Timeseries Multi-Step Multi-Output

Data treatment for model 1 Features:

- local lateral position Local_X, to account for different behaviors depending on the driving lane,
- local longitudinal position Local_Y, to account for different behaviors when approaching the merging lan
- lateral and longitudinal velocities vx and vy,
- type (motorcycle, car or truck)

Import packages

In [32]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf

plt.rcParams['figure.figsize'] = (8, 6)
```

Load dataset

In [33]:
```
url_1 = 'https://github.com/duonghung86/Vehicle-trajectory-tracking/raw/master/Data/NGS
zip_path = tf.keras.utils.get_file(origin=url_1, fname=url_1.split('/')[-1], extract=Tr
csv_path = zip_path.replace('zip','csv')
csv_path
```

Out[33]:
```
'C:\\Users\\Duong Hung\\.keras\\datasets\\0750_0805_us101_smoothed_11_.csv'
```

Let's take a glance at the data. Here are the first few rows:

In [34]:
```
df = pd.read_csv(csv_path)
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 18 columns):
 #   Column        Non-Null Count    Dtype  
---  ------        --------------    -----  
 0   Vehicle_ID    1048575 non-null  int64  
 1   Frame_ID      1048575 non-null  int64  
 2   Total_Frames  1048575 non-null  int64  
 3   Global_Time   1048575 non-null  int64  
 4   Local_X       1048575 non-null  float64
 5   Local_Y       1048575 non-null  float64
 6   Global_X      1048575 non-null  float64
 7   Global_Y      1048575 non-null  float64
 8   v_Length      1048575 non-null  float64
 9   v_Width       1048575 non-null  float64
 10  v_Class       1048575 non-null  int64  
 11  v_Vel         1048575 non-null  float64
 12  v_Acc         1048575 non-null  float64
 13  Lane_ID       1048575 non-null  int64  
 14  Preceeding    1048575 non-null  int64  
 15  Following     1048575 non-null  int64  
 16  Space_Hdwy    1048575 non-null  float64
 17  Time_Hdwy     1048575 non-null  float64
dtypes: float64(10), int64(8)
memory usage: 144.0 MB
```

Out[34]:

	Vehicle_ID	Frame_ID	Total_Frames	Global_Time	Local_X	Local_Y	Global_X	Global_Y	v_
0	2	13	437	1118846980200	16.467196	35.380427	6451137.641	1873344.962	
1	2	14	437	1118846980300	16.446594	39.381608	6451140.329	1873342.000	
2	2	15	437	1118846980400	16.425991	43.381541	6451143.018	1873339.038	
3	2	16	437	1118846980500	16.405392	47.380780	6451145.706	1873336.077	
4	2	17	437	1118846980600	16.384804	51.379881	6451148.395	1873333.115	

Next look at the statistics of the dataset:

In [35]:
```
df.describe().transpose().round(3)
```

Out[35]:

	count	mean	std	min	25%	50%	7
Vehicle_ID	1048575.0	1.533080e+03	790.271	2.000000e+00	9.320000e+02	1.574000e+03	2.210000e
Frame_ID	1048575.0	4.518249e+03	2412.479	8.000000e+00	2.455000e+03	4.586000e+03	6.598000e
Total_Frames	1048575.0	5.608770e+02	146.577	1.770000e+02	4.640000e+02	5.180000e+02	6.400000e
Global_Time	1048575.0	1.118847e+12	241247.914	1.118847e+12	1.118847e+12	1.118847e+12	1.118848e
Local_X	1048575.0	2.940600e+01	16.666	5.340000e-01	1.728400e+01	2.955700e+01	4.187500e
Local_Y	1048575.0	1.002056e+03	596.357	1.796600e+01	4.883960e+02	9.640280e+02	1.491548e
Global_X	1048575.0	6.451838e+06	446.275	6.451107e+06	6.451450e+06	6.451808e+06	6.452205e
Global_Y	1048575.0	1.872677e+06	397.006	1.871875e+06	1.872352e+06	1.872699e+06	1.873015e
v_Length	1048575.0	1.463500e+01	4.870	4.000000e+00	1.200000e+01	1.450000e+01	1.650000e
v_Width	1048575.0	6.132000e+00	1.037	2.000000e+00	5.400000e+00	6.000000e+00	6.900000e
v_Class	1048575.0	2.009000e+00	0.191	1.000000e+00	2.000000e+00	2.000000e+00	2.000000e
v_Vel	1048575.0	3.877400e+01	14.110	0.000000e+00	3.031700e+01	3.989800e+01	4.820200e
v_Acc	1048575.0	3.610000e-01	5.852	-3.193080e+02	-1.752000e+00	1.700000e-02	2.754000e
Lane_ID	1048575.0	2.956000e+00	1.469	1.000000e+00	2.000000e+00	3.000000e+00	4.000000e
Preceeding	1048575.0	1.459864e+03	844.319	0.000000e+00	7.880000e+02	1.519000e+03	2.186000e
Following	1048575.0	1.477269e+03	843.679	0.000000e+00	8.120000e+02	1.533000e+03	2.202000e
Space_Hdwy	1048575.0	7.815800e+01	48.615	0.000000e+00	4.984000e+01	6.911000e+01	9.725000e
Time_Hdwy	1048575.0	1.090800e+02	1027.551	0.000000e+00	1.460000e+00	1.970000e+00	2.730000e

In [36]:
```
df.columns
```

Out[36]:
```
Index(['Vehicle_ID', 'Frame_ID', 'Total_Frames', 'Global_Time', 'Local_X',
       'Local_Y', 'Global_X', 'Global_Y', 'v_Length', 'v_Width', 'v_Class',
       'v_Vel', 'v_Acc', 'Lane_ID', 'Preceeding', 'Following', 'Space_Hdwy',
       'Time_Hdwy'],
      dtype='object')
```

In [37]:
```
#  keep only columns that are useful for now
kept_cols = ['Vehicle_ID', 'Frame_ID', 'Total_Frames', 'Local_X','Local_Y','v_Vel']
df = df[kept_cols]
df.head()
```

Out[37]:

	Vehicle_ID	Frame_ID	Total_Frames	Local_X	Local_Y	v_Vel
0	2	13	437	16.467196	35.380427	40.000000
1	2	14	437	16.446594	39.381608	40.012349
2	2	15	437	16.425991	43.381541	39.999855
3	2	16	437	16.405392	47.380780	39.992920
4	2	17	437	16.384804	51.379881	39.991544

In [38]:
```
'the number of vehicles is {}'.format(len(df.Vehicle_ID.unique()))
```

Out[38]:
```
'the number of vehicles is 1993'
```

In [39]:
```
# let use only 1000 vehicle to reduce the computation workload
vehicle_list = df.Vehicle_ID.unique()
n_veh = 100 # number of vehicles
np.random.seed(48)
new_veh_list = np.random.choice(vehicle_list,n_veh)
print(new_veh_list)
```

```
[2149  901 1903 2061  567 1362 2593 1396  736 1841  570  346  394 2349
   10 1727 1339 2571 2741 2444 1176 1744 2587 2270 2328  869 2218 2167
  711 2167  389 1518  458 2255  686  190 1175 2589 1675  701  204 2567
 1690 2706  641   73 1785 1844 2126 1975 1942 2171  905 2161 1038  854
  212 1702  798 1259 2336  585 2691 1183 2202  729 2219  635 1106 2164
 1455 2136 1767 1044 2310   75  888 1605 1976 2209  950  790 1001 2249
 2255 2150 2172 1552  417 1149  923 1577 1368  825 1717 1953 2336 2211
  374 2384]
```

In [40]:
```
new_df = df[df.Vehicle_ID.isin(new_veh_list)]
new_df.info()
new_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 51316 entries, 2454 to 1029946
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Vehicle_ID    51316 non-null  int64  
 1   Frame_ID      51316 non-null  int64  
 2   Total_Frames  51316 non-null  int64  
 3   Local_X       51316 non-null  float64
 4   Local_Y       51316 non-null  float64
 5   v_Vel         51316 non-null  float64
dtypes: float64(3), int64(3)
memory usage: 2.7 MB
```

Out[40]:

	Vehicle_ID	Frame_ID	Total_Frames	Local_X	Local_Y	v_Vel
2454	10	39	436	4.311965	35.406783	40.920000
2455	10	40	436	4.289860	39.935881	45.291518
2456	10	41	436	4.268287	44.330462	43.946334
2457	10	42	436	4.247104	48.609480	42.790711
2458	10	43	436	4.226170	52.791893	41.824650

Data transformation

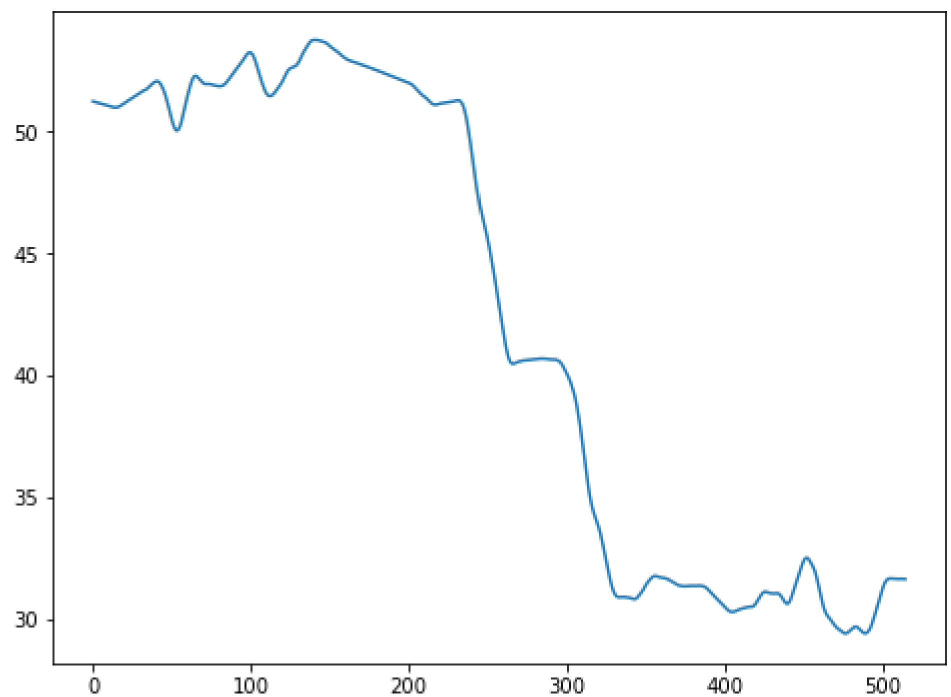
1 object and 1 target variable: Predict at time *t*

Prepare the data set

In [41]:
```
simple_df = new_df[new_df.Vehicle_ID == new_veh_list[0]].copy()
simple_df = simple_df[['Frame_ID','Local_X']]
simple_df.set_index('Frame_ID', inplace = True)
simple_df.sort_index(inplace=True)
simple_df.reset_index(drop=True, inplace=True)
plt.plot(simple_df)
simple_df.head()
```

Out[41]:

	Local_X
0	51.214252
1	51.196441
2	51.178494
3	51.160446
4	51.142331



In [ ]:

In [42]:
```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    """
    Frame a time series as a supervised learning dataset.
    Arguments:
    data: Sequence of observations as a list or NumPy array.
    n_in: Number of lag observations as input (X).
    n_out: Number of observations as output (y).
    dropnan: Boolean whether or not to drop rows with NaN values.
    Returns:
    Pandas DataFrame of series framed for supervised learning.
    """
    #n_vars = 1 if type(data) is list else data.shape[1]
    variables = list(data.columns)
    df = data.copy()
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += ['{}(t-{})'.format(j, i) for j in variables]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += ['{}(t)'.format(j) for j in variables]
        else:
            names += ['{}(t+{})'.format(j, i) for j in variables]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

In [43]:
```
series_to_supervised(simple_df, n_in=4, n_out=1, dropnan=False).head()
```

Out[43]:

	Local_X(t-4)	Local_X(t-3)	Local_X(t-2)	Local_X(t-1)	Local_X(t)
0	NaN	NaN	NaN	NaN	51.214252
1	NaN	NaN	NaN	51.214252	51.196441
2	NaN	NaN	51.214252	51.196441	51.178494
3	NaN	51.214252	51.196441	51.178494	51.160446
4	51.214252	51.196441	51.178494	51.160446	51.142331

In [44]:
```
transformed_df = series_to_supervised(simple_df, n_in=4, n_out=1, dropnan=True)
transformed_df.head()
```

Out[44]:

	Local_X(t-4)	Local_X(t-3)	Local_X(t-2)	Local_X(t-1)	Local_X(t)
4	51.214252	51.196441	51.178494	51.160446	51.142331
5	51.196441	51.178494	51.160446	51.142331	51.124182
6	51.178494	51.160446	51.142331	51.124182	51.106019
7	51.160446	51.142331	51.124182	51.106019	51.088021
8	51.142331	51.124182	51.106019	51.088021	51.069916

In [45]:
```
### Split the data set
from sklearn.model_selection import train_test_split
```

In [46]:
```
transformed_df.iloc[:,:-1]
```

Out[46]:

	Local_X(t-4)	Local_X(t-3)	Local_X(t-2)	Local_X(t-1)
4	51.214252	51.196441	51.178494	51.160446
5	51.196441	51.178494	51.160446	51.142331
6	51.178494	51.160446	51.142331	51.124182
7	51.160446	51.142331	51.124182	51.106019
8	51.142331	51.124182	51.106019	51.088021
...	...	...	...	...
510	31.668082	31.660184	31.653473	31.653089
511	31.660184	31.653473	31.653089	31.655713
512	31.653473	31.653089	31.655713	31.657416
513	31.653089	31.655713	31.657416	31.656984
514	31.655713	31.657416	31.656984	31.653203

511 rows × 4 columns

In [47]:
```
X_train, X_test, y_train, y_test = train_test_split(transformed_df.iloc[:,:-1],transfor
                                                    test_size=0.3, random_state=42)
print(X_train.shape,X_test.shape, y_train.shape, y_test.shape)
```

```
(357, 4) (154, 4) (357,) (154,)
```

In [48]:
```
### Standardize the data
train_mean = X_train.mean()
train_std = X_train.std()

X_train = (X_train - train_mean) / train_std
X_test = (X_test - train_mean) / train_std
```

In [49]:
```
print(X_train.describe())
X_train.shape
```

```
       Local_X(t-4)  Local_X(t-3)  Local_X(t-2)  Local_X(t-1)
count  3.570000e+02  3.570000e+02  3.570000e+02  3.570000e+02
mean   1.475322e-15  1.174286e-15  2.308766e-15 -6.692437e-16
std    1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
min   -1.367345e+00 -1.361572e+00 -1.358854e+00 -1.354419e+00
25%   -1.166080e+00 -1.161489e+00 -1.156922e+00 -1.152961e+00
50%    2.488613e-01  2.166698e-01  1.828833e-01  1.472498e-01
75%    9.506863e-01  9.562820e-01  9.600661e-01  9.638601e-01
max    1.136656e+00  1.140431e+00  1.144156e+00  1.147849e+00
```

Out[49]: (357, 4)

Apply prediction model

In [50]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
#from sklearn.preprocessing import MinMaxScaler
#from sklearn.metrics import mean_squared_error
```

Vanilla LSTM

A Vanilla LSTM is an LSTM model that has a single hidden layer of LSTM units, and an output layer used to

In [51]:
```
n_steps = 4
n_features = 1
```

In [52]:
```
X_train = X_train.values
X_test = X_test.values
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], n_features))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], n_features))
```

In [53]:
```
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse', metrics=['mse'])

# For saving the best model during the whole training process.
#checkpointer = callbacks.ModelCheckpoint(filepath='BestModel.h5', monitor='val_loss',

#### Interrupt training if `val_loss` stops improving for over 10 epochs #######
stop_learn= tf.keras.callbacks.EarlyStopping(patience=10, monitor='val_loss')


# fit model
Monitor = model.fit(X_train, y_train, epochs=100,
                    callbacks=[stop_learn],
                    validation_data=(X_test, y_test), verbose=1)
```

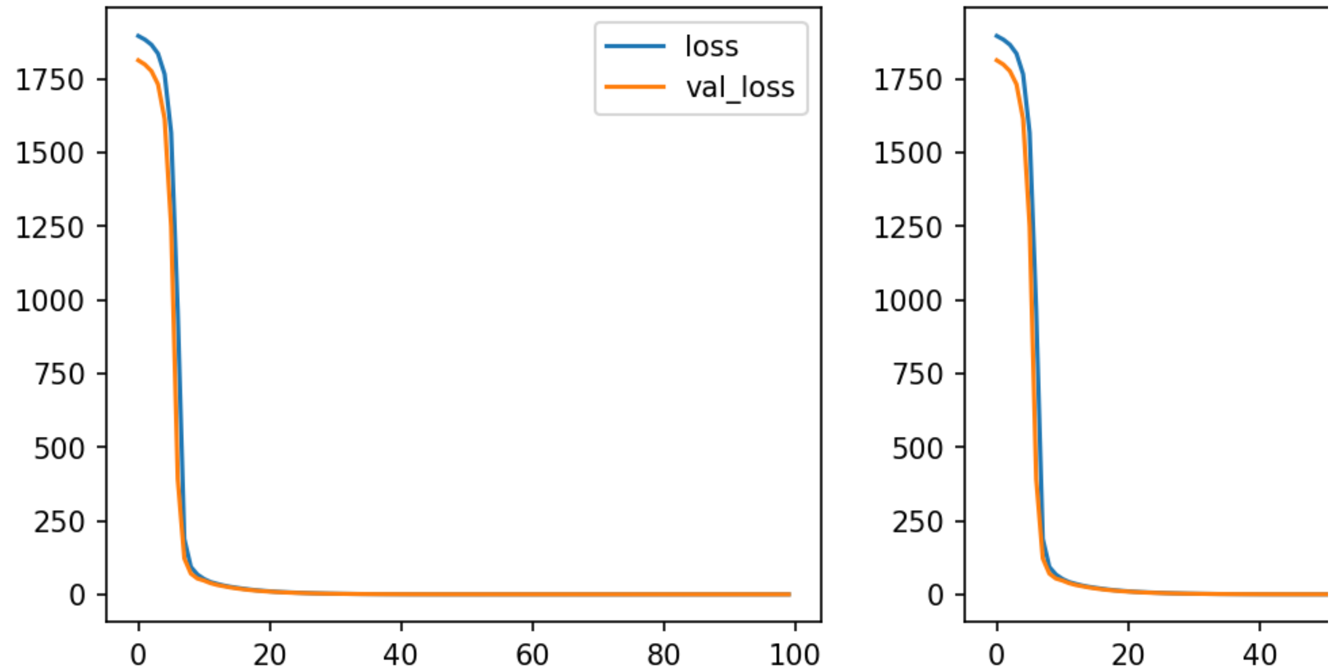
```
Epoch 30/100
357/357 [==============================] - 0s 235us/sample - loss: 2.8951 - mse: 2.8951
val_mse: 2.6058
Epoch 31/100
357/357 [==============================] - 0s 235us/sample - loss: 2.5373 - mse: 2.5373
val_mse: 2.3798
Epoch 32/100
357/357 [==============================] - 0s 235us/sample - loss: 2.2130 - mse: 2.2130
val_mse: 2.0367
Epoch 33/100
357/357 [==============================] - 0s 246us/sample - loss: 1.9300 - mse: 1.9300
val_mse: 1.8067
Epoch 34/100
357/357 [==============================] - 0s 258us/sample - loss: 1.6847 - mse: 1.6847
val_mse: 1.5203
Epoch 35/100
357/357 [==============================] - 0s 260us/sample - loss: 1.4381 - mse: 1.4381
val_mse: 1.3333
Epoch 36/100
357/357 [==============================] - 0s 249us/sample - loss: 1.2499 - mse: 1.2499
```

In [54]:
```
hist = pd.DataFrame(Monitor.history)
hist['epoch'] = Monitor.epoch
hist.tail()
```

Out[54]:

	loss	mse	val_loss	val_mse	epoch
95	0.212819	0.212819	0.235612	0.235612	95
96	0.209801	0.209801	0.232804	0.232804	96
97	0.212999	0.212999	0.242234	0.242234	97
98	0.209579	0.209579	0.226788	0.226788	98
99	0.207336	0.207336	0.235213	0.235213	99

In [55]:
```
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(10,4),dpi=150)
hist[['loss','val_loss']].plot(ax=axes[0])
hist[['mse','val_mse']].plot(ax=axes[1])
plt.show()
```



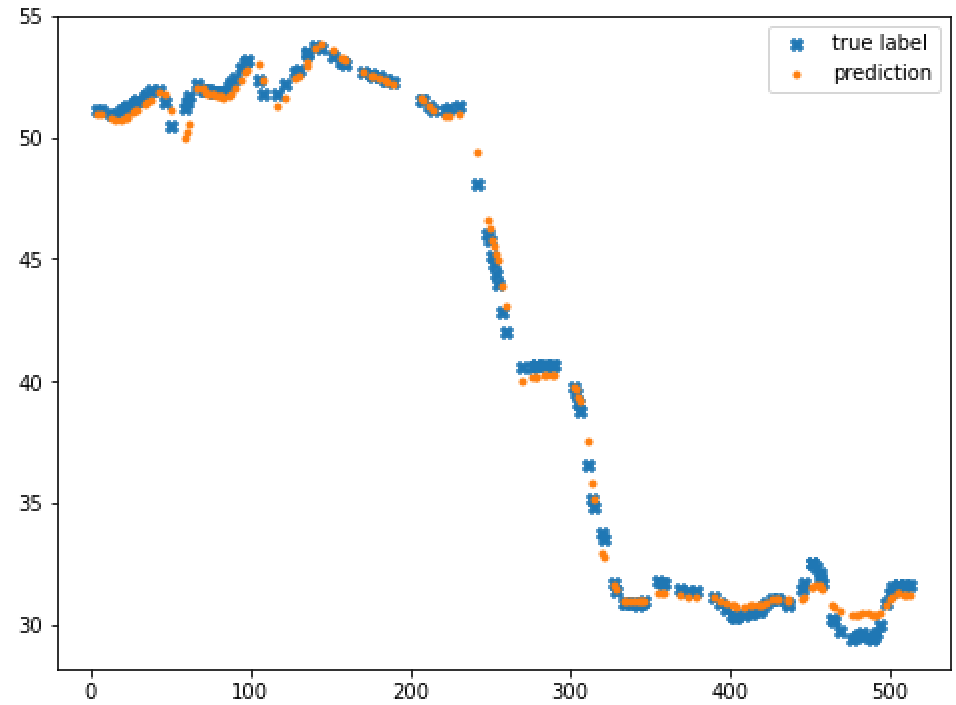
In [56]:
```
yhat = model.predict(X_test, verbose=1)
#print(yhat)
```

```
154/154 [==============================] - 0s 3ms/sample
```

In [57]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt

rms = sqrt(mean_squared_error(y_test, yhat))
rms
```

Out[57]: 0.4849871459835071

In [58]:
```
plt.scatter(y_test.index,y_test, label = "true label",marker = 'X', )
plt.scatter(y_test.index,yhat, label = "prediction",marker = '.')
plt.legend()
plt.show()
```



1 object and 1 target variable: Predict at time `t` and `t+1` (Multi-S Models)

Prepare the dataset

In [59]:
```
series_to_supervised(simple_df, n_in=4, n_out=2, dropnan=False).head()
```

Out[59]:

	Local_X(t-4)	Local_X(t-3)	Local_X(t-2)	Local_X(t-1)	Local_X(t)	Local_X(t+1)
0	NaN	NaN	NaN	NaN	51.214252	51.196441
1	NaN	NaN	NaN	51.214252	51.196441	51.178494
2	NaN	NaN	51.214252	51.196441	51.178494	51.160446
3	NaN	51.214252	51.196441	51.178494	51.160446	51.142331
4	51.214252	51.196441	51.178494	51.160446	51.142331	51.124182

In [60]:
```
transformed_df = series_to_supervised(simple_df, n_in=4, n_out=2, dropnan=True)
transformed_df.head()
```

Out[60]:

	Local_X(t-4)	Local_X(t-3)	Local_X(t-2)	Local_X(t-1)	Local_X(t)	Local_X(t+1)
4	51.214252	51.196441	51.178494	51.160446	51.142331	51.124182
5	51.196441	51.178494	51.160446	51.142331	51.124182	51.106019
6	51.178494	51.160446	51.142331	51.124182	51.106019	51.088021
7	51.160446	51.142331	51.124182	51.106019	51.088021	51.069916
8	51.142331	51.124182	51.106019	51.088021	51.069916	51.052000

In [61]:
```
transformed_df[['Local_X(t)','Local_X(t+1)']].values
```

Out[61]:
```
array([[51.142331  , 51.12418182],
       [51.12418182, 51.10601865],
       [51.10601865, 51.08802098],
       ...,
       [31.65741608, 31.65698368],
       [31.65698368, 31.6532028 ],
       [31.6532028 , 31.64486014]])
```

In [62]:
```
X_train, X_test, y_train, y_test = train_test_split(transformed_df.iloc[:,:-2],transfor
                                                    test_size=0.3, random_state=42)
print(X_train.shape,X_test.shape, y_train.shape, y_test.shape)
```

```
(357, 4) (153, 4) (357, 2) (153, 2)
```

In [63]:
```
### Standardize the data
train_mean = X_train.mean()
train_std = X_train.std()

X_train = (X_train - train_mean) / train_std
X_test = (X_test - train_mean) / train_std
```

In [64]:
```
print(X_train.describe())
X_train.shape
```

```
       Local_X(t-4)  Local_X(t-3)  Local_X(t-2)  Local_X(t-1)
count  3.570000e+02  3.570000e+02  3.570000e+02  3.570000e+02
mean   1.470346e-15  2.059977e-15  2.268960e-15 -6.468526e-16
std    1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
min   -1.372996e+00 -1.366166e+00 -1.360197e+00 -1.355126e+00
25%   -1.169918e+00 -1.164507e+00 -1.158710e+00 -1.153098e+00
50%    1.782954e-01  1.432778e-01  1.072958e-01  7.183136e-02
75%    9.580206e-01  9.626895e-01  9.666225e-01  9.709811e-01
max    1.145475e+00  1.149578e+00  1.153589e+00  1.157543e+00
```

Out[64]:
```
(357, 4)
```

Apply prediction model

In [65]:
```
n_steps = 4
n_features = 1
```

In [66]:
```
X_train = X_train.values
X_test = X_test.values
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], n_features))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], n_features))
```

In [67]:
```
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(2))
model.compile(optimizer='adam', loss='mse', metrics=['mse'])

# For saving the best model during the whole training process.
#checkpointer = callbacks.ModelCheckpoint(filepath='BestModel.h5', monitor='val_loss',

#### Interrupt training if `val_loss` stops improving for over 10 epochs #######
stop_learn= tf.keras.callbacks.EarlyStopping(patience=10, monitor='val_loss')


# fit model
Monitor = model.fit(X_train, y_train, epochs=100,
                    callbacks=[stop_learn],
                    validation_data=(X_test, y_test), verbose=1)
```

```
Train on 357 samples, validate on 153 samples
Epoch 1/100
357/357 [==============================] - 2s 5ms/sample - loss: 1892.6820 - mse: 1892.
5.7256 - val_mse: 1835.7257
Epoch 2/100
357/357 [==============================] - 0s 238us/sample - loss: 1884.5107 - mse: 188
27.0042 - val_mse: 1827.0043
Epoch 3/100
357/357 [==============================] - 0s 288us/sample - loss: 1875.1749 - mse: 187
15.6651 - val_mse: 1815.6650
Epoch 4/100
357/357 [==============================] - 0s 219us/sample - loss: 1861.3159 - mse: 186
95.5646 - val_mse: 1795.5647
Epoch 5/100
357/357 [==============================] - 0s 258us/sample - loss: 1833.8622 - mse: 183
50.0271 - val_mse: 1750.0270
Epoch 6/100
357/357 [==============================] - 0s 269us/sample - loss: 1763.3380 - mse: 176
19.1072 - val_mse: 1619.1072
```

In [68]:
```
yhat = model.predict(X_test, verbose=1)
print(yhat[:5])
```

```
153/153 [==============================] - 0s 2ms/sample
[[30.451542 30.433975]
 [31.449556 31.375954]
 [30.330221 30.31709 ]
 [30.84705  30.795261]
 [31.012201 30.952255]]
```

In [69]:
```
yhat[:,0]
```

Out[69]:
```
array([30.451542, 31.449556, 30.330221, 30.84705 , 31.012201, 50.07107 ,
       31.561111, 51.709846, 31.108248, 31.01705 , 30.725054, 50.94543 ,
       51.63701 , 45.205605, 52.48072 , 52.636215, 31.062275, 40.055927,
       30.842058, 51.776825, 31.103573, 50.84607 , 44.883106, 30.697992,
       30.956812, 31.429634, 52.5047  , 51.430065, 31.417961, 52.37558 ,
       31.016317, 52.861446, 31.06396 , 51.761257, 51.18972 , 30.830133,
       31.006628, 52.928963, 50.908325, 53.150864, 50.99476 , 30.67101 ,
       52.027397, 50.9787  , 30.95997 , 30.827019, 51.723495, 40.118008,
       51.24432 , 30.368048, 52.292885, 51.696655, 39.85156 , 31.378326,
       30.828548, 30.926392, 51.537987, 51.67636 , 30.756475, 52.63371 ,
       51.882347, 51.016136, 51.84381 , 50.802673, 51.05282 , 51.07395 ,
       31.38052 , 51.68508 , 45.770897, 51.76172 , 30.324259, 30.860281,
       50.88908 , 40.104992, 52.402206, 31.037542, 45.50051 , 51.861073,
       30.676344, 51.107216, 39.11026 , 51.936466, 31.11601 , 52.332577,
       40.11299 , 53.51126 , 53.087997, 31.300829, 52.26713 , 51.81888 ,
       46.65056 , 39.590576, 30.72103 , 31.135645, 38.907677, 30.662043,
       52.31929 , 31.007666, 53.006123, 30.824387, 30.74117 , 52.415924,
       35.492786, 31.093414, 30.34043 , 53.614845, 30.985943, 40.019554,
       30.87068 , 31.158268, 52.0187  , 51.72918 , 31.361637, 50.784332,
       51.635796, 40.046646, 30.814016, 50.93708 , 31.311607, 30.848604,
       50.32016 , 30.863628, 31.130419, 49.410004, 30.667803, 37.18395 ,
       30.41274 , 30.915312, 30.386095, 51.20677 , 31.050913, 51.775482,
       53.737354, 30.798483, 32.632153, 52.4695  , 30.856878, 51.469612,
       51.304443, 53.208176, 51.127754, 30.947506, 50.611404, 39.895718,
       51.162598, 50.84495 , 52.189796, 52.02947 , 30.640272, 30.746332,
       52.727676, 30.909714, 31.155054], dtype=float32)
```

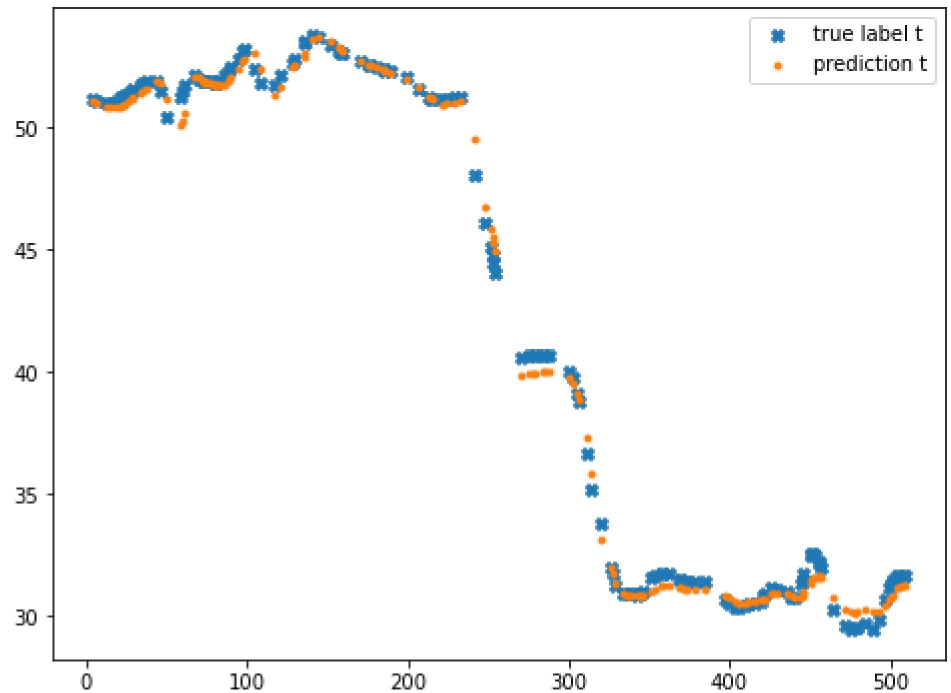
In [70]:
```
y_test.iloc[:,1]
```

Out[70]:
```
484    29.606848
453    32.351807
479    29.598867
438    30.641876
372    31.360688
         ...
406    30.349110
400    30.411604
98     53.220900
500    31.473545
358    31.718217
Name: Local_X(t+1), Length: 153, dtype: float64
```

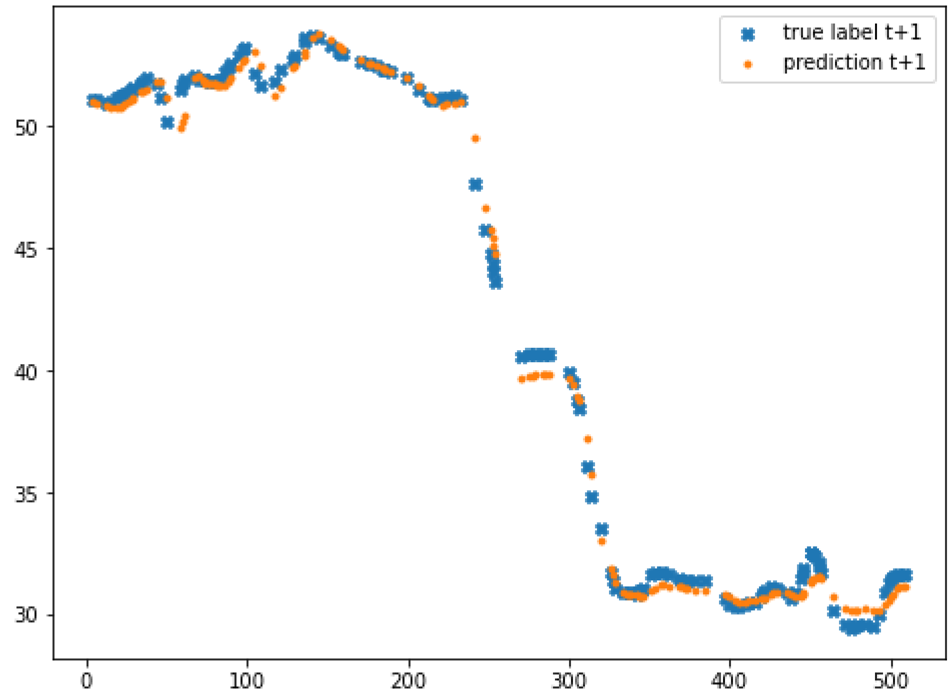
In [71]:
```
rms1 = sqrt(mean_squared_error(y_test.iloc[:,0], yhat[:,0]))
rms1
```

Out[71]:
```
0.47934308613682397
```

In [57]:
```
plt.scatter(y_test.index,y_test.iloc[:,0], label = "true label t",marker = 'X', )
plt.scatter(y_test.index,yhat[:,0], label = "prediction t",marker = '.')
plt.legend()
plt.show()
```



In [56]:
```
plt.scatter(y_test.index,y_test.iloc[:,1], label = "true label t+1",marker = 'X', )
plt.scatter(y_test.index,yhat[:,1], label = "prediction t+1",marker = '.')
plt.legend()
plt.show()
```

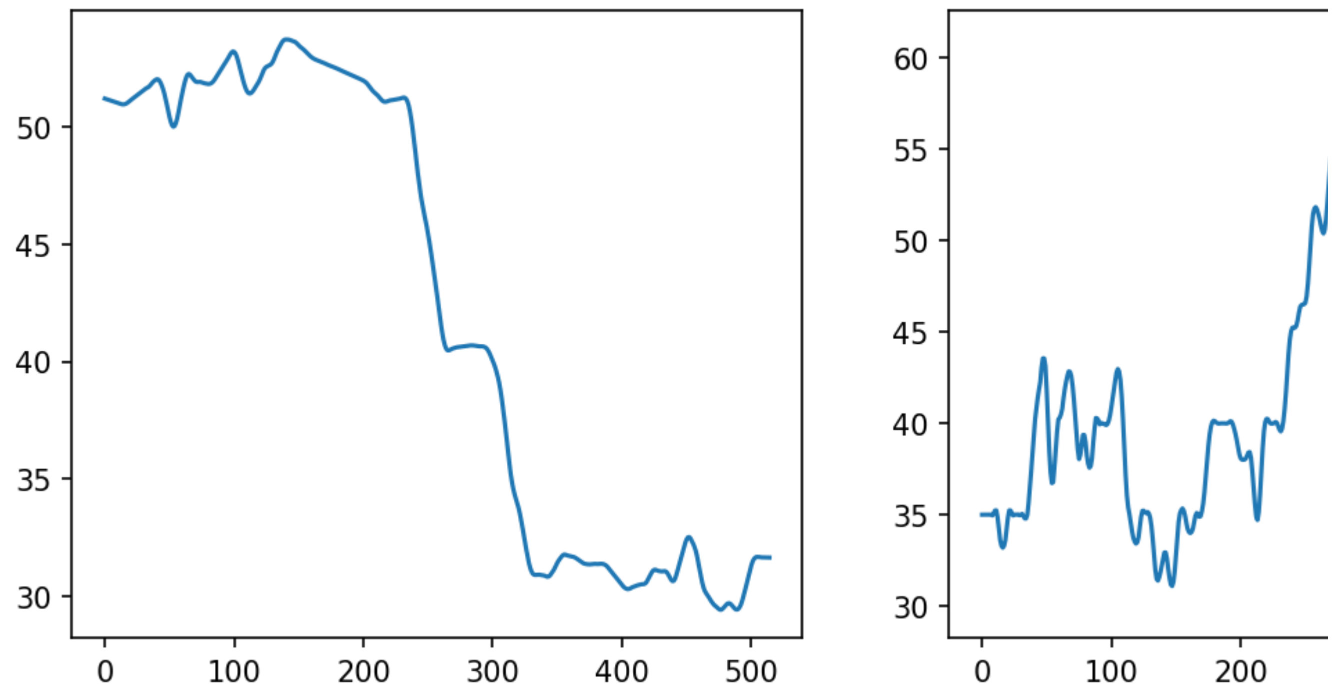


1 object, 2 target variables, Predict at time `t` (Multiple Parallel S

2 target variables are Local_X and v_Vel

Prepare the data set

In [75]:
```
simple_df = new_df[new_df.Vehicle_ID == new_veh_list[0]].copy()
simple_df = simple_df[['Frame_ID','Local_X','v_Vel']]
simple_df.set_index('Frame_ID', inplace = True)
simple_df.sort_index(inplace=True)
simple_df.reset_index(drop=True, inplace=True)
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(10,4),dpi=150)
simple_df['Local_X'].plot(ax=axes[0])
simple_df['v_Vel'].plot(ax=axes[1])
plt.show()
simple_df.head()
```



Out[75]:

	Local_X	v_Vel
0	51.214252	35.000000
1	51.196441	34.998425
2	51.178494	35.000227
3	51.160446	35.001526
4	51.142331	35.002322

In [76]:
```
series_to_supervised(simple_df, n_in=4, n_out=2, dropnan=False).head()
```

Out[76]:

	Local_X(t-4)	v_Vel(t-4)	Local_X(t-3)	v_Vel(t-3)	Local_X(t-2)	v_Vel(t-2)	Local_X(t-1)	v_Vel(t-1)	Local_X(t)	v_
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	51.214252	35.0(
1	NaN	NaN	NaN	NaN	NaN	NaN	51.214252	35.000000	51.196441	34.9!
2	NaN	NaN	NaN	NaN	51.214252	35.000000	51.196441	34.998425	51.178494	35.0(
3	NaN	NaN	51.214252	35.000000	51.196441	34.998425	51.178494	35.000227	51.160446	35.0(
4	51.214252	35.0	51.196441	34.998425	51.178494	35.000227	51.160446	35.001526	51.142331	35.0(

In [79]:
```
transformed_df = series_to_supervised(simple_df, n_in=4, n_out=1, dropnan=True)
transformed_df.head()
```

Out[79]:

	Local_X(t-4)	v_Vel(t-4)	Local_X(t-3)	v_Vel(t-3)	Local_X(t-2)	v_Vel(t-2)	Local_X(t-1)	v_Vel(t-1)	Local_)
4	51.214252	35.000000	51.196441	34.998425	51.178494	35.000227	51.160446	35.001526	51.142:
5	51.196441	34.998425	51.178494	35.000227	51.160446	35.001526	51.142331	35.002322	51.1241
6	51.178494	35.000227	51.160446	35.001526	51.142331	35.002322	51.124182	35.002615	51.106(
7	51.160446	35.001526	51.142331	35.002322	51.124182	35.002615	51.106019	35.000914	51.088(
8	51.142331	35.002322	51.124182	35.002615	51.106019	35.000914	51.088021	35.001255	51.069!

In [80]:
```
X_train, X_test, y_train, y_test = train_test_split(transformed_df.iloc[:,:-2],transfor
                                                     test_size=0.3, random_state=42)
print(X_train.shape,X_test.shape, y_train.shape, y_test.shape)
```

```
(357, 8) (154, 8) (357, 2) (154, 2)
```

In [81]:
```
### Standardize the data
train_mean = X_train.mean()
train_std = X_train.std()

X_train = (X_train - train_mean) / train_std
X_test = (X_test - train_mean) / train_std
X_train.describe()
```

Apply prediction model

In [84]:
```
n_steps = 4
n_features = 1
X_train = X_train.values
X_test = X_test.values
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], n_features))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], n_features))
```

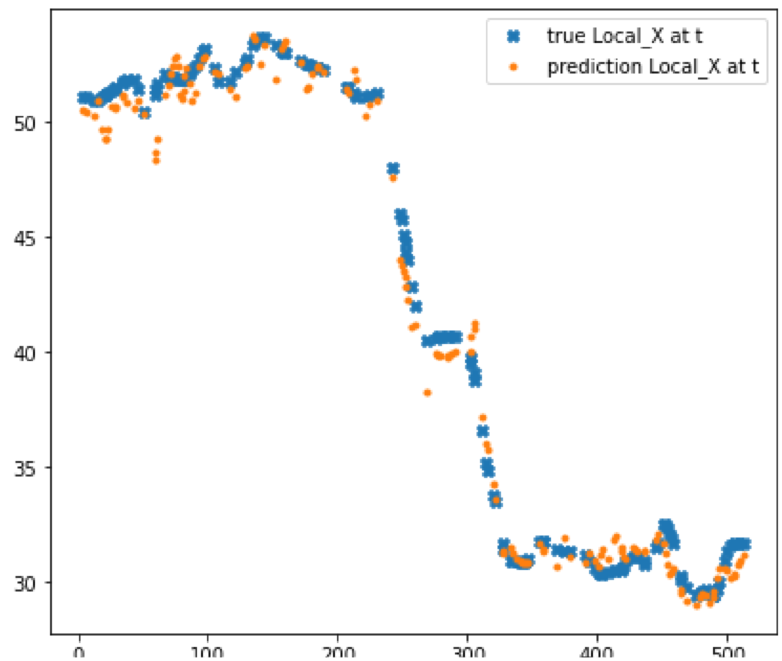
In [86]:
```
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps*2, n_features)))
model.add(Dense(2))
model.compile(optimizer='adam', loss='mse', metrics=['mse'])

# For saving the best model during the whole training process.
#checkpointer = callbacks.ModelCheckpoint(filepath='BestModel.h5', monitor='val_loss',

#### Interrupt training if `val_loss` stops improving for over 10 epochs #######
stop_learn= tf.keras.callbacks.EarlyStopping(patience=10, monitor='val_loss')


# fit model
Monitor = model.fit(X_train, y_train, epochs=100,
                    callbacks=[stop_learn],
                    validation_data=(X_test, y_test), verbose=1)
```

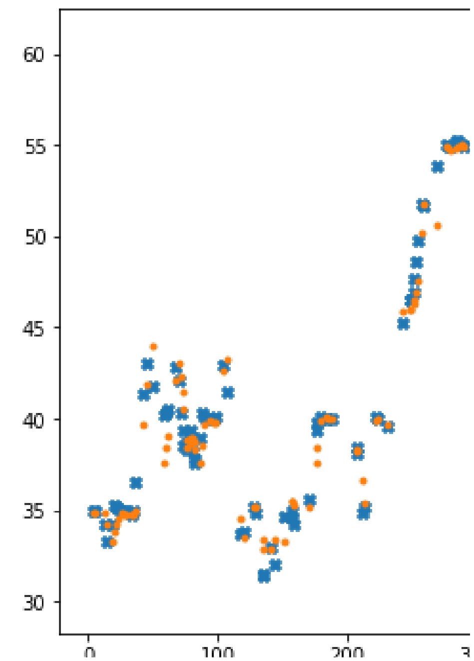
```
Train on 357 samples, validate on 154 samples
Epoch 1/100
357/357 [==============================] - 2s 5ms/sample - loss: 1799.9222 - mse: 1799.
9.3664 - val_mse: 1749.3663
Epoch 2/100
357/357 [==============================] - 0s 314us/sample - loss: 1788.8126 - mse: 178
35.8922 - val_mse: 1735.8922
Epoch 3/100
357/357 [==============================] - 0s 291us/sample - loss: 1769.7496 - mse: 176
01.9643 - val_mse: 1701.9642
Epoch 4/100
357/357 [==============================] - 0s 314us/sample - loss: 1692.7764 - mse: 169
66.9931 - val_mse: 1466.9930
Epoch 5/100
357/357 [==============================] - 0s 336us/sample - loss: 1041.0888 - mse: 104
4.9053 - val_mse: 544.9053
Epoch 6/100
357/357 [==============================] - 0s 314us/sample - loss: 219.0419 - mse: 219.
6629 - val_mse: 258.6629
```

In [87]:
```
yhat = model.predict(X_test, verbose=1)
print(yhat[:5])
```

```
154/154 [==============================] - 0s 3ms/sample
[[52.40429  35.17813 ]
 [50.92724  38.53888 ]
 [31.337734 40.129745]
 [41.184547 51.725895]
 [52.149414 42.270027]]
```

In [95]:
```
plt.figure(figsize=(15,6))
plt.subplot(1,2,1)
plt.scatter(y_test.index,y_test.iloc[:,0], label = "true Local_X at t",marker = 'X', )
plt.scatter(y_test.index,yhat[:,0], label = "prediction Local_X at t",marker = '.')
plt.legend()
plt.subplot(1,2,2)
plt.scatter(y_test.index,y_test.iloc[:,1], label = "true speed at t",marker = 'X', )
plt.scatter(y_test.index,yhat[:,1], label = "prediction speed at t",marker = '.')
plt.legend()
plt.show()
```

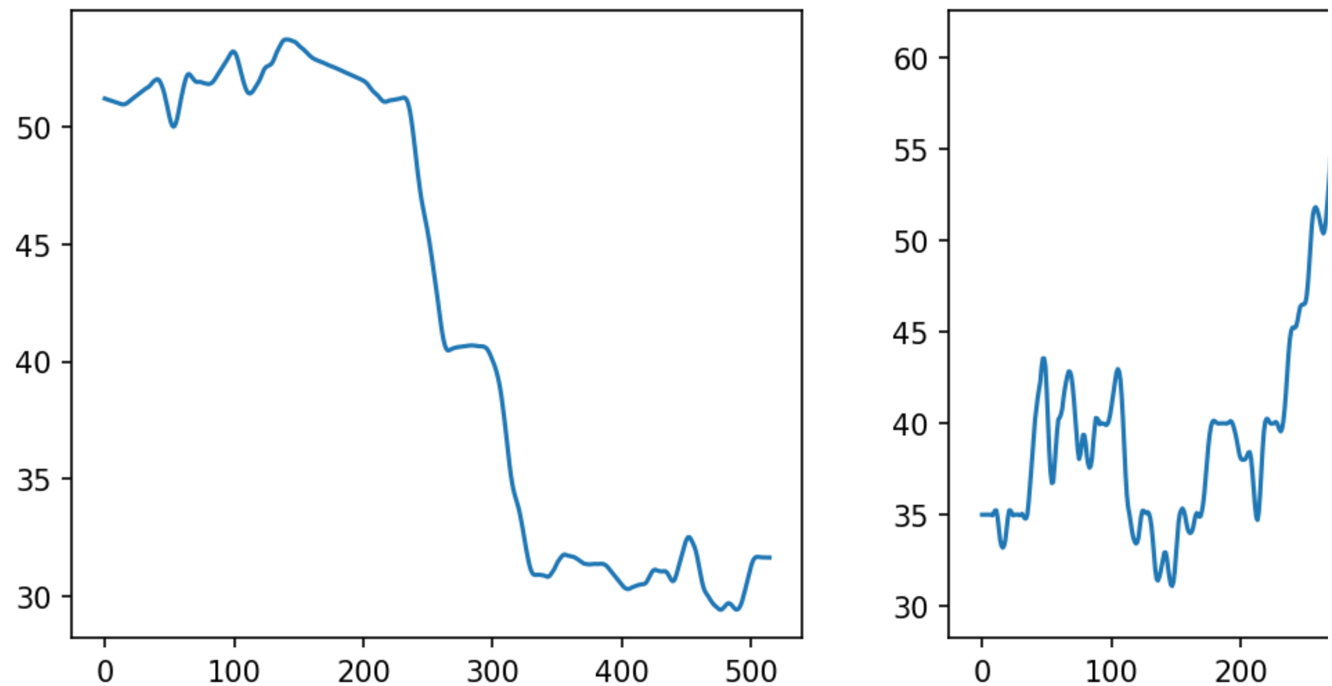




1 object, 2 target variables, predict at time `t`, multiple input seri

Data preparation

In [103]:
```
simple_df = new_df[new_df.Vehicle_ID == new_veh_list[0]].copy()
simple_df = simple_df[['Frame_ID','Local_X','Local_Y','v_Vel']]
simple_df.set_index('Frame_ID', inplace = True)
simple_df.sort_index(inplace=True)
simple_df.reset_index(drop=True, inplace=True)
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(10,4),dpi=150)
simple_df['Local_X'].plot(ax=axes[0])
simple_df['v_Vel'].plot(ax=axes[1])
plt.show()
simple_df.head()
```



Out[103]:

	Local_X	Local_Y	v_Vel
0	51.214252	39.647098	35.000000
1	51.196441	43.146895	34.998425
2	51.178494	46.646872	35.000227
3	51.160446	50.146978	35.001526
4	51.142331	53.647163	35.002322

In [102]:
```
transformed_df = series_to_supervised(simple_df, n_in=4, n_out=1, dropnan=True)
transformed_df.head()
```

Out[102]:

	Local_X(t-4)	Local_Y(t-4)	v_Vel(t-4)	Local_X(t-3)	Local_Y(t-3)	v_Vel(t-3)	Local_X(t-2)	Local_Y(t-2)	v_Vel(t-2)	L
4	51.214252	39.647098	35.000000	51.196441	43.146895	34.998425	51.178494	46.646872	35.000227	5
5	51.196441	43.146895	34.998425	51.178494	46.646872	35.000227	51.160446	50.146978	35.001526	5
6	51.178494	46.646872	35.000227	51.160446	50.146978	35.001526	51.142331	53.647163	35.002322	5
7	51.160446	50.146978	35.001526	51.142331	53.647163	35.002322	51.124182	57.147378	35.002615	5
8	51.142331	53.647163	35.002322	51.124182	57.147378	35.002615	51.106019	60.647422	35.000914	5

In [108]:
```
transformed_df.drop(columns=['Local_Y(t)'],inplace=True)
```

In [109]:
```
X_train, X_test, y_train, y_test = train_test_split(transformed_df.iloc[:,:-2],transfor
                                                    test_size=0.3, random_state=42)
print(X_train.shape,X_test.shape, y_train.shape, y_test.shape)
```

```
(357, 12) (154, 12) (357, 2) (154, 2)
```

In [110]:
```
### Standardize the data
train_mean = X_train.mean()
train_std = X_train.std()

X_train = (X_train - train_mean) / train_std
X_test = (X_test - train_mean) / train_std
X_train.describe()
```

Out[110]:

	Local_X(t-4)	Local_Y(t-4)	v_Vel(t-4)	Local_X(t-3)	Local_Y(t-3)	v_Vel(t-3)	Local_
count	3.570000e+02	3.570000e+02	3.570000e+02	3.570000e+02	3.570000e+02	3.570000e+02	3.57000(
mean	1.475322e-15	3.284021e-16	-2.639656e-15	1.174286e-15	-1.243947e-17	-3.288997e-15	2.30876
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.00000(
min	-1.367345e+00	-1.667882e+00	-1.626871e+00	-1.361572e+00	-1.668580e+00	-1.624022e+00	-1.358854
25%	-1.166080e+00	-8.356201e-01	-8.437423e-01	-1.161489e+00	-8.370283e-01	-8.317377e-01	-1.156922
50%	2.488613e-01	-1.055727e-01	-9.665806e-02	2.166698e-01	-1.044232e-01	-9.495026e-02	1.82883
75%	9.506863e-01	8.979312e-01	3.542775e-01	9.562820e-01	8.976975e-01	3.542584e-01	9.60066
max	1.136656e+00	1.739317e+00	3.055842e+00	1.140431e+00	1.738859e+00	2.991404e+00	1.14415(

In [111]:
```
X_train = X_train.values
X_test = X_test.values
```

In [112]:
```
n_steps = 4
n_features = 1
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], n_features))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], n_features))
```

In [113]:
```
X_train.shape
```

Out[113]:
```
(357, 12, 1)
```

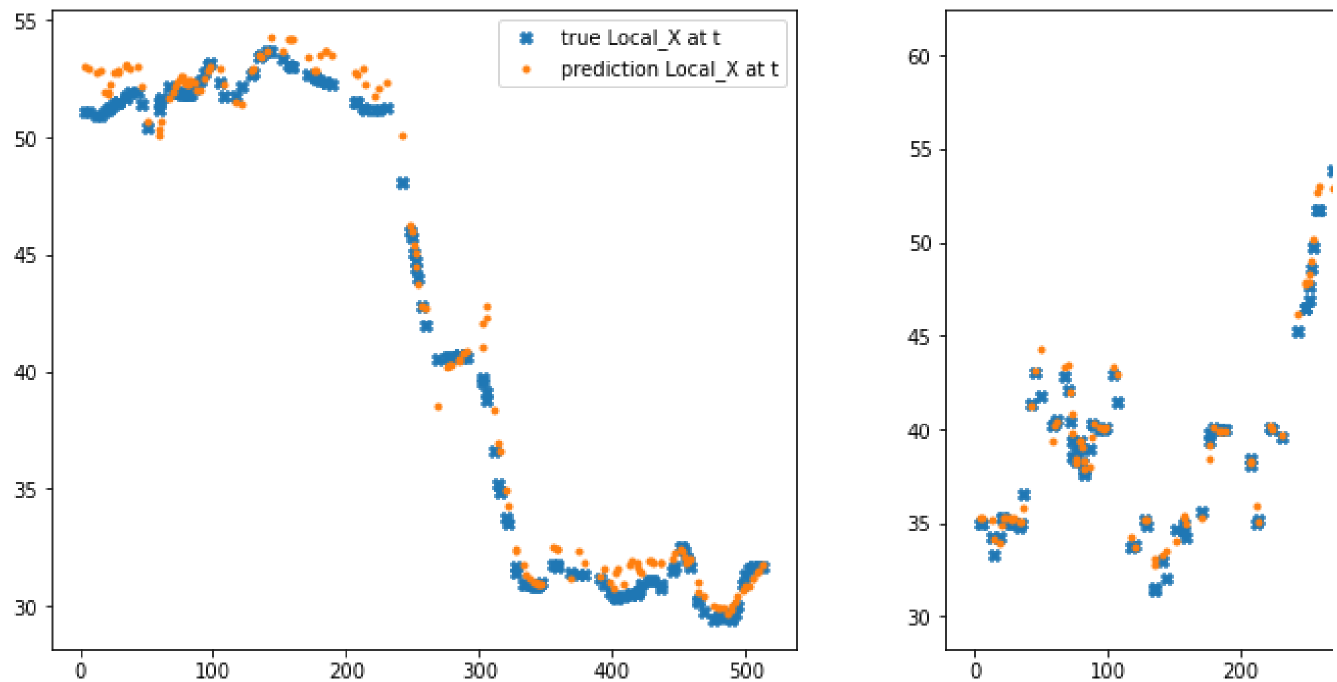
In [115]:
```
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps*3, n_features)))
model.add(Dense(2))
model.compile(optimizer='adam', loss='mse', metrics=['mse'])

# For saving the best model during the whole training process.
#checkpointer = callbacks.ModelCheckpoint(filepath='BestModel.h5', monitor='val_loss',

#### Interrupt training if `val_loss` stops improving for over 10 epochs #######
stop_learn= tf.keras.callbacks.EarlyStopping(patience=10, monitor='val_loss')


# fit model
Monitor = model.fit(X_train, y_train, epochs=100,
                    callbacks=[stop_learn],
                    validation_data=(X_test, y_test), verbose=1)
```

```
Train on 357 samples, validate on 154 samples
Epoch 1/100
357/357 [==============================] - 2s 7ms/sample - loss: 1797.1003 - mse: 1797.
4.3626 - val_mse: 1744.3627
Epoch 2/100
357/357 [==============================] - 0s 442us/sample - loss: 1780.2005 - mse: 178
16.9481 - val_mse: 1716.9480
Epoch 3/100
357/357 [==============================] - 0s 549us/sample - loss: 1717.4922 - mse: 171
73.6404 - val_mse: 1473.6404
Epoch 4/100
357/357 [==============================] - 0s 445us/sample - loss: 910.8924 - mse: 910.
4942 - val_mse: 512.4943
Epoch 5/100
357/357 [==============================] - 0s 448us/sample - loss: 600.4077 - mse: 600.
0654 - val_mse: 667.0654
Epoch 6/100
357/357 [==============================] - 0s 434us/sample - loss: 371.5620 - mse: 371.
1720 - val_mse: 321.1720
```

In [116]:
```
yhat = model.predict(X_test, verbose=1)
print(yhat[:5])
```

```
154/154 [==============================] - 0s 2ms/sample
[[52.861416 35.15887 ]
 [52.02561  39.586807]
 [31.811306 40.620678]
 [42.695164 53.04565 ]
 [52.164997 41.95745 ]]
```

In [117]:
```
plt.figure(figsize=(15,6))
plt.subplot(1,2,1)
plt.scatter(y_test.index,y_test.iloc[:,0], label = "true Local_X at t",marker = 'X', )
plt.scatter(y_test.index,yhat[:,0], label = "prediction Local_X at t",marker = '.')
plt.legend()
plt.subplot(1,2,2)
plt.scatter(y_test.index,y_test.iloc[:,1], label = "true speed at t",marker = 'X', )
plt.scatter(y_test.index,yhat[:,1], label = "prediction speed at t",marker = '.')
plt.legend()
plt.show()
```



In [119]:
```

```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-119-46a8a2031c9c> in <module>
----> 1 import tensorflow_datasets as tfds

ModuleNotFoundError: No module named 'tensorflow_datasets'
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: