Trajectory prediction using RNNs for a Dubins UAV

Oleg Russu, Rohan Walia

10/27/2024

Background

Dubins curves are used in modelling motion of non-holonomic vehicles that are constrained by a minimum turning radius. While 2D dubins curves can model Ackermann style robots, 3D Dubins curves can model motion of fixed wing aircraft or drones. Dubins curves are helpful in providing a reference trajectory to follow, or in simplifying the dynamics of the actual system to verify other components in a simulation (such as a given control policy).

In this project, we leverage the fact that RNNs can process sequential data to predict the trajectory of a Dubins UAV and evaluate it's performance.

Methodology

Inputs to RNN

The RNN predicts a trajectory based on the following inputs describing the starting configuration and dynamical constraints of the UAV:

- Initial point $x_0 = [0, 0, 0]$ (fixed)
- Goal point x_g (variable)
- Initial heading $\psi \in [0, 2\pi]$ (variable)
- Climb angle $\gamma \in [-30, 30]$ (variable)

The turning radius of this UAV is set to 100m. We vary the waypoint distance (`steplength`) between points in the predicted trajectory to study the affect of this parameter on the performance of the RNN.

Generating ground truth data

The ground truth reference trajectory for training the RNN is generated using the `dubinEHD3d` function. This function finds shortest Dubins path in 3D between two points, considering an initial heading and constant flight path angle, turning radius, and waypoint distance. It calculates the turning circles based on the start position and radius, checks if a feasible path exists, and then selects the minimum-length path by comparing left and right turn options. The path is built by combining an arc (turning) and a straight-line segment, producing a sequence of 3D points. The function returns the path, the final heading at the destination, and the number of path points.

For simplicity, the goal location is always greater than the step length. Considering a 1000x100 grid cell, 36 heading values and 12 climb angle values, the input data contains approximately 4 million permutations of starting configurations (ignoring any goal locations within 1 step size distance of the starting position).

Architecture

We based our model on a PyTorch's LSTM module. We attempted to create a "one-to-many" architecture, that takes in a single input (goal coordinates, heading and climb angle), and generates a sequence of trajectory waypoints.

```python
# RNN model
class DubinsRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers=1):
        super(DubinsRNN, self).__init__()
        self.LSTM = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.dropout = nn.Dropout(0.4)  # Dropout layer
        self.fc = nn.Linear(hidden_size, output_size)

        self.num_layers = num_layers
        self.hidden_size = hidden_size

    def forward(self, x):

        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

        # Initialize first hidden and cell states to 0
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)

        LSTM_out, _ = self.LSTM(x, (h0, c0))
        out = self.fc(LSTM_out)
        return out
```

Figure 1: Enter Caption

We initialize the first hidden and cell states (h0, c0) to 0 in order to start the trajectory at the origin of the grid. We used 1 hidden layer. We used two versions of this architecture. For Model 1, we used 128 neurons. For model 2, we used 256 neurons.

Training, Evaluation and Regularization Attempt

We created a dataset of 432000 samples and used a 80-10-10 split for training, validation and testing (testing loss not shown, it was not considered). The dataset contained valid 3D dubins paths (targets) and corresponding input parameters x, y, ψ, γ (inputs). We used `nn.MSELoss()` in combination with an Adam optimizer. Training took place on an NVIDIA RTX 3060 6GB graphics card with 15 - 20 % memory utilization.

Model 1

For our first model, we used 128 neurons for the hidden LSTM layer. We trained this version of the network with a learning rate of 10^{-5} for 100 epochs. We used a dropout rate of 0.4 to avoid over fitting the data. However, as shown in figure 2, our model ended up over fitting the training data by a large margin (result might improve if we simply use a 80-20 split for testing and validation, instead of a 80-10-10 split for training, validation and testing as initially planned). This model took 3 hours to train over 100 epochs.

Parameter Name	**Value**
Batch Size	64
Learning Rate	0.00001
Dropout rate	0.4
Number of epochs	100
Hidden layer neurons	128

Table 1: Hyperparameter Values for Model 1

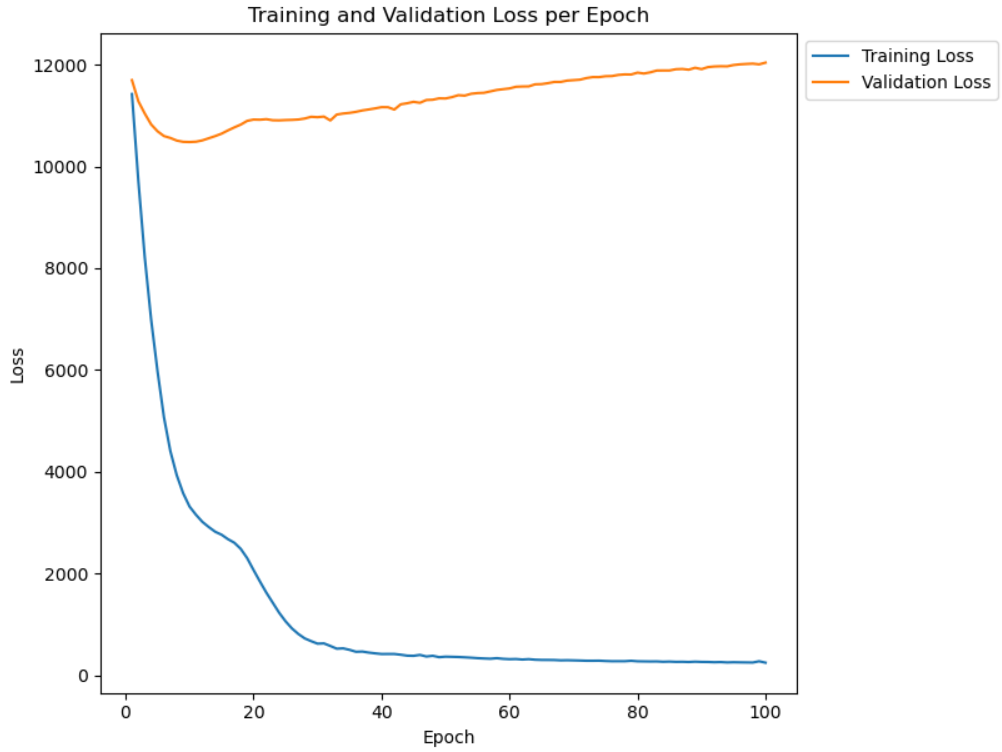


Figure 2: Training and validation losses for model 1. The final training loss was 244.8237

Model 2

Our second attempt was to create a denser model with double the number of neurons for reducing the training loss. This model was trained with a lower learning rate of 10^{-7}, and for 200 epochs instead of 100 epochs. This model took 17.5 hours to train.

Parameter Name	**Value**
Batch Size	64
Learning Rate	0.0000001
Dropout rate	0.4
Number of epochs	200
Hidden layer neurons	256

Table 2: Hyperparameter Values for Model 2

We did manage to reduce overfitting slightly, however the performance of this model was worse as compared to that of Model 1 (please refer to the results section). We believe this poor performance is related to the final training loss of 450.1369.

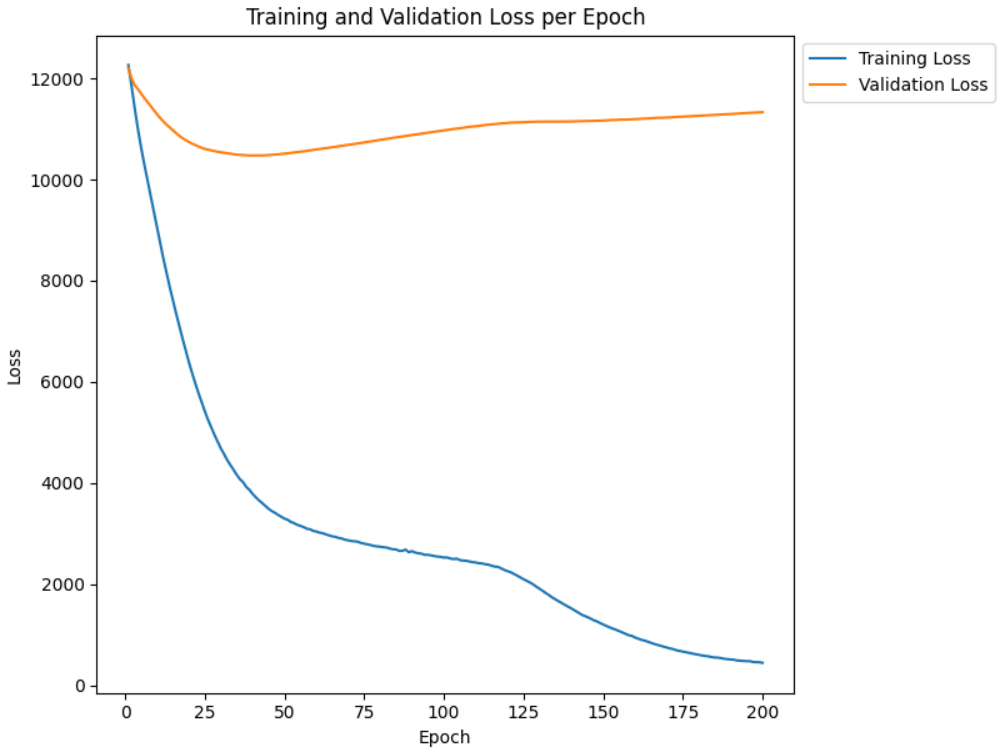
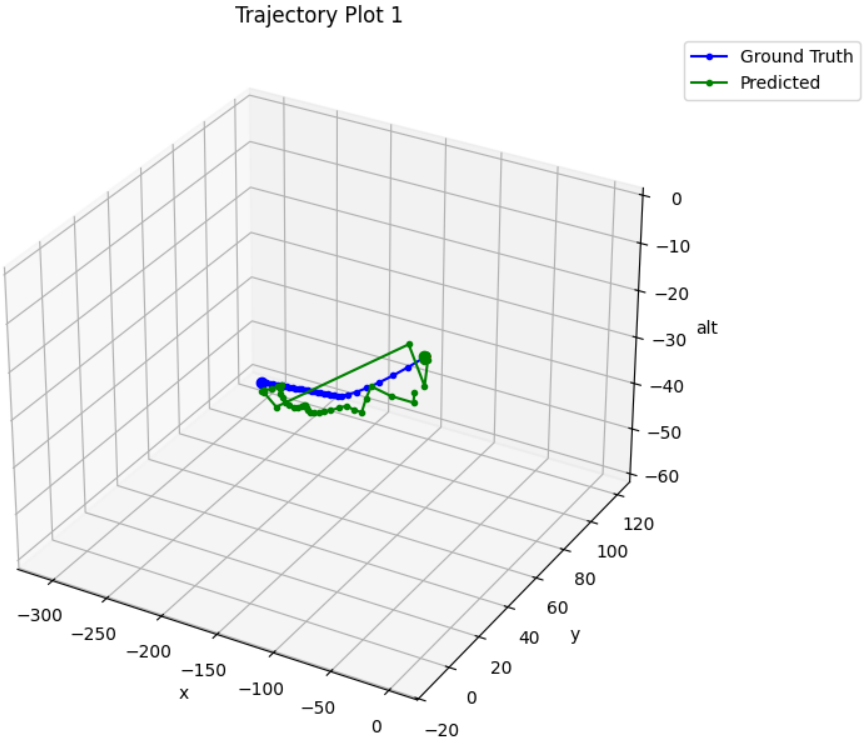


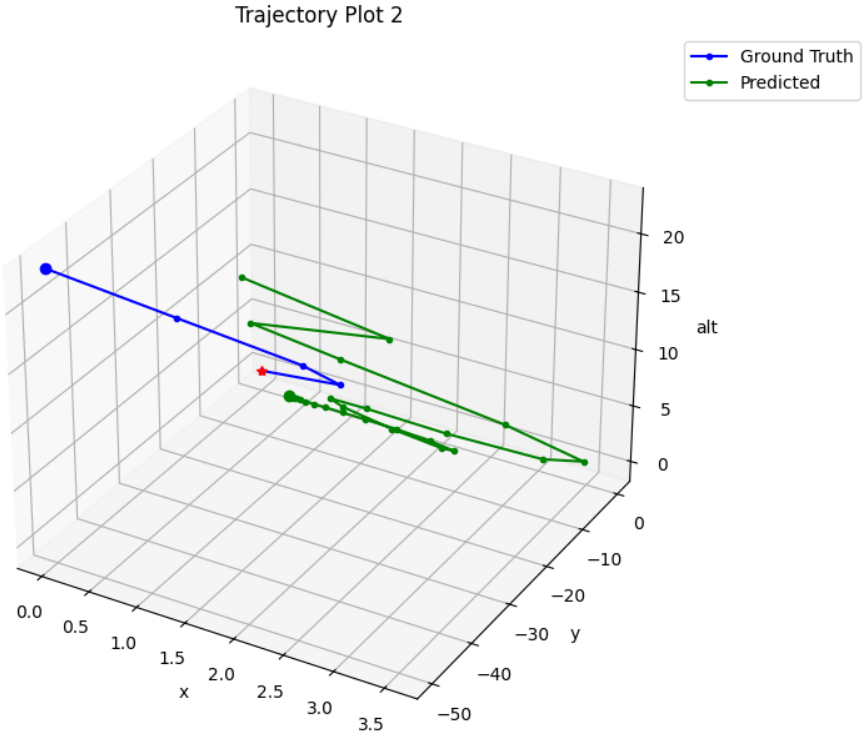
Figure 3: Training and validation losses for model 2. The final training loss was 450.1369

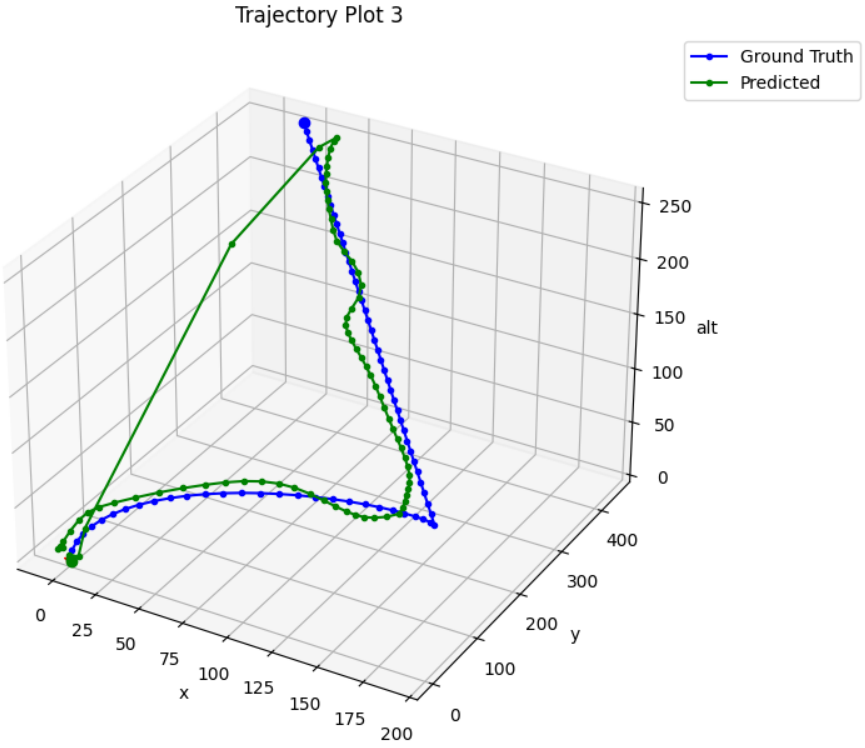
Results

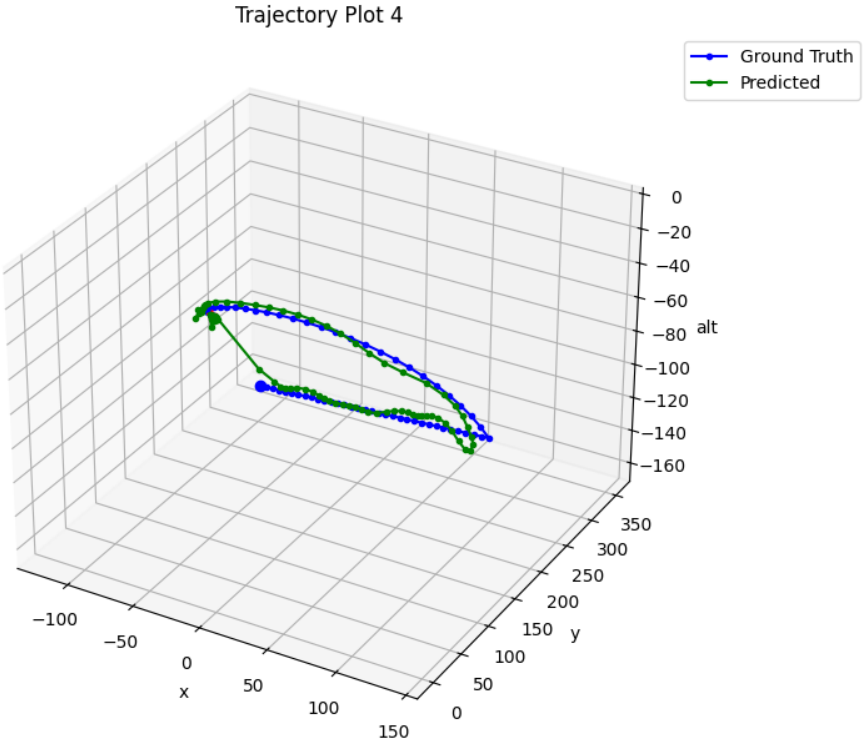
Model 1

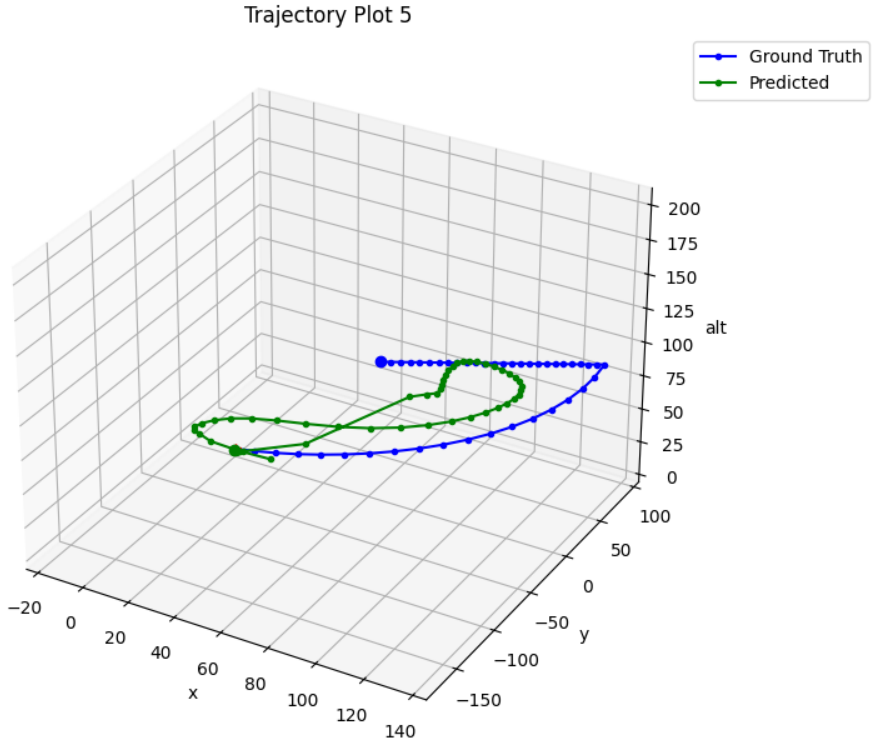
As shown in the plots below, the first model is roughly able to reconstruct smooth trajectories. Note that we did not create a loss for explicitly enforcing steplength, which is why some of the waypoints in the predicted trajectories are not equally spaced. One odd behavior we noticed with this model is the start and goal coordinates being directly connected with each other. We were not able to point out the reason for this behavior, but we suspect this is also partly due to no restrictions on steplength.

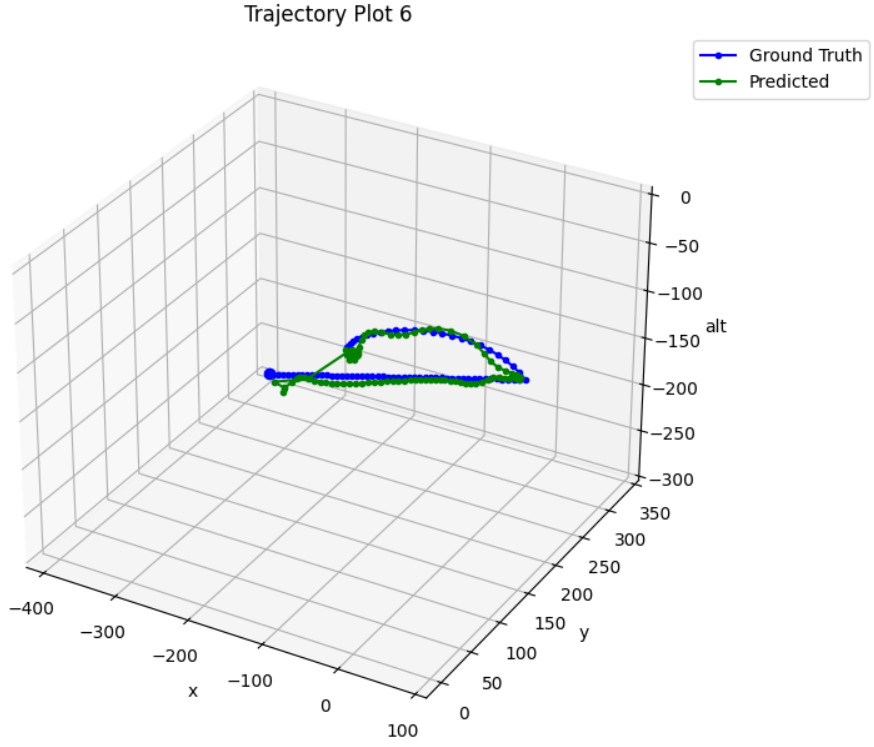

Trajectory Plot 1
Ground Truth
Predicted
alt
x
y

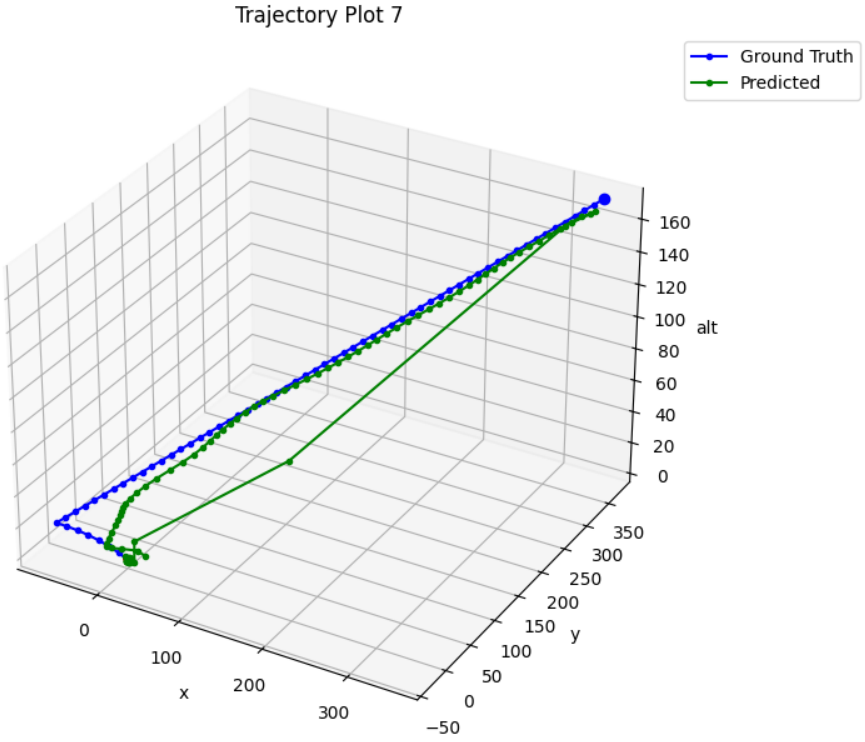

Trajectory Plot 2
Ground Truth
Predicted
alt
x
y

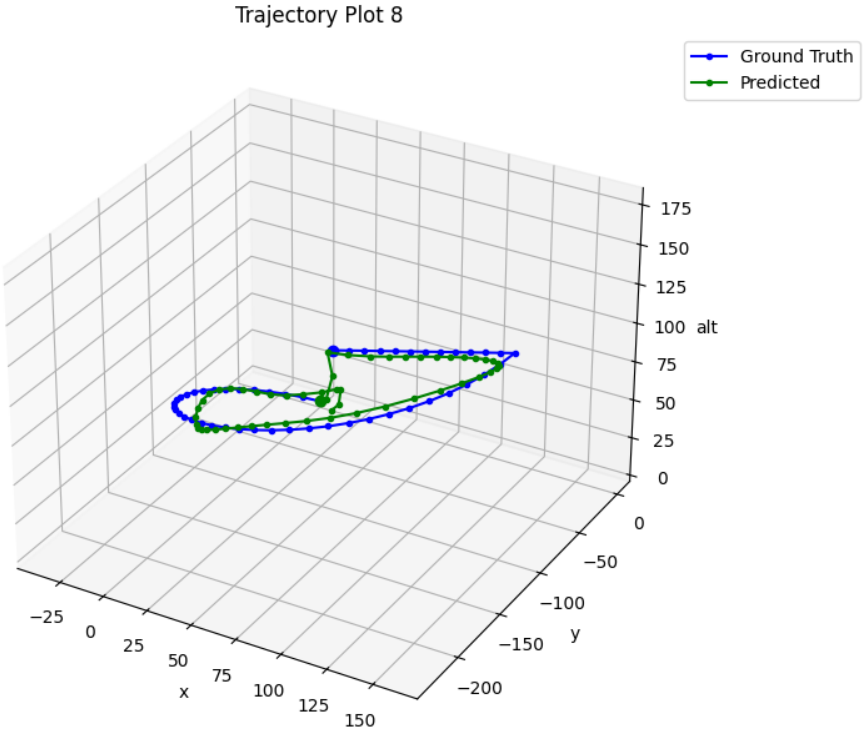

Trajectory Plot 3
Ground Truth
Predicted
alt
x
y

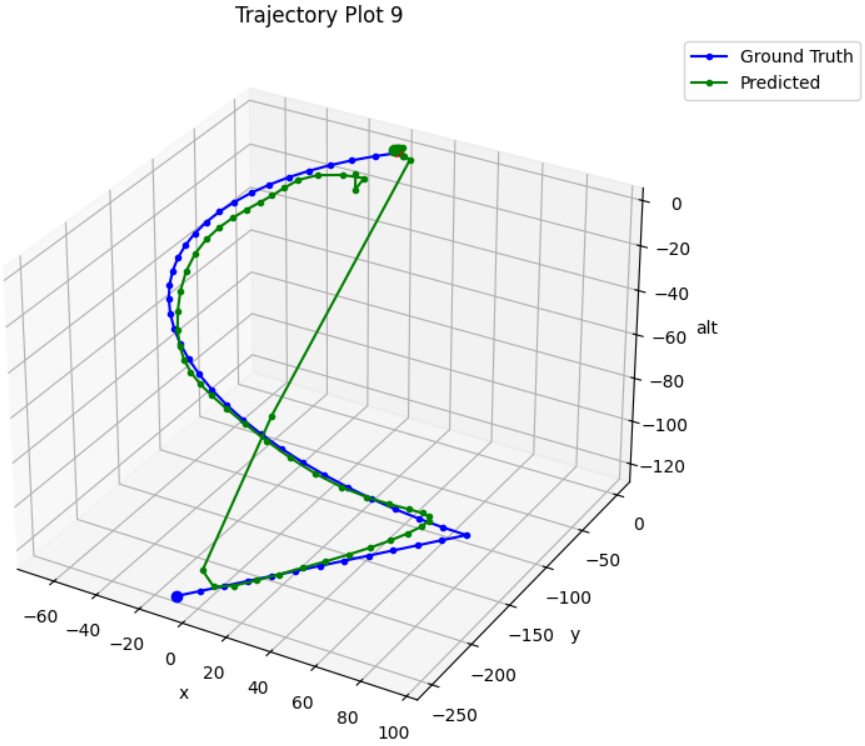

Trajectory Plot 4
Ground Truth
Predicted
alt
x
y

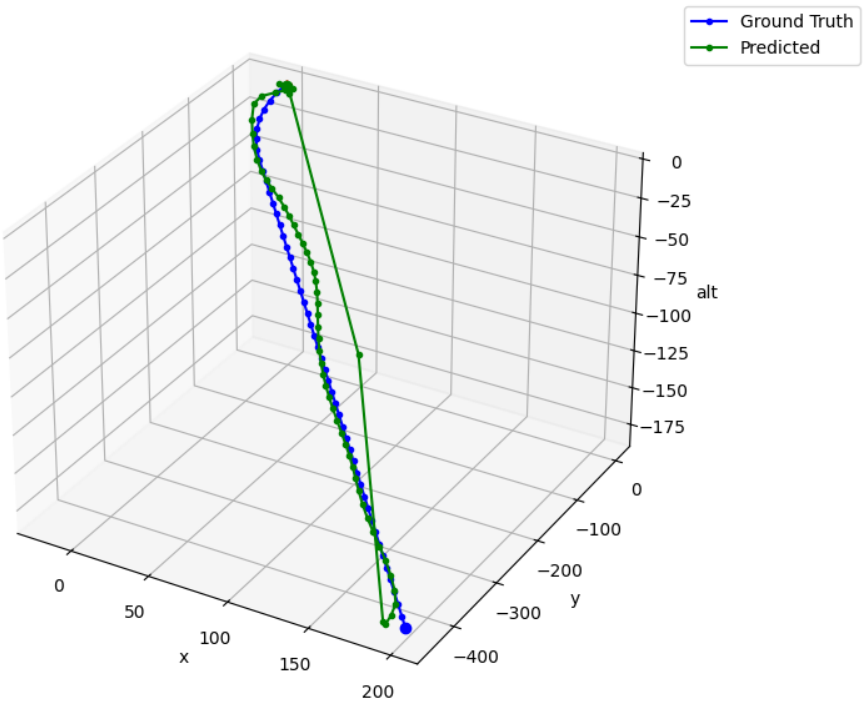

Trajectory Plot 5
Ground Truth
Predicted
alt
x
y


Trajectory Plot 6
Ground Truth
Predicted
alt
x
y

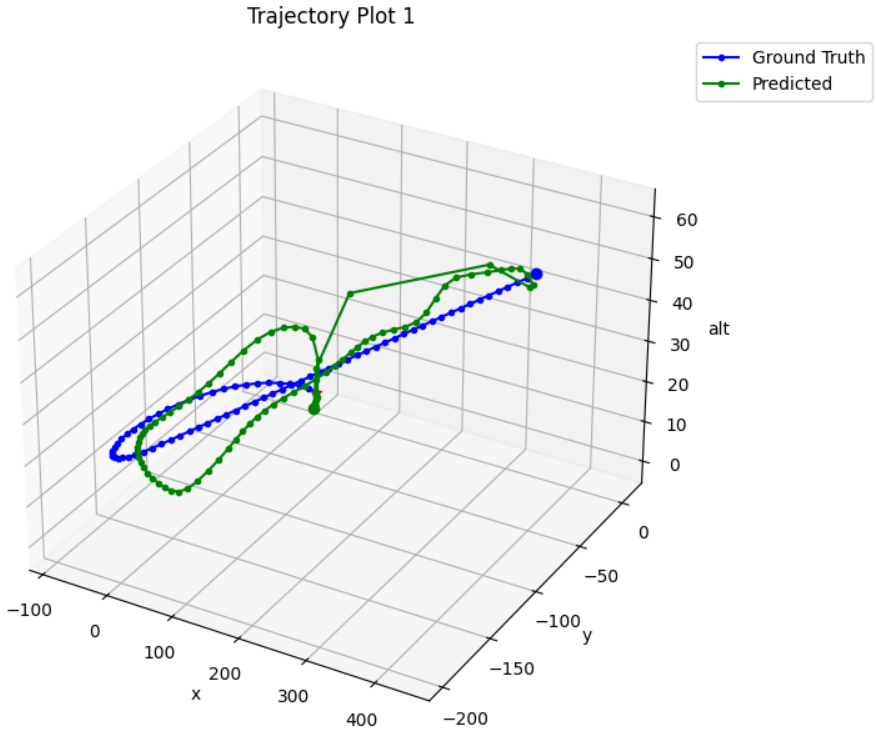

Trajectory Plot 7
Ground Truth
Predicted
alt
x
y

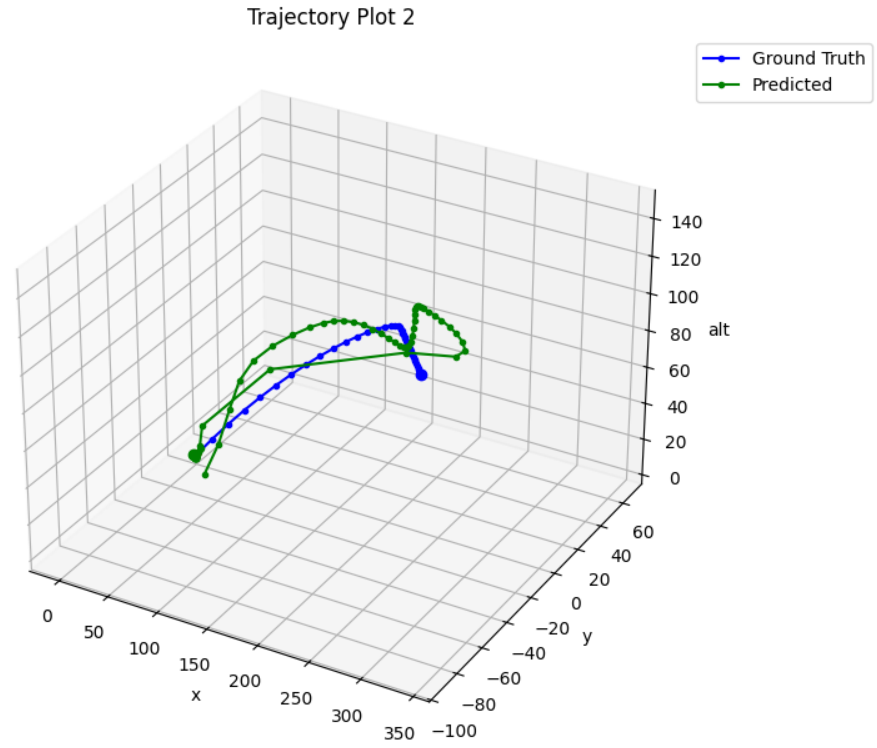

Trajectory Plot 8
Ground Truth
Predicted
alt
x
y

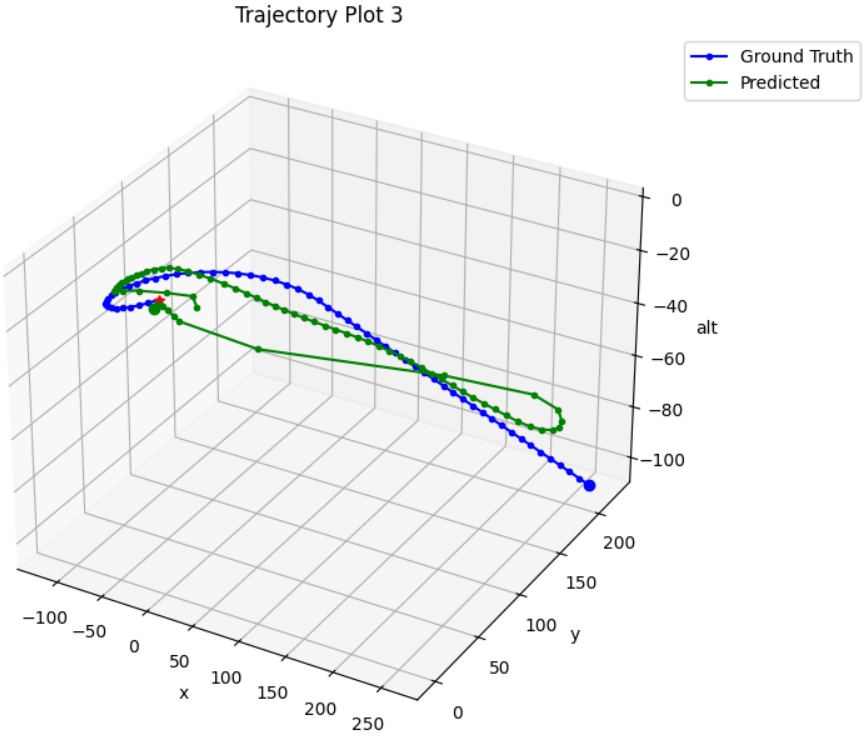

Trajectory Plot 9
Ground Truth
Predicted
alt
x
y

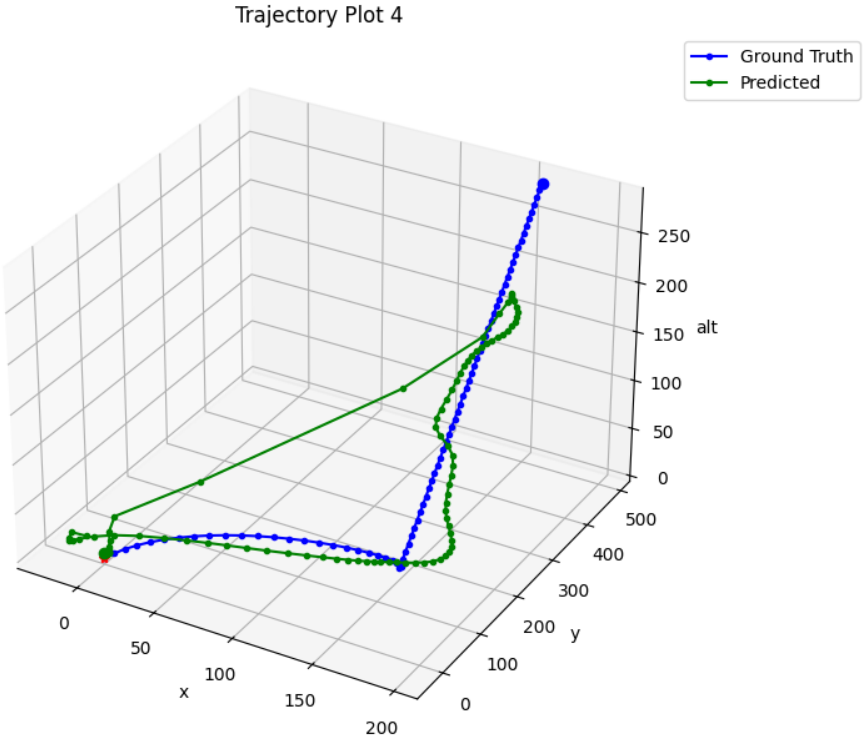

Trajectory Plot 10
Ground Truth
Predicted
alt
x
y

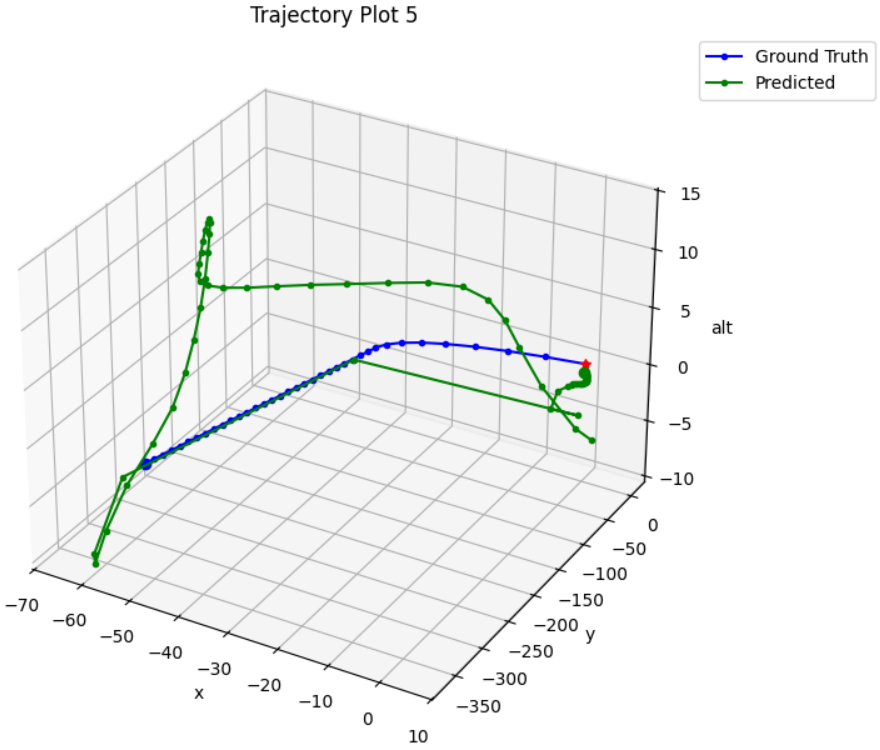
Model 2

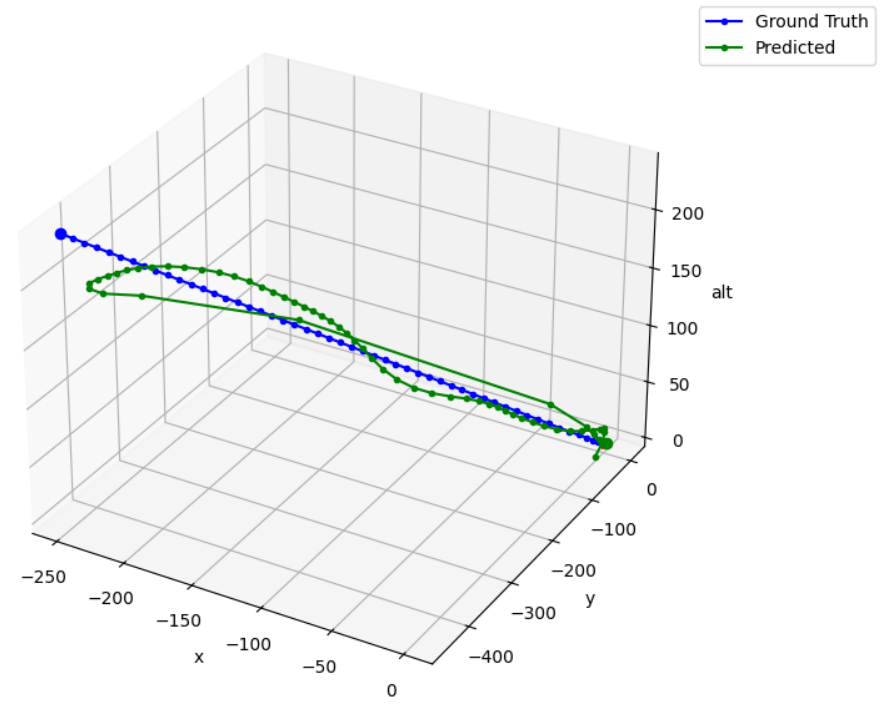

Trajectory Plot 1
Ground Truth
Predicted
alt
x
y

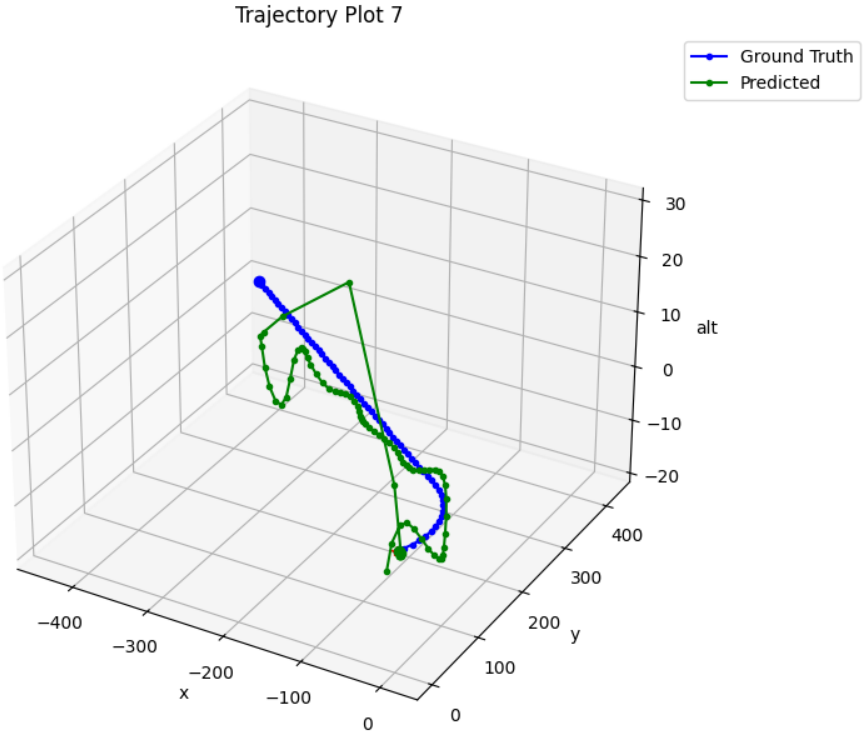

Trajectory Plot 2
Ground Truth
Predicted
alt
x
y

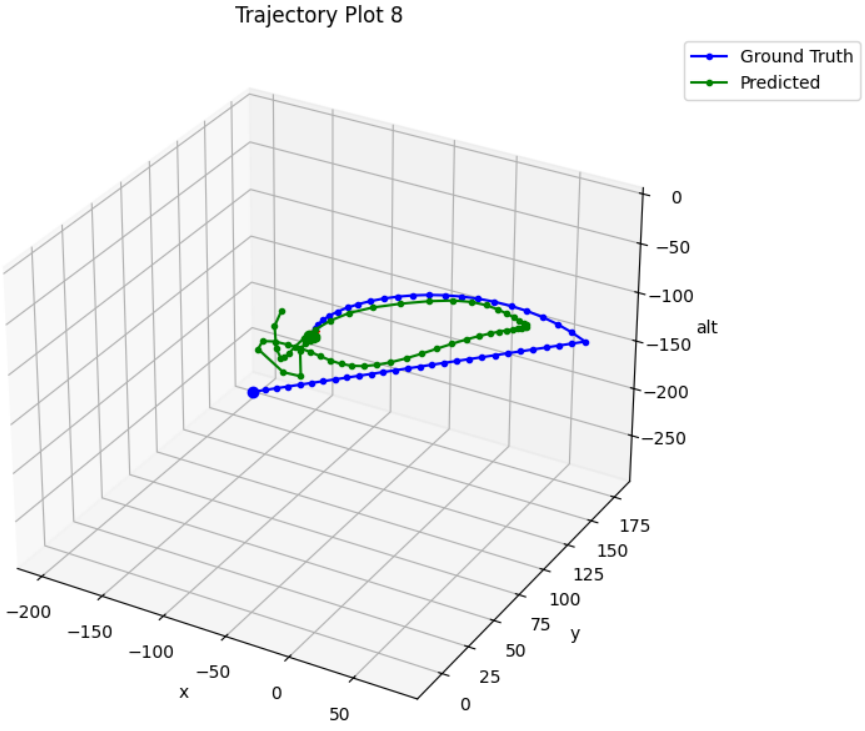

Trajectory Plot 3
Ground Truth
Predicted
alt
x
y

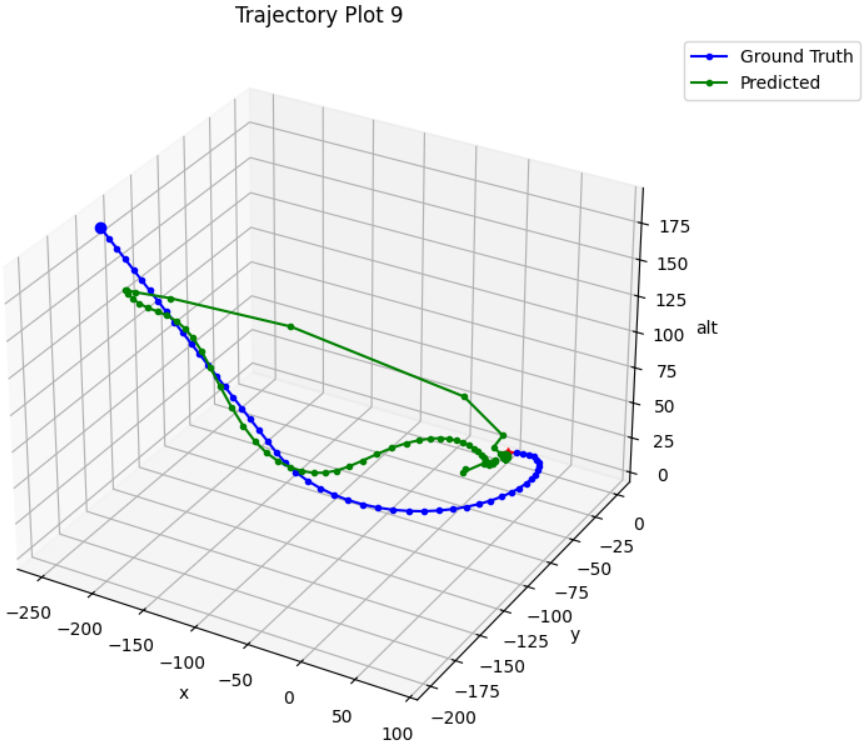

Trajectory Plot 4
Ground Truth
Predicted
alt
x
y

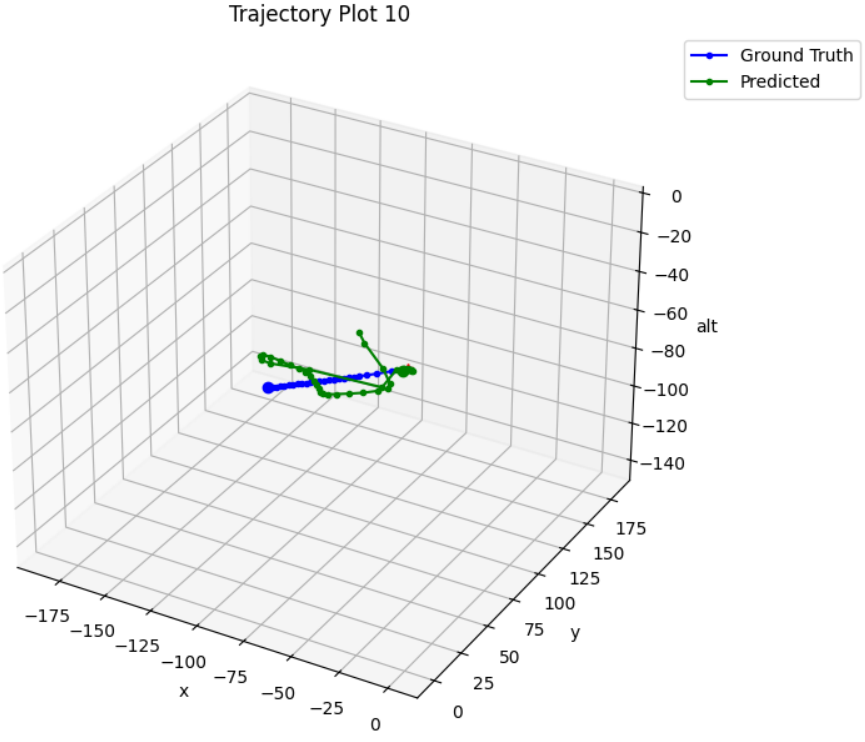

Trajectory Plot 5
Ground Truth
Predicted
alt
x
y


Trajectory Plot 6
Ground Truth
Predicted
alt
x
y


Trajectory Plot 7
Ground Truth
Predicted
30
20
10
0
−10
−20
alt
−400
−300
−200
−100
0
x
0
100
200
300
400
y


Trajectory Plot 8
Ground Truth
Predicted
0
−50
−100
−150
−200
−250
alt
−200
−150
−100
−50
0
50
x
0
25
50
75
100
125
150
175
y


Trajectory Plot 9
Ground Truth
Predicted
alt
x
y


Trajectory Plot 10
Ground Truth
Predicted
alt
x
y

Future Improvements

We believe that early stopping would be very effective in further reducing overfitting for our model. However, that would reduce the performance of the model due to a higher training loss. In order to reduce training loss while avoiding overfitting, we need to design loss functions that capture more attributes of the reference dubins trajectories, particularly:

1. Step length: Distance between each point should be capped by a fixed step length. This would result in a more uniform trajectory and also avoid confusing the model by preventing direct connection of start and goal points.
2. Smoothness: To penalize sharp turns, smoothness metrics can be applied to reduce unrealistic trajectories with sporadic turns.

In addition to loss functions that better characterize reference trajectories, we would experiment with different initialization techniques (example xavier), more regularization techniques such as input or batch normalization, and performance functions such as `reduce_lr_on_plateau`.

References