CSE 643: Artificial Intelligence

Assignment 3: Report

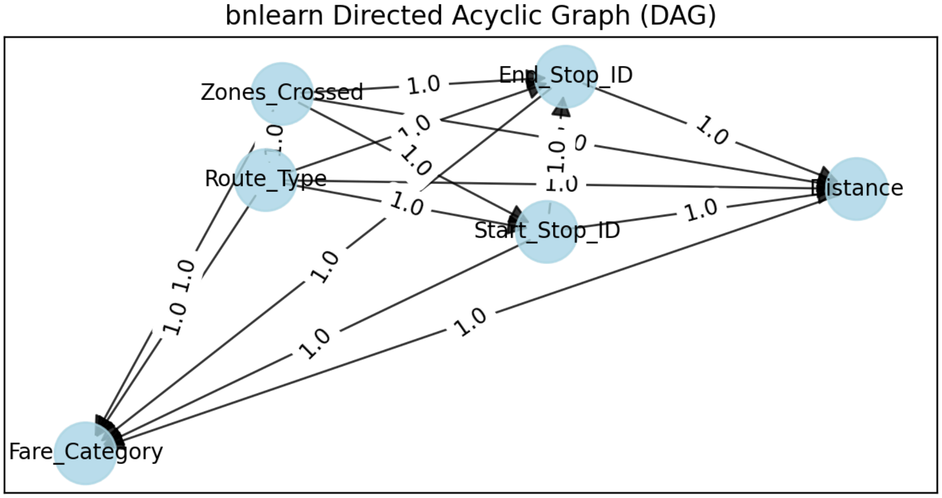
Aditya Aggarwal **2022028**

Question 1` - Bayesian Network for Fare Classification

Base Model

To make the Directed Acyclic Graph (DAG) for the base model, every pair of features is connected to each other. Using the make_DAG function, a Bayesian network with $\frac{n(n-1)}{2} = 15$ edges is obtained. We train this model using the bn.parameter_learning.fit function.

bnlearn Directed Acyclic Graph (DAG)



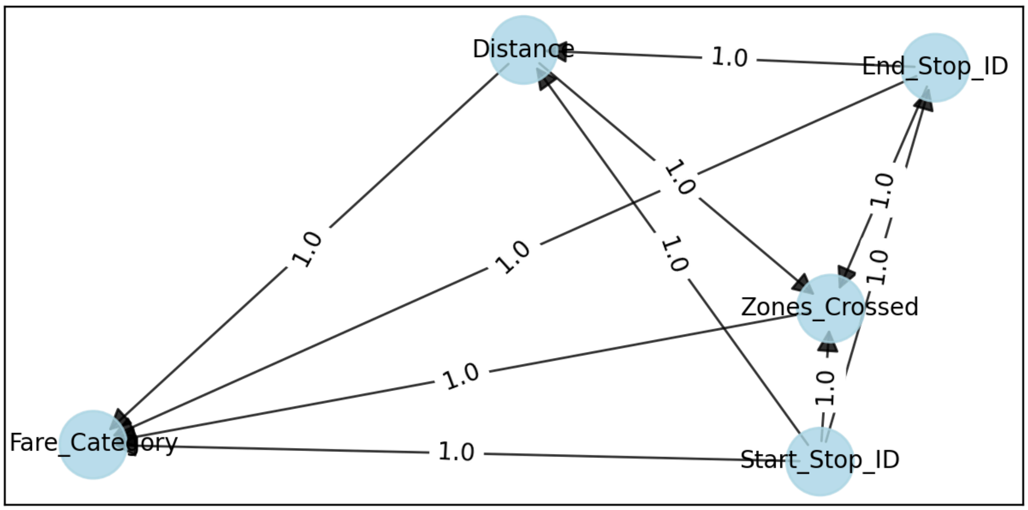
Pruned Model

To prune the earlier created DAG, the bn.independence function is used with parameter prune= True. This function calculates the correlation and dependence between each pair of features to obtain a p-value based on the chi-square test and returns an adjacency matrix with pruned edges.

For the above graph, it prunes 5 edges. The network is recreated based on pruned edges and trained following the same procedure as before.

```
[bnlearn] >bayes DAG created.
[bnlearn] >Compute edge strength with [chi_square]
[bnlearn] >5 edges are removed with P-value > 0.05 based on chi_square
[bnlearn] >Converting source-target into adjacency matrix..
[bnlearn] >Making the matrix symmetric..
[bnlearn] >bayes DAG created.
```

bnlearn Directed Acyclic Graph (DAG)



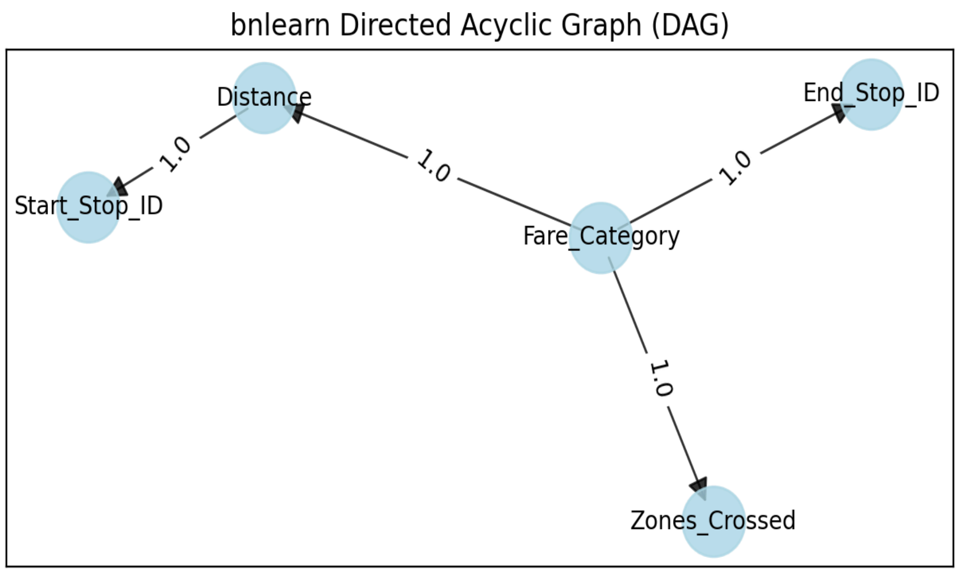
Optimized Model

In order to optimize the model, the bn.structure_learning.fit function is used with parameter methodtype= 'hc' representing the Hill Climbing method for optimization. The resultant optimized network is then used to train the model following the same procedure as before.

Results

In all three models, 100.00% accuracy was achieved on the classification of Fare_Category, with all 350/350 test cases passing.

```
Total Test Cases: 350
Total Correct Predictions: 350 out of 350
Model accuracy on filtered test cases: 100.00%
```

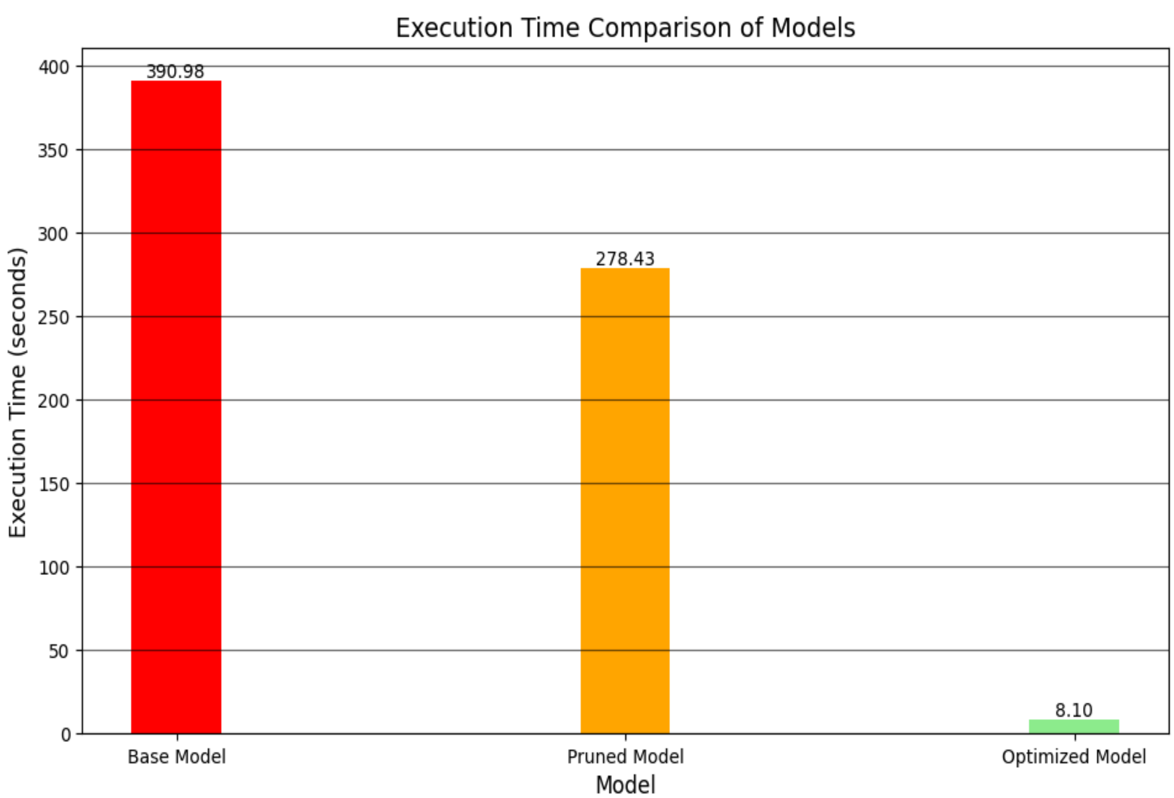


Execution Time

```
Execution time for base model: 390.97514033317566

Execution time for pruned model: 278.43317794799805

Execution time for optimized model: 8.100509643554688
```



Question 2` - Tracking a Roomba Using the Viterbi Algorithm

Defining States

The state of the bot is defined by its position and heading.

$state \equiv ((x, y), h);\ \exists x \exists y \in \{1, 2, \dots, 9\};\ \exists h \in \{'N', 'S', 'E', 'W'\}$

Thus, we have $10 \times 10 \times 4 = 400$ possible states.

Emission Probability

$$log\left(P_E\right) = -\frac{(x_{obs} - x_{true})^2 + (y_{obs} - y_{true})^2}{2\sigma^2} - ln\left(2\pi\sigma^2\right)$$

Transmission Probability

Given the movement policy, the log-likelihood of transmission probability was calculated by observing the likelihood of the bot transitioning from the previous state to the current state.

- For impossible transitions (for instance, a transition between states with a Manhattan distance greater than 1 unit), $log(0) = -\infty$ was returned by the function.
- For deterministic transitions (for instance, a transition in straight_until_obstacle policy in a non-boundary cell), $log(1) = 0$ was returned by the function.
- For probabilistic transitions (for instance, a transition in random_walk policy), $log(0.25)$ was returned by the function.

Viterbi Algorithm | reference

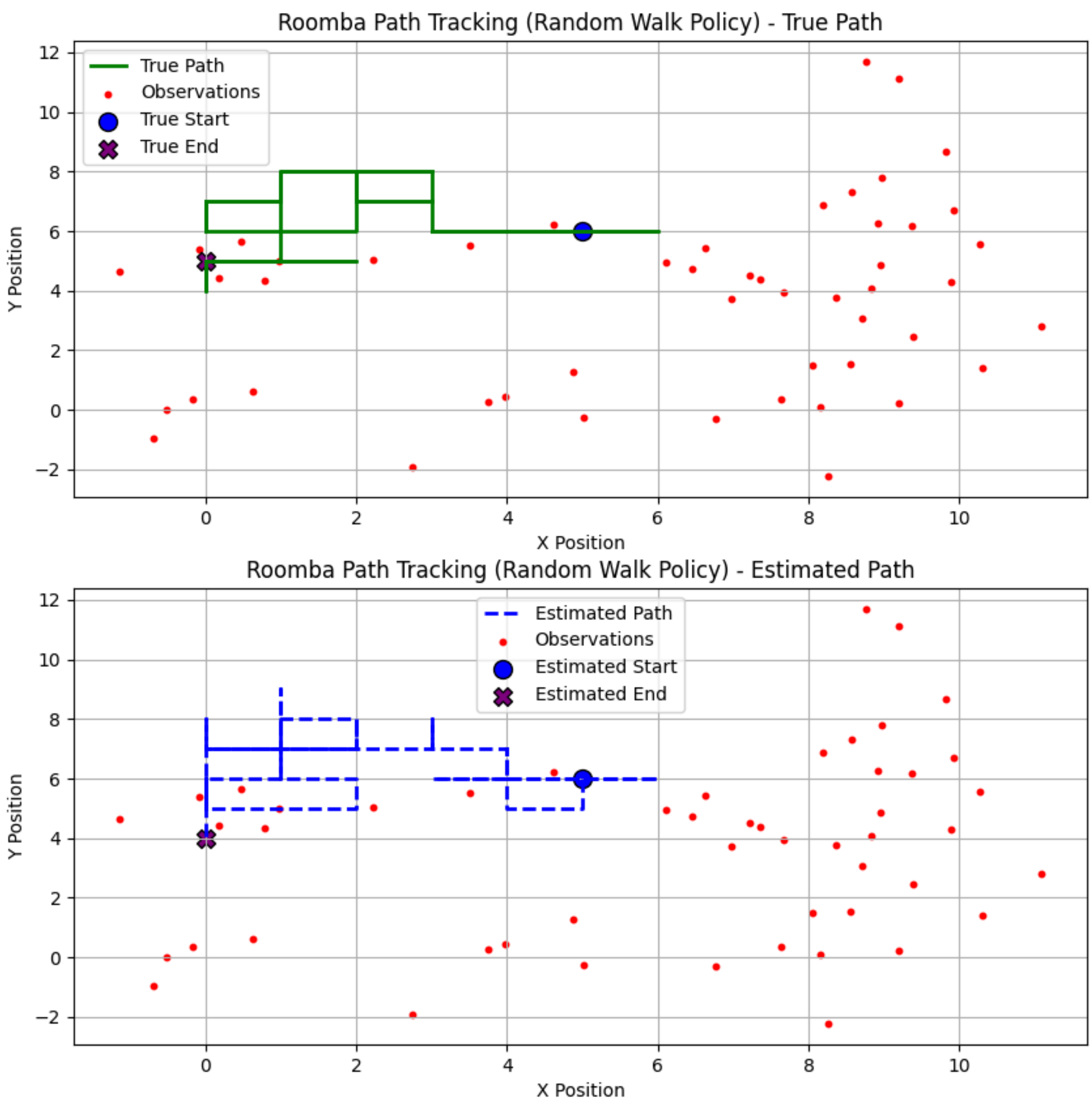
The algorithm uses dynamic programming to find the most likely sequence of states in a Hidden Markov Model (HMM) given a series of observations. It maintains V such that V[t][y] represents the log probability of the most probable state sequence ending in state y at time t. At each time step, it selects the maximum probability state transition path for each state and stores the sequence. The most likely sequence is reconstructed based on the highest probability in the last time step.

Results

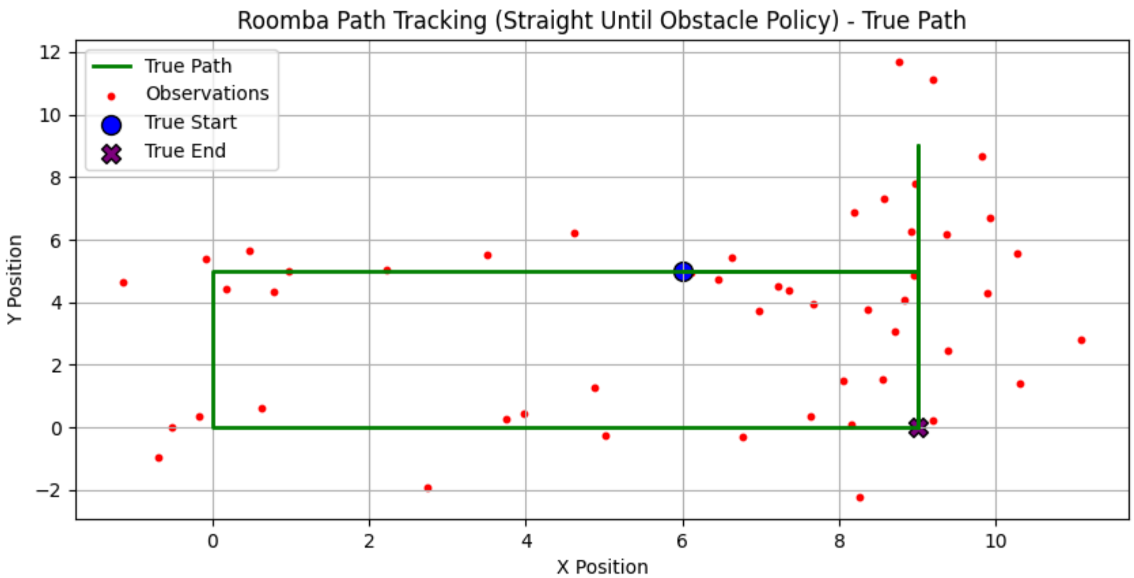
1. Seed 111

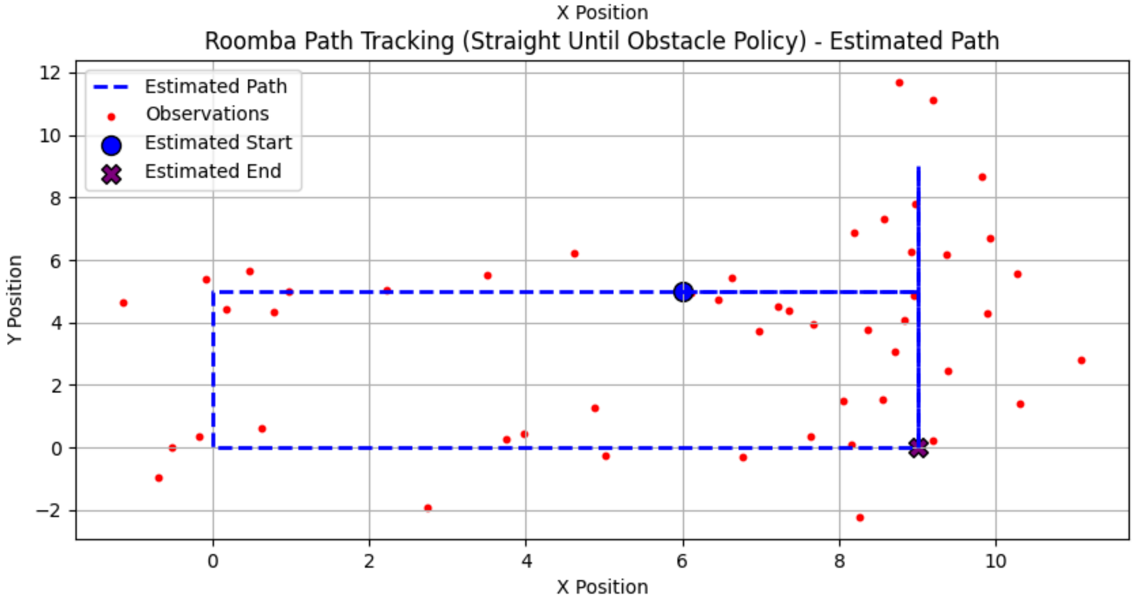
Accuracy on random_walk = 42.00%

Accuracy on straight_until_obstacle = 100.00%



Paths for Random Walk Policy with Seed 111



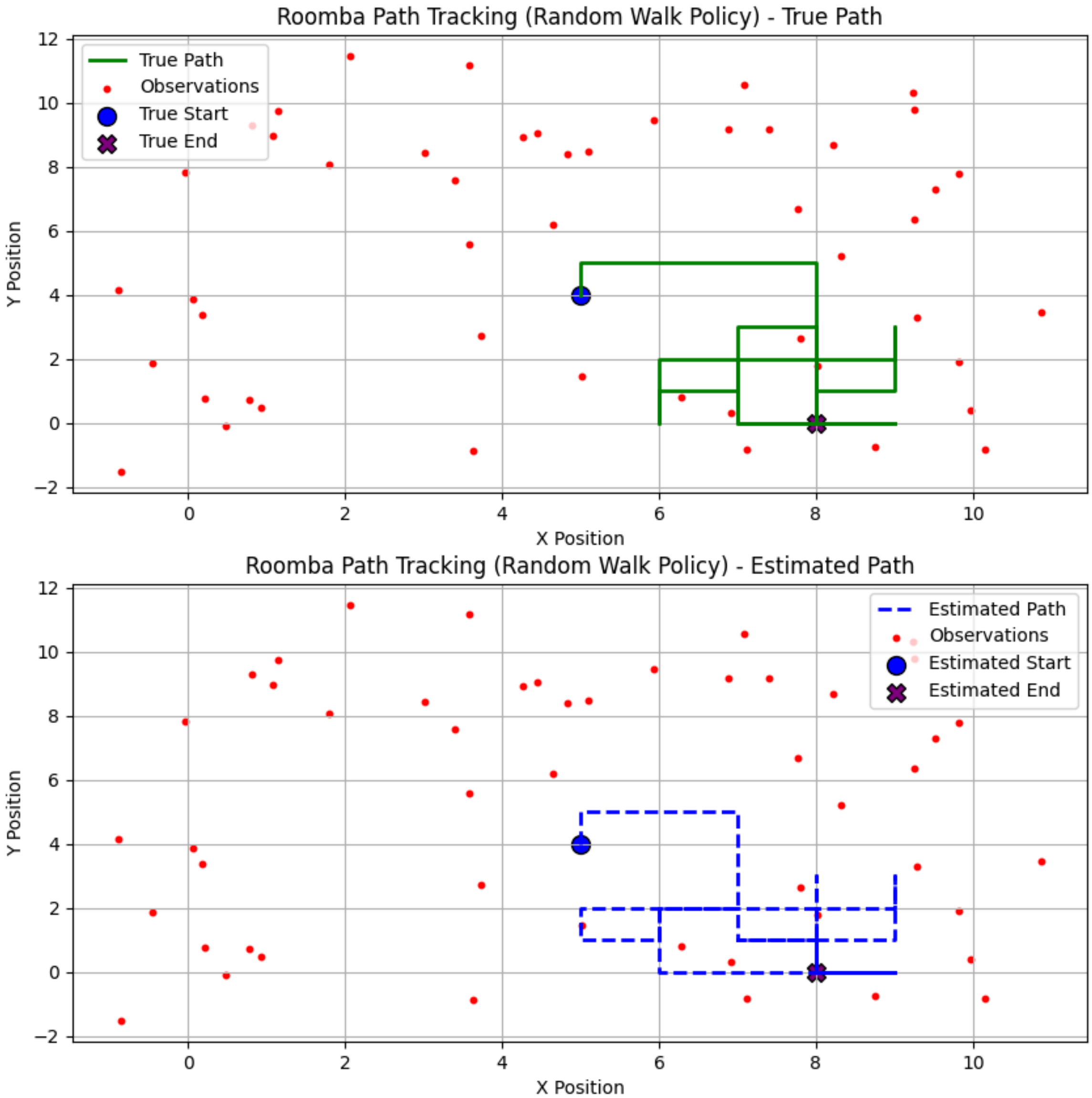


Paths for Straight Until Obstacle Policy with Seed 111

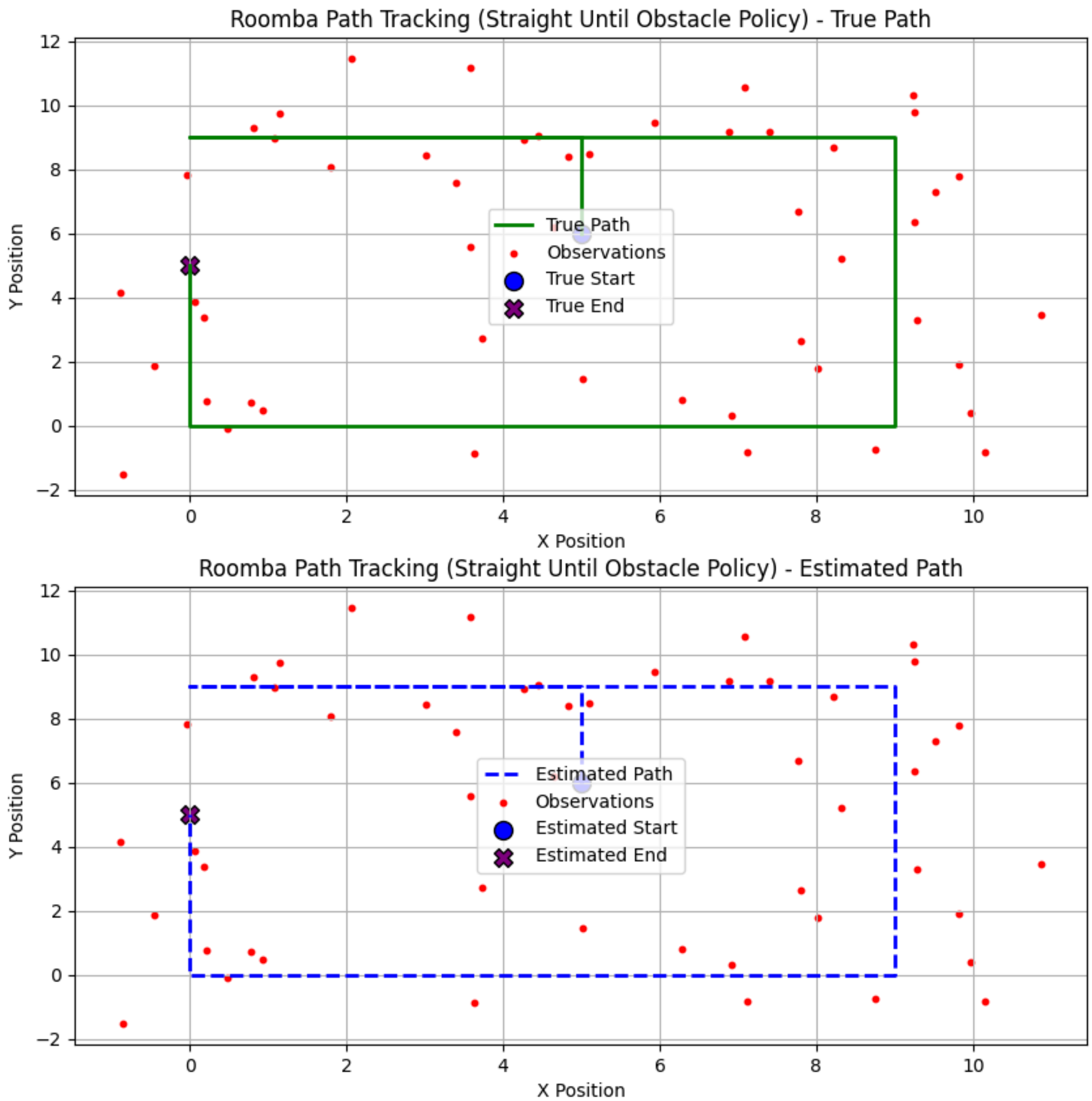
2. Seed 42

Accuracy on random_walk = 64.00%

Accuracy on straight_until_obstacle = 100.00%



Paths for Random Walk Policy with Seed 42

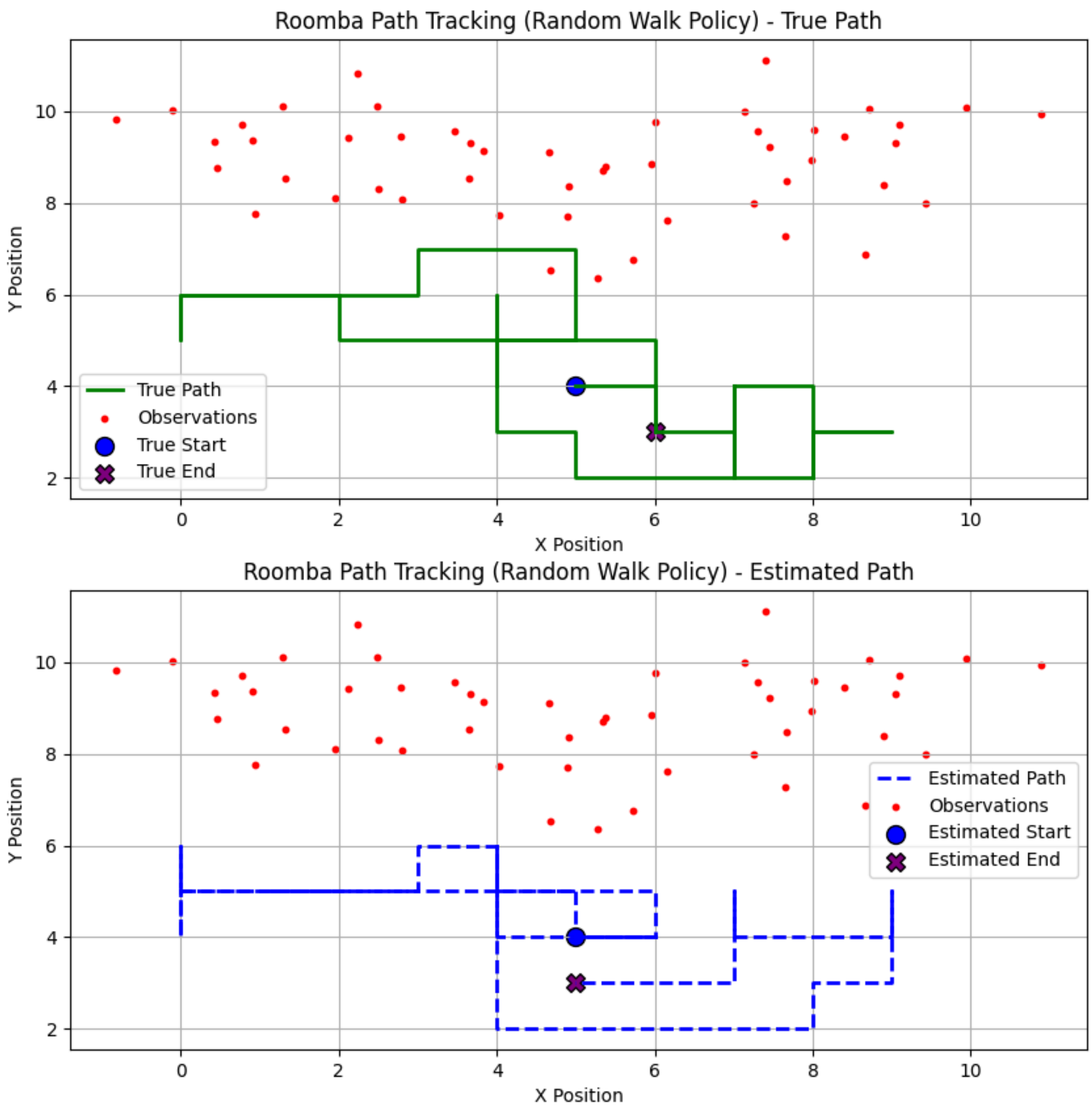


Paths for Straight Until Obstacle Policy with Seed 42

3. Seed 69

Accuracy on random_walk = 44.00%

Accuracy on straight_until_obstacle = 82.00%



Paths for Random Walk Policy with Seed 69

Paths for Straight Until Obstacle Policy with Seed 69

Conclusion

While the Roomba bot does not perfectly follow the true path for the given movement policy in some seeds, it is able to capture the approximate movement using the noisy observations it receives from its sensors and the logic for transitioning from one state to another in the Hidden Markov Model.

The accuracy in the case of straight_until_obstacle will always be higher than the accuracy in random_walk because the former is a much more deterministic movement policy, thus reducing the number of options for the next state and making the decision easier.

The random_walk policy, on the other hand, is probabilistic, which forces the model to depend on the emission probabilities derived from the noisy observations, introducing the chance of error in estimating the true path.

It is also to be noted that even straight_until_obstacle will not give 100% accuracy on all seeds due to its probabilistic nature when the expected position of the bot is an obstacle.