Project III Report for COM S 4/5720 Spring 2025: Rotation-Aware Probabilistic Escape-Pursuit Planning

Jesus Soto Gonzalez

***Abstract*— Project III builds upon the dynamic pursue-escape setting of Project II by introducing probabilistic movement uncertainty through rotation-based deviations. The agent must adapt to potentially rotated actions while still planning robustly against moving opponents. This report describes an A*-based planner with probabilistic compensation via rotation estimation, combined with heuristics for spatial awareness, safety, and pursuit. I integrated a dynamic estimator using exponential moving averages and sliding windows to track transition noise and adapted A*-based routing for reactive decisions in real time. The resulting agent balances offensive and defensive behavior and avoids unstable traps while pursuing long-term goals.**

I. Introduction

In Project II, my agent simulated short-term futures using a shallow Monte Carlo Tree Search (MCTS) strategy, performing randomized rollouts to select the most promising direction. In Project III, the challenge changed: instead of rollout simulation, I had to account for noisy transitions, where the selected action may be rotated left or right. This required robust, adaptive planning capable of reasoning under uncertainty. I reintroduced A*-style shortest path planning and added compensation mechanisms to adjust actions based on observed transition bias

II. Strategy and Integration of Project II Concepts

Key ideas retained from Project II:

- Using grid-based heuristics like Manhattan distance and center bonuses.
- Avoiding walls and dead-ends to maintain flexibility and escape routes.
- Switching between pursuit and evasion depending on context.

III. New ideas for Project III included:

- Probabilistic rotation modeling via a hybrid estimator.
- A*-based shortest-path computation toward a dynamic target.
- Fallback movement scoring when pathfinding fails.
- Integration of three movement likelihoods (left, unchanged, right) into action evaluation.

IV. Development of Final Planner

A. Early Strategies

I tested various versions before finalizing the current strategy:

- **Static Greedy**: Always follow the A* path to the pursued. This failed under rotation noise.
- **Wall Avoidance Heuristic**: Penalized being near walls, but lacked pursuit behavior.
- **Random Correction**: Adjusted action after execution, which didn't work since the outcome had already occurred.

The final planner needed to not only estimate likelihoods of rotation error but also score actions in a way that respected those estimates while balancing pursuit, safety, and map control.

B. Final Strategy

The final agent uses:

- An A* pathfinding routine to determine ideal direction toward the goal.
- Rotation bias tracking via a sliding window and exponential moving average.
- Movement scoring that considers rotated variants of a direction.
- Mode switching between `defensive`, `offensive`, and `normal` based on proximity.

If A* yields a path, the agent chooses a direction and adjusts it based on rotation estimates. If no path exists, a fallback heuristic scores all actions using the weighted likelihoods of success.

This evaluation balances competing goals dynamically based on current surroundings, producing adaptive and robust behavior.

C. Tracking and Estimating Rotation Probabilities (p_1, p_2, p_3)

In this environment, moves don't always execute exactly as planned; they can randomly rotate left or right. To deal with this, the agent keeps track of how often each type of rotation happens. After every action, it compares the direction it tried to move with the direction it actually moved. If it was rotated to the left, it updates p_1; if no rotation, it updates p_2; and if it was rotated to the right, it updates p_3.

To estimate these probabilities reliably, the agent uses two tracking methods:

- A **sliding window** that stores the last 50 outcomes and updates quickly when the pattern changes.
- An **exponential moving average (EMA)** that changes more slowly and gives a smoother long-term trend.

These two trackers are blended using a weighting factor based on how many steps the agent has taken so far. Early on, the sliding window has more influence. Over time, the agent gradually shifts to trusting the EMA. The final combined

values for p_1, p_2, and p_3 are used to model how likely each kind of rotation is in future actions.

D. How p_1, p_2, and p_3 Affect Decisions

The rotation probabilities influence the agent's choices in two main ways.

First, when the agent finds a direction to move using A*, it needs to decide if that move is likely to succeed. If p_2, the chance of the move going as intended is high, it just proceeds. But if p_2 is very low (less than 0.2), the agent assumes the move will likely get rotated. It then compares p_1 (left rotation) and p_3 (right rotation). If one is much more likely, the agent anticipates it by selecting a move that will end up in the correct direction after the rotation occurs. If no rotation clearly dominates, it sticks with the original direction.

Second, if A* fails to find a usable path (for example, due to obstacles), the agent falls back to evaluating all directions. For each possible move, it considers the three outcomes: going straight, rotating left, or rotating right. Each of these is scored using the usual heuristics (distance to agents, wall penalties, center bonus, etc.), and the total score is a weighted average based on p_1, p_2, and p_3. This lets the agent pick actions that are most likely to result in good positions, even if the action gets rotated.

In both cases, the agent is not guessing. It is using learned data to plan around rotation noise and improve its decision-making.

E. Example: Planning with Rotation in Mind

Let's say the agent wants to move north, but based on recent experience it has learned:

- $p_1 = 0.2$ (left rotation is uncommon),
- $p_2 = 0.1$ (it rarely goes straight),
- $p_3 = 0.7$ (right rotation is very likely).

Since p_2 is low, the agent expects the move won't execute as planned. Because p_3 is highest, it plans ahead by choosing a move that, if rotated right, will still move it north. This way, the agent gets closer to its target even if rotation occurs.

In a different situation, if the probabilities were:

- $p_1 = 0.3$, $p_2 = 0.4$, $p_3 = 0.3$,

the agent would likely trust the original direction because going straight is reasonably likely and the two rotation directions are balanced.

These examples show that the agent isn't just reacting to mistakes; it actively adjusts its choices based on what's likely to happen. This leads to smarter, more reliable behavior in unpredictable environments.

V. Implementation Details

A. plan_action

Each timestep, `plan_action()` performs the following:

- Updates the estimator if the previous move changed due to rotation.
- Predicts the pursuer's next position using velocity.
- Switches mode based on distance: defensive (evade), offensive (chase), or normal.
- Attempts A* pathfinding to the selected target. If successful, chooses the next move adjusted by rotation estimates.
- If no path is found, scores all candidate moves and selects the best one based on expected value.

B. evaluate_moves

The fallback movement evaluation works the following way:

- For each of the 8 base directions, evaluate all three possible outcomes (left, unchanged, right).
- Score each using weighted likelihood (p1, p2, p3).
- Heuristics include distance to pursued or pursuer, wall proximity, and center alignment.
- Apply special bonuses or penalties for closeness, being cornered, or reaching the pursued.

The action with the highest expected score is chosen, using randomized tie-breaking

C. update_estimators

Whenever a move is executed, this function updates the estimator:

- Tracks if the action was rotated left, right, or unchanged.
- Updates exponential moving average.
- Updates sliding window with weighted frequency.

Over time, the agent adapts to persistent rotation noise patterns.

D. a_star and get_escape_target

The `a_star()` function computes the shortest path from the current location to a target tile using Manhattan distance as the heuristic. When evading, the agent selects the furthest walkable tile from the pursuer using `get_escape_target()`.

E. is_valid_move, is_robust_move

- `is_valid_move()` ensures a move stays within bounds and avoids walls.
- Special checks prevent diagonal corner-cutting.
- `is_robust_move()` verifies that a move and its rotated variants are all valid.

F. wall_penalty, center_bonus, manhattan

- `wall_penalty()` penalizes positions next to edges or surrounded by obstacles.
- `center_bonus()` rewards positions closer to the map center.
- `manhattan()` provides distance estimates for scoring and A*.

VI. Results and Observations

Testing showed the agent was able to:

- Escape threats in noisy environments.
- Pursue targets intelligently when safe.
- Maintain safe and central positions when uncertain.
- Handle action rotation without losing stability or strategy.

It consistently outperformed static or naive A*-based agents that lacked compensation or heuristics.

VII. Conclusion

This project introduced realistic uncertainty into planning by allowing actions to rotate. I designed an agent that learns rotation tendencies and uses weighted action evaluation to maintain robust, adaptive behavior. Combined with A* for ideal targeting, fallback heuristics for safety, and behavior switching modes, the agent performs well across many map types and adversaries.

References

1 P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, 1968.

2 B. Weng, "Notes on A-Star Searching," Department of Computer Science, Iowa State University, March 1, 2025.