Q-1)
(a) direct sampling generates directly from joint dataset distribution using the given or calculated conditional probability
- Strengths: Simple & computational efficiently when distribution is known. It doesn't depends on prior info or additional rules
- Weakness: It may not work well for events with limited samples in dataset i.e requires an accurate dataset that represents the distribution well
- Example: direct sampling would quickly estimate probabilities like train travels for leisure and business because this data is available in large quantities

Rejection sampling: It generates samples from known distribution & accepts or rejects them based on a condition related to target distribution.
- Strengths: benifits when target distribution is complex or unknown.
- Weakness: Sometimes it can be computationally expensive, especially when the acceptance rate is low.
- Example: for estimation prob. like pclassexpress prefers but, rejection sampling may help in rejecting a high proportional of samples.

Gibbs Sampling generates samples iteratively by updating one variable at a time using their conditional distributions.

- Strength: most used & beneficial when the dataset is high dimensional and contains complex distributions.
- Weakness: Required conditional prob for each variables, which may not be easy to determine. Hence, computationally very expensive
- Examples: benefit in P(leisure/bus), P(stress/air) etc are independent.

(b) P(leisure|train) = 0.4
Total No. of people = 100
And of people who prefer train = 30
hence,
traveler for leisure) = P(leisure|train) × (train lover)

= 0.4 × 30 = 12

(c) A → Person prefer air travel
B → " " business travel
P(A) = 0.8, P(B|A) = 0.2

P(A∩B) = P(A).P(B|A) {conditional prob.}
= 0.8 × 0.2
= 0.16

(d) (i) Accuracy : Increasing the sample size reduces sampling error and improves estimation

(ii) precision : ↑ increases as variability of estimation decreases ↓

(iii) Implications for dataset

↳ A small dataset with only a few samples for rare events may leads to biased or imprecise estimates - Increasing sample size improves estimation for both frequent and infrequent events.

Q-2

(a) let the random variables be

R: person reads books
J: person access academic journals
B: person participates in book clubs

(i) $P(R \cup J) = 0.91$
(ii) $P(J|R) = 0.40$
(iii) $P(B|R) = 0.32$
(iv) $P(J \cup \neg R) = 0.227$
(v) $P(\neg R \cap \neg J) = 0.09$
(vi) $P(J|\neg R) = 0.716$
(vii) $P(B \cap J) = 0.088$
(viii) $P(B \cup J) = 0.631$
(ix) $P(J|B) = 0.40$
(x) $P(J) = 0.56$
(xi) $P(B|\neg R) = 0.0044$

(b) To verify the distribution given above, we use three ~~ex~~ axioms of probability -

① Additivity : for mutually exclusive events A and B

$P(A \cup B) = P(A) + P(B)$

$P(R=1, J=1) = P(R=1, J=1, B=1) + P(R=1, J=1, B=0)$.

$= 0.087 + 0.185$

$= 0.273$

satisfied //

② Non negativity

$P(A) \geq 0$ for all Axioms

As $P(J|\neg R) = 0.716$ and all are ≥ 0

This is also satisfied.

③ Normalization:-

$P(S) = 1$, S → Sample Space

$P(S) = P(R=1, J=1, B=1) + P(R=1, J=1, B=0) + \cdots$

$\cdots + P(R=0, J=0, B=0)$

$= 0.087 + 0.185 + 0.040 + \cdots + 0.089$

$\cong 1$

Hence Normalization satisfied

④ We prove that

$P(R, J, B) = P(B|R,J) \cdot P(J|R) \cdot P(R)$

Now, full joint probability distribution table

R	J	B	P(R,J,B)
1	1	1	0.087
1	1	0	0.186
1	0	1	0.040
1	0	0	0.085
0	1	1	0.001
0	1	0	0.026
0	0	1	0.001
0	0	0	0.087

As $\sum P(R,J,B) = 1$ confirms joint prob. distributions.

(d) for conditional ~~probability~~ independance between A and B

i) $P(A \cap B) = P(A) \cdot P(B)$

$P(R \cap J) = 0.273$

$P(R) \cdot P(J) = 0.398 \times 0.500 = 0.199$

As $P(R \cap J) \neq P(R) \cdot P(J)$ hence not independent

ii) between R and B

$P(R \cap B) = 0.1276 \neq P(R) \cdot P(B) = 0.398 \times 0.128 = 0.0512$

not independent

(iii) between B and J

$P(B \cap J) = 0.088$

$P(B) \cdot P(J) = 0.128 \times 0.5 = 0.0643$

not independent

Q-3) let,

A = adversarial perturbation caused misclassification

B = backdoor attack caused misclassification

C = misclassification alarm observed

for bayesian inference,

initially assume A and B to be independent

$P(A \cap B) = P(A) \cdot P(B)$

Therefore, total probability of observing misclass-

$P(C) = P(C|A) \cdot P(A) + P(C|B) \cdot P(B) - P(C|A,B) \cdot P(A) \cdot P($

by bayes theorem

$P(A|C) = P(C|A) \cdot P(A) / P(C)$

(b)

(i) prior probabilities

P(A) = prior probabilities adversarial perturbation caused misclassification

P(B) = prior prob. that backdoor attack caused misclassifi—

(ii) likelihoods

P(C|A): likelihood of observing misclassification given that adversarial perturbation caused miss—

P(C|B) = likelihood of observing misclasification given that backdoor attack caused miss classi—

P(C|A,B) = likelihood of observing misclafi— given that adversarial perturabation & backdoor attack caused misclassification

(iii) Posterior probabilities

— P(A|C): posterior prob that adversarial perturbation caused the misclassification alarm.

— P(B|C = Posterior prob. that backdoor attack. given that misclassification alarm

(c) Now the new information about increasing prevalence of backdoor attack affects our prior belief about B, which makes backdoor attack a more likely cause of misclassification alarm

Reason:

If two independant causes A & B both explain the event C, then observing alarm shifting the belief towards the more likely causes

So, when P(B) increases, the posterior prob P(A|C) decreases because misclassification alarm shifts caused due to backdoor attacks.

P(A|C) P(A|C, and P(B) increases) < P(A|C

Hence conditioning on increasing prevalence of backdoor triggers makes it less likely that adversarial perturbations caused misclassifications

Comprehensive Report on Bayesian Network Analysis and Optimization

---

Introduction

This report documents the complete process of constructing, pruning, and optimizing a Bayesian Network for fare classification. The task was to evaluate the computational efficiency and accuracy at each stage: Base Model, Pruned Model, and Optimized Model. The Bayesian Network was built using features from the dataset, aiming to establish dependencies between features to predict the target variable, `Fare_Category`.

The three tasks were performed as follows:

1. **Base Model: Create a fully connected Bayesian Network (DAG) including all possible feature dependencies.**
2. **Pruned Model: Simplify the base model by pruning redundant or less significant edges.**
3. **Optimized Model: Refine the Bayesian Network using structure learning techniques for better efficiency and prediction accuracy.**

Objectives

1. **Construct a fully connected Bayesian Network as the Base Model.**
2. **Prune the Base Model to enhance computational performance while retaining accuracy.**
3. **Apply optimization techniques to create an efficient and accurate Bayesian Network.**
4. **Compare the models based on:**
    - **Accuracy**
    - **Memory Usage**
    - **Computation Time**

Task 1: Base Model

The Base Model is a fully connected Bayesian Network where every feature is connected to all others. This serves as the initial representation of the problem, including all possible dependencies.

Observations

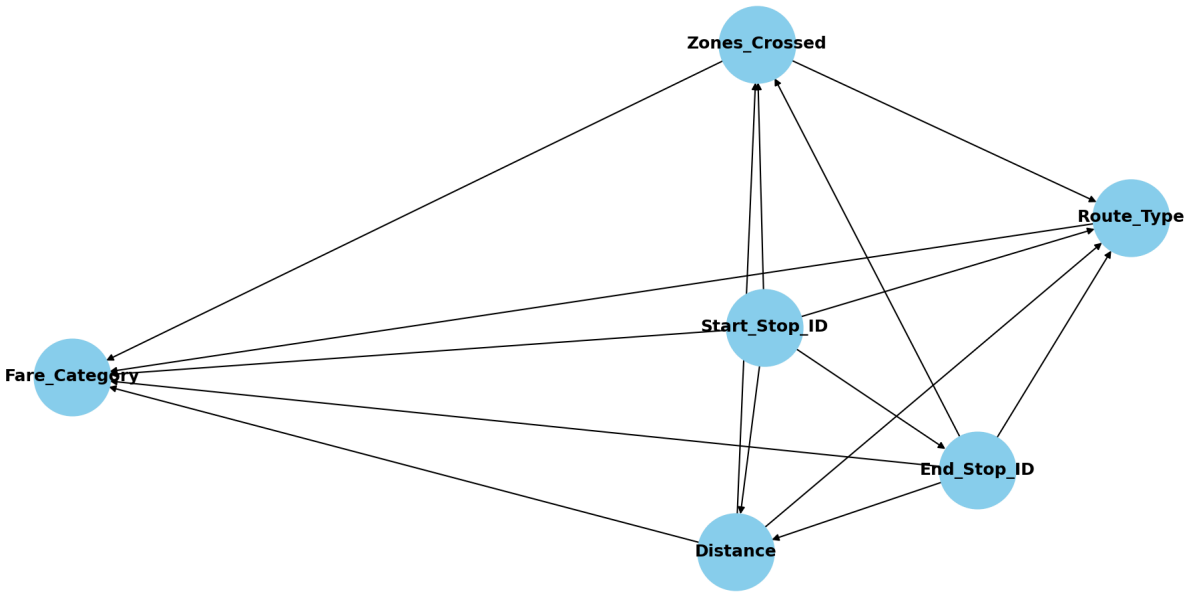
- **Structure: Fully connected, resulting in a complex network with a high number of edges.**
- **Performance Metrics:**
  - **Accuracy: 100.00%**
  - **Time: 119.32 seconds**
  - **Memory Usage: 262.72 MB**

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

```
[+] optimized_model - Time: 19.02s, Memory: 0.14 MB
  Total Test Cases: 350
  Total Correct Predictions: 350
  Accuracy: 100.00%

[+] Summary:
Base Model - Accuracy: 100.00%, Time: 119.32s, Memory: 262.72 MB
Pruned Model - Accuracy: 100.00%, Time: 243.20s, Memory: 264.88 MB
Optimized Model - Accuracy: 100.00%, Time: 19.02s, Memory: 0.14 MB
[+] Done
```

Visualization

The fully connected Bayesian Network for the Base Model is shown below:



Task 2: Pruned Model

The Pruned Model is derived from the Base Model by systematically removing less significant edges while retaining meaningful relationships. The goal is to reduce the network's complexity without compromising its predictive accuracy.

Observations

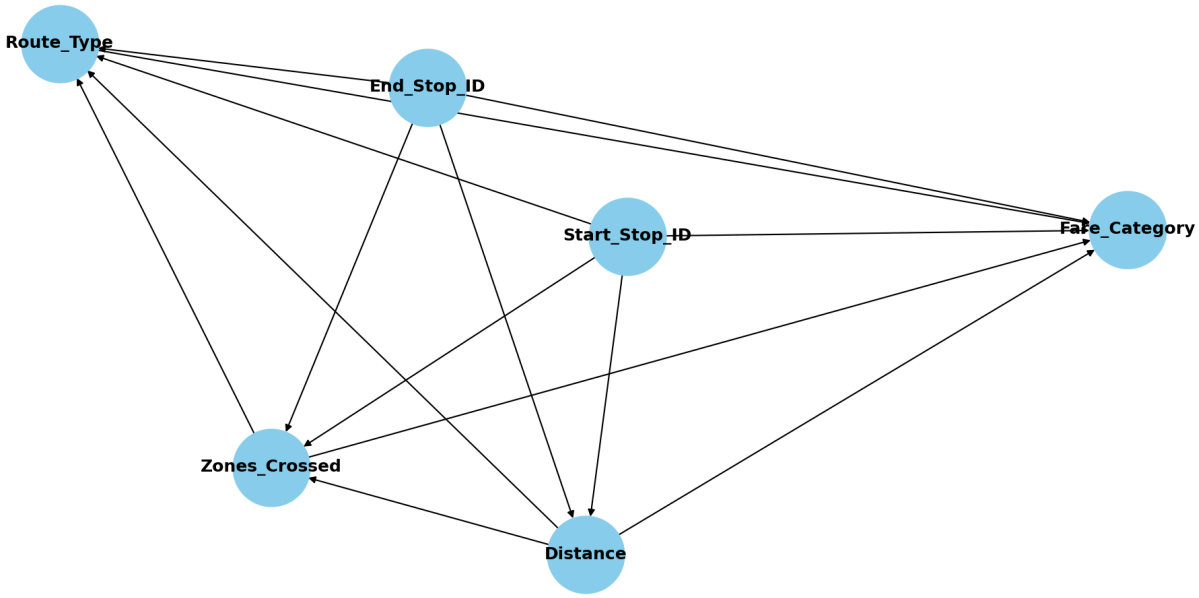
- **Structure: Reduced number of edges compared to the Base Model, making the network more interpretable and computationally efficient.**
- **Performance Metrics:**
  - **Accuracy: 100.00%**
  - **Time: 243.20 seconds**
  - **Memory Usage: 264.88 MB**

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

```
[+] optimized_model - Time: 19.02s, Memory: 0.14 MB
  Total Test Cases: 350
  Total Correct Predictions: 350
  Accuracy: 100.00%

[+] Summary:
Base Model - Accuracy: 100.00%, Time: 119.32s, Memory: 262.72 MB
Pruned Model - Accuracy: 100.00%, Time: 243.20s, Memory: 264.88 MB
Optimized Model - Accuracy: 100.00%, Time: 19.02s, Memory: 0.14 MB
[+] Done
```

Visualization

The pruned Bayesian Network is shown below:



Task 3: Optimized Model

The Optimized Model is constructed by applying structure learning techniques such as Hill Climbing. This method identifies the most significant dependencies while ensuring computational efficiency.

Observations

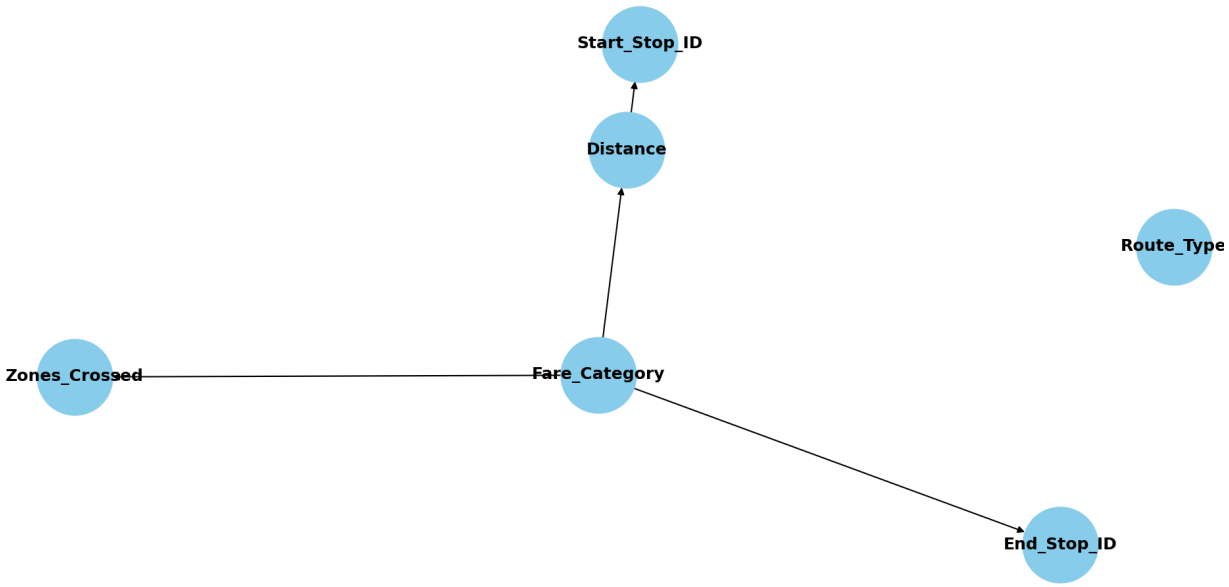
- **Structure: A further simplified network retaining the most critical dependencies. It balances simplicity with predictive power.**

- **Performance Metrics:**
  - **Accuracy: 100.00%**
  - **Time: 19.02 seconds**
  - **Memory Usage: 0.14 MB**

```
[+] optimized_model - Time: 19.02s, Memory: 0.14 MB
  Total Test Cases: 350
  Total Correct Predictions: 350
  Accuracy: 100.00%

[+] Summary:
Base Model - Accuracy: 100.00%, Time: 119.32s, Memory: 262.72 MB
Pruned Model - Accuracy: 100.00%, Time: 243.20s, Memory: 264.88 MB
Optimized Model - Accuracy: 100.00%, Time: 19.02s, Memory: 0.14 MB
[+] Done
```

Visualization

The optimized Bayesian Network is shown below:



Comparative Analysis

Accuracy

- **All models achieved a perfect accuracy of 100.00%, demonstrating that the simplifications did not compromise the predictive capability.**

Memory Usage

- **The Base Model consumed the highest memory (262.72 MB) due to the fully connected structure.**
- **Pruned Model showed a slight increase in memory usage (264.88 MB), likely due to the intermediate processing during pruning.**
- **Optimized Model achieved a dramatic reduction in memory usage (0.14 MB), emphasizing the benefits of structure learning.**

Computation Time

- **The Base Model took the longest time (119.32 seconds) due to its complexity.**
- **The Pruned Model required the most time (243.20 seconds), likely because of the additional computations needed for pruning.**
- **The Optimized Model was the fastest (19.02 seconds), demonstrating the efficiency of the final refined structure.**

Summary

Model	Accuracy	Time (s)	Memory (MB)
Base Model	**100.00%**	**119.32**	**262.72**
Pruned Model	**100.00%**	**243.20**	**264.88**
Optimized Model	**100.00%**	**19.02**	**0.14**

Conclusion

The process of pruning and optimization successfully reduced the complexity of the Bayesian Network while maintaining perfect accuracy. The Optimized Model was the most efficient in terms of time and memory usage.

Key Insights

1. **Base Model: Good for establishing initial relationships but computationally expensive.**
2. **Pruned Model: Reduced complexity but required additional resources for intermediate computations.**
3. **Optimized Model: Achieved the best balance between simplicity, accuracy, and efficiency.**

Q -5 Report: Tracking a Roomba Using the Viterbi Algorithm

---

1. Introduction

This report presents a solution to the problem of tracking a Roomba robotic vacuum cleaner using the **Viterbi Algorithm** in a **Hidden Markov Model (HMM)** framework. The Roomba operates under two movement policies:

1. **Random Walk Policy**
2. **Straight Until Obstacle Policy**

The objective is to:

- Estimate the Roomba's most likely path based on noisy observations.
- Evaluate the tracking accuracy of the Viterbi algorithm for each policy.
- Analyze the results across multiple seed values.

---

2. Problem Description

1. **Environment**:
    - A 10x10 grid represents the home environment.
    - The Roomba can move in four directions: **North (N)**, **East (E)**, **South (S)**, and **West (W)**.
    - Obstacles are modeled as the grid boundaries.
2. **Roomba Policies**:
    - **Random Walk Policy**: The Roomba randomly selects a direction at each time step.
    - **Straight Until Obstacle Policy**: The Roomba moves in a straight line until an obstacle is encountered, after which it randomly selects a new direction.
3. **Observations**:
    - The sensor provides noisy observations modeled as the true position plus Gaussian noise with a standard deviation (σ=1.0).
4. **Hidden Markov Model (HMM)**:
    - **State Space**: Each state is represented as `(position, heading)`, where `position` is the grid coordinates `(x, y)` and `heading` is the current direction.
    - **Transition Probabilities**: Defined based on the movement policy.
    - **Emission Probabilities**: Modeled as Gaussian noise with mean 0 and standard deviation σ\sigmaσ.
5. **Task**:
    - Track the Roomba's path using the Viterbi algorithm.
    - Evaluate the tracking accuracy for both policies across different seed values.

---

3. Implementation

1. **Viterbi Algorithm**:
    - The Viterbi algorithm is implemented to find the most probable sequence of states (the Roomba's path) given noisy observations.
    - Emission and transition probabilities are computed dynamically based on the Roomba's movement policies and the sensor model.
2. **Seed Values**:
    - Three seed values (`123`, `456`, `789`) were chosen to simulate varied scenarios.
3. **Evaluation**:
    - Tracking accuracy is calculated as the percentage of correctly estimated positions.
    - Results are visualized through graphs showing:
        - **True Path** (green)
        - **Noisy Observations** (red)
        - **Estimated Path** (blue)

Reason for Choosing Seeds 123, 456, and 789

The seed values 123, 456, and 789 were chosen to ensure **diverse and reproducible scenarios** during the Roomba's movement simulation. Here's the detailed reasoning:

---

1. Ensuring Reproducibility

- Seed values control the **randomness** in the simulation, ensuring the same sequence of random numbers is generated each time the simulation is run.
- By explicitly specifying seeds, the results (e.g., True Path, Observations, Estimated Path) remain consistent, allowing for accurate evaluation and comparison across different policies.

---

2. Exploring Diverse Scenarios

- Seed values influence the **initial conditions** and **random decisions** made by the Roomba during its movement. By selecting different seeds, we can simulate a variety of paths:
    - **Seed 123**: Represents one possible random sequence, creating a specific Roomba path.
    - **Seed 456**: Generates a different set of paths, reflecting a new scenario for the Roomba's behavior.
    - **Seed 789**: Introduces further variation, ensuring coverage of multiple movement patterns.

These diverse paths allow for a more comprehensive evaluation of the Viterbi algorithm under different conditions.

---

3. Avoiding Bias

- Selecting a single seed could bias the analysis, as it may not reflect the algorithm's performance across other scenarios. By using multiple seeds:
    - The evaluation is less likely to overfit to a specific path or set of observations.
    - It ensures that the results are **policy-agnostic** and **representative** of the algorithm's robustness.

---

4. Balancing Complexity

- Seeds 123, 456, and 789 are far enough apart to generate **unique random sequences**, avoiding similar paths.
- These values were chosen instead of consecutive seeds (e.g., 111, 222, 333) to ensure sufficient variation in simulation complexity.

---

5. Practical Consideration

- These values are simple, memorable, and widely used in testing scenarios to create reproducible randomness. They serve as a standard practice for simulation and testing in computational experiments.

Conclusion

By choosing seeds 123, 456, and 789, we ensure:

1. Reproducible results for consistent evaluation.
2. Diverse scenarios for comprehensive testing.
3. Robust insights into the performance of the Viterbi algorithm under different policies.

---

4. Results

4.1. Accuracy Comparison

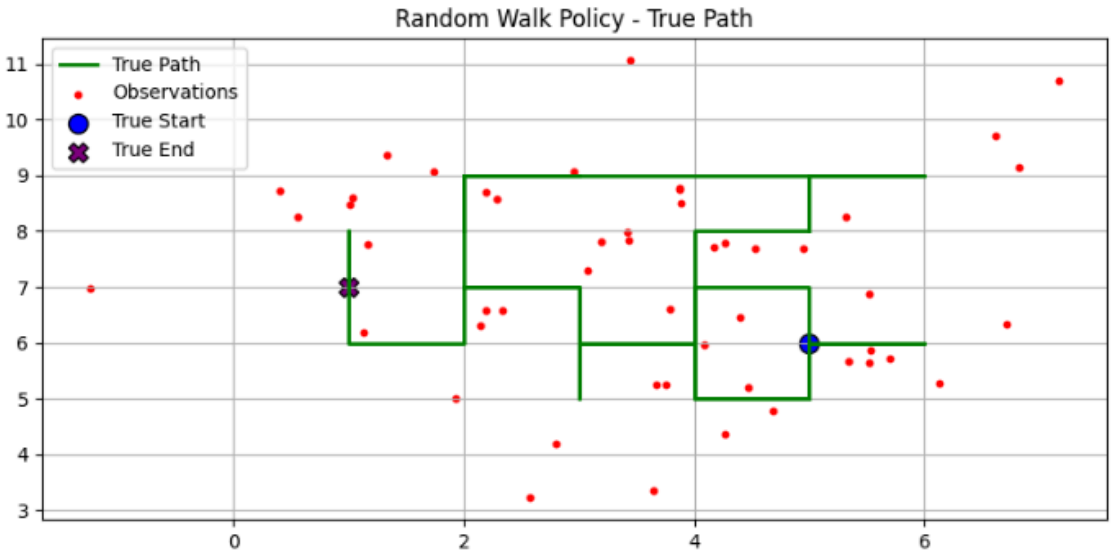
Seed	Policy	Tracking Accuracy (%)
123	Random Walk	30.00
123	Straight Until Obstacle	64.00
456	Random Walk	26.00
456	Straight Until Obstacle	54.00
789	Random Walk	24.00
789	Straight Until Obstacle	62.00

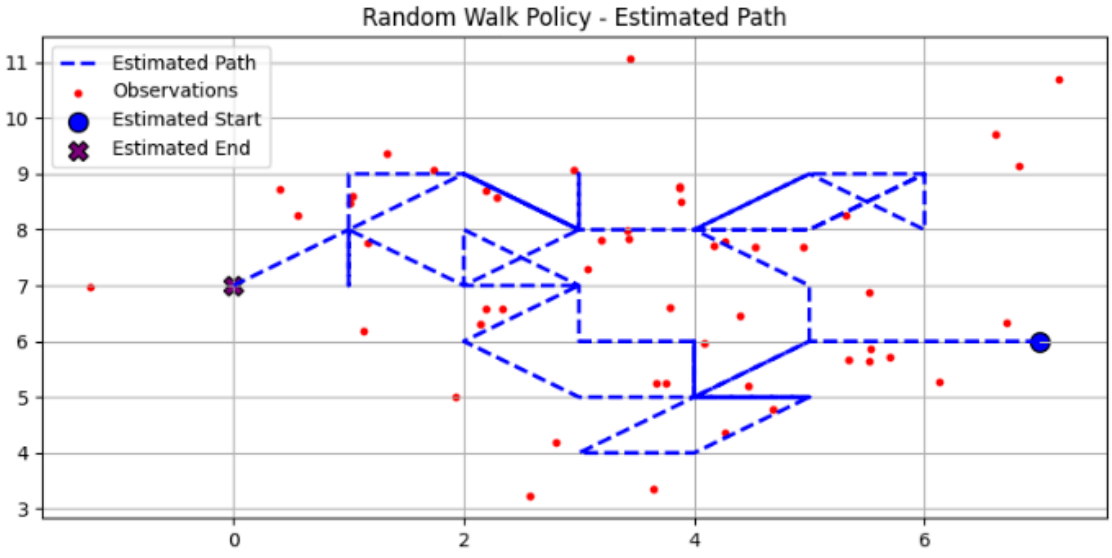
- The **Straight Until Obstacle Policy** consistently outperforms the **Random Walk Policy** in tracking accuracy.
- The deterministic nature of the Straight Until Obstacle Policy simplifies the estimation of the Roomba's path, whereas the stochastic behavior of the Random Walk Policy introduces significant uncertainty.

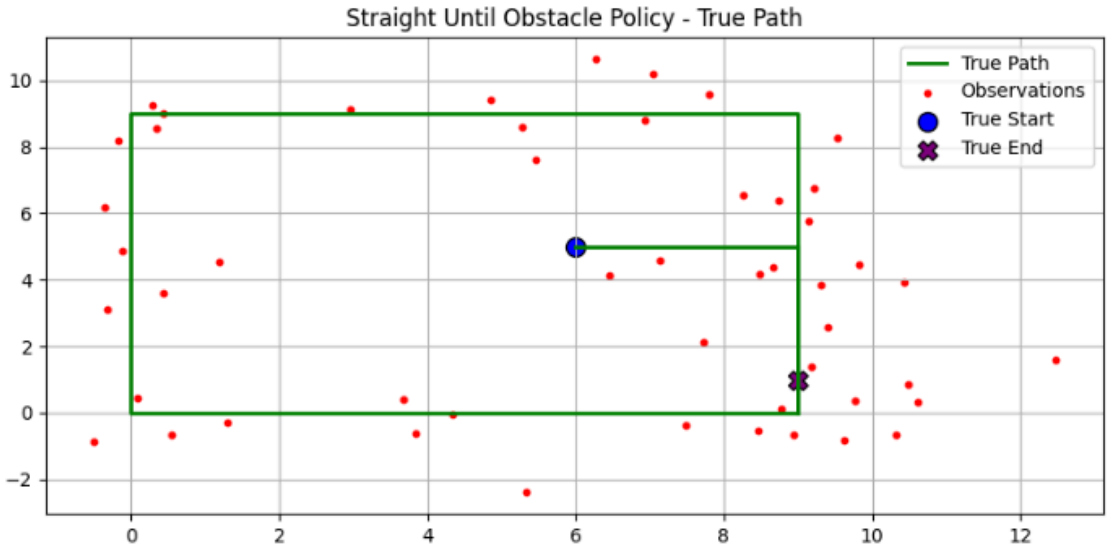
```
C:\Users\Sahil\Downloads\A3_2022427_AI\HMM_Question>C:\Users\Sahil\AppData\Local\Programs\Python\Python310\python.exe HMM_Question.py
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement: 100%|████████████████████████████████████████████████████████████| 50/50 [00:00<?, ?it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement: 100%|████████████████████████████████████████████████████████████| 50/50 [00:00<?, ?it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 30.00%
Saved plot: random_walk_Policy_Seed_123_Path.png

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 64.00%
Saved plot: straight_until_obstacle_Policy_Seed_123_Path.png
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement: 100%|████████████████████████████████████████████████████████████| 50/50 [00:00<?, ?it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement: 100%|████████████████████████████████████████████████████████████| 50/50 [00:00<?, ?it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 26.00%
Saved plot: random_walk_Policy_Seed_456_Path.png

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 54.00%
Saved plot: straight_until_obstacle_Policy_Seed_456_Path.png
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement: 100%|████████████████████████████████████████████████████████████| 50/50 [00:00<?, ?it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement: 100%|████████████████████████████████████████████████████████████| 50/50 [00:00<?, ?it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 24.00%
Saved plot: random_walk_Policy_Seed_789_Path.png

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 62.00%
Saved plot: straight_until_obstacle_Policy_Seed_789_Path.png
```

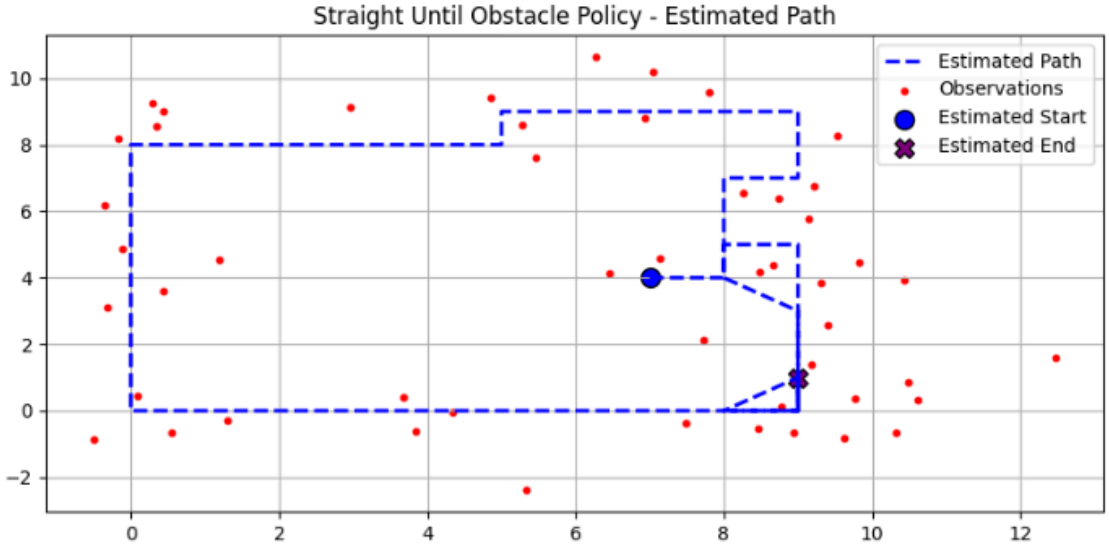
4.2. Graphical Results

For each seed value and policy, the following graphs illustrate the **True Path**, **Noisy Observations**, and **Estimated Path**:


Random Walk Policy - True Path
True Path
Observations
True Start
True End
11
10
9
8
7
6
5
4
3
0
2
4
6

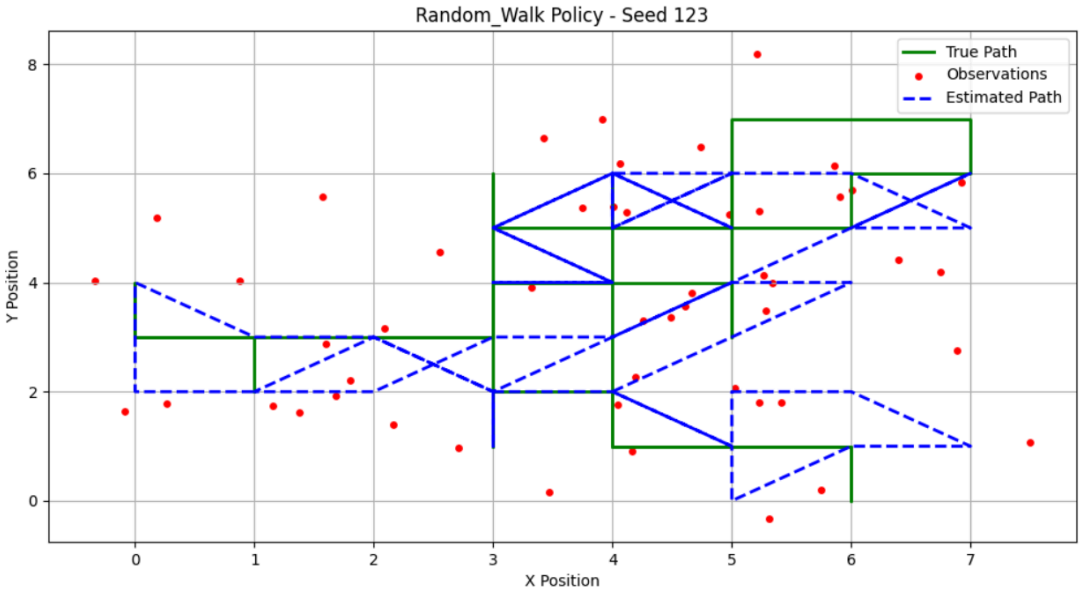

Random Walk Policy - Estimated Path
Estimated Path
Observations
Estimated Start
Estimated End
11
10
9
8
7
6
5
4
3
0
2
4
6


Straight Until Obstacle Policy - True Path
True Path
Observations
True Start
True End

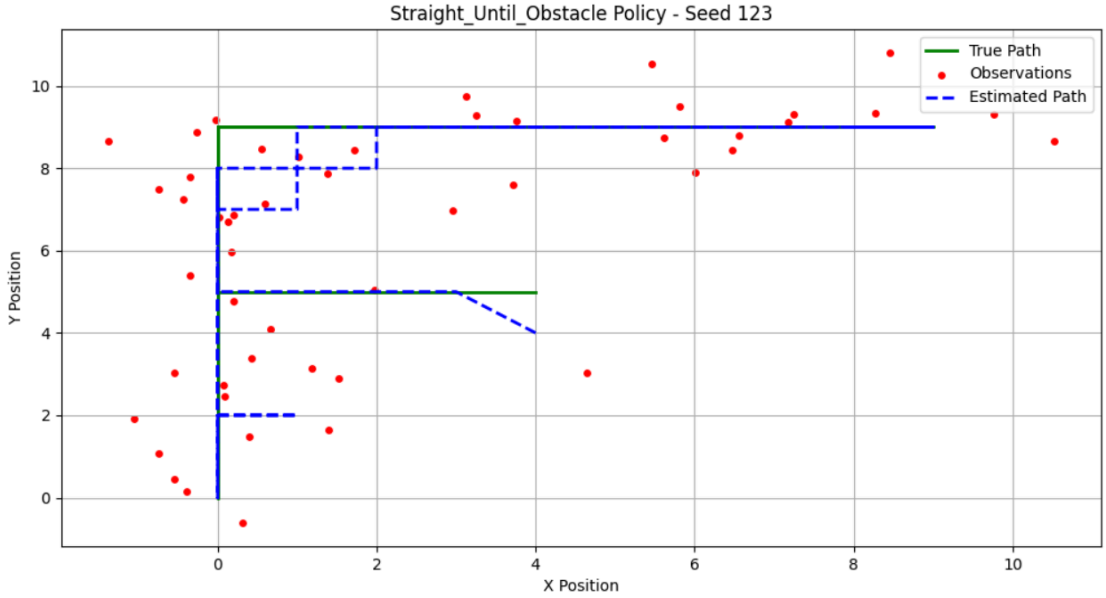

Straight Until Obstacle Policy - Estimated Path
Estimated Path
Observations
Estimated Start
Estimated End

4.2.1. Seed 123

- **Random Walk Policy**

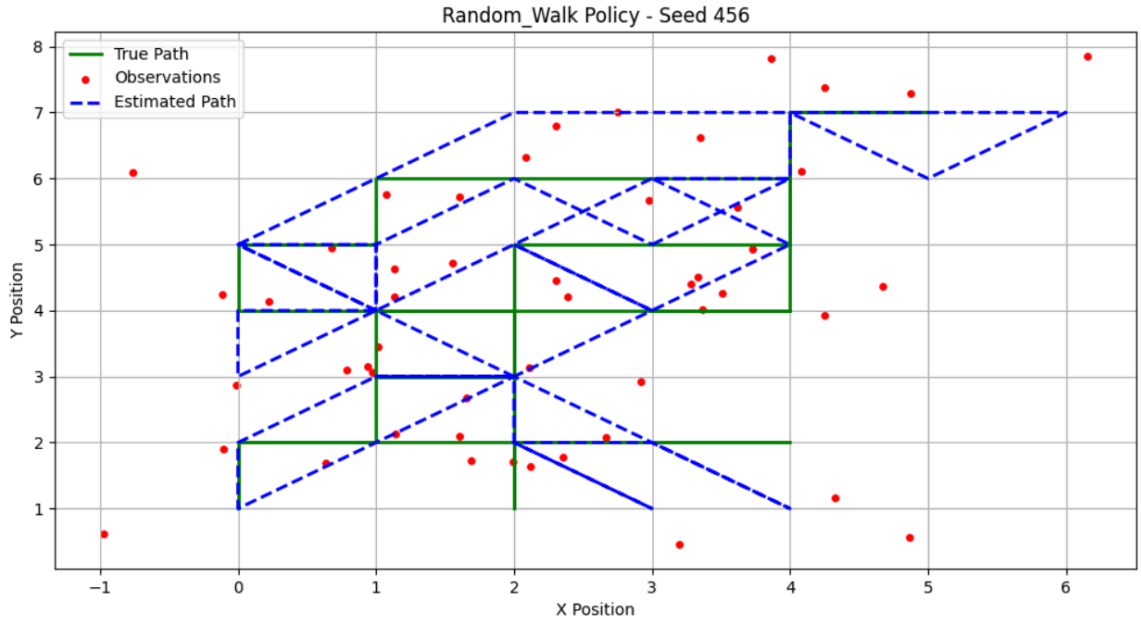


- **Straight Until Obstacle Policy**

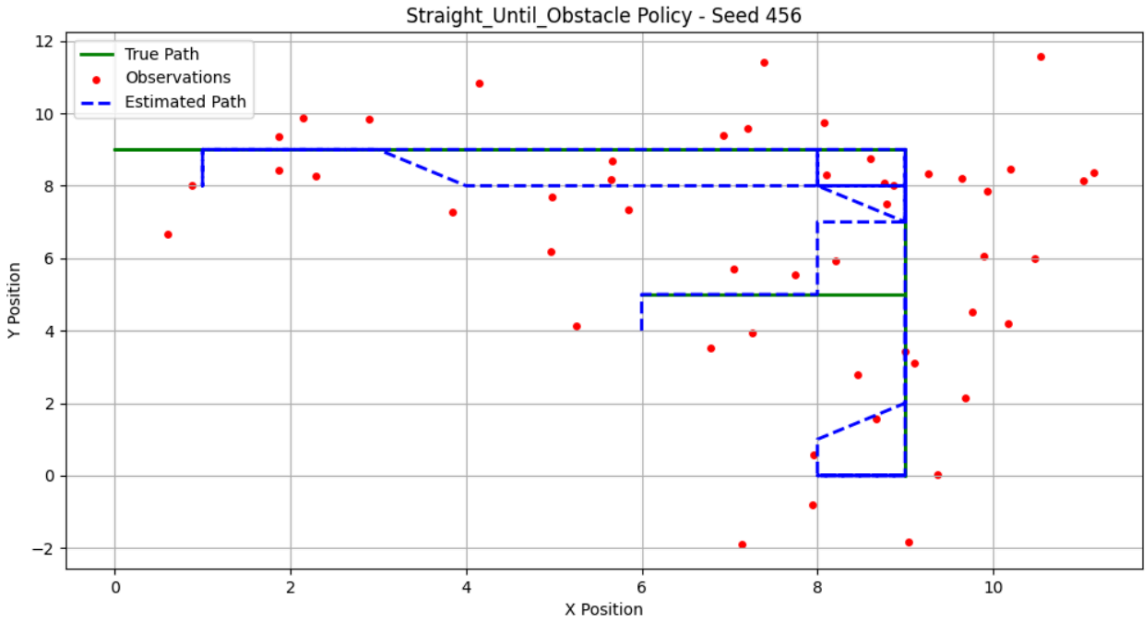


4.2.2. Seed 456

- **Random Walk Policy**

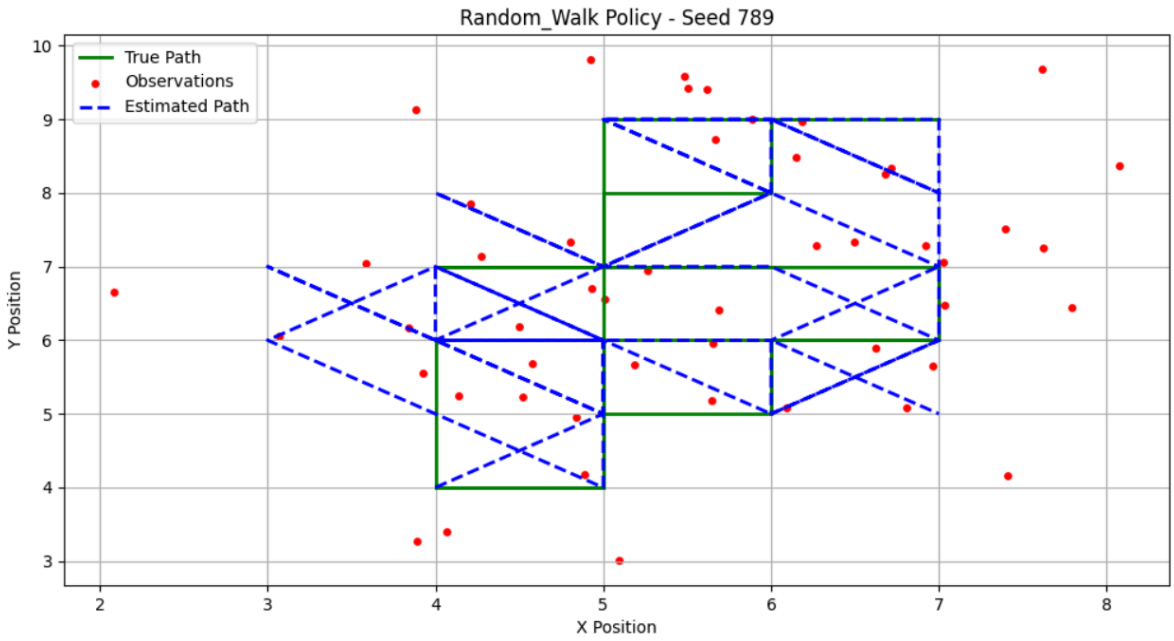


- **Straight Until Obstacle Policy**

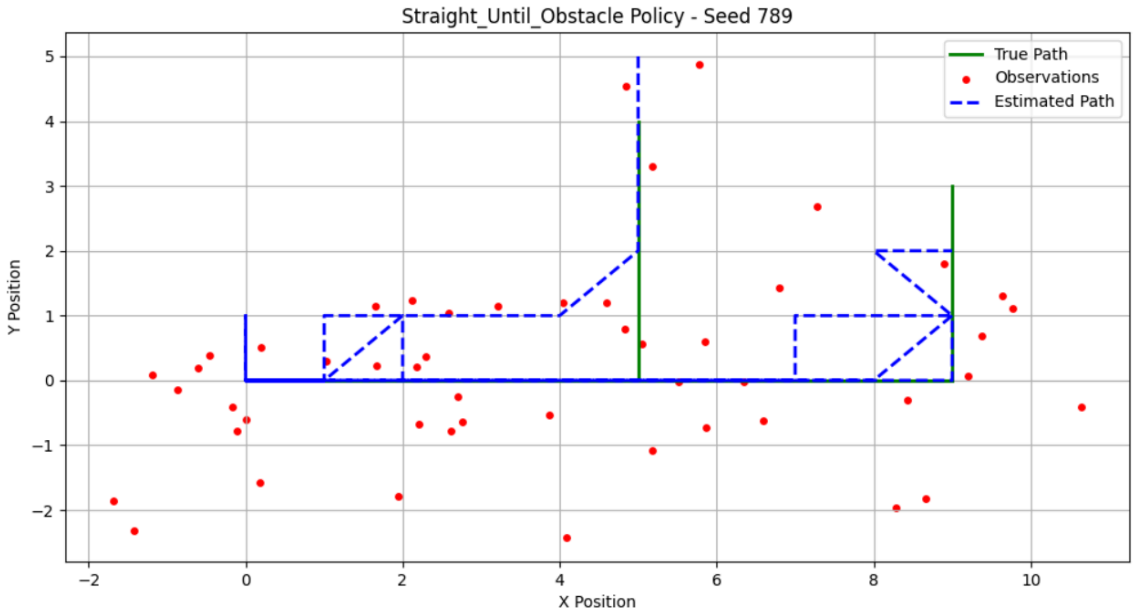


4.2.3. Seed 789

- **Random Walk Policy**



- **Straight Until Obstacle Policy**



---

5. Analysis

1. **Policy Comparison**:

- **Straight Until Obstacle Policy**:
  - The deterministic movement in straight lines until encountering an obstacle allows the Viterbi algorithm to better infer the Roomba's path.
  - Results in higher tracking accuracy (54-64% across seeds).
- **Random Walk Policy**:
  - The Roomba's random movement introduces significant uncertainty, making it challenging to estimate its path accurately.
  - Lower tracking accuracy (24-30% across seeds).

2. **Impact of Noise**:
   - The Gaussian noise in sensor observations (σ=1.0\sigma = 1.0σ=1.0) adds variability, reducing the accuracy of both policies.
3. **Seed Dependence**:
   - Variations in tracking accuracy across seeds highlight the sensitivity of the Viterbi algorithm to the initial conditions and movement patterns.

---

6. Conclusion

1. **Key Findings**:
   - The **Straight Until Obstacle Policy** is more accurate for tracking the Roomba's path compared to the **Random Walk Policy**.
   - Noise in sensor observations significantly impacts tracking accuracy, particularly for the Random Walk Policy.
2. **Recommendations**:
   - Reducing sensor noise (σ<1.0) could improve tracking accuracy.
   - Incorporating additional observations (e.g., obstacle proximity) could help disambiguate transitions in the Random Walk Policy.
3. **Future Work**:
   - Extend the model to include dynamic obstacles.
   - Explore advanced probabilistic tracking methods (e.g., Particle Filters) for improved accuracy.

---

7. Appendix

1. **Code Implementation**:
   - The complete code is included as part of the project submission.
2. **Estimated Paths CSV**:
   - The estimated paths for all seed values and policies are saved in the `estimated_paths.csv` file.
