Trajectory Correction Research – Research Project

Student: Ethan Higa, Mentor: John Tadrous

(All the code for and iterations of this project can be found here: https://github.com/eleehiga/Trajectory_Correction_Research)

Abstract

In this report, we work into implementing the Automatic Trajectory Repair (ATR) and Long Short Term Memory (LSTM) algorithms, in the programming language Python to correct a given trajectory dataset of an object. This work will eventually culminate into combining these two algorithms in order to create a novel solution towards correcting trajectory data. Through this work, snippets of the Python version of these algorithms and results from test data are shown.

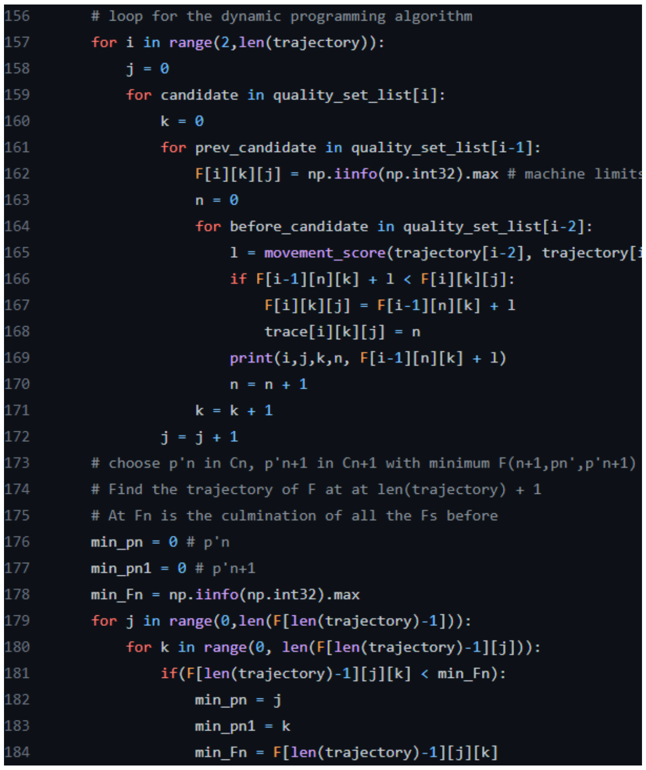
Summary

Geographic information with the position about an object whether it be a person, an airplane, or an automobile is normally found by triangulation or internal sensors, however, sometimes there can be errors in the position measurements and those can be reflected in the trajectory dataset as viewed from the whole. An example of this problem in real life could be if a shipping company collects trajectory data from all their boats. Say they want to have a presentation of how the boats moved in one day of work for evaluating how they could improve their shipping throughout the day. When they present the raw trajectory data they do not want random jagged movements from the boat, but rather, they want smooth movements from the boat as that would be reflective of how the boats actually moved. To solve this problem, solutions are investigated such as the ATR algorithm to repair noisy points and LSTM neural network algorithm to fill in gaps in data. These solutions are more effective than the Kalman Filter algorithm because that algorithm changes unnecessary points and does not learn from the trajectory.

A. System Model

The data used for this project will be a set of (x,y) position data points throughout time. For testing, an artificial trajectory dataset will be used that will be created to have the problem noise and gaps in it. Eventually though, real data will be used that has these types of error, and possibly more types in it.

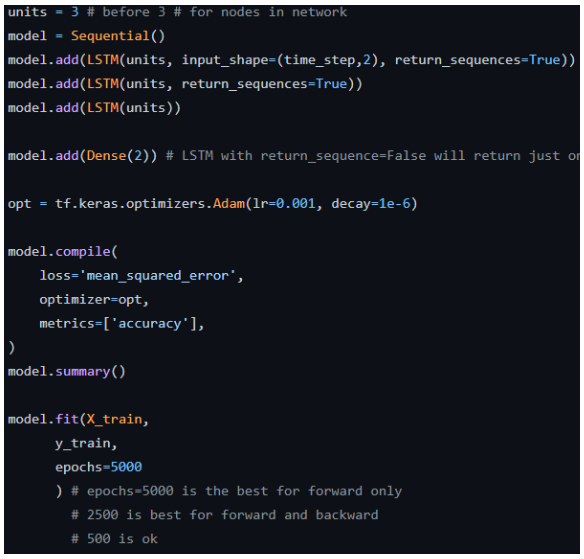
The automatic trajectory repair algorithm assumes that most of the given trajectory points are correct but only a few are incorrect. With that, each trajectory point has a radius around it which contains candidate points the corrected point could be instead. However, because there are a lot of points for each candidate set, an algorithm for quality selection is used to cut down the candidate set. With this new candidate set, each point from one candidate set to another will have a movement tendency score calculated with the repair distance from the given point to the selected candidate point, travel distance with one point to the next one, and speed with the change in the points before and after. From that, dynamic programming, as seen in Figure 1 for this project, is used to select the points in the candidate sets that minimize the overall movement tendency score.



```python
    # loop for the dynamic programming algorithm
    for i in range(2,len(trajectory)):
        j = 0
        for candidate in quality_set_list[i]:
            k = 0
            for prev_candidate in quality_set_list[i-1]:
                F[i][k][j] = np.iinfo(np.int32).max # machine limits
                n = 0
                for before_candidate in quality_set_list[i-2]:
                    l = movement_score(trajectory[i-2], trajectory[i
                    if F[i-1][n][k] + l < F[i][k][j]:
                        F[i][k][j] = F[i-1][n][k] + l
                        trace[i][k][j] = n
                    print(i,j,k,n, F[i-1][n][k] + l)
                    n = n + 1
                k = k + 1
            j = j + 1
    # choose p'n in Cn, p'n+1 in Cn+1 with minimum F(n+1,pn',p'n+1)
    # Find the trajectory of F at at len(trajectory) + 1
    # At Fn is the culmination of all the Fs before
    min_pn = 0 # p'n
    min_pn1 = 0 # p'n+1
    min_Fn = np.iinfo(np.int32).max
    for j in range(0,len(F[len(trajectory)-1])):
        for k in range(0, len(F[len(trajectory)-1][j])):
            if(F[len(trajectory)-1][j][k] < min_Fn):
                min_pn = j
                min_pn1 = k
                min_Fn = F[len(trajectory)-1][j][k]
```

Figure 1 ATR Dynamic Programming Algorithm in Python

The LSTM neural network algorithm will take in a sequence of coordinate points and output the predicted next point. This algorithm is trained on a series of points, each with its length as a specified time step. After training, the gap is filled with the previous points used before the gap to predict the points missing in the gap. However, because the forward direction of the gap filling does not work, a more optimal solution is one that does a weighted sum of the forward and backward pass LSTM predictions, as shown by Equation 1.



```
units = 3 # before 3 # for nodes in network
model = Sequential()
model.add(LSTM(units, input_shape=(time_step,2), return_sequences=True))
model.add(LSTM(units, return_sequences=True))
model.add(LSTM(units))

model.add(Dense(2)) # LSTM with return_sequence=False will return just o

opt = tf.keras.optimizers.Adam(lr=0.001, decay=1e-6)

model.compile(
    loss='mean_squared_error',
    optimizer=opt,
    metrics=['accuracy'],
)
model.summary()

model.fit(X_train,
      y_train,
      epochs=5000
      ) # epochs=5000 is the best for forward only
        # 2500 is best for forward and backward
        # 500 is ok
```

Figure 2 LSTM Implemented with the Tensorflow library

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_f \\ y_f \end{bmatrix} * \left(1 - \frac{i}{N}\right) + \begin{bmatrix} x_b \\ y_b \end{bmatrix} * \left(\frac{i}{N}\right)$$
$$f = forward, b = backward, i = 0\ for\ start\ of\ gap\ and\ N\ for\ end, N = size\ of\ the\ gap$$

Equation 1: forward and backward prediction

B. Simulation Results

The ATR implementation in Python is a work in progress with it not predicting very accurately, see Figure 3. More work needs to be done with how to speed up this program as it is quite slow, as this algorithm is O(n^3), and figuring out more efficient methods to debug the dynamic programming component of this approach.

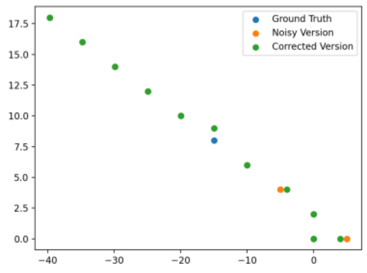


Figure 3 ATR program results

The LSTM implementation in Python uses a broken path as for example shown by Figure 4.

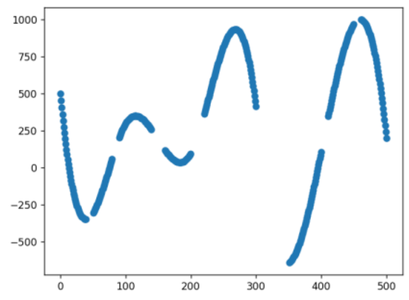


Figure 4 Neural Network Reconstruction Broken Path

The forward pass only implementation, as proposed by the original authors of this idea, is not accurate, see Figure 5.

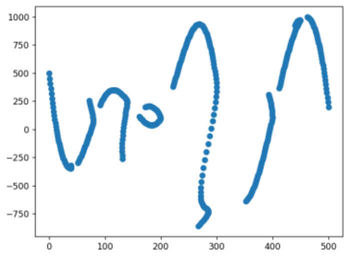


Figure 5 Neural Network Reconstruction Forward Pass Results

The forward and backward pass implementation does produce a much more accurate result, see Figure 6. Therefore, the forward only implementation by the authors is not accurate because they test their algorithm on trajectories that have much less curvature than curves like Figure 4.

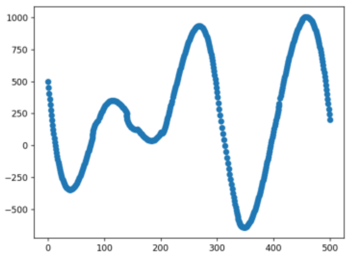


Figure 6 Neural Network Reconstruction Forward and Backward Pass Results

As seen by Figure 6 this algorithm is not completely accurate and this is probably due to how the neural network optimizes to a local minimum and not to a global minimum, that would be the most accurate predictor, and how the forward and backward only passes in general do not connect the end of the prediction to the start of the points on the other side of the gap.

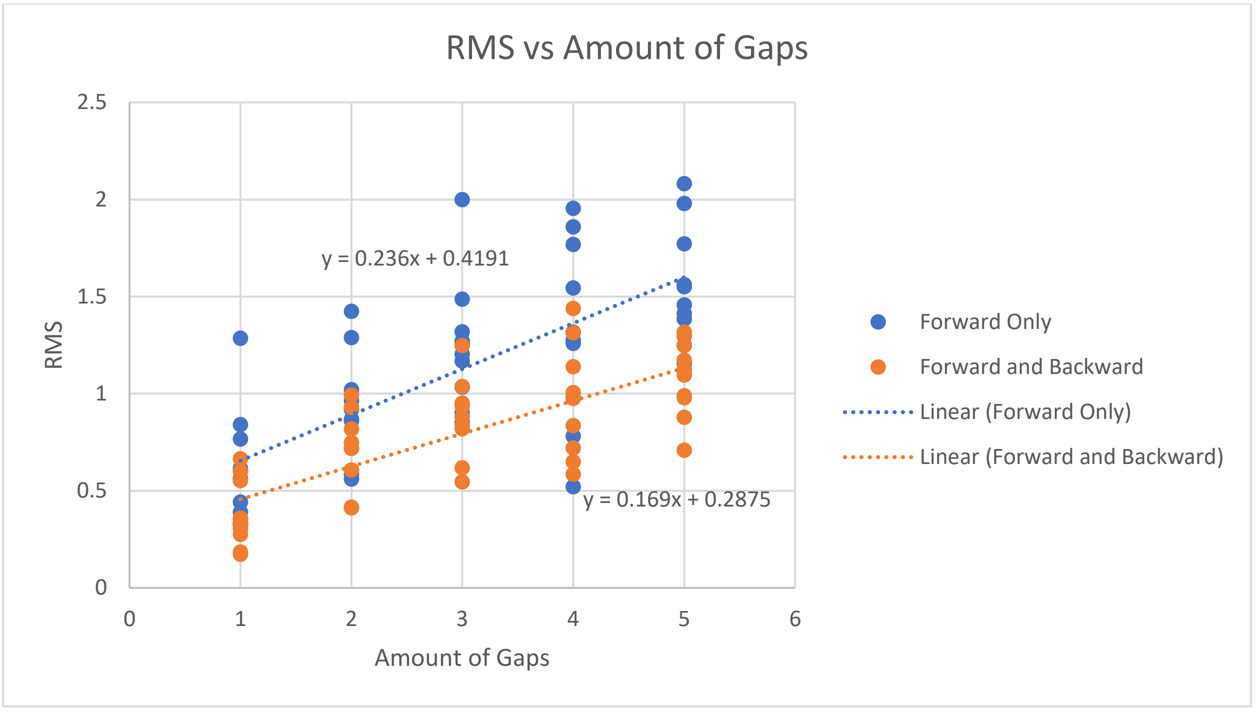


Figure 7 Graph of the RMS vs Amount of Gaps from 50 runs of the reconstruction program

For the Figure 7 the forward only and forward and backward reconstruction algorithms were run ten times for each amount of gaps so for 5 gaps that yields to 50 runs in total (the run time of all of this was around twelve hours). This proposed algorithm of forward and backward passes does perform better than the previously proposed algorithm of a forward pass only. This is as in that graph the forward and backward pass algorithm will overall have less error than forward only and will not have as much a increase of RMS per gap.

Therefore, other algorithms may be tested in the future like reinforcement learning. That algorithm is projected to further improve efficiency as it allows for an intelligent agent to be trained, much like how a person could be trained for drawing. This intelligent agent then could be able to correct and reconstruct trajectory points. For example, in the training the agent could be heavily penalized when its trajectory prediction does not end up where the start of the points at the other side of the gap are.

The Research Experience: This experience has taught me how to do artificial intelligence research with incorporating ideas from different papers and our own along with technologies of Python with different libraries. Dr. Tadrous provided me great guidance for what paths this research should go on, help to understand complex parts of the problem and solutions in papers, and providing great ideas on how to solve aspects of this problem. Aside from that, Dr. Tadrous also gave me great career and education advice that definitely helped me out.

Acknowledgment: Special thanks to Naval Undersea Warfare Center Pacific Detachment as in the summer of 2021 I worked on Kalman Filter, Low Pass Filter, Spearman Coefficient, Dynamic Mode Decomposition, and Expectation Maximization algorithm solutions to this problem (note: these implementations also did not lead to accurate results)

References

P. Zhao, A. Zhang, C. Zhang, J. Li, Q. Zhao and W. Rao, "ATR: Automatic Trajectory Repairing with Movement Tendencies," 2019.

M. Liang, R. Liu, Q. Zhong, J. Liu, and J. Zhang, “Neural Network-Based Automatic Reconstruction of Missing Vessel Trajectory Data,” 2019.