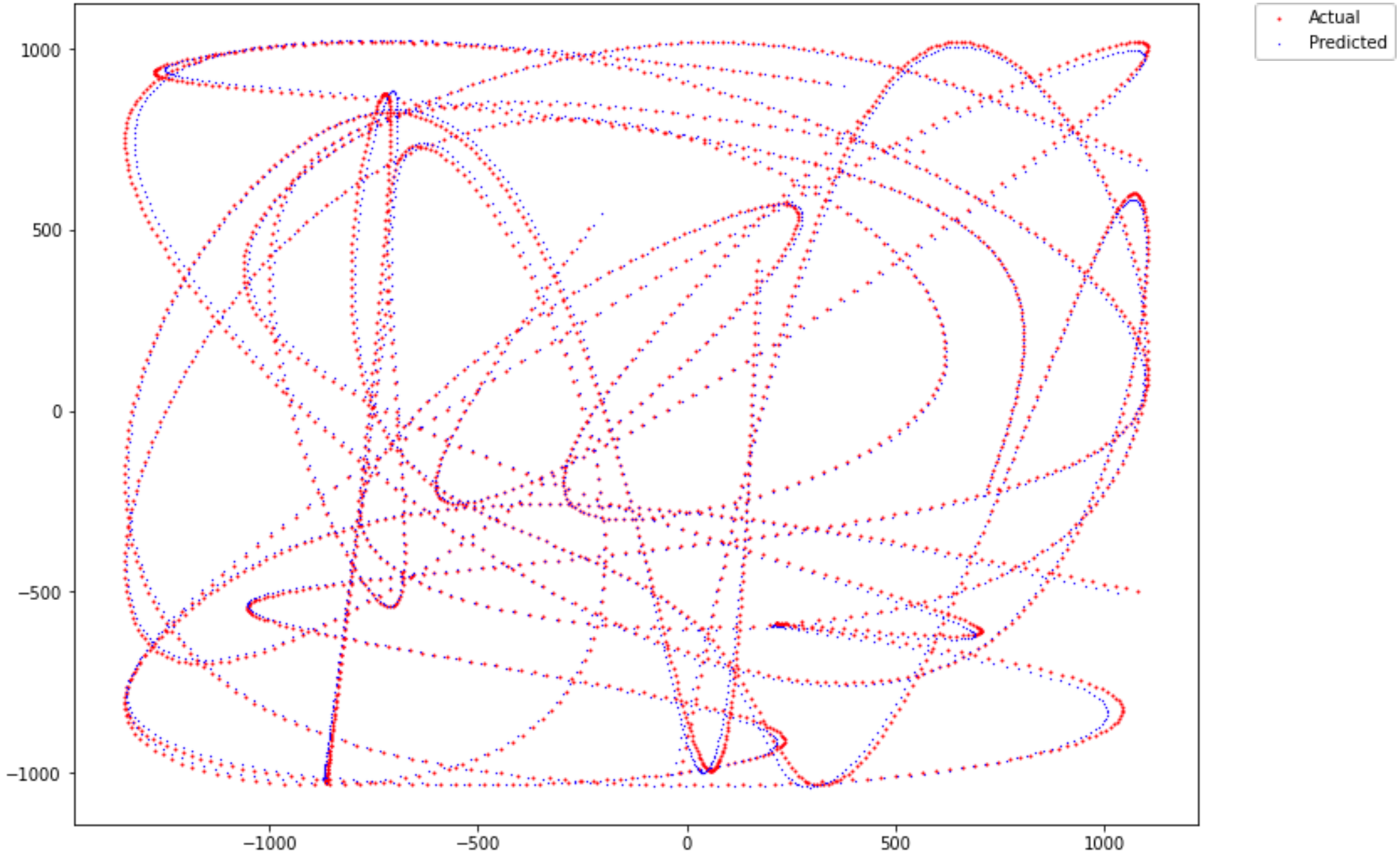
Plot of X positions



Plot of Y positions

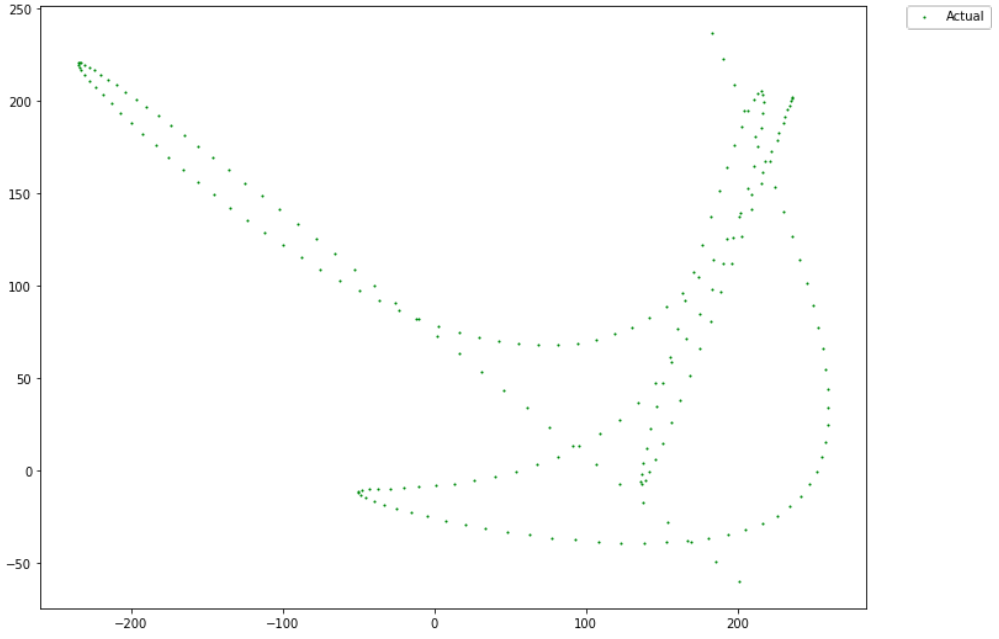


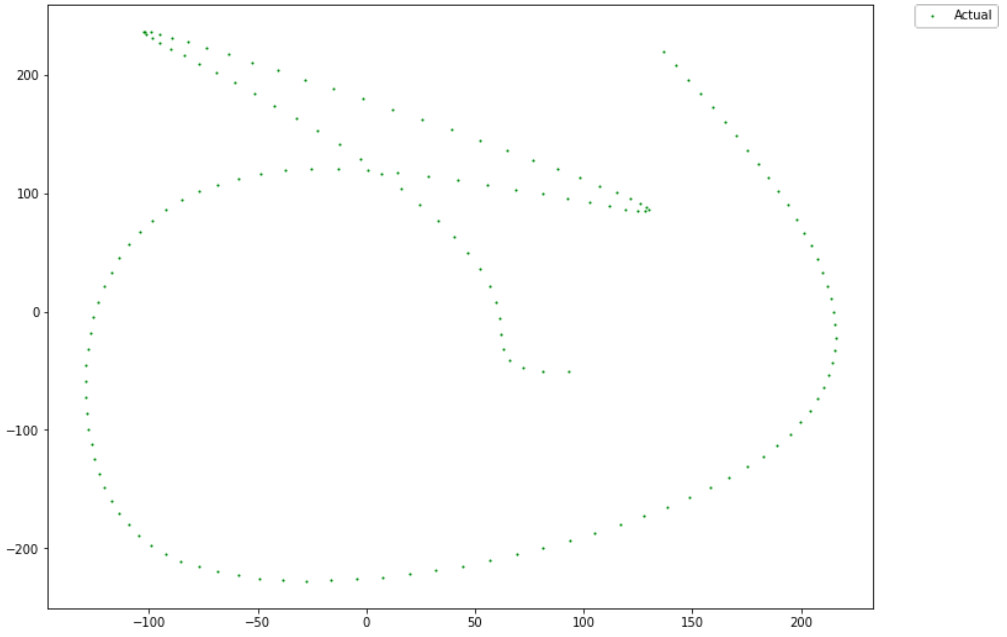
Average RMSE: 20.084332 m

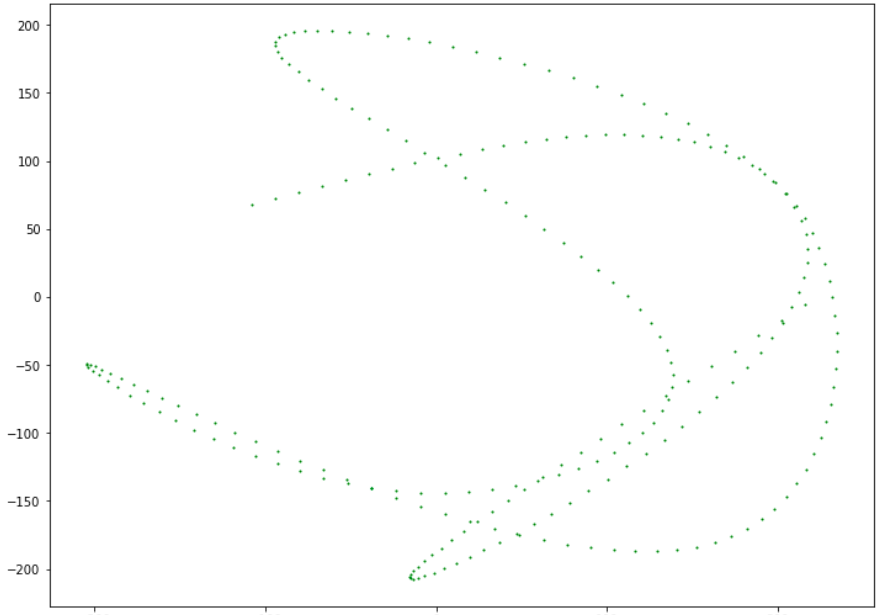

Actual
Predicted
1000
500
0
−500
−1000
−1000
−500
0
500
1000

UMI with speed at 11.11 m/s

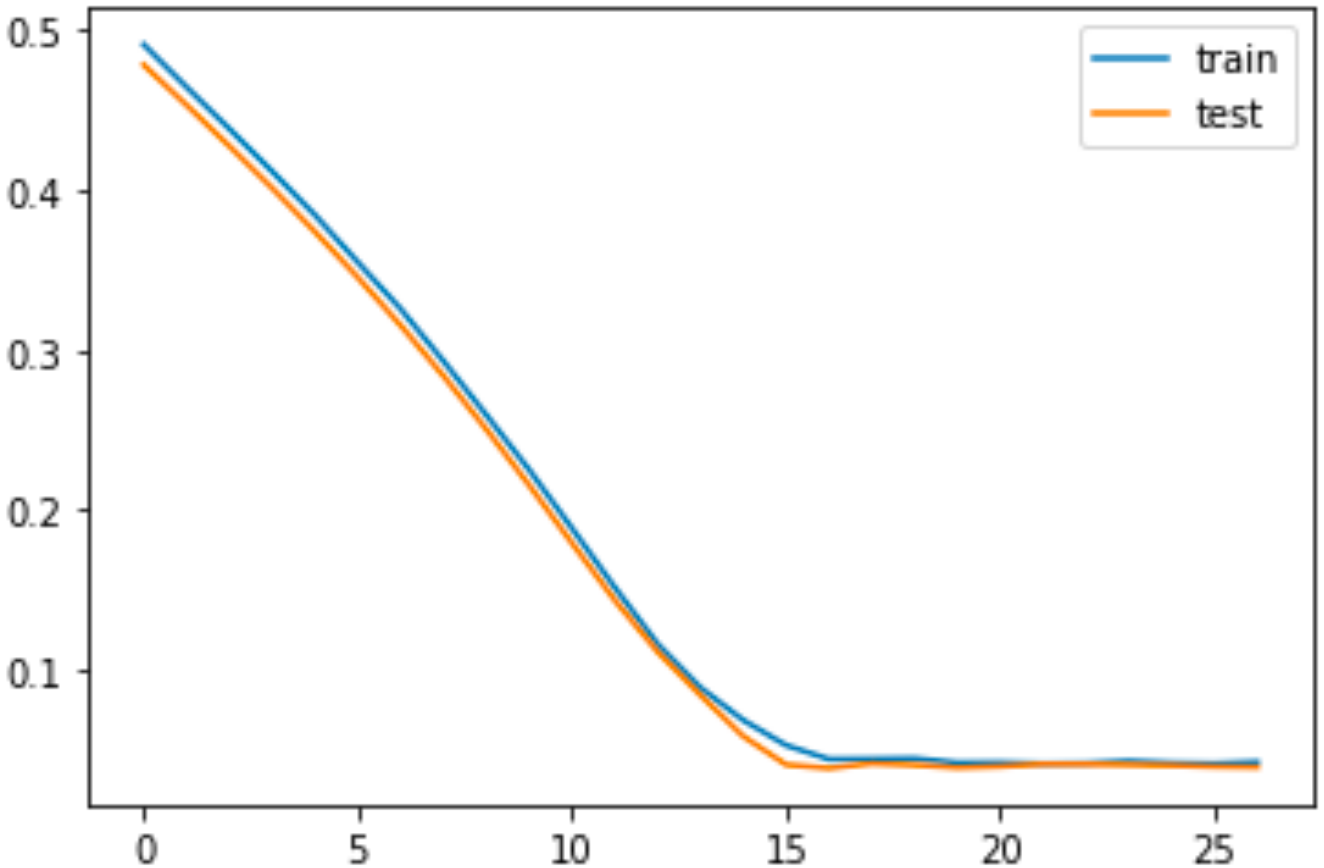
Sample plots from all routes







Epoch 27/600 6/6 - 0s - loss: 0.0428 - accuracy: 0.9806 - val_loss: 0.0396 - val_accuracy: 0.9758

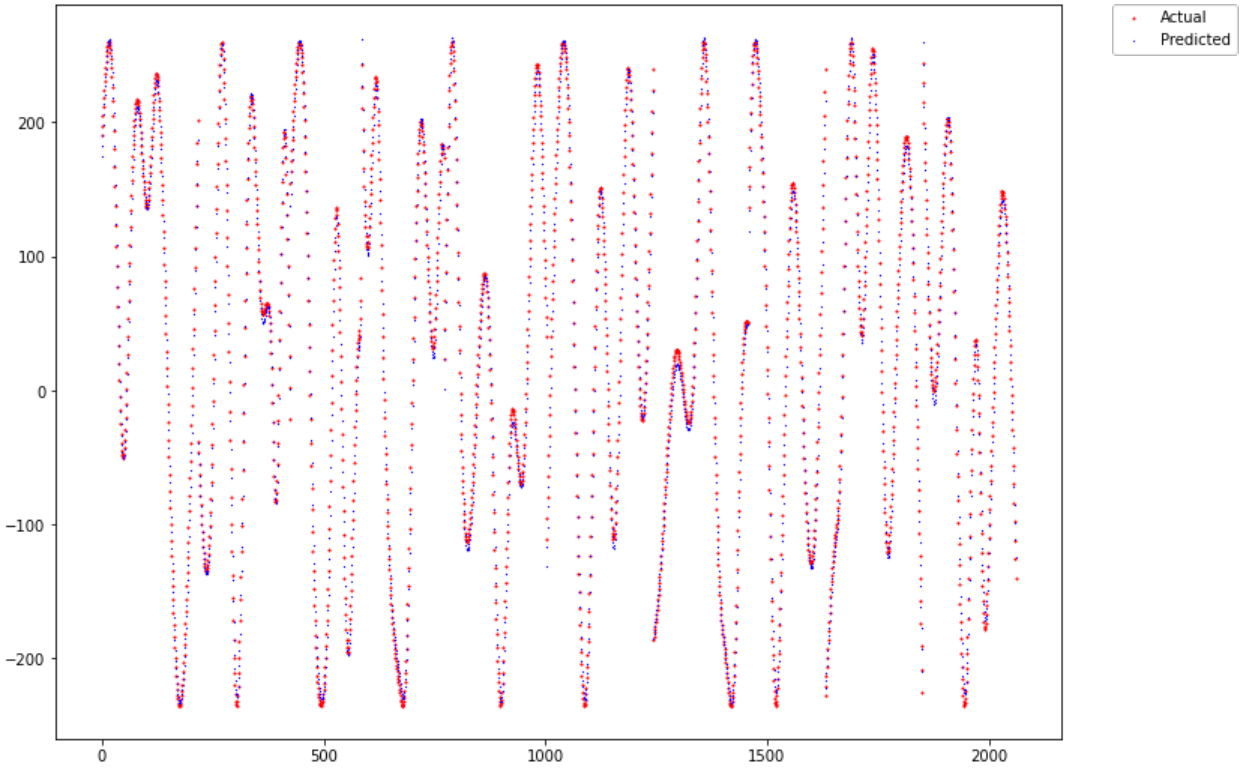


52/52 [==============================] - 0s
2ms/step - loss: 0.0389 - accuracy: 0.9667

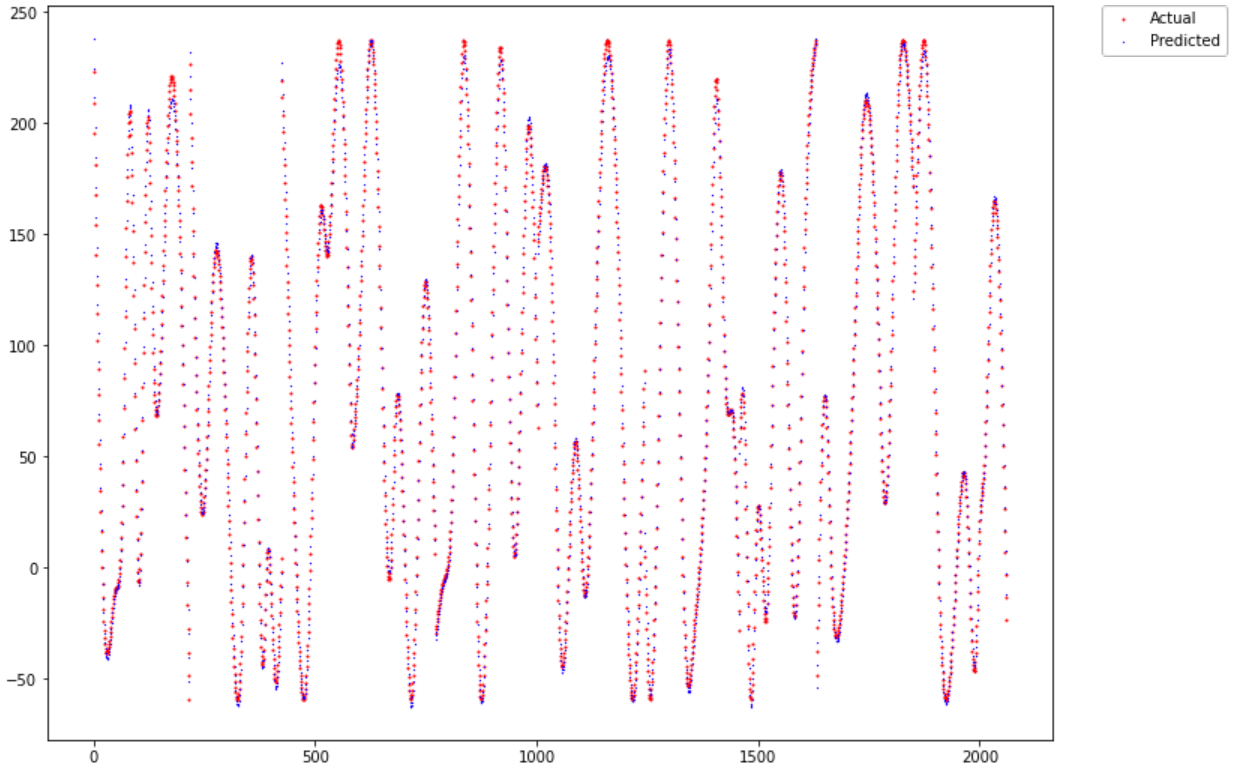
Accuracy: 96.6667
Loss: 0.038939

Average RMSE for Test RMSE: 9.610334 m
Average RMSE for Train RMSE: 9.512330 m

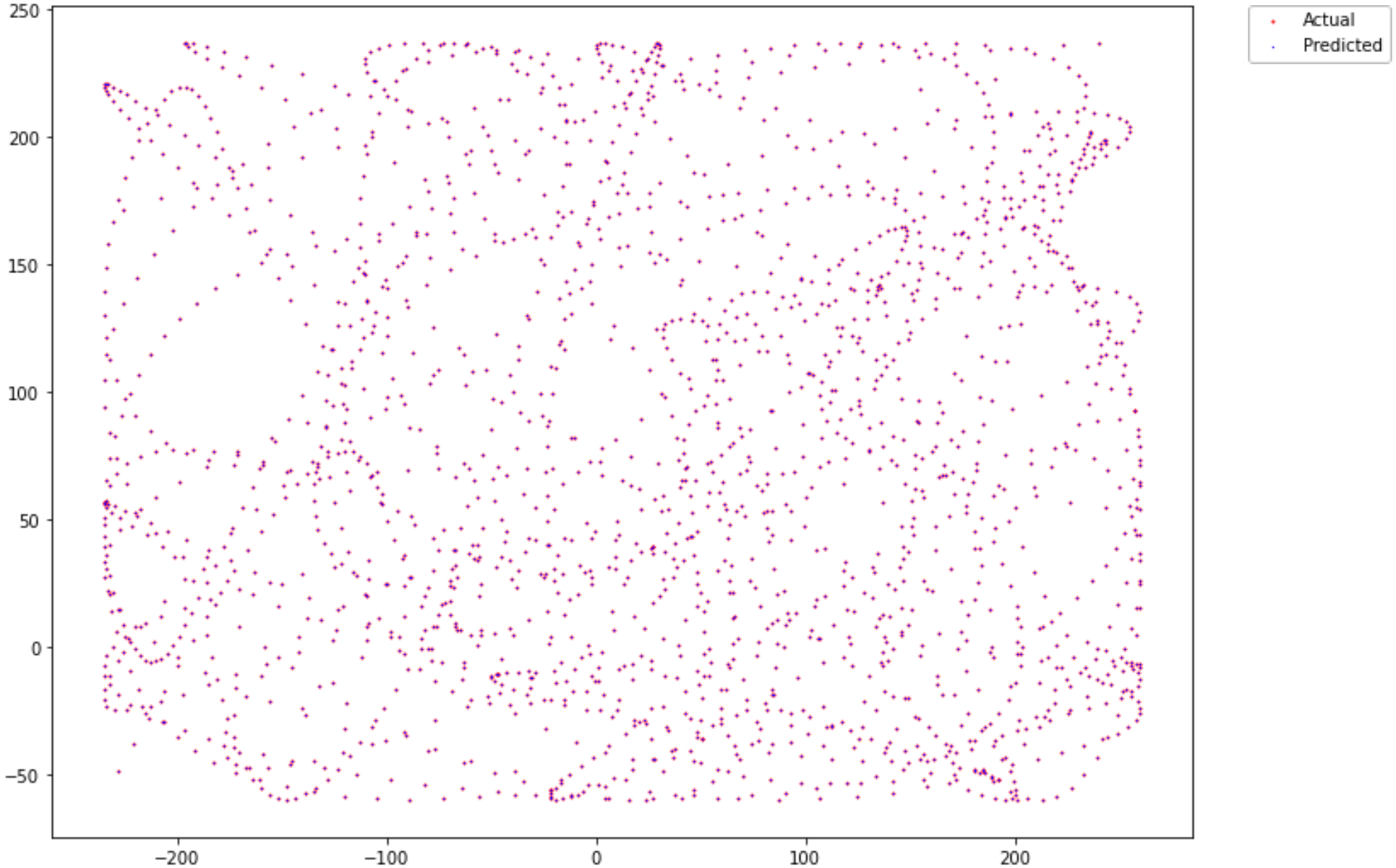
Plot of X positions



Plot of Y positions



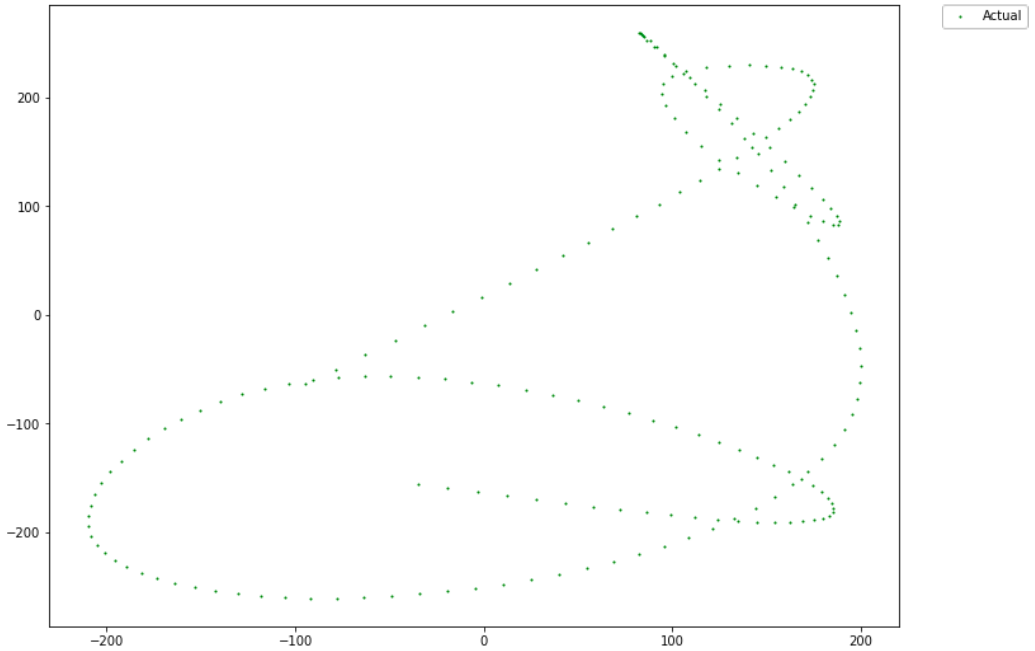
Plot of X and Y positions

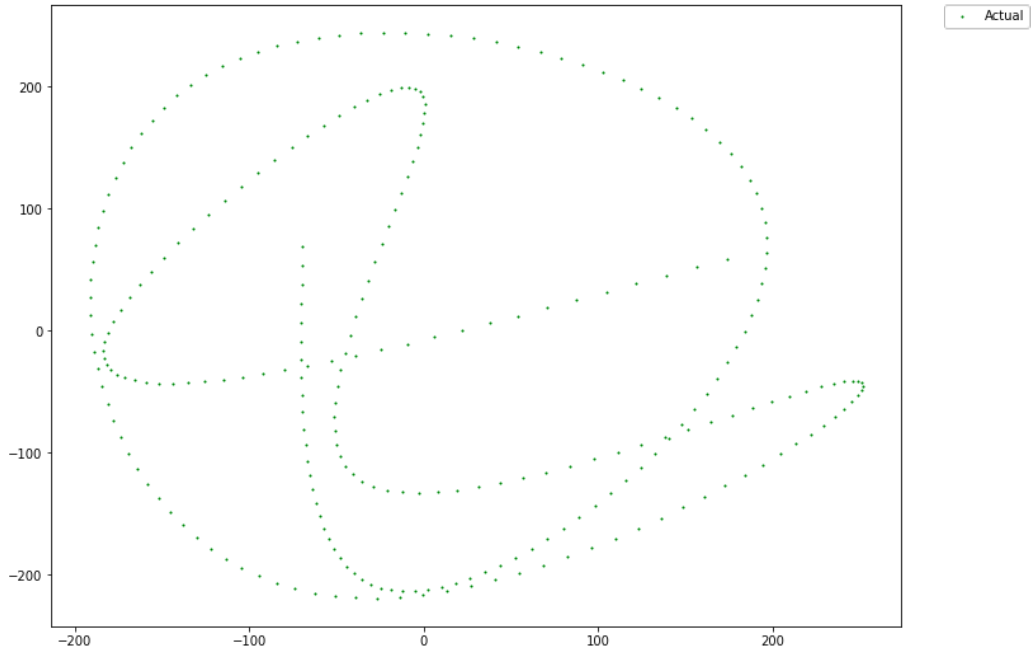


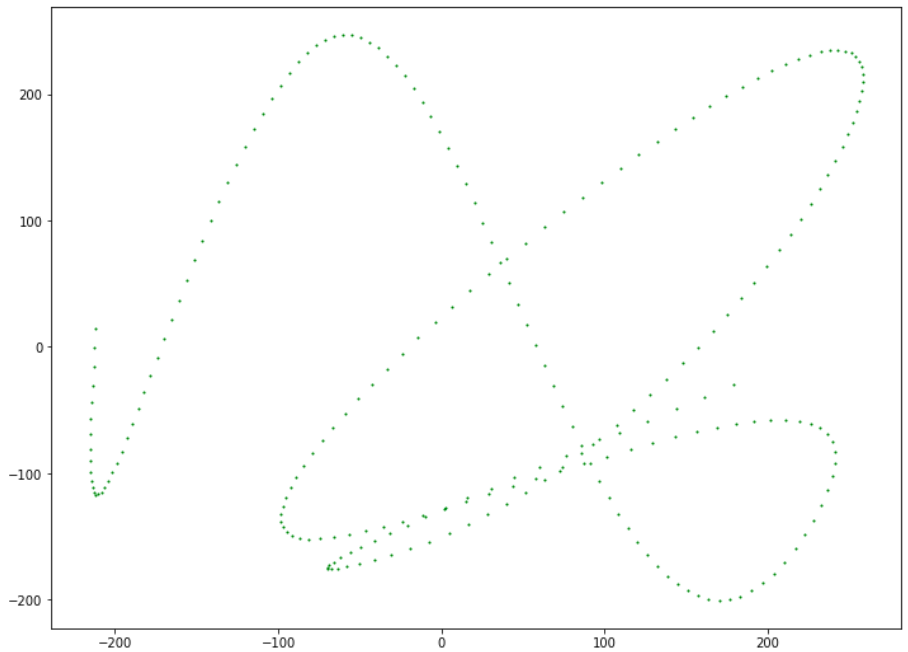
Try feeding 6 different paths and Let ML predicts

```
traj11 = pd.read_csv('UE5_18-11-2021_23-40-28.csv', low_memory=False,  header=None)
traj12 = pd.read_csv('UE5_18-11-2021_23-40-49.csv', low_memory=False,  header=None)
traj13 = pd.read_csv('UE6_18-11-2021_23-40-28.csv', low_memory=False,  header=None)
traj14 = pd.read_csv('UE7_18-11-2021_23-40-28.csv', low_memory=False,  header=None)
traj15 = pd.read_csv('UE8_18-11-2021_23-40-28.csv', low_memory=False,  header=None)
traj16 = pd.read_csv('UE9_18-11-2021_23-40-28.csv', low_memory=False,  header=None)

traj11.columns = ["X", "Y", "Speed", "Time"]
traj12.columns = ["X", "Y", "Speed", "Time"]
traj13.columns = ["X", "Y", "Speed", "Time"]
traj14.columns = ["X", "Y", "Speed", "Time"]
traj15.columns = ["X", "Y", "Speed", "Time"]
traj16.columns = ["X", "Y", "Speed", "Time"]
```

Sample plots



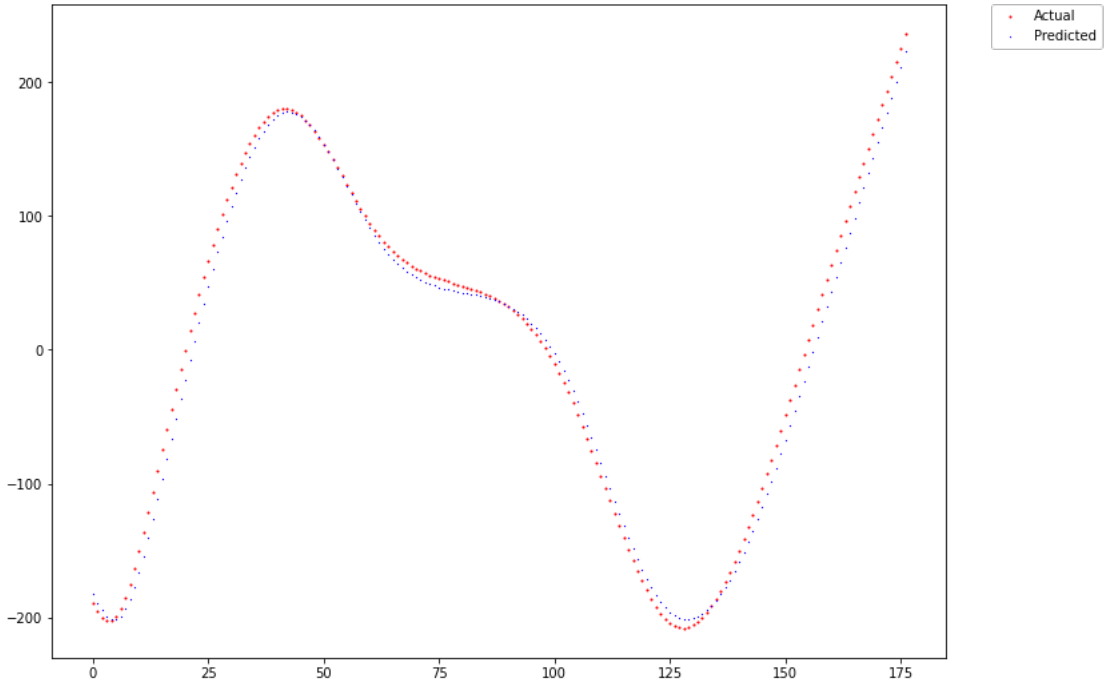




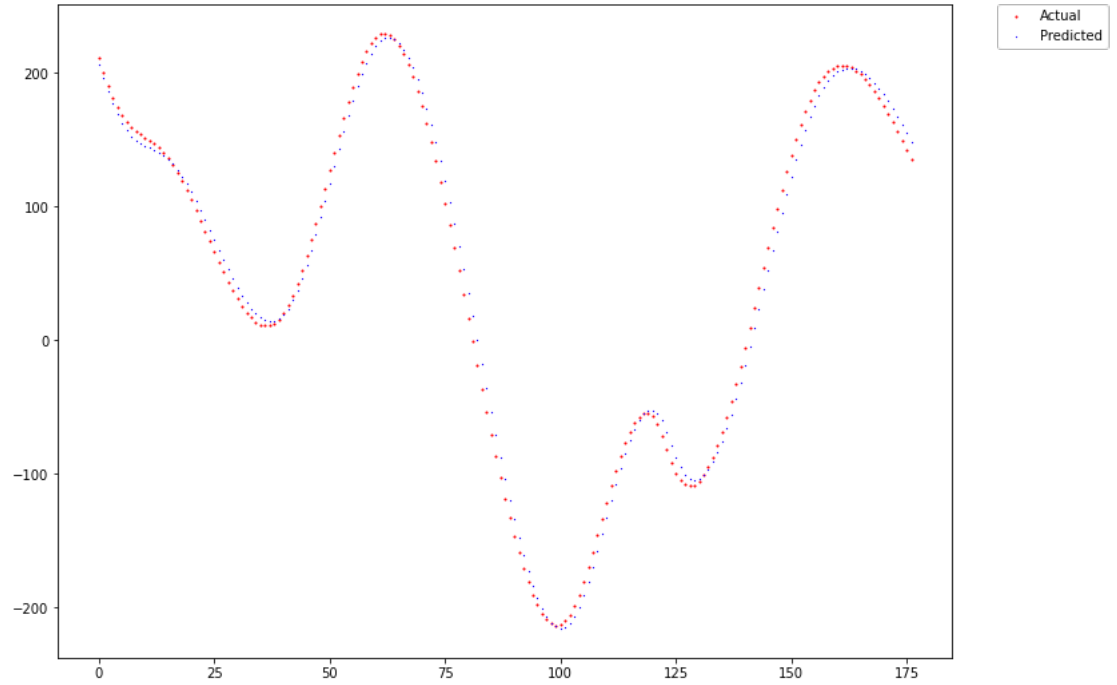
Prediction results

Actual vs Predicted plots for trajectory 11

Plot of X positions

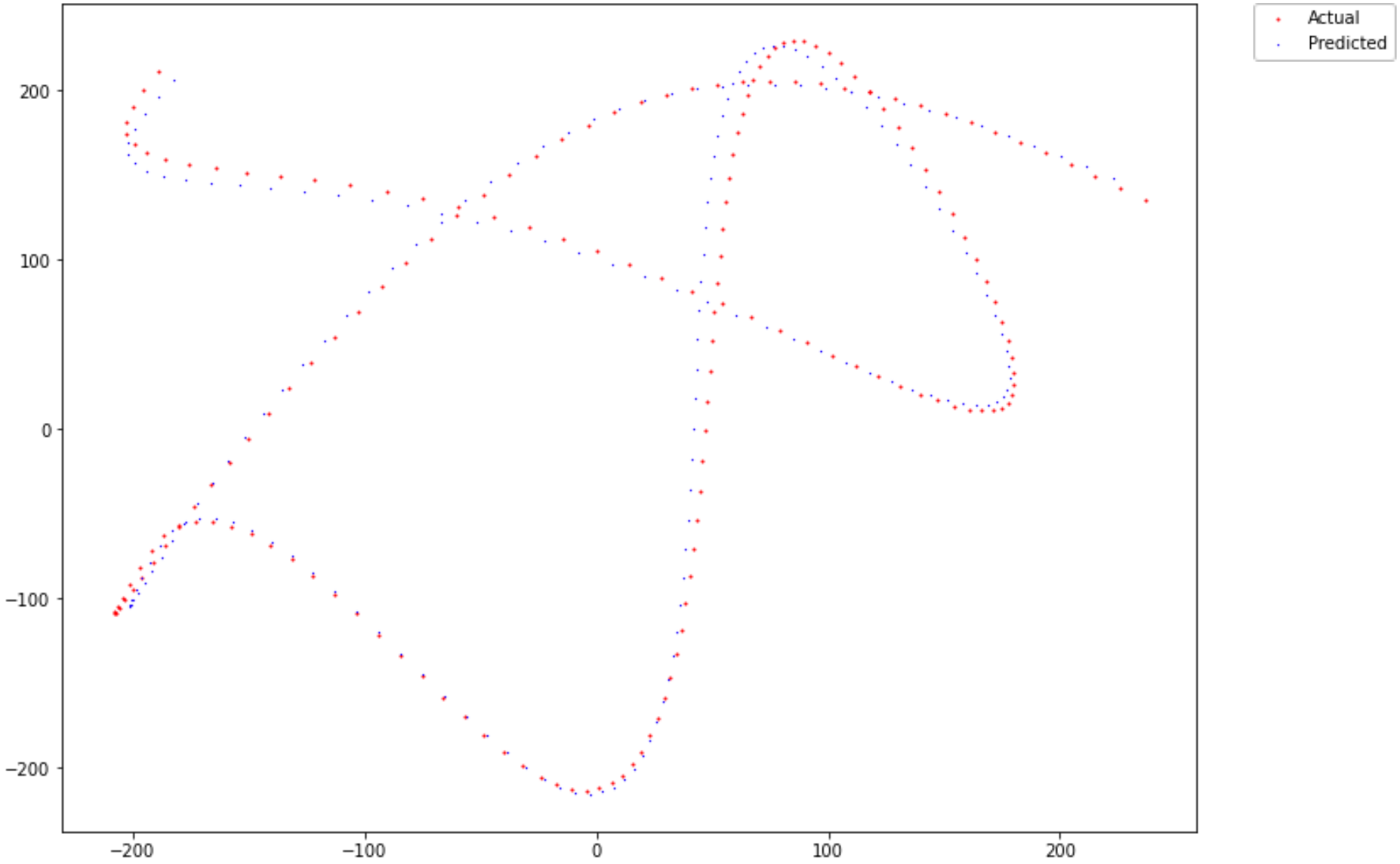


Plot of Y positions



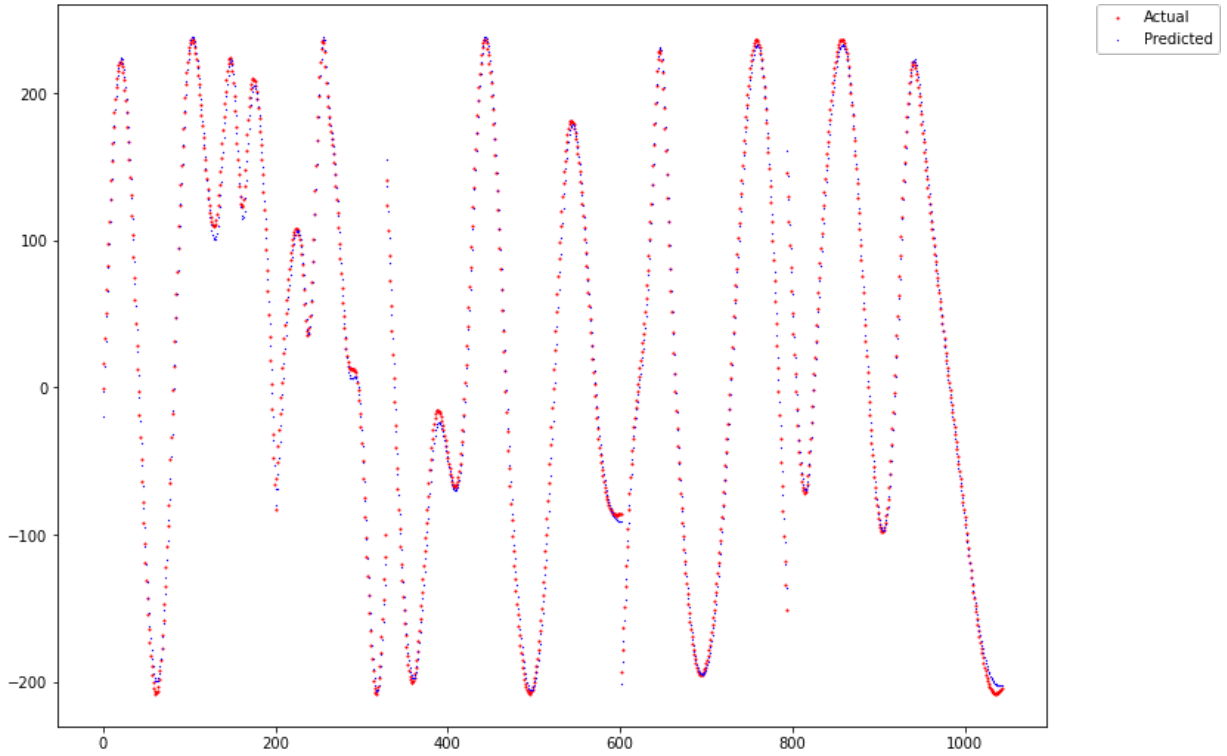
Average RMSE: 10.743610 m

Actual vs Predicted plot for trajectory 11

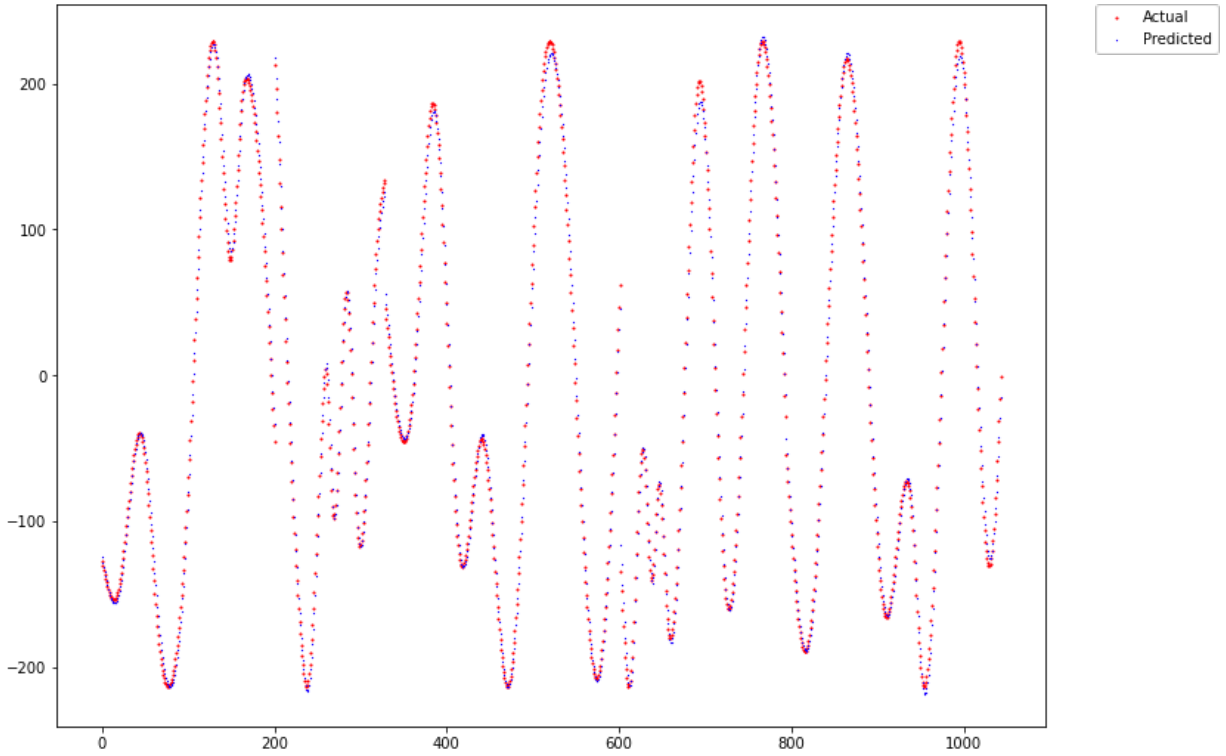


Actual vs Predicted plots for trajectory 12-16

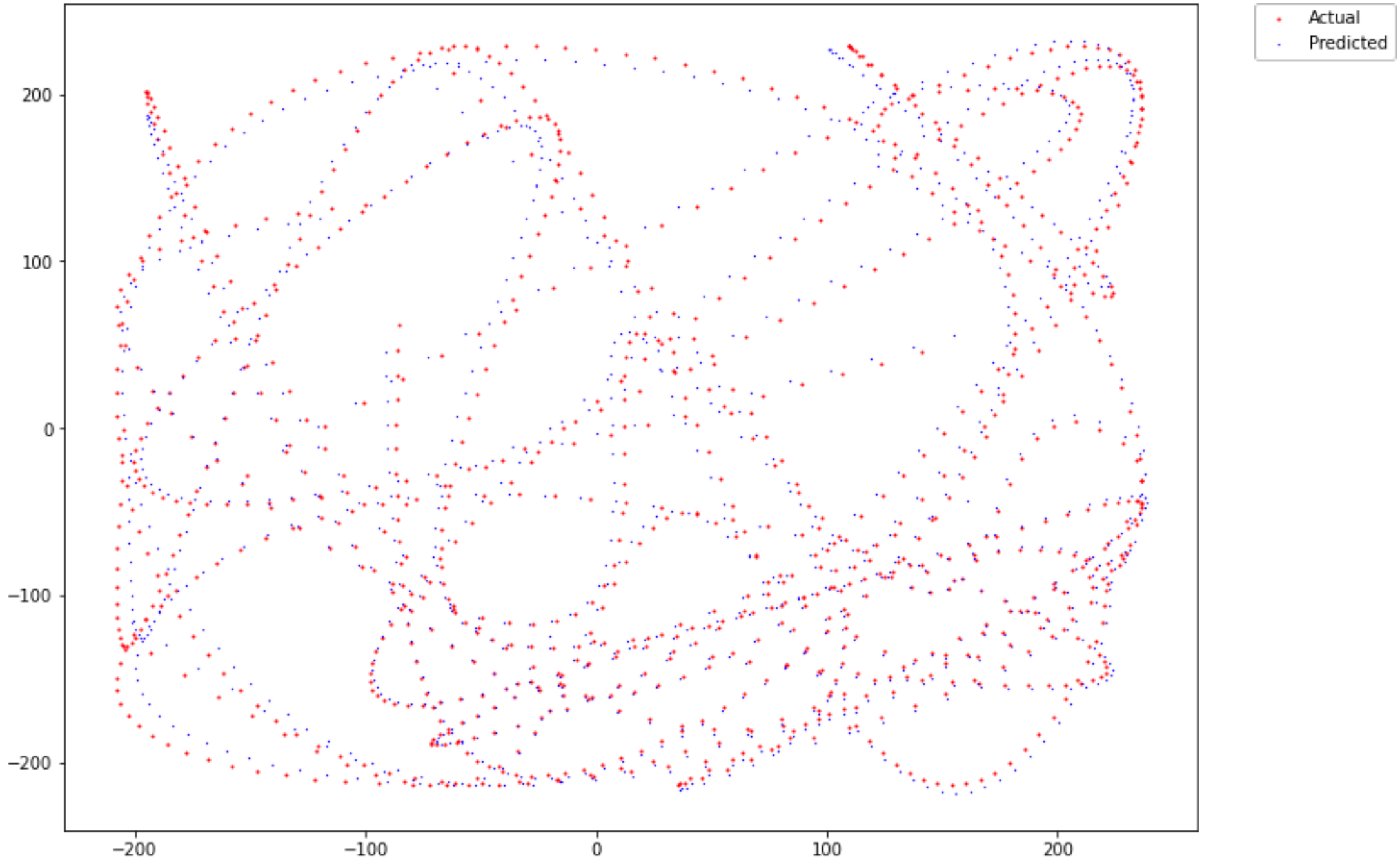
Plot of X positions



Plot of Y positions

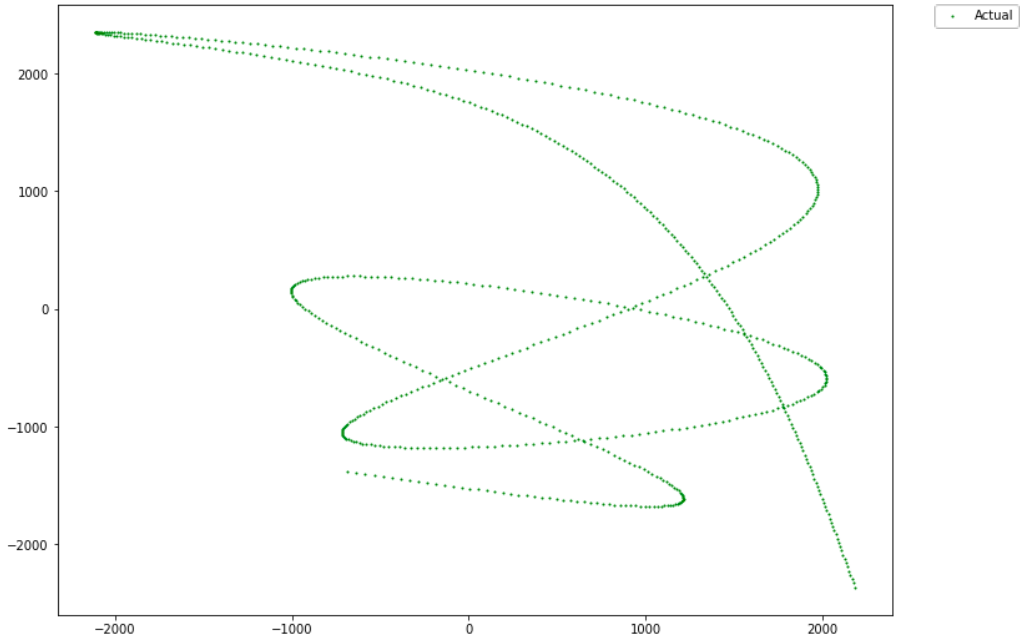


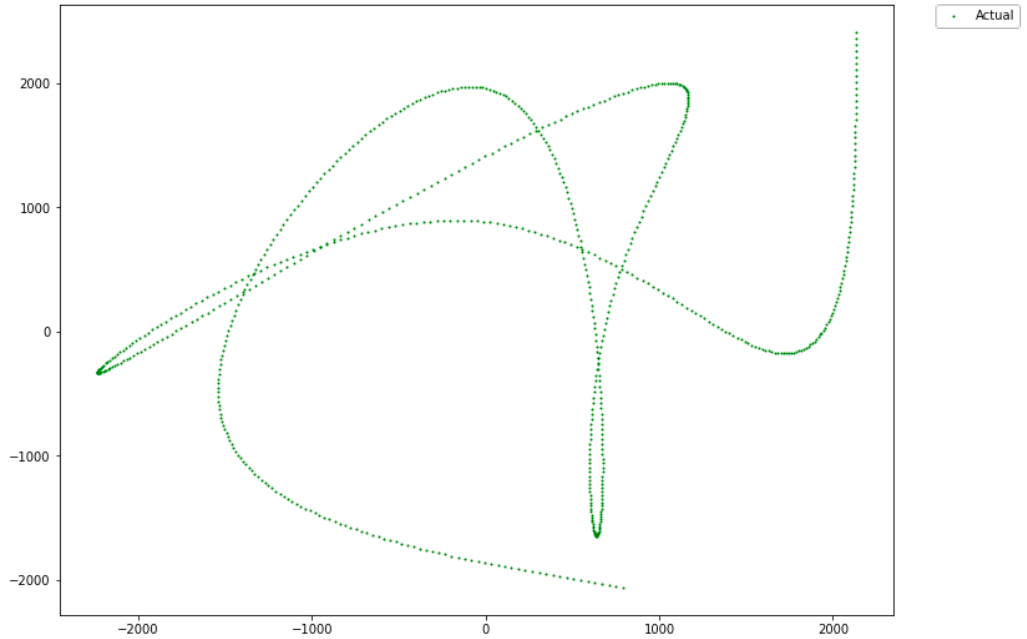
Average RMSE: 9.769335 m

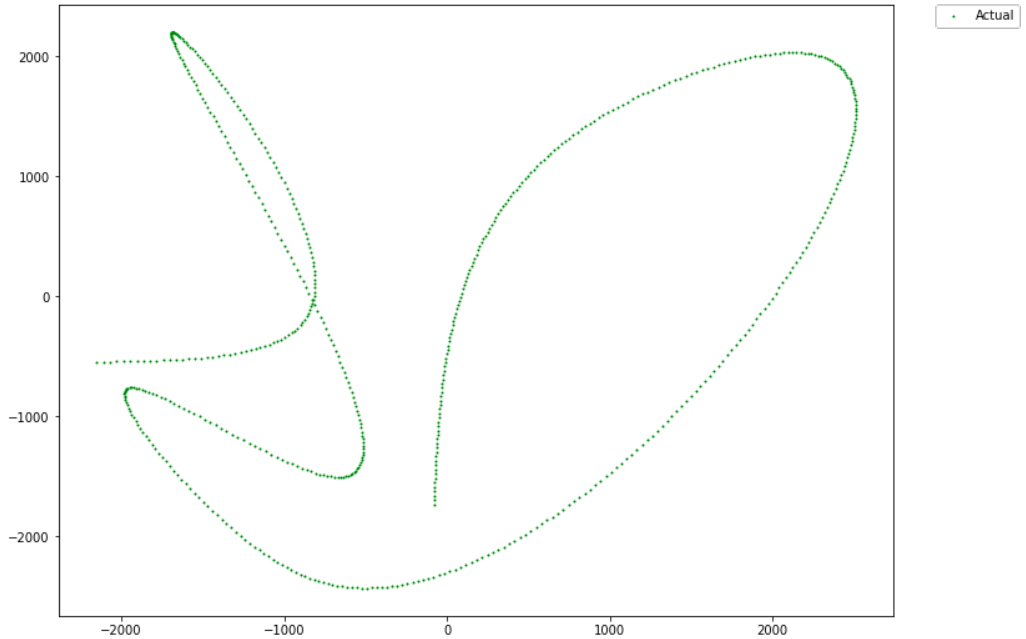

Actual
Predicted
200
100
0
−100
−200
−200
−100
0
100
200

RMA with speed at 33.33 m/s

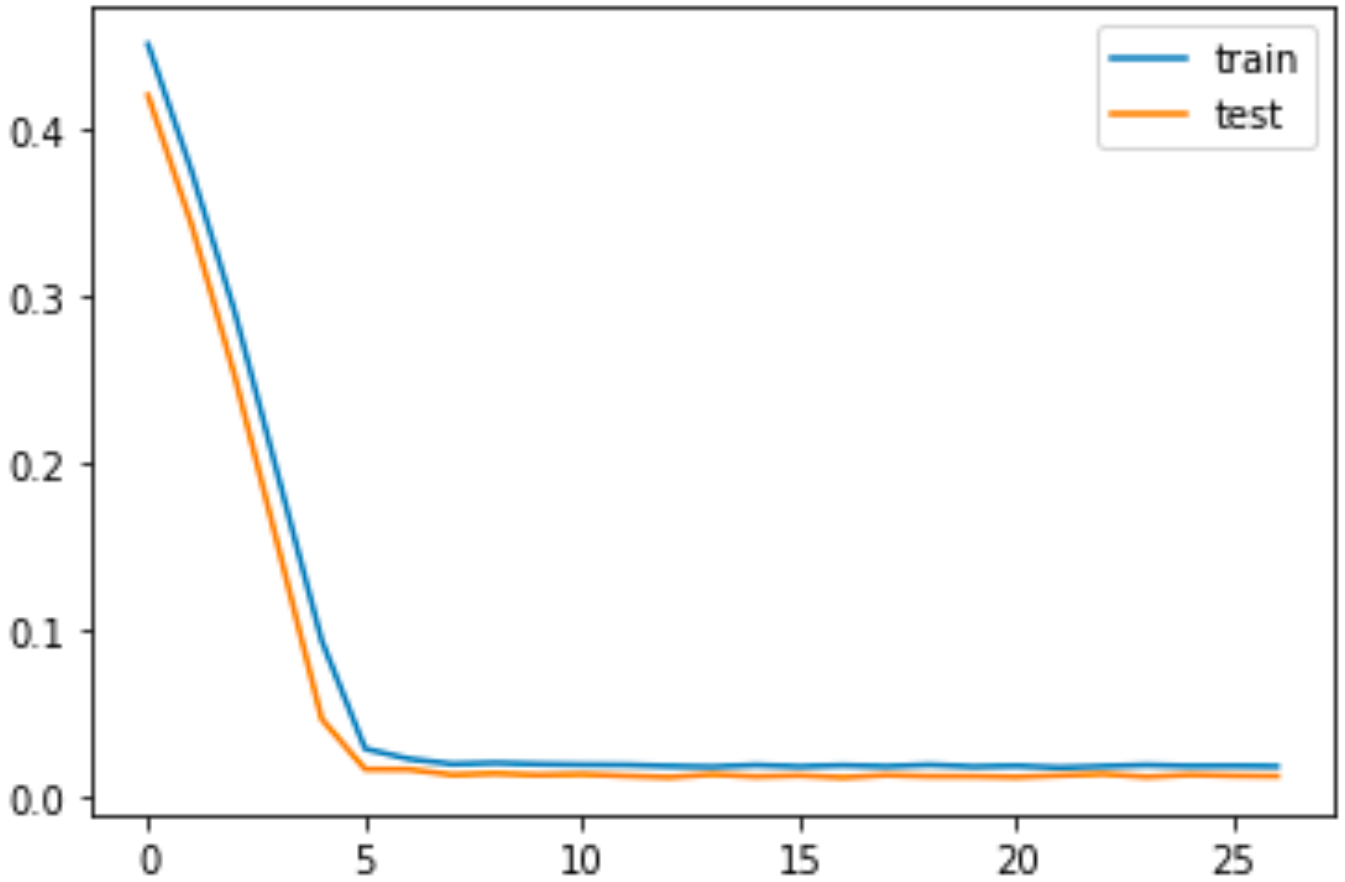
Sample plots from all routes







Epoch 27/600 20/20 - 0s - loss: 0.0181 - accuracy: 0.9950 - val_loss: 0.0123 - val_accuracy: 0.9931

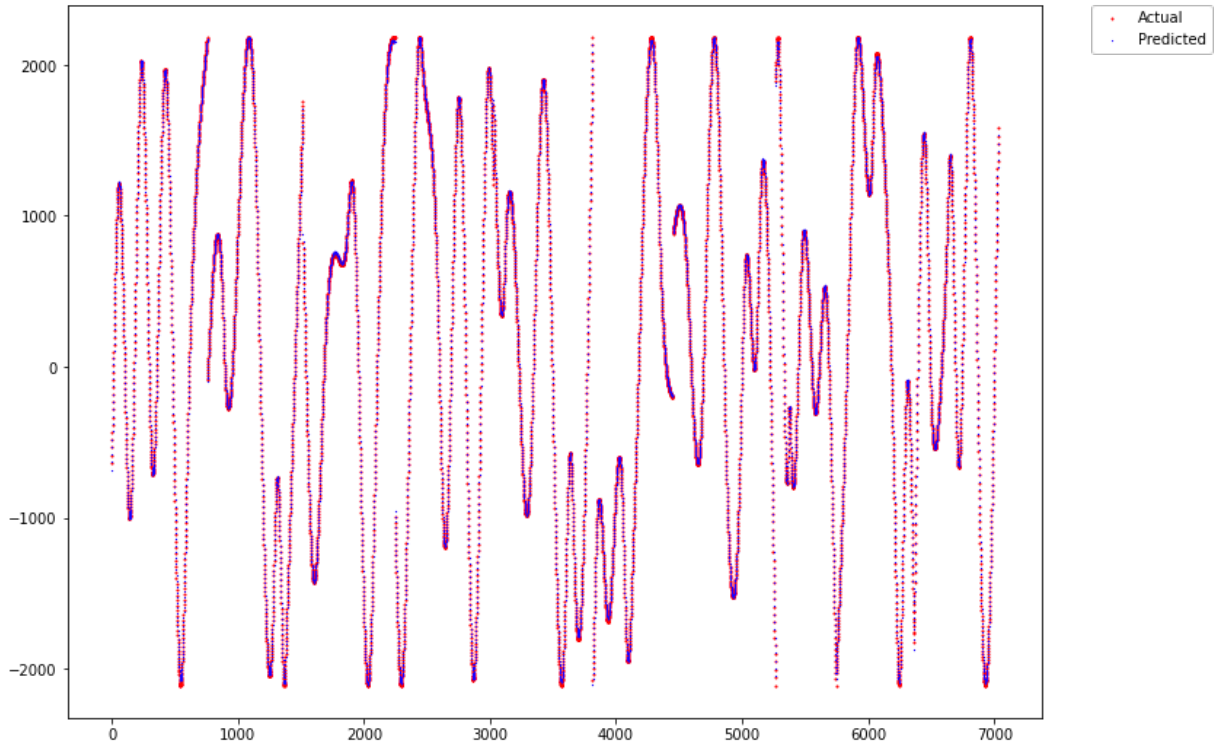


176/176 [==============================] - 0s 1ms/step - loss: 0.0114 - accuracy: 0.9927

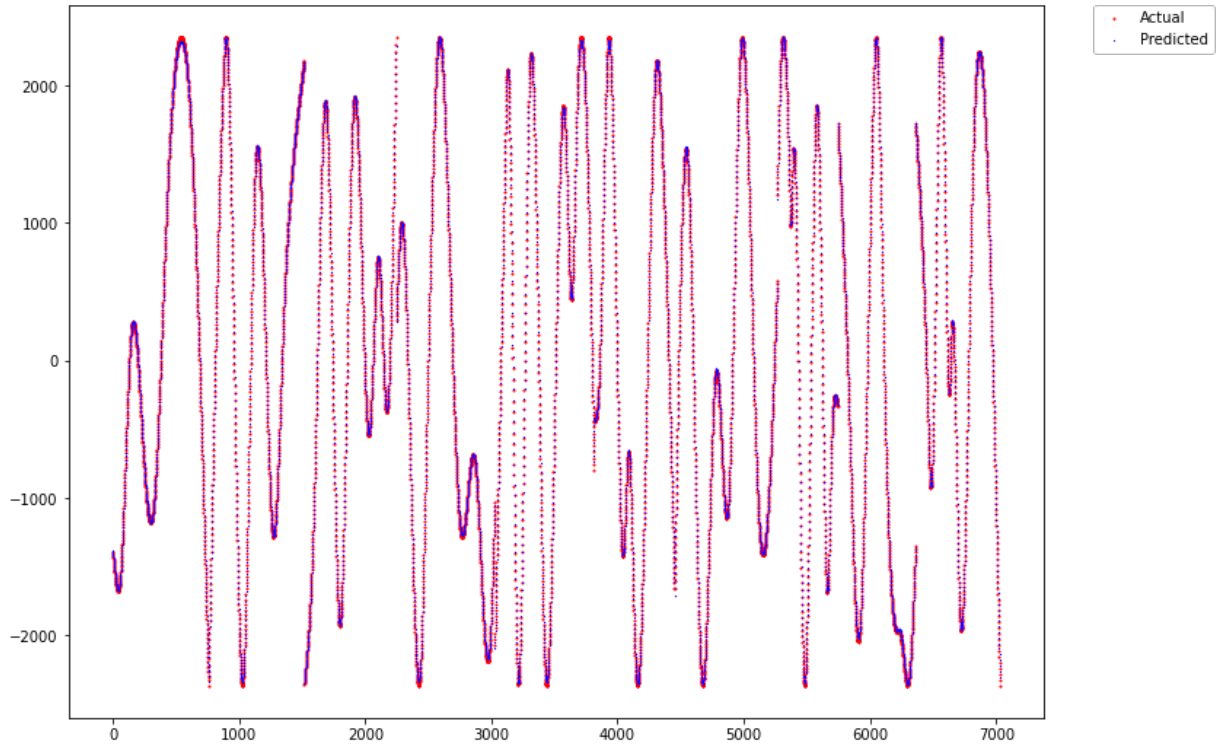
Accuracy: 99.2711
Loss: 0.011410

Average RMSE for Test RMSE: 30.410962 m
Average RMSE for Train RMSE: 27.584458 m

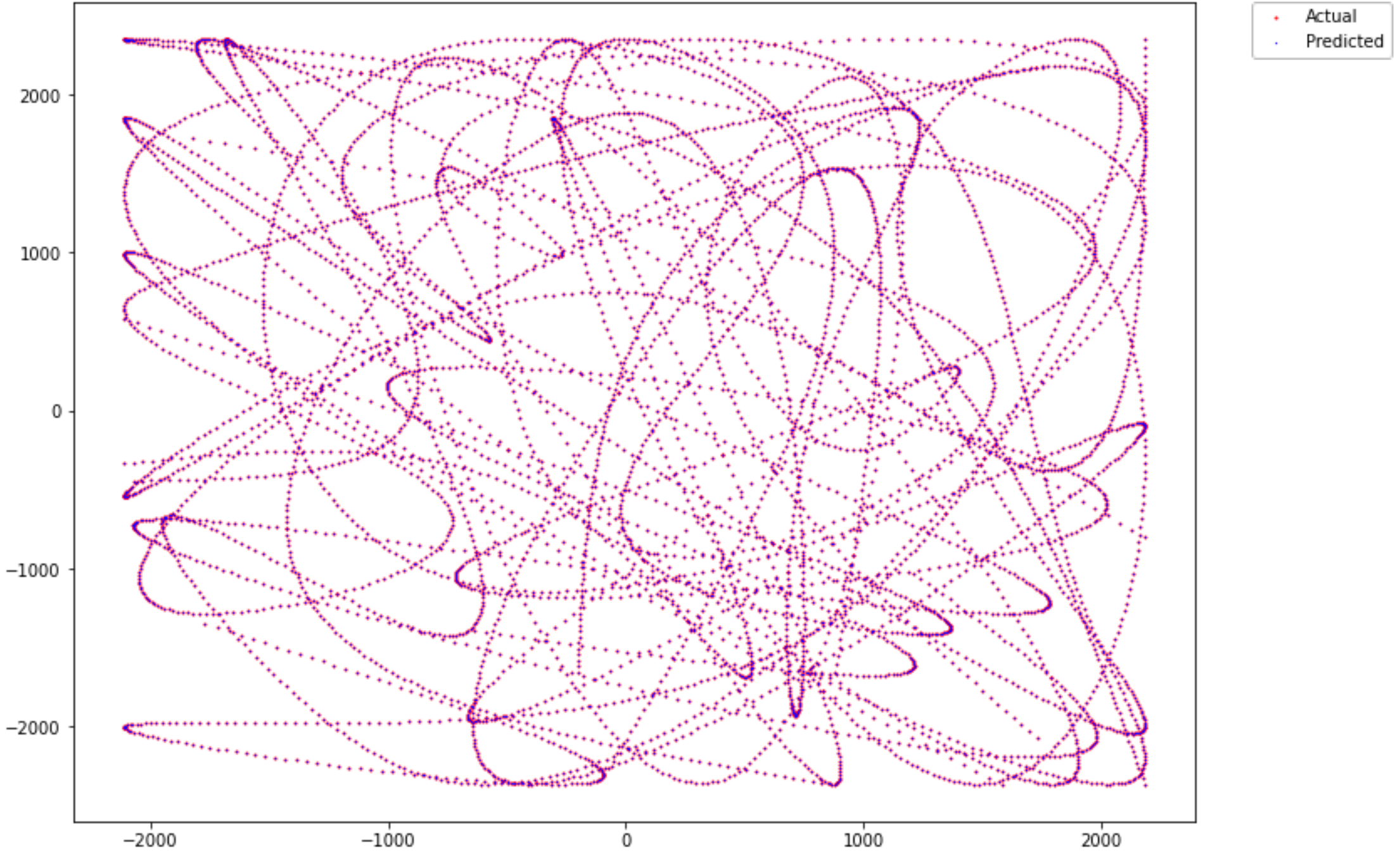
Plot of X positions



Plot of Y positions



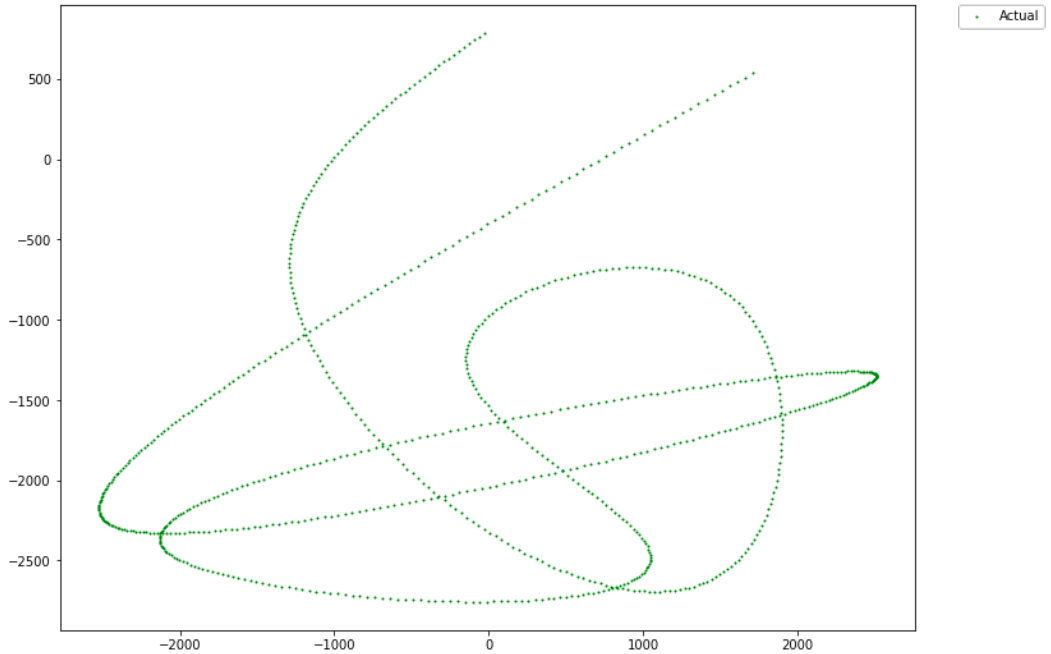
Plot of X and Y positions

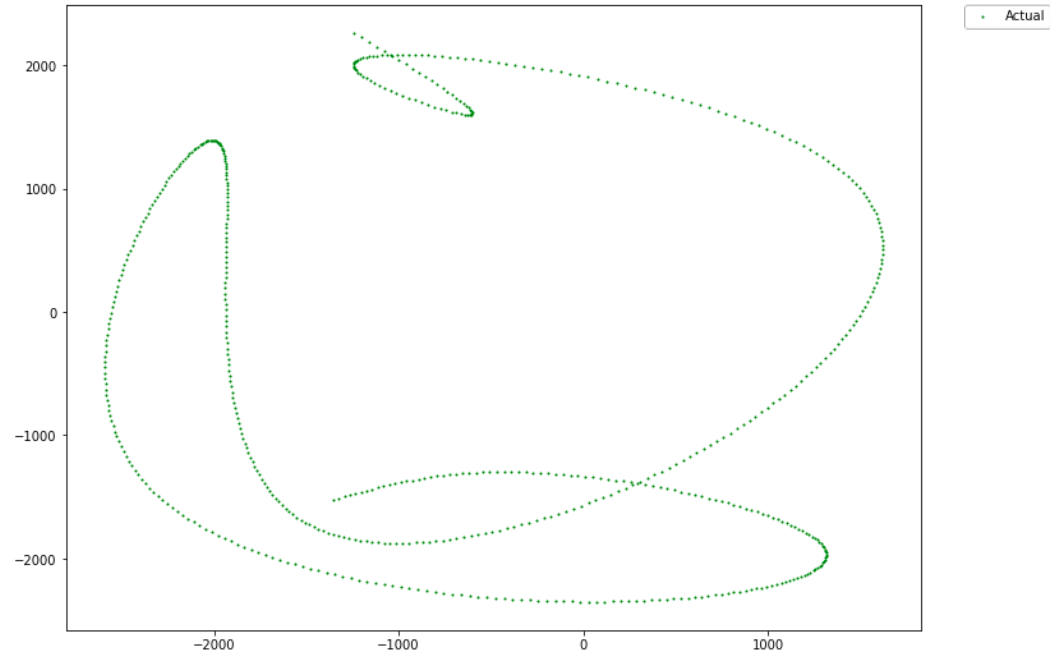


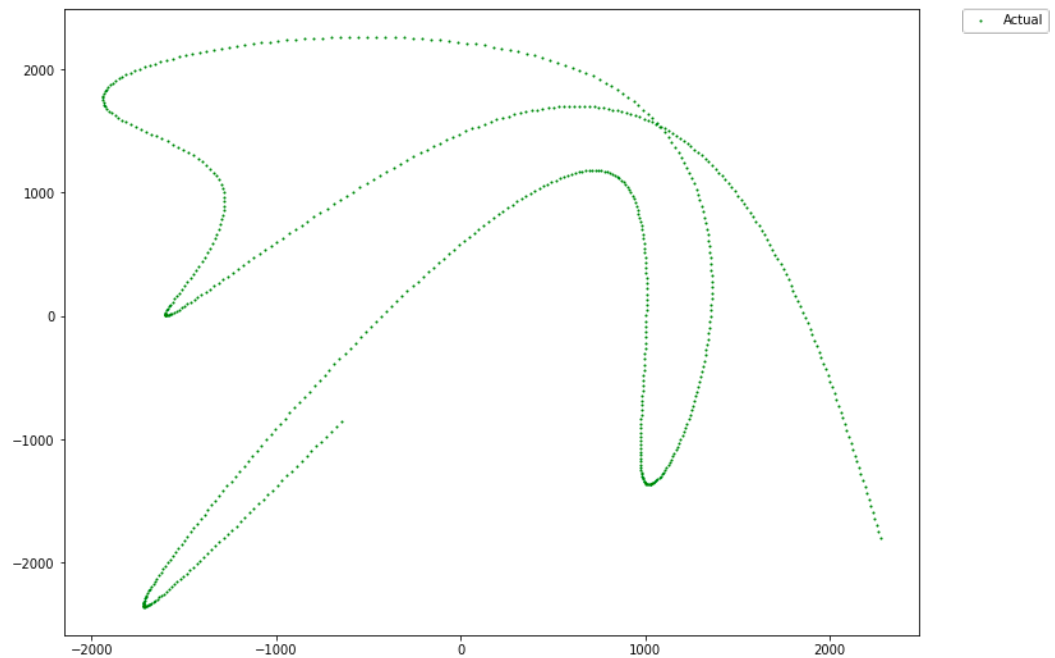
Try feeding 6 different paths and Let ML predicts

```
traj11 = pd.read_csv('UE5_18-11-2021_23-53-03.csv', low_memory=False,  header=None)
traj12 = pd.read_csv('UE5_18-11-2021_23-53-21.csv', low_memory=False,  header=None)
traj13 = pd.read_csv('UE6_18-11-2021_23-53-03.csv', low_memory=False,  header=None)
traj14 = pd.read_csv('UE7_18-11-2021_23-53-03.csv', low_memory=False,  header=None)
traj15 = pd.read_csv('UE8_18-11-2021_23-53-03.csv', low_memory=False,  header=None)
traj16 = pd.read_csv('UE9_18-11-2021_23-53-04.csv', low_memory=False,  header=None)

traj11.columns = ["X", "Y", "Speed", "Time"]
traj12.columns = ["X", "Y", "Speed", "Time"]
traj13.columns = ["X", "Y", "Speed", "Time"]
traj14.columns = ["X", "Y", "Speed", "Time"]
traj15.columns = ["X", "Y", "Speed", "Time"]
traj16.columns = ["X", "Y", "Speed", "Time"]
```

Sample plots



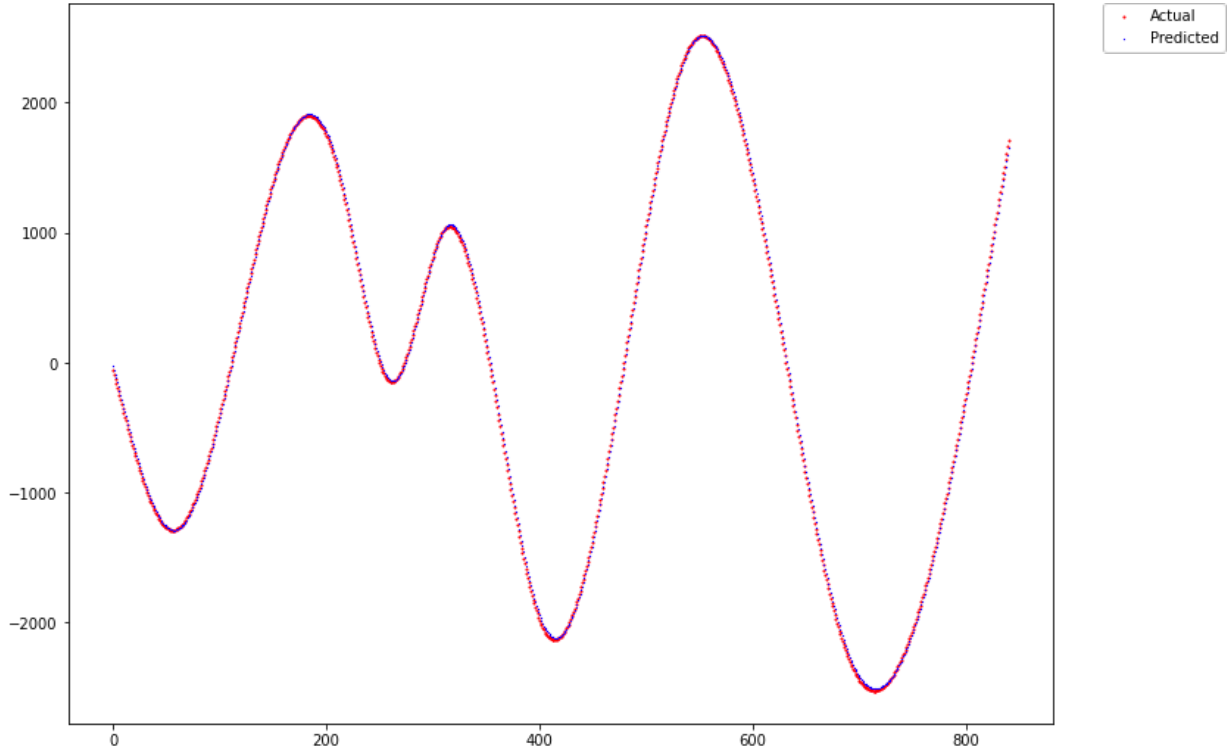




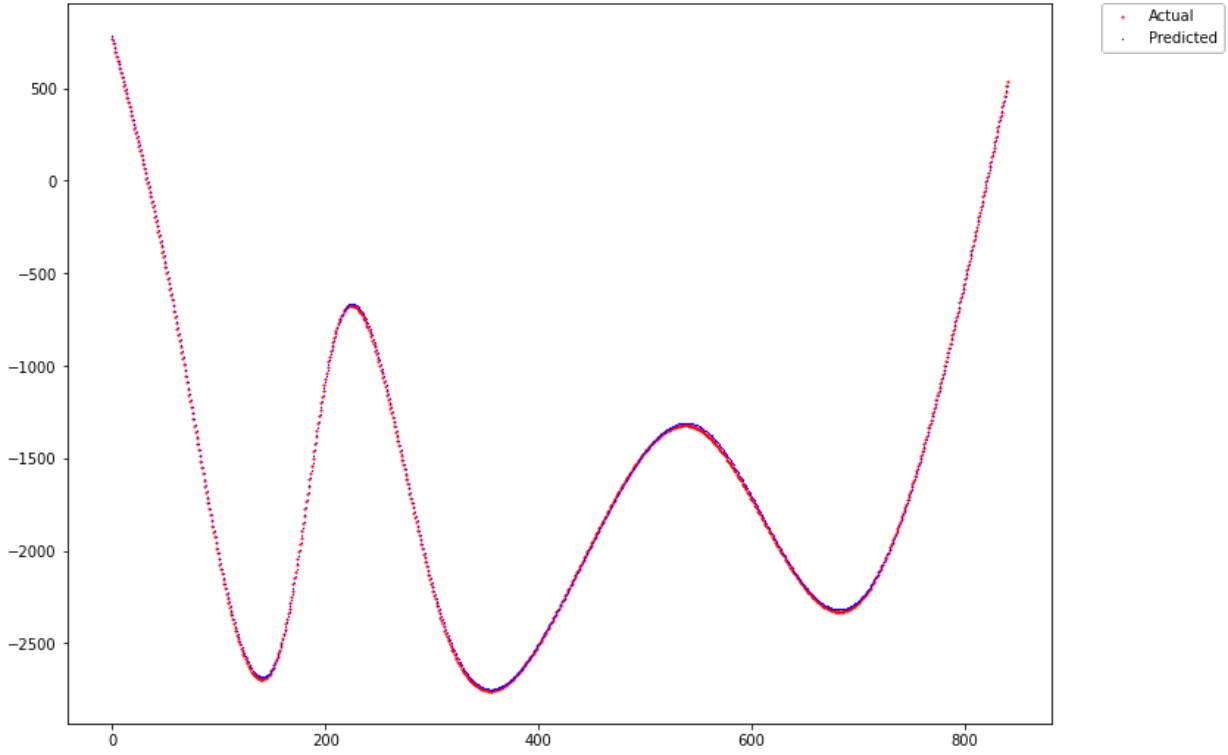
Prediction results

Actual vs Predicted plots for trajectory 11

Plot of X positions

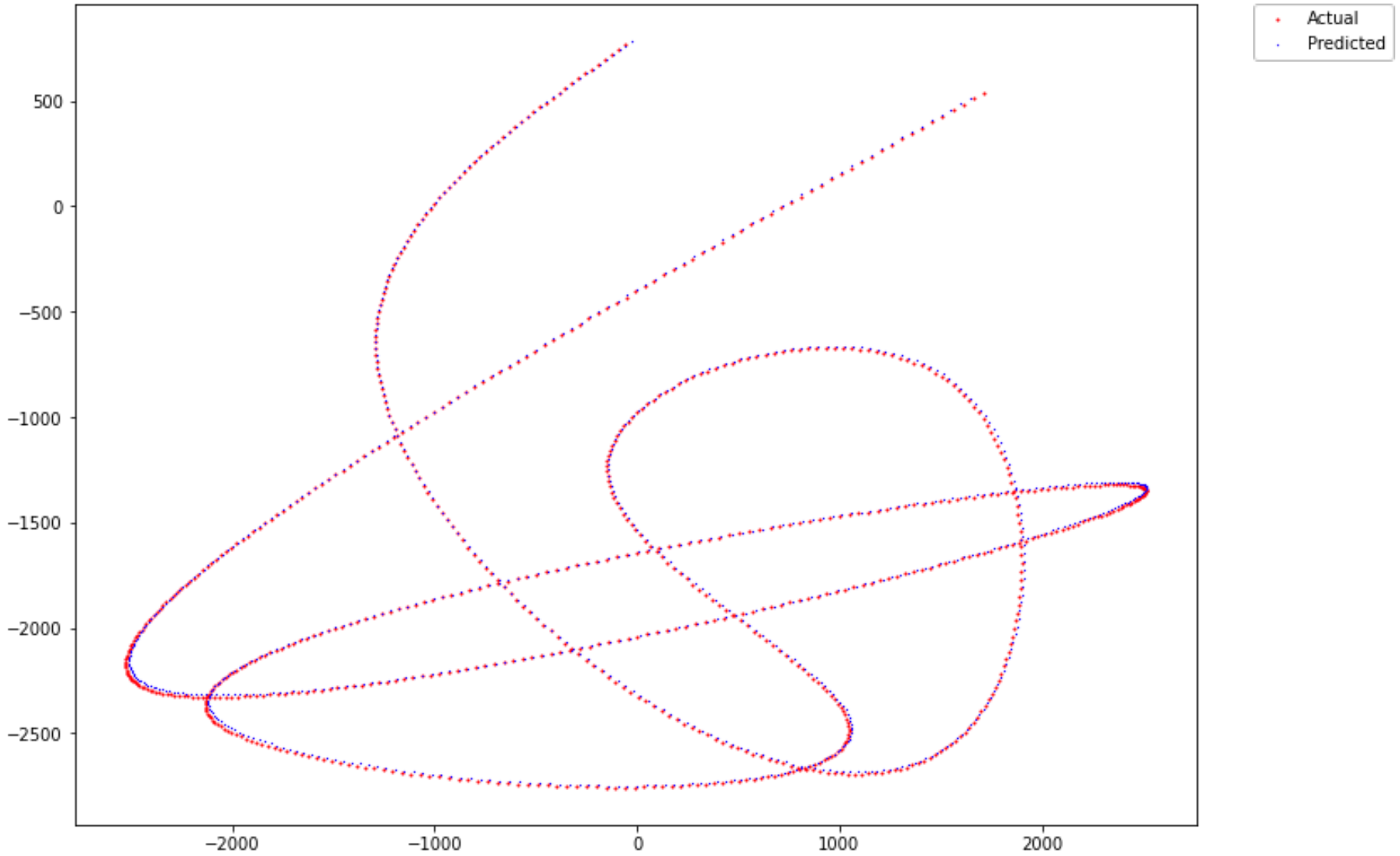


Plot of Y positions



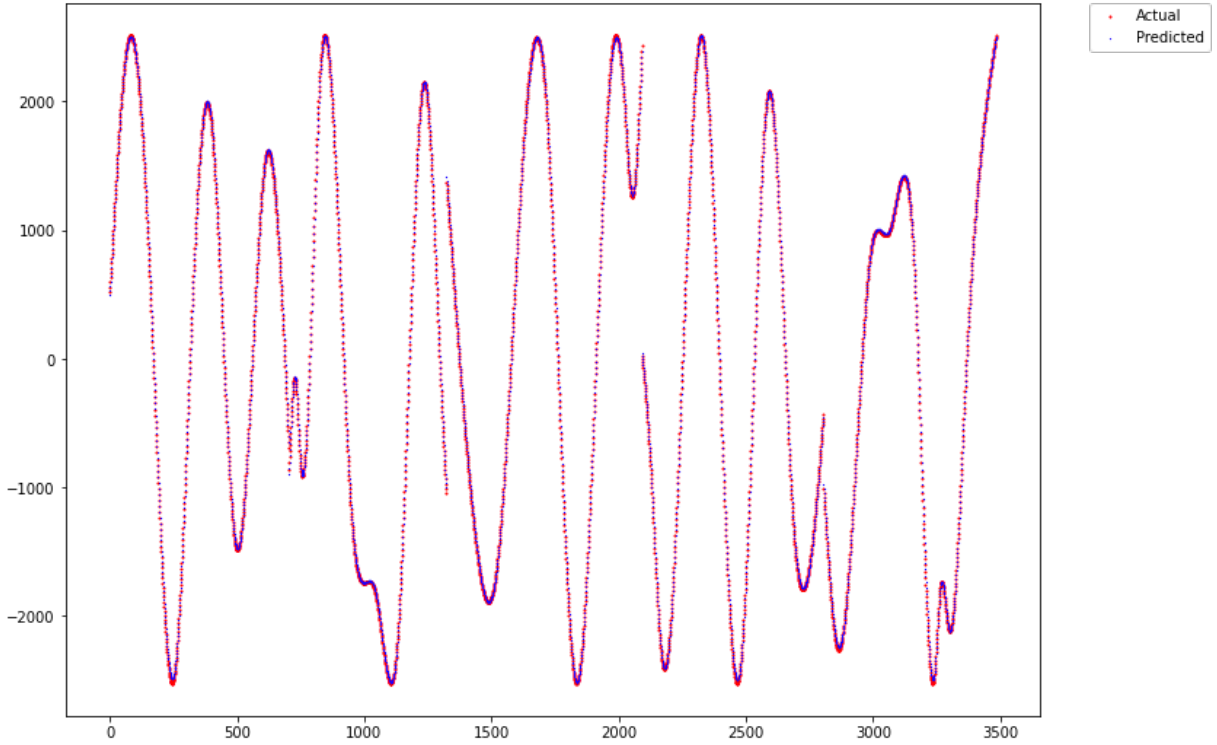
Average RMSE: 28.093843 m

Actual vs Predicted plot for trajectory 11

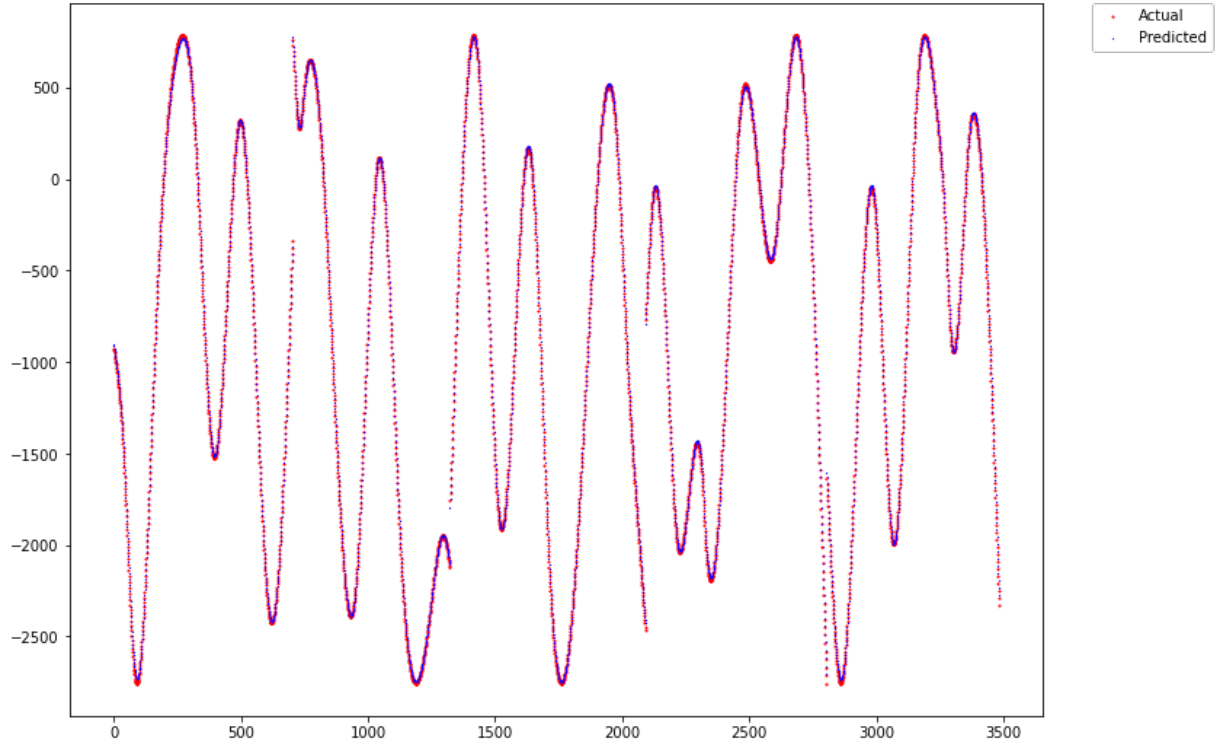


Actual vs Predicted plots for trajectory 12-16

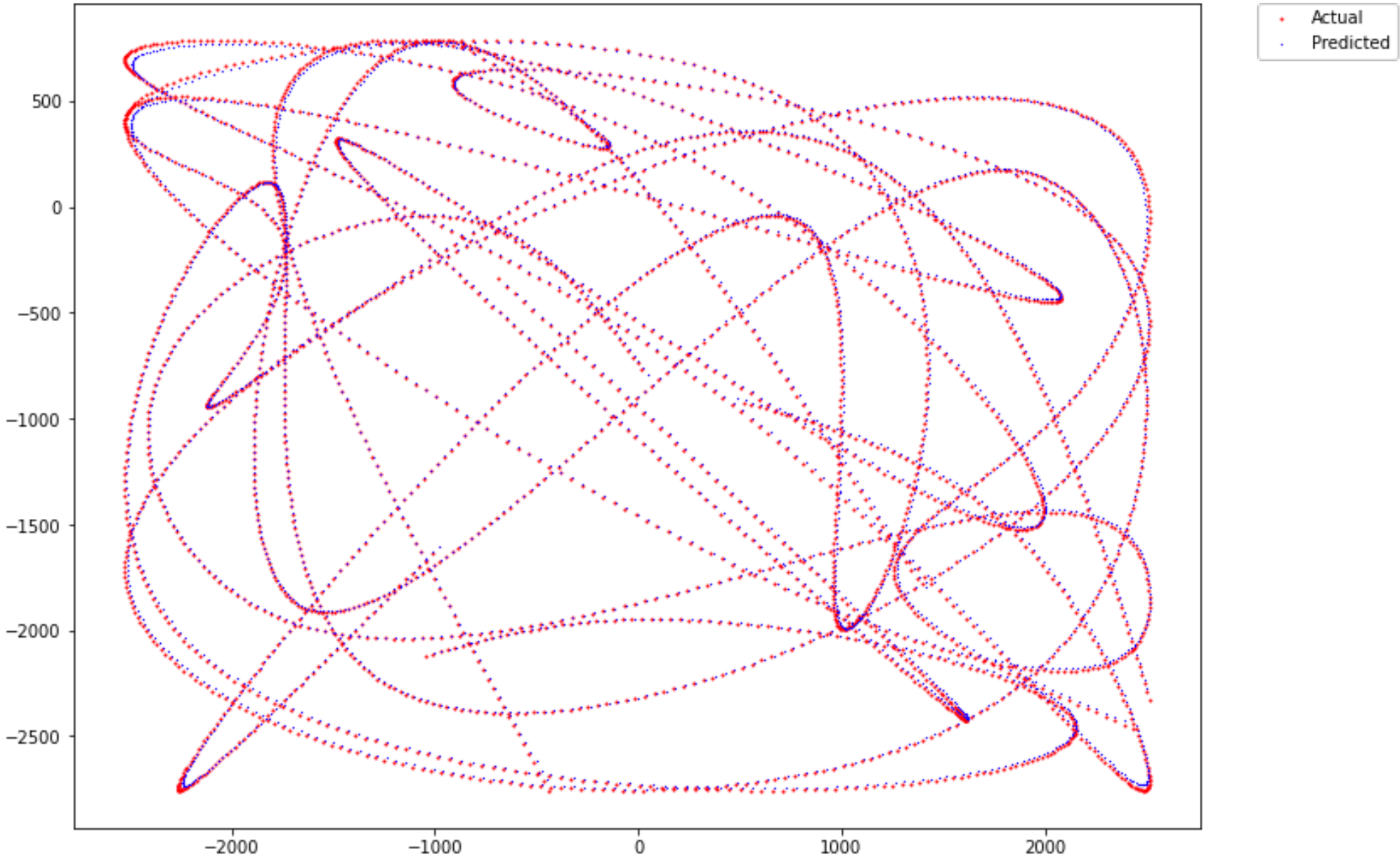
Plot of X positions



Plot of Y positions



Average RMSE: 29.834959 m


Actual
Predicted
500
0
−500
−1000
−1500
−2000
−2500
−2000
−1000
0
1000
2000

Conclusion

Scenario	Uma (80 km/hr)	UMi-StreetCanyon (40km/hr)	Rma (120 km/hr)
Map size (m)	X: (-2500,2500) Y: (-2500,2500)	X: (-500,500) Y: (-500,500)	X: (-5000,5000) Y: (-5000,5000)
RMSE or Step size (m)	20.084332 m	9.769335 m	29.834959 m

Test case

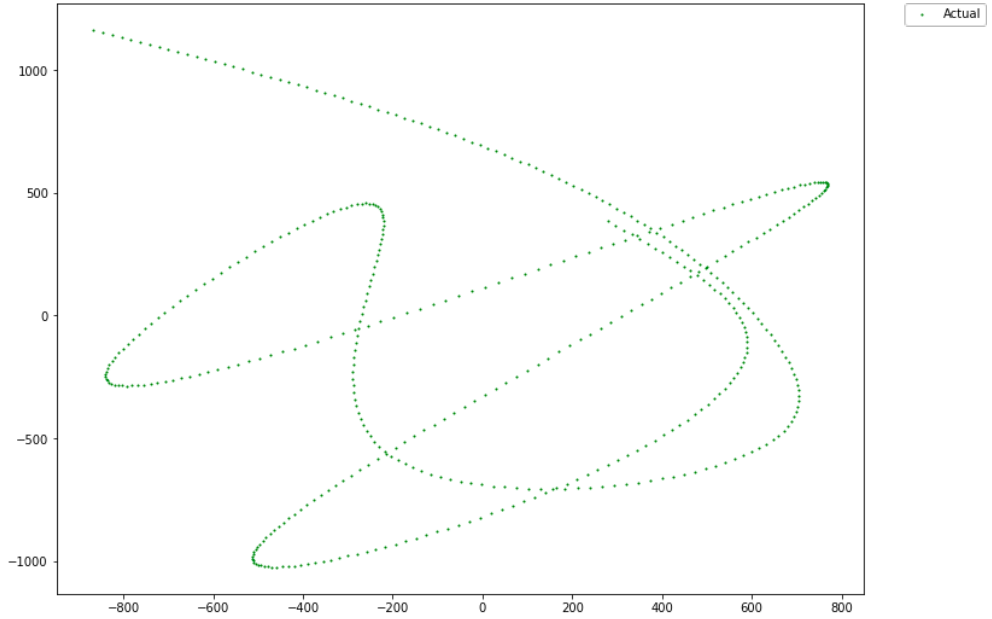
Scenario	Uma (80 km/hr)
Map size (m)	X: (-2500,2500) Y: (-2500,2500)
RMSE or Step size (m)	20.084332 m

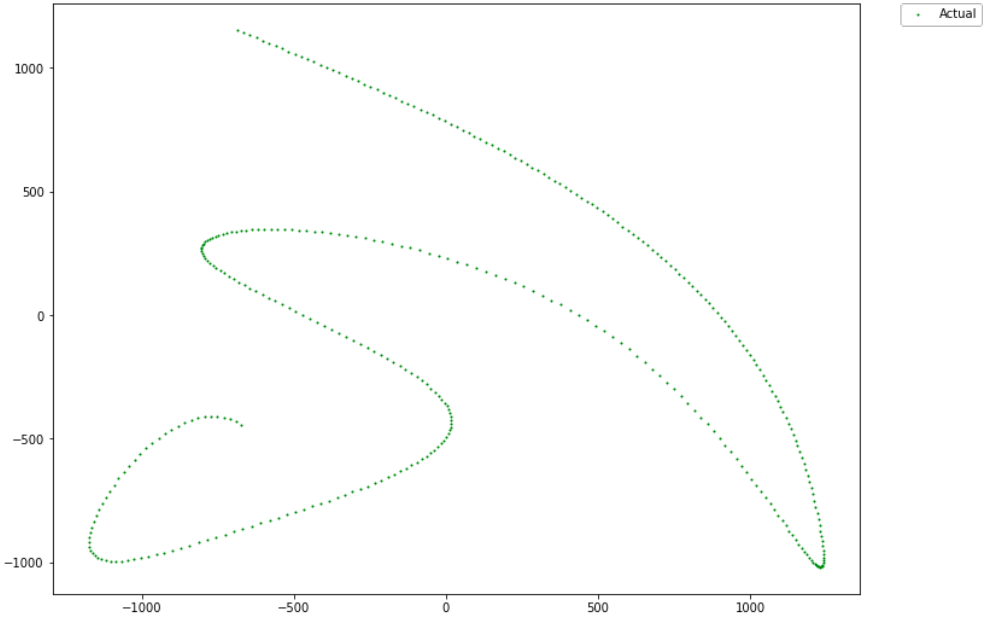
1. Generate 10 routes dataset for training the model

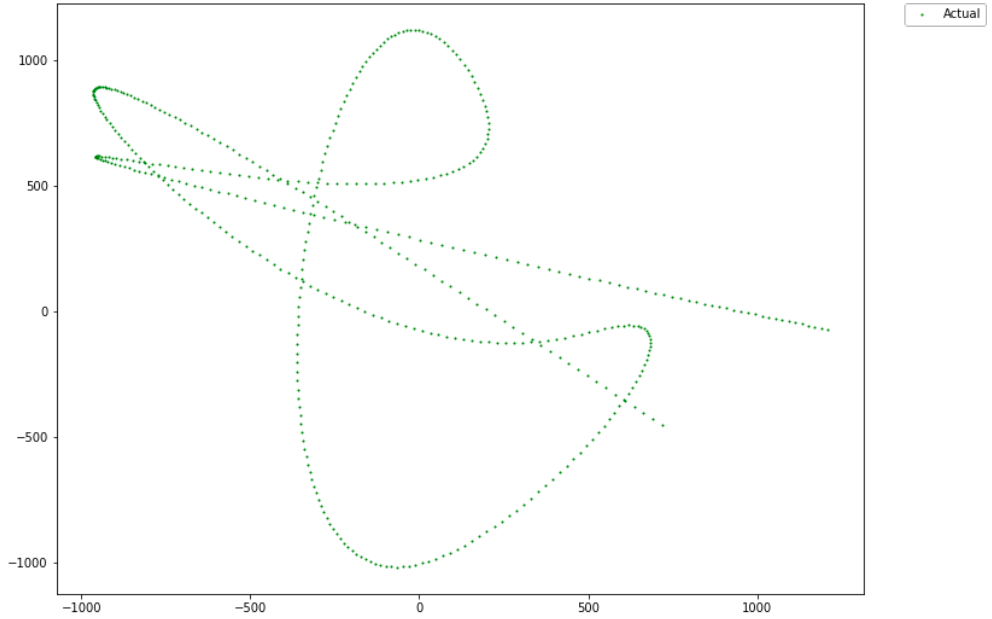
2. Train model with training dataset and evaluate model with test dataset

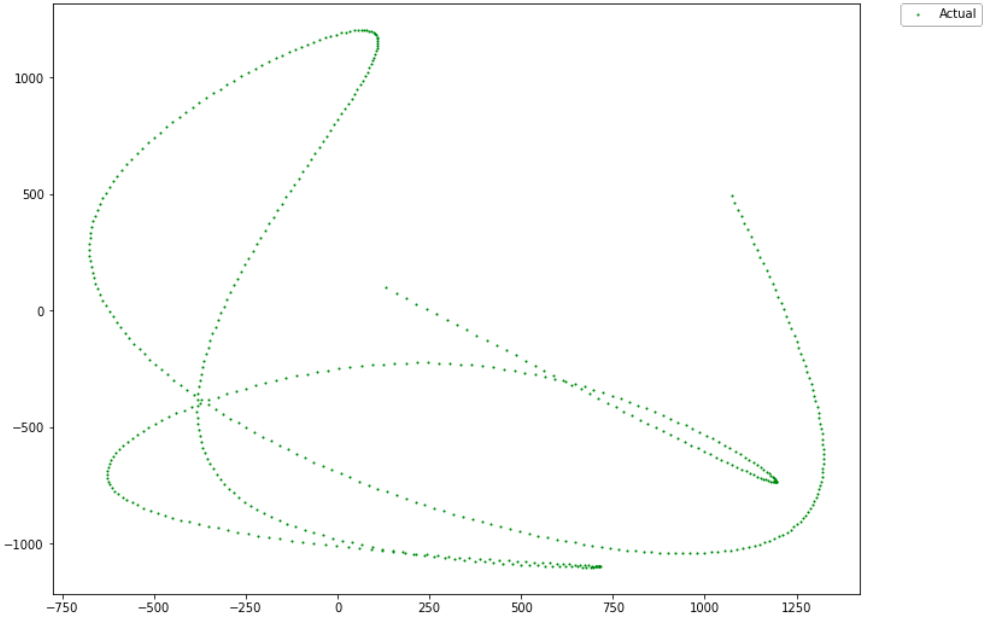
3. Evaluate model again with 50 starting points of 50 random routes

Sample plots from all routes









X and Y data was chosen from the 10 datsets to be fed into ML

```
traj01.columns = ["X", "Y", "Speed", "Time"]
traj02.columns = ["X", "Y", "Speed", "Time"]
traj03.columns = ["X", "Y", "Speed", "Time"]
traj04.columns = ["X", "Y", "Speed", "Time"]
traj05.columns = ["X", "Y", "Speed", "Time"]
traj06.columns = ["X", "Y", "Speed", "Time"]
traj07.columns = ["X", "Y", "Speed", "Time"]
traj08.columns = ["X", "Y", "Speed", "Time"]
traj09.columns = ["X", "Y", "Speed", "Time"]
traj10.columns = ["X", "Y", "Speed", "Time"]
```

```
traj01,traj10
```

(	X	Y	Speed	Time
0	-1309.000000	-8881.000000	0.000000	0
1	-1364.323797	-8802.824599	95.771164	1
2	-1419.631820	-8724.650237	95.761204	2
3	-1474.908294	-8646.477955	95.741288	3
4	-1530.137444	-8568.308792	95.711426	4
..	...	...	...	...
835	1330.265530	-6807.994029	103.153956	835
836	1228.247784	-6825.039314	103.431921	836
837	1126.008103	-6842.050377	103.645206	837
838	1023.613862	-6859.037608	103.793769	838
839	921.132439	-6876.011395	103.877580	839

Transform time series data to supervised problem and concatenate all X Y dataset into 1 dataset

Feature Variable

Target Variable

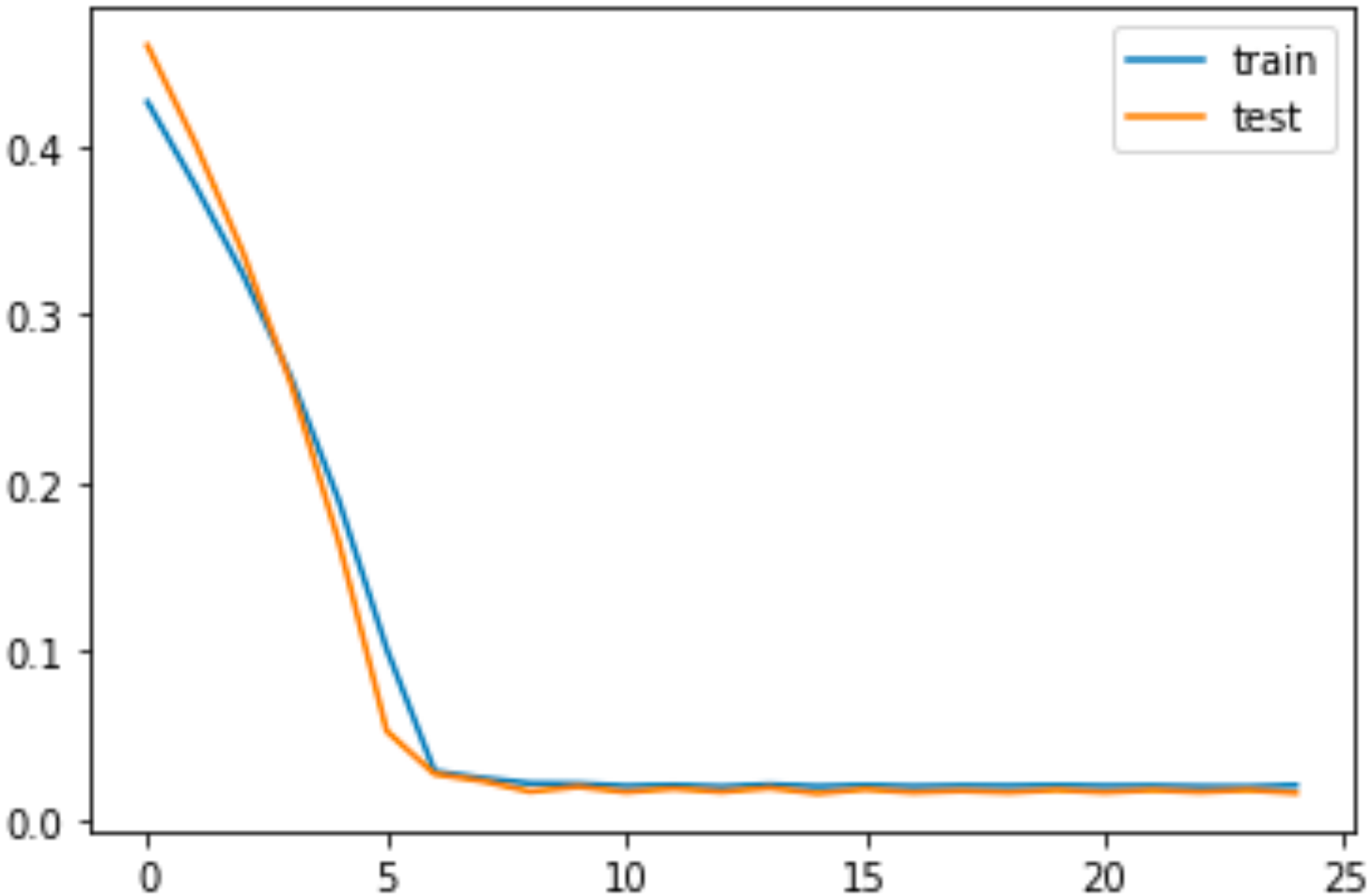
	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	-0.133186	-1.000000	-0.139738	-0.989572
2	-0.139738	-0.989572	-0.146288	-0.979145
3	-0.146288	-0.979145	-0.152834	-0.968717
4	-0.152834	-0.968717	-0.159374	-0.958290
5	-0.159374	-0.958290	-0.165907	-0.947864

	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	0.617546	0.027658	0.625560	0.019306
2	0.625560	0.019306	0.633571	0.010955
3	0.633571	0.010955	0.641574	0.002607
4	0.641574	0.002607	0.649567	-0.005739
5	0.649567	-0.005739	0.657545	-0.014080

Bidirectional LSTM model

```
from keras.layers import Bidirectional

# design network
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',mode='auto', patience=10,restore_best_weights=True)
model1 = Sequential()
model1.add(Bidirectional(LSTM(150, input_shape=(train_X.shape[1], train_X.shape[2]))))
model1.add(Dropout(0.1))
model1.add(Dense(test_y.shape[1]))
model1.compile(loss='mae', optimizer='adam',metrics=['accuracy'])

# fit network
history1 = model1.fit(train_X, train_y, epochs=600, callbacks=[callback], batch_size=72, validation_data=(test_X,
test_y), verbose=2, shuffle=False)

# plot history
pyplot.plot(history1.history['loss'], label='train')
pyplot.plot(history1.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

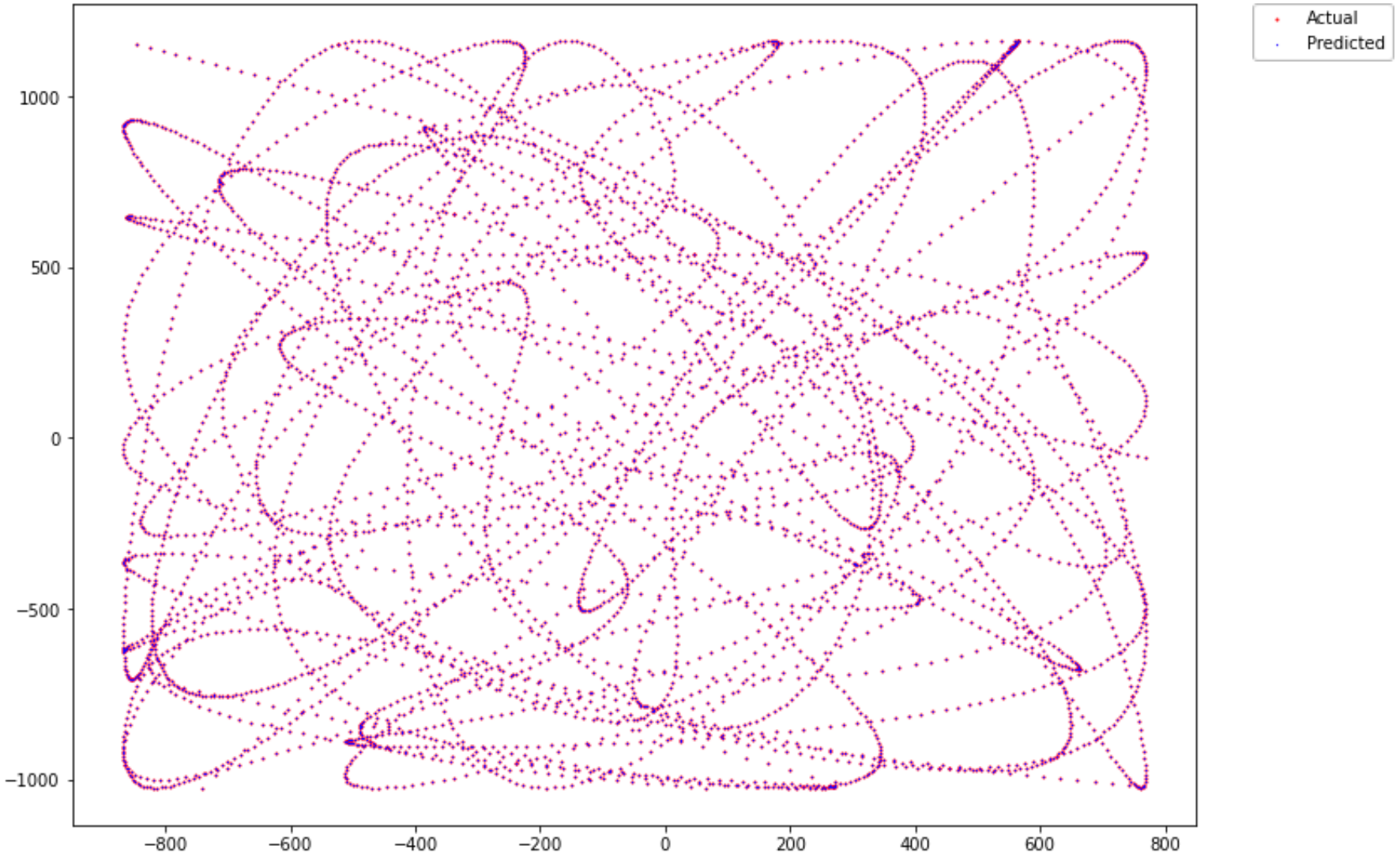
Epoch 25/600 15/15 - 0s - loss: 0.0206 - accuracy: 0.9952 - val_loss: 0.0160 - val_accuracy: 0.9902



132/132 [==============================]
- 0s 863us/step - loss: 0.0158 - accuracy: 0.9914

Accuracy: 99.1435
Loss: 0.015781

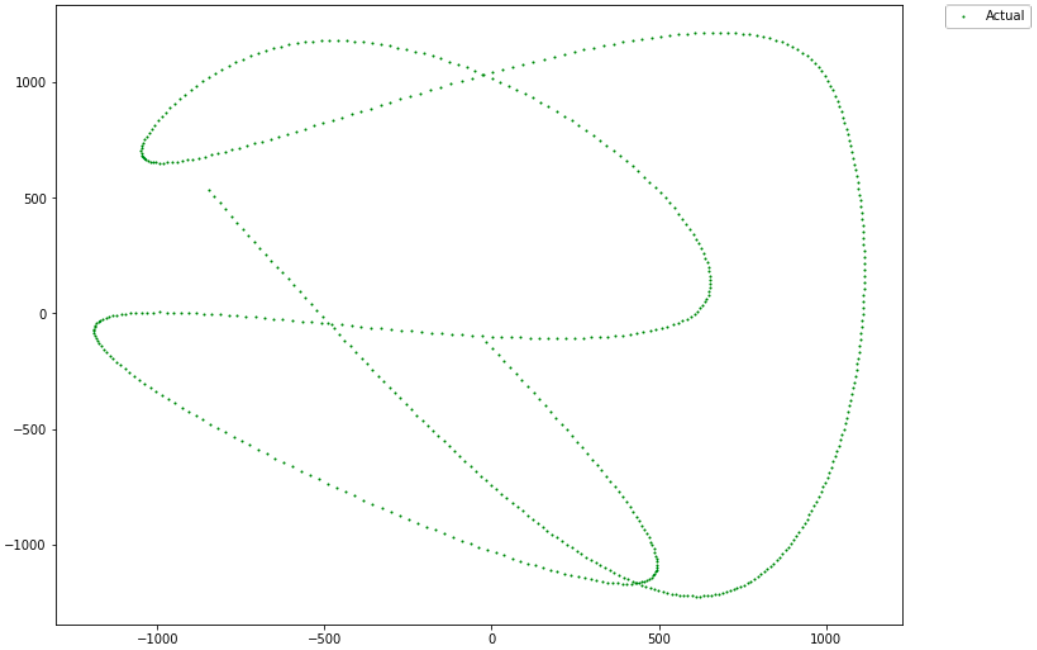
Test RMSE: 17.660842m , Train RMSE: 15.724601m

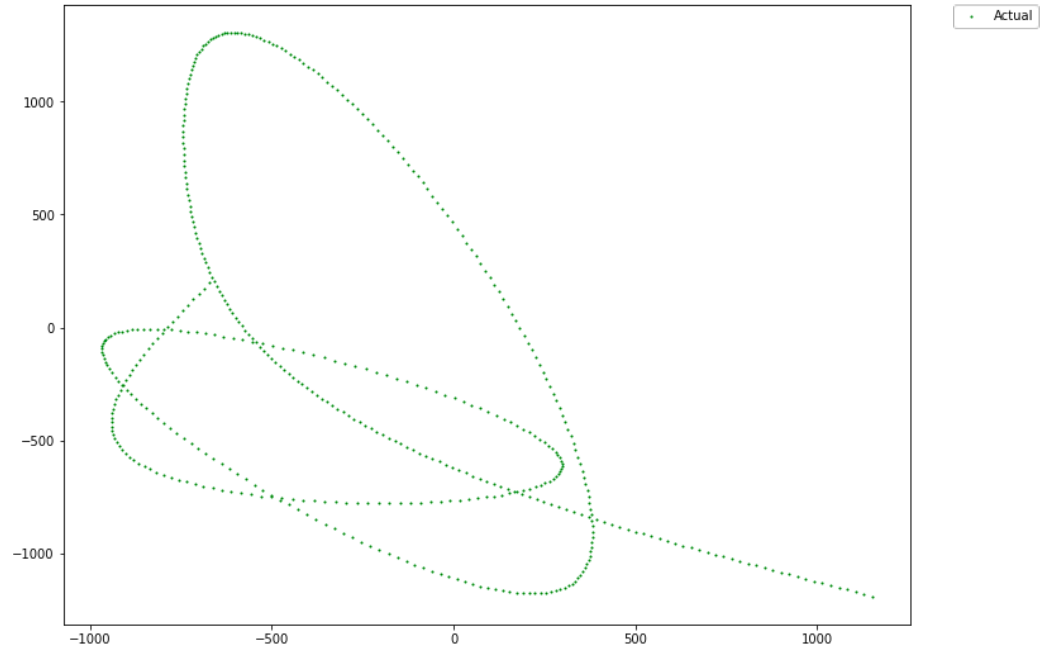


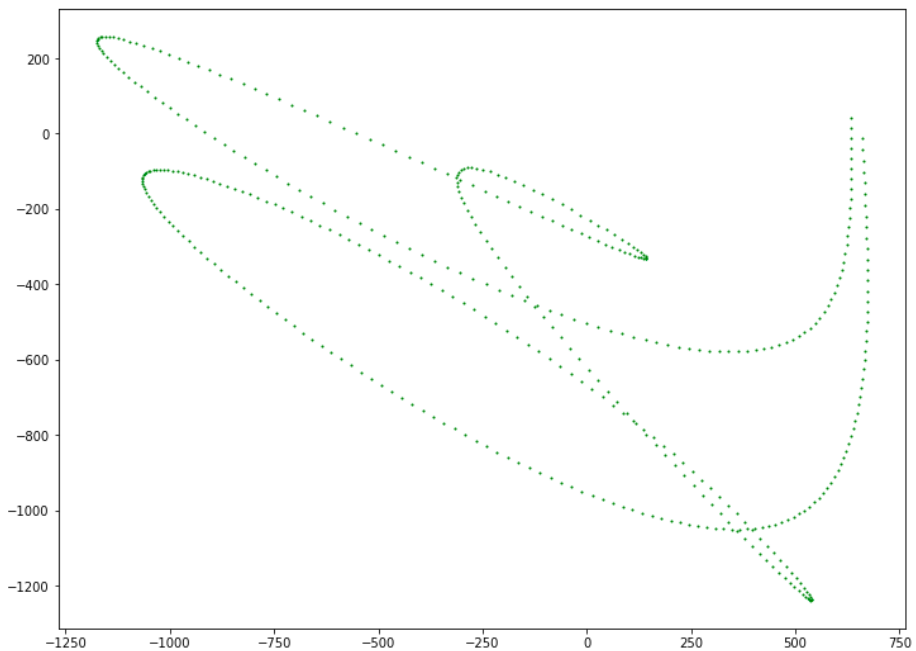
Try feeding just the starting point of the path (1 point) and Let ML predicts

```
traj00 = pd.read_csv('UE0_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj01 = pd.read_csv('UE1_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj02 = pd.read_csv('UE2_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj03 = pd.read_csv('UE3_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj04 = pd.read_csv('UE4_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj05 = pd.read_csv('UE5_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj06 = pd.read_csv('UE6_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj07 = pd.read_csv('UE7_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj08 = pd.read_csv('UE8_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj09 = pd.read_csv('UE9_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj10 = pd.read_csv('UE10_07-12-2021_20-27-58.csv', low_memory=False,  header=None)

traj11 = pd.read_csv('UE11_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj12 = pd.read_csv('UE12_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj13 = pd.read_csv('UE13_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj14 = pd.read_csv('UE14_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj15 = pd.read_csv('UE15_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj16 = pd.read_csv('UE16_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj17 = pd.read_csv('UE17_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj18 = pd.read_csv('UE18_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj19 = pd.read_csv('UE19_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
traj20 = pd.read_csv('UE20_07-12-2021_20-27-58.csv', low_memory=False,  header=None)
```

Sample plots

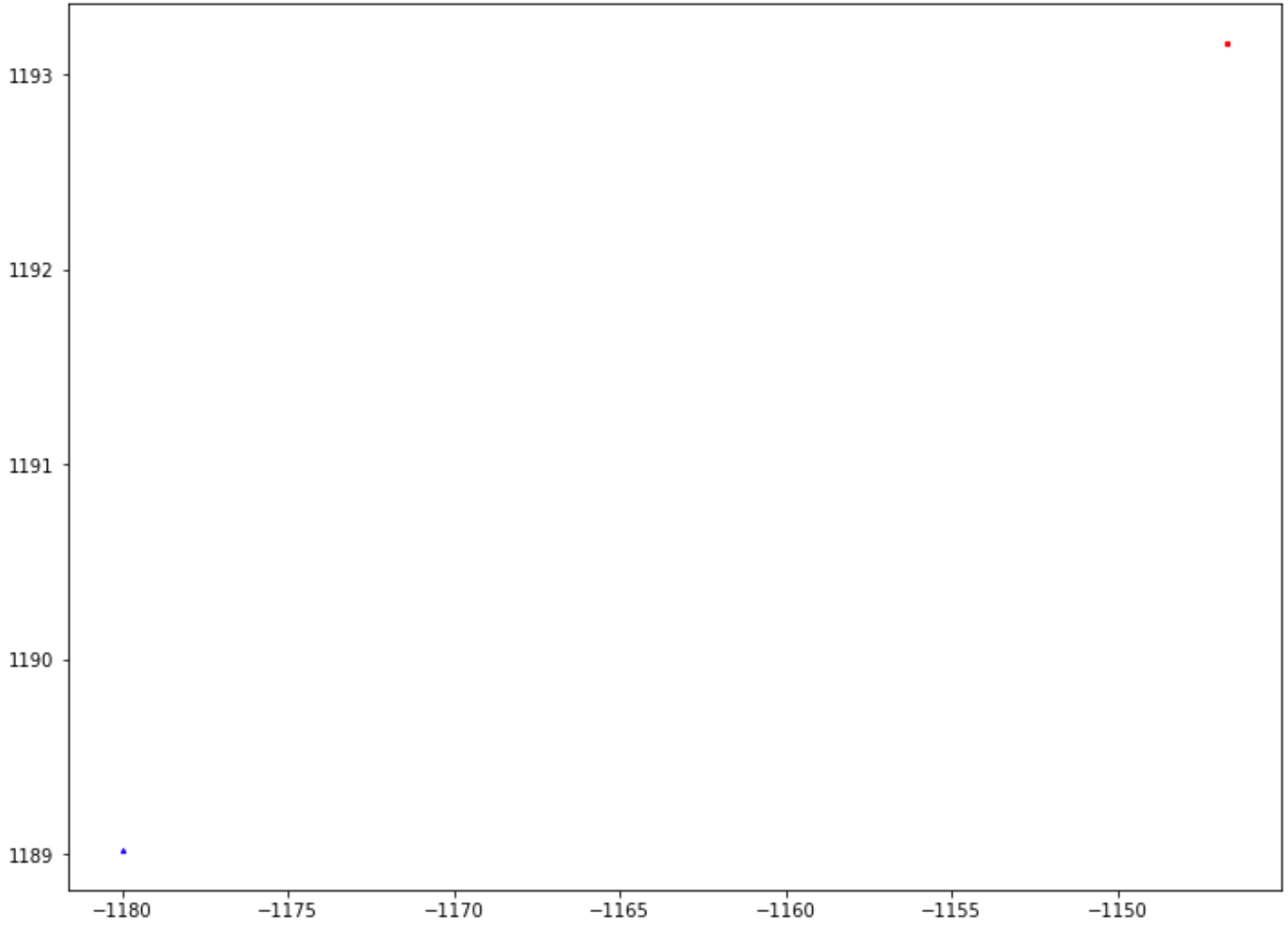






Prediction results

Actual vs Predicted plots of 1st point for trajectory 00



Input

	X	Y	Speed	Time
0	-1180.000000	1189.000000	0.000000	0
1	-1146.705463	1193.160693	33.553503	1

Expected outcome

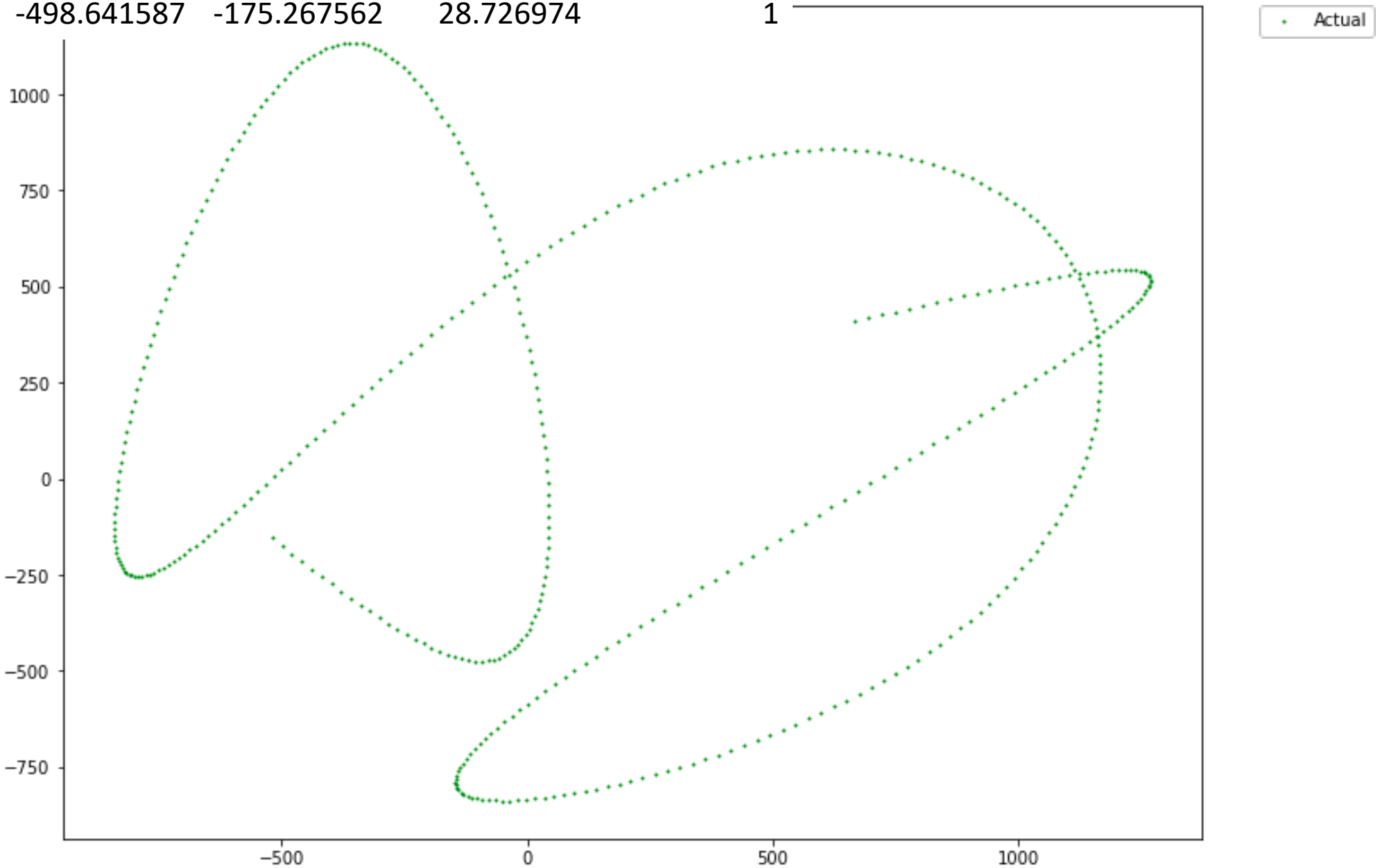
Average RMSE: 23.708999 m

Actual outcome:

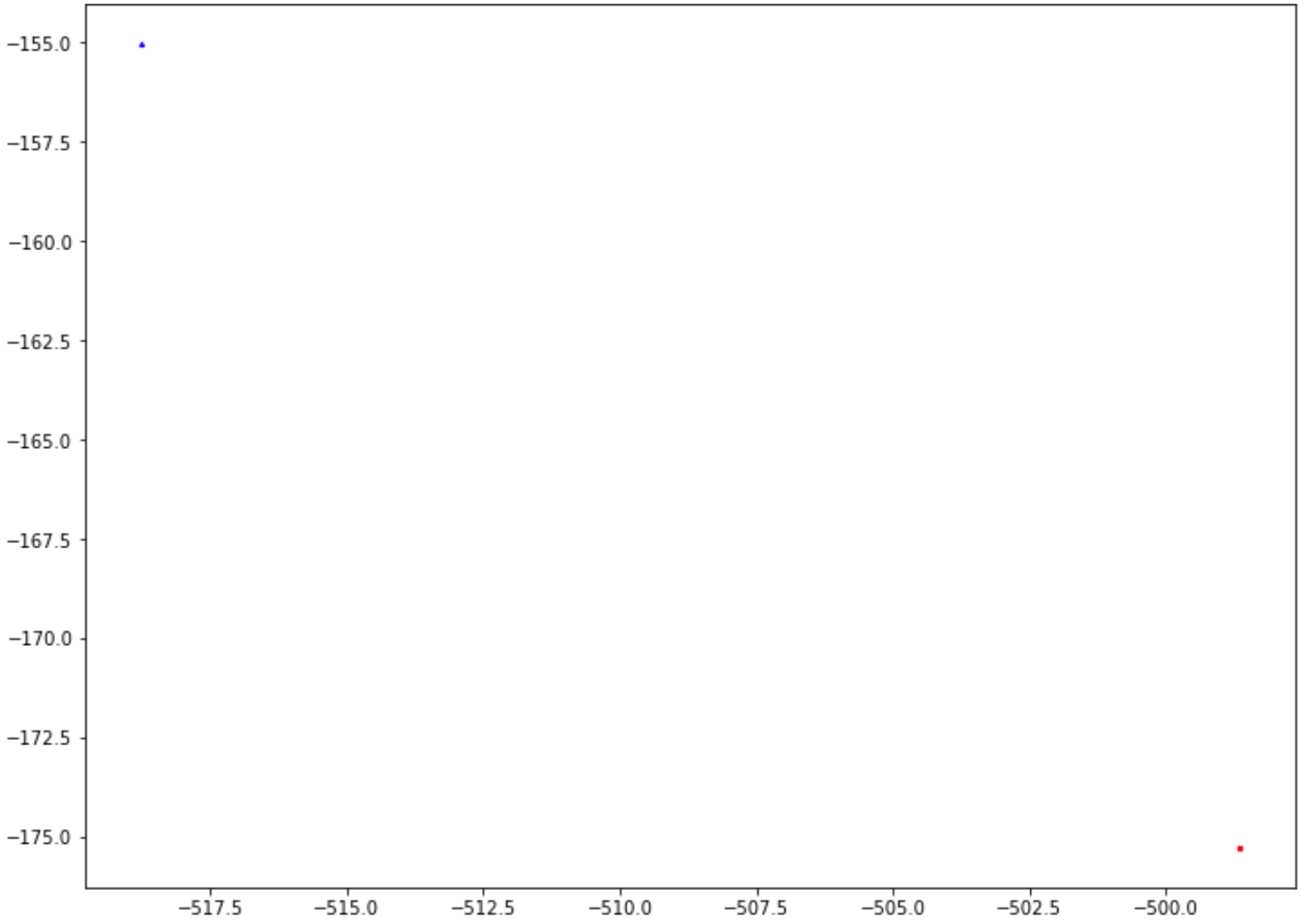
array([[-1146.7054, 1193.1606]]

Predicted outcome:
array([[1179.9791, 1189.026]]

	X	Y	Speed	Time
0	-519.000000	-155.000000	0.000000	0
1	-498.641587	-175.267562	28.726974	1



Actual vs Predicted plots of 1st point for trajectory 01



Input

	X	Y	Speed	Time
0	-519.000000	-155.000000	0.000000	0
1	-498.641587	-175.267562	28.726974	1

Expected outcome

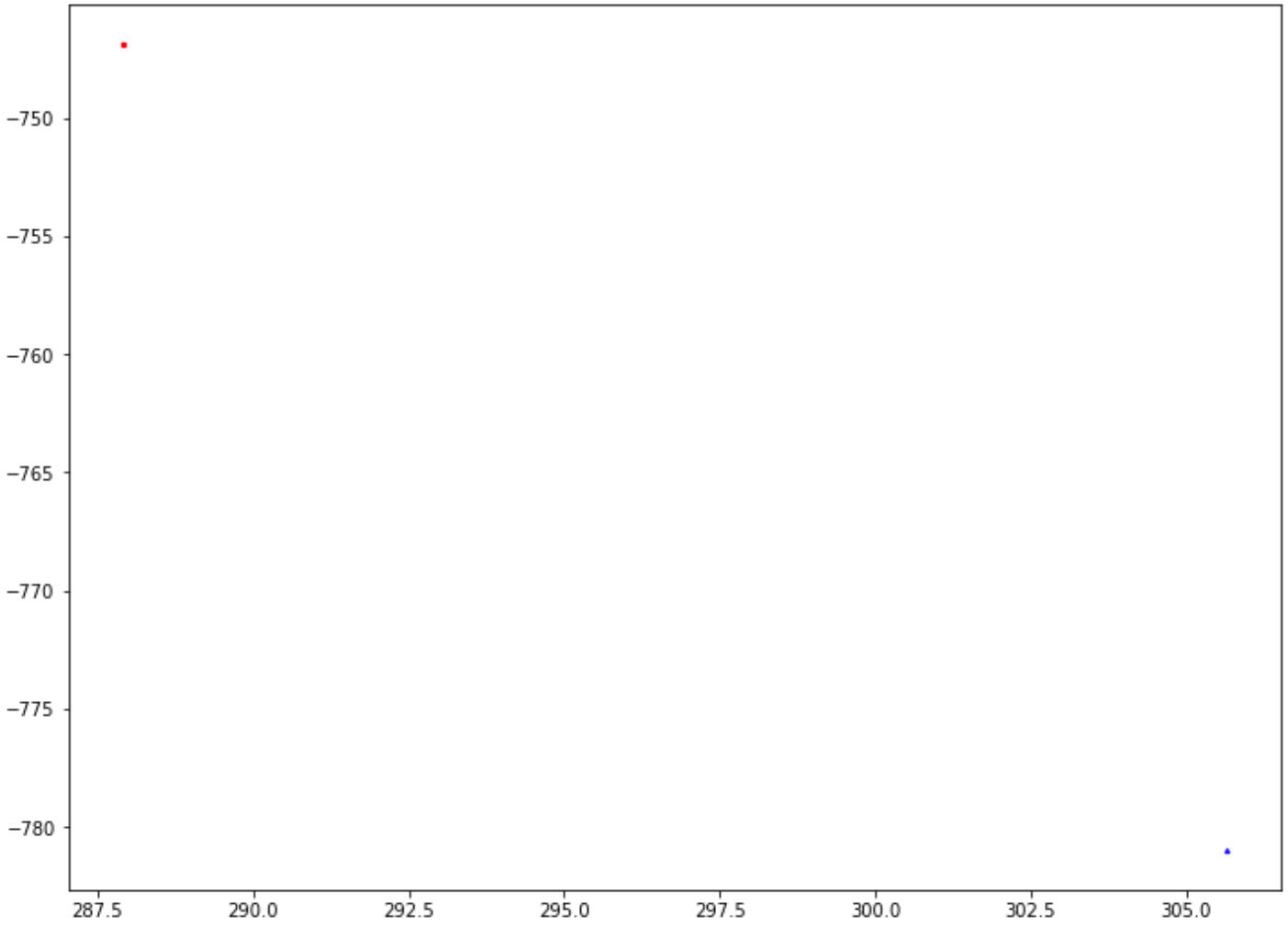
Average RMSE: 20.185940 m

Actual outcome:

array([[-498.6416 , -175.26756]]

Predicted outcome:
array([[-518.7653 , -155.01959]]

Actual vs Predicted plots of 1st point for trajectory 02



	X	Y	Speed	Time
0	306.000000	-781.000000	0.000000	0
1	287.913757	-746.879159	38.617923	1

Input

Expected outcome

Average RMSE: 27.163783 m

Actual outcome:

[[287.91376 -746.87915]]

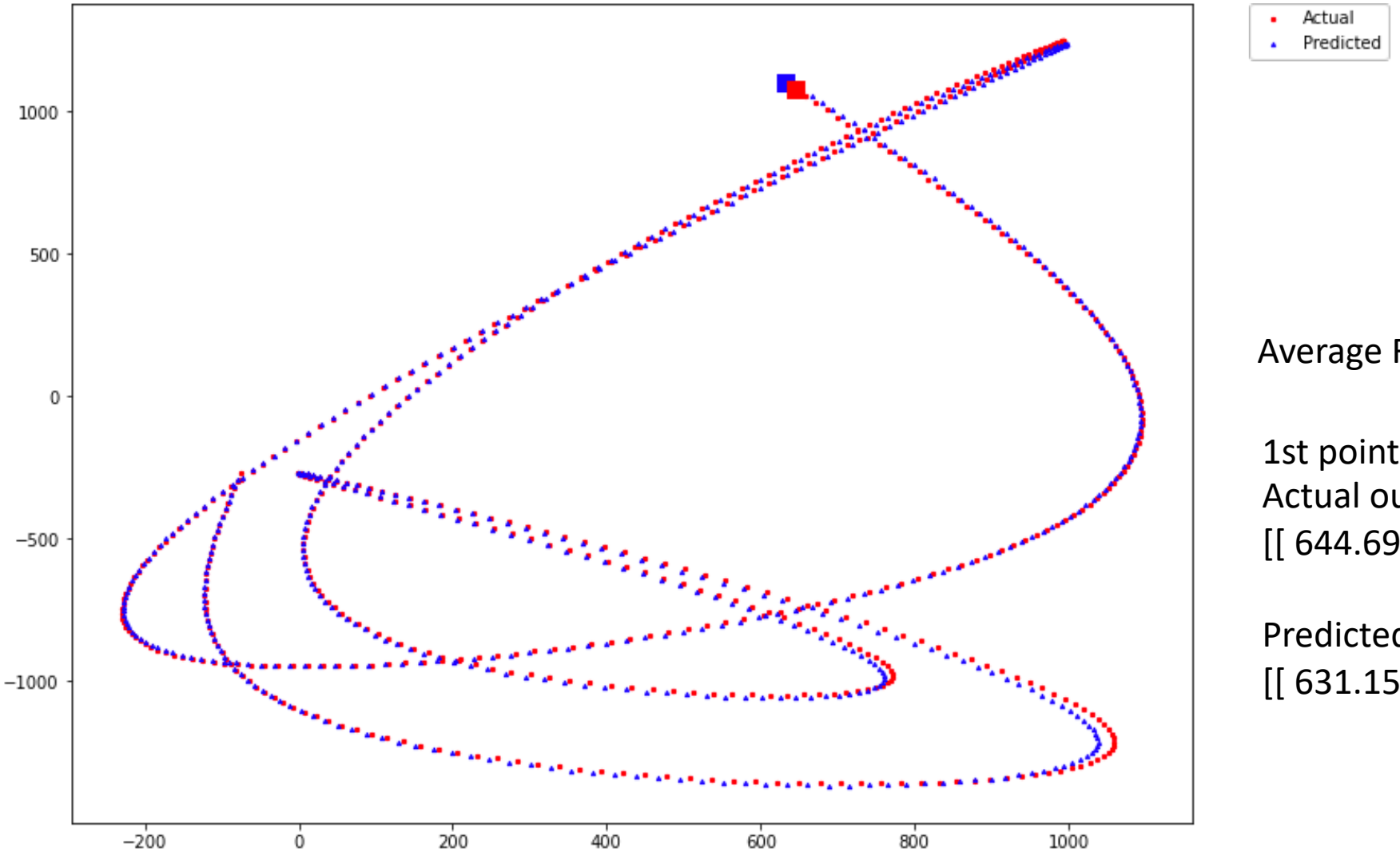
Predicted outcome:

[[305.64188 -780.9593]]

Route Number	Average RMSE(m)	Actual outcome	Predicted outcome
traj03	23.375563	[[-420.53052 1248.557]]	[[-387.66394 1245.0043]]
traj04	27.56115	[[-12.040055 -137.72424]]	[[-30.988033 -171.78607]]
traj05	28.73715	[[1194.0282 791.0483]]	[[1234.0765 797.96045]]
traj06	17.396096	[[556.53253 -68.40773]]	[[540.1906 -50.017788]]
traj07	24.095802	[[-16.493214 315.29327]]	[[6.04305 340.85364]]
traj08	28.215966	[[1156.3525 -773.8077]]	[[1196.0759 -770.02167]]
traj09	28.136374	[[779.74677 -644.0747]]	[[815.2821 -661.97864]]
traj10	16.428861	[[606.8783 -1005.8786]]	[[629.5421 -1010.9939]]
traj11	22.867931	[[-828.9273 503.38547]]	[[-845.80316 530.9733]]
traj12	17.241216	[[1128.121 -1179.418]]	[[1150.5469 -1188.9886]]
traj13	20.76738	[[412.8217 -814.1401]]	[[441.05392 -806.0463]]
traj14	26.668663	[[-249.9264 -1177.7723]]	[[-273.9848 -1206.8176]]
traj15	20.05866	[[267.10873 -623.30927]]	[[276.80414 -649.9682]]
traj16	19.254963	[[634.27203 13.744358]]	[[634.0031 40.97366]]
traj17	18.635087	[[1077.9507 -723.154]]	[[1086.8208 -747.9704]]
traj18	21.18135	[[-646.66205 372.16565]]	[[-637.9834 400.83585]]
traj19	26.325498	[[-176.31052 702.6458]]	[[-194.78459 734.96875]]
traj20	14.624681	[[-897.7342 -1026.1322]]	[[-913.98975 -1038.9197]]
traj21	23.332483	[[-979.9048 342.5515]]	[[-972.98676 374.81528]]
traj22	20.374743	[[-599.09357 -215.81651]]	[[-614.99 -239.84908]]
traj23	24.696206	[[858.45013 -787.46124]]	[[887.41486 -806.9766]]
traj24	22.955109	[[-962.7131 217.78612]]	[[-993.9803 209.05482]]
traj25	30.393334	[[-925.61456 1154.2216]]	[[-956.63824 1183.9712]]

Route Number	Average RMSE(m)	Actual outcome	Predicted outcome
traj26	24.03476	[[996.50555 -1140.4313]]	[[1030.0641 -1135.0309]]
traj27	21.768322	[[-421.29816 838.9247]]	[[-413.98602 868.82874]]
traj28	25.954005	[[-538.36584 -36.65161]]	[[-533.99164 -20.868343]]
traj29	25.070993	[[718.5209 -768.08875]]	[[737.61426 -797.96436]]
traj30	18.172453	[[-498.6889 273.94504]]	[[-485.27106 252.02615]]
traj31	21.714256	[[-962.34985 -208.07155]]	[[-969.9952 -237.81322]]
traj32	16.971309	[[-354.24734 -1016.942]]	[[-368.9907 -1035.881]]
traj33	17.658008	[[-559.2095 348.85083]]	[[-540.96515 365.90237]]
traj34	21.684942	[[8.825372 -553.9888]]	[[35.0501 -538.09106]]
traj35	21.05559	[[744.0259 -435.2382]]	[[728.18475 -410.0244]]
traj36	15.736444	[[-1043.1824 -961.6871]]	[[-1055.9918 -979.8857]]
traj37	27.456148	[[-1114.816 825.1265]]	[[-1144.652 849.976]]
traj38	21.702828	[[-703.9108 -92.54139]]	[[-726.7339 -72.01985]]
traj39	19.933048	[[890.55005 950.43555]]	[[870.2369 969.9811]]
traj40	16.948901	[[-917.78674 -1130.4629]]	[[-907.21356 -1151.9742]]
traj41	21.908874	[[985.29645 276.16553]]	[[972.0084 248.17578]]
traj42	24.233669	[[-863.70886 -954.61707]]	[[-897.97833 -954.99756]]
traj43	28.97179	[[998.45807 -290.77368]]	[[1037.0737 -277.07843]]
traj44	19.124011	[[-888.5246 995.03253]]	[[-884.9933 1021.84644]]
traj45	13.75447	[[-174.7279 747.5702]]	[[-175.98535 766.98126]]
traj46	15.657306	[[-586.3517 1125.707]]	[[-598.8542 1143.9824]]
traj47	27.159612	[[-822.5466 413.75876]]	[[-859.5683 423.9901]]
traj48	27.002002	[[-441.1503 -691.4758]]	[[-462.9862 -722.8033]]
traj49	22.894387	[[1193.5542 918.9691]]	[[1210.0315 946.84033]]

Actual vs Predicted plots of 1st point and whole points for trajectory 50



Average RMSE: 16.962235m

1st point
Actual outcome:
[[644.6901 1076.7563]]

Predicted outcome:
[[631.15784 1100.9767]]

Test case

Scenario	Uma (80 km/hr)
Map size (m)	X: (-2500,2500) Y: (-2500,2500)
RMSE or Step size (m)	20.084332 m

1. Generate 10 routes dataset for training the model

2. Train model with training dataset and evaluate model with test dataset

3. Evaluate model again with 50 starting points of 50 random routes