Tejas Shetty
u7984216

COMP8620 - Advanced Topics in Artificial Intelligence

Assignment 2

Part A

1. Effect of Discount Factor

In Value Iteration, the discount factor γ controls how much future rewards influence the value of each state. A higher γ means the agent cares more about future outcomes, while a smaller γ makes it more short sighted.

When γ is large (close to 1), changes in the value function propagate more slowly through the state space, because each update depends on many future rewards. As a result, the algorithm needs more iterations before the values stabilise. In contrast, when γ is small, only immediate rewards matter, so the value function converges much faster.

Therefore, as γ increases, the lower bound on the number of iterations required for Value Iteration to reach an ε-optimal solution increases. Convergence becomes slower because the algorithm places greater emphasis on long-term rewards.

2.

A) What to pass to Algorithm 1 after discretising the POMDP:

Each state is a belief (a probability distribution over the original states). A POMDP can be viewed as an MDP in the belief space, where beliefs are distributions over states and the policy maps beliefs to actions.

Inputs to Algorithm 1 (ValueIteration⟨S,A,T,R⟩) for the discretised belief MDP:

- S (states): a finite set of beliefs B = {b1,…,b_M} obtained by discretising or covering the reachable belief space (e.g., grid/sampling/reachable set). Beliefs are distributions over the POMDP state space. The belief space is a simplex of dimension $|S^P|$ - 1.
- A (actions): exactly the POMDP's action set A = A^P.

- T (transition): for each b ∈ B and a ∈ A, branch on observations o ∈ O^P compute the Bayes-updated belief b′ = τ(b,a,o), and then project b′ to the nearest node in (or include it if growing B). The probability of reaching b′ is the marginal observation probability P(o | b,a) aggregated over all o that project to b′.
- R (reward): the expected immediate reward under the belief, i.e., R(b,a) = $\Sigma^s b(s)R^P(s,a)$

These map directly to Algorithm 1's inputs and inner loop (lines 4-5) once S is the discretised belief set and T, R are as above.

B) Time complexity of the inner loops:

- For one belief b and one action a, we must consider all observations o ∈ O^P, compute the observation likelihood P(o ｜ b,a), and perform a Bayes belief update to get τ(b,a,o).
    - A straightforward (uncached) implementation touches the original state space in nested sums, which is O($|S^P|^2$) per observation.
    - Doing this for all observations gives O($|O^P| . |S^P|^2$) per (b,a).
- Taking the max over all actions multiplies by $|A^P|$.

So the inner update per belief costs:

- Naïve (no caching / factoring): O($|A^P| \cdot |O^P| \cdot |S^P|^2$).
- With common caching/vectorisation (e.g., precompute $b^\top T^P$ so updates are single passes): O($|A^P| . |O^P| . |S^P|$).

3.

Given:

- Each move (L/R/U/D) costs -1.
- stay is 0.
- Reaching the goal gives a one-off R_g, then the process ends.
- The number of moves to reach the goal is uncertain in [8, 15]. Discount factor is γ=0.95.

For the optimal policy to favor moving towards the goal, the expected discounted return of the shortest path to the goal must be greater than the expected return of the staying policy (i.e., staying at the start cell forever).

- Optimal Value of Staying at s(V_{stay}): If the robot stays at s forever, the only reward it receives is the immediate reward for staying, which is 0.

$$V_{stay} = R_{stay} + \gamma R_{stay} + \gamma^2 R_{stay} + \cdots = 0 + \gamma(0) + \gamma^2(0) + \cdots = 0$$

- Optimal Value of Moving to g(V_{reach}): The path to the goal involves a sequence of moves, each costing -1, followed by the final goal reward Rg. To ensure the robot reaches g rather than staying, we should consider the worst case path that the robot would still prefer over staying. The worst case is the path that yields the lowest return, which is the path with the maximum number of steps (N_{max} = 15) before the goal is reached.

  The discounted return for reaching the goal in N steps is:

$$V_N = \text{Total move cost} + \text{Discounted goal reward}$$

$$= (-1) + \gamma(-1) + \cdots + \gamma^{N-1}(-1) + \gamma^N R_g$$

  The move cost is a geometric series:

$$\sum_{k=0}^{N-1} \gamma^k(-1) = -1 \text{ x } \frac{1-\gamma^N}{1-\gamma}$$

  For the worst case, N=15 steps.

$$V_{15} = -\frac{1-\gamma^{15}}{1-\gamma} + \gamma^{15} R_g$$

Solving for Rg:

To ensure that the robot reaches the goal, the lowest possible value of the path (V_{15}) must be strictly greater than V_{stay}:

$$V_{15} > V_{stay}$$

$$-\frac{1-\gamma^{15}}{1-\gamma} + \gamma^{15} R_g > 0$$

Using $\gamma = 0.95$:

$$1 - \gamma = 1 - 0.95 = 0.05$$

$$\gamma^{15} = (0.95)^{15} \approx 0.4633$$

$$-\frac{1-0.4633}{0.05} + 0.4633 R_g > 0$$

$$-10.734 + 0.4633 R_g > 0$$

$$0.4633 R_g > 10.734$$

$$R_g > \frac{10.7343}{0.4633} \approx 23.168$$

The reward for reaching the goal should be R_g > **23.168** to ensure the optimal policy chooses to move to the goal, even in the worst case path of 15 steps.

4. Model of smartwatch coach as a POMDP

POMDP P = $\langle S,A,O,T,Z,R,\gamma,b_0 \rangle$

States S:

Latent "user motivation type" (what the model must infer):

$$S = \{High, Medium, Low\}$$

Actions A:

Notification tone/intensity choices:

$$A = \{Gentle, Warning, Stern\}$$

Observations O:

What the watch detects in the hour after the prompt (coarsened activity):

$$O = \{HighAct, MedAct, LowAct\}$$

Transition $T(s' \mid s,a)$:

Simplest stationary type (keeps the focus on partial observability rather than nonstationarity):

$$T(s' \mid s,a) = 1[s' = s].$$

Observation model $Z(o \mid s,a)$:

Two-stage view: each latent type induces a true typical activity level, then the sensor is noisy.

- Map type -> true activity (action independent baseline):
    - High -> HighAct, Medium -> MedAct, Low -> LowAct.
- Sensing noise (given true activity): correct with prob 0.6, each of the two off-diagonals with 0.2.

  Thus a compact specification:

$$Z(o \mid s,a) = \begin{cases} 0.6, & o = Activity(s) \\ 0.2, & o \neq Activity(s) \end{cases}$$

Reward R(s,a):

Encodes matching the tone to the person and value of eliciting activity, plus a small interaction cost. One consistent scheme:

- Matching matrix (tone ↔ type):
  Gentle - High: +2,
  Warning - Medium: +2,
  Stern - Low: +2.
  mild mismatch: 0
  strong mismatch (e.g., Stern - High): -1.
- Activity bonus by type (proxy for health/engagement): High: +3, Med: +1, Low: 0.
- Message cost: -0.1 per prompt.

Combined: R(s,a) = Match(s,a) + ActBonus(s) - 0.1

Discount γ:

We can use $\gamma = 0.95$ to align with the rest of Part A.

Initial belief b_0:

If the user is new or unknown: $b_0 = (1/3, 1/3, 1/3)$.

Part B

1. MDP framing for centralized multi-drone navigation

We need an MDP ⟨S,A,T,R,γ⟩ for centralised, collision-free navigation on a 3D grid with non deterministic action effects (wind), where each drone can move to one of its 26 neighbours per step.

State space S:

A state is the joint grid positions of all N drones:

- S ={x = (x1,…,xN)} where $x_i \in$ G is a 3D integer cell (x,y,z) inside the grid,

  with the constraints:
  - $x_i \notin$ O(obstacle cells), and
  - $x_i \neq x_j$ for i ≠ j (no two drones in the same cell)

Action space A:

A joint action is a Cartesian product of per-drone moves:

- Per-drone primitive moves: the set $\Delta = \{(\delta_x, \delta_y, \delta_z) \in \{-1,0,1\}^3 \setminus \{(0,0,0)\}\}$.
  - If altitude changes are allowed, $|\Delta| = 26$.
  - If altitude changes are disallowed, restrict to the 8 neighbours in the xy plane (dz=0).
- Joint action a = $(a_1,\ldots,a_N) \in \Delta^N$. (In code it is often encoded as an integer index in [0, $|\Delta|^N$ - 1].)

Transition function T(s,a,s′):

Stochastic due to action uncertainty and collisions:

A. Independent per-drone success model (T_{known}): For each drone i, the intended neighbour $x_i + a_i$ is reached with probability $p_i(\alpha, x_i, a_i)$ and some other feasible neighbour (including staying or side slips) with the remaining probability mass split according to your environment's uncertainty model.

B. Obstacle handling: If a sampled target is an obstacle cell, either (a) convert to “no move”, or (b) transition to a terminal collision state.
C. Inter drone collision: If two (or more) drones land in the same cell at the same time, transition to a terminal collision state.
D. Goal reaching or absorption: If every drone is at its goal, transition to an absorbing success state.

Formally, T factors as a product of per drone categorical distributions followed by a deterministic projection that resolves obstacles, checks joint collisions, and maps to terminal/success absorbing states where appropriate.

Reward function R(s,a,s′):

- Use shaped, per-step rewards consistent with the config:
  - Step cost: $c_{step} < 0$ every time step (encourages shorter solutions).
  - Goal reward: $r_{goal} > 0$ granted when all drones reach their goals (absorbing state).
  - Collision penalty: $r_{coll} < 0$ upon entering a collision terminal state.
- These map directly to the YAML fields: step_cost, goal_reward, collision_penalty.

Discount factor γ:

$\gamma \in (0,1)$. Using as given. This ensures convergence and encodes a preference for sooner success.

Termination:

- Success absorbing state (all goals reached).
- Collision absorbing state (obstacle hit or inter-drone collision).
- Hard step cap (e.g., max_num_steps). Exceeding it transitions to a terminal timeout with zero or small negative reward.

2. Design of an Approximate Online Solver

Chosen approach:

For this task, I am choosing to design an online approximate solver based on Monte Carlo Tree Search (MCTS) using the Upper Confidence Tree (UCT) algorithm with progressive widening.

This method performs real time decision making in stochastic multi drone navigation by repeatedly simulating future outcomes using the environment's transition model T(s,a,s′) and updating action value estimates online.

The UCT planner incrementally builds a search tree rooted at the current joint state of all drones.

At each decision step, it uses the generative model (env.simulate(state, action)) to sample transitions and estimates the expected discounted return of actions.

After a fixed computation time, the action with the highest estimated value at the root is executed in the environment.

Algorithm overview:

The planner follows the standard MCTS four phase cycle, modified for large stochastic action spaces:

A. Selection:

   Starting from the current state, the algorithm recursively selects actions that maximise the UCT criterion:

$$a^{*} = argmax_{a}[Q(s,a) + c_{uct}\sqrt{\frac{ln\,N(s)}{N(s,a)+1}}]$$

   where N(s) is the state's visit count and N(s,a) is the number of times action a was tried. This balances exploration of new actions and exploitation of high-value actions.

B. Progressive Widening:

   To control branching in the large joint-action space ($|A| = 8^{N}$ or 26^{N}), new actions are only added when

$$|A_{s}| \leq C_{pw}N(s)^{\alpha_{pw}}$$

with constants C_{pw} = 1.5 and α_{pw} = 0.5.

This ensures the number of explored actions grows sub-linearly with the number of visits, keeping computation tractable as more drones are added.

C. Expansion:

When widening allows, a previously untried joint action is sampled and added as a new edge in the tree.

This node is initialised with Q(s,a) = 0 and N(s,a) = 0.

D. Simulation (Rollout):

From the newly expanded state, a rollout of up to 60 steps is simulated using an ε-greedy policy:

a. with probability ε (0.2), a random action is taken, otherwise
b. the algorithm samples k (= 8) candidate actions and picks the one with the highest immediate reward.

This approximates short term goal directed behaviour without an explicit heuristic.

E. Backup:

The total discounted return $G = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$ is propagated backwards along the visited path.

Each node's statistics are updated incrementally:

$$Q(s,a) \leftarrow Q(s,a) + \frac{G - Q(s,a)}{N(s,a)}$$

allowing the action values to converge to expected returns over many simulations.

F. Action Selection:

After the per-step planning time expires (e.g., 1 s), the root node's action with the highest Q(s,a) is returned and executed.

The tree is then reused for the next step, enabling continuous online replanning as new observations arrive.

Pseudo-code:

```
function PLAN(current_state, time_budget):
    root ← NODE(current_state)
    while time < time_budget:
        s ← current_state
        path ← []
        while not terminal(s) and depth < expand_limit: #selection and expansion
            n ← NODE(s)
            if |A_tried(s)| < C_pw * N(s)^α_pw:
                a ← new random untried action
            else:
                a ← argmax_a [ Q(s,a) + c_uct * sqrt(ln N(s)/N(s,a)) ]
            s', r, done ← simulate(s, a)
            path.append((s, a, r))
            s ← s'
            if done: break
        G ← simulate_rollout(s, ε, k, horizon) #Rollout
        for (s_i, a_i, r_i) in reversed(path): #Backup
            G ← r_i + γ * G
            update Q(s_i,a_i) and counts
    return argmax_a Q(root,a)
```

Rationale:

- It is model based and online, ideal for stochastic environments where transitions are known only through simulation.
- Progressive widening allows MCTS to scale to multi-drone joint actions without exhaustive enumeration.
- The ε-greedy rollout provides a simple but effective balance between exploration and goal oriented exploitation.
- The approach naturally supports any number of drones and adapts to varying per-step time budgets.

Expected behaviour:

The planner searches deeper around the most promising joint actions while maintaining stochastic robustness.

With larger planning time or smaller uncertainty, trajectories become smoother and more coordinated. Under tight budgets, drones may reach their goals at different times.

3. Implementation of the Online Approximate Planner

Algorithm implemented:

I implemented a Monte Carlo Tree Search (MCTS) variant, specifically the Upper Confidence Tree (UCT) algorithm with progressive widening and an ε-greedy rollout policy.

This algorithm plans online in real time by repeatedly simulating the effects of possible joint drone actions using the environment's stochastic transition model env.simulate(state, action).

Each node in the tree represents a joint state of all drones. The algorithm expands actions gradually (progressive widening) to limit branching when the joint action space is large, and estimates the action values Q(s,a) from discounted simulation returns.

Key Implementation Details:

Component	Description
Selection	For a visited state s, actions are chosen using the UCT rule: $a^* = argmax_a[Q(s,a) + c_{uct}\sqrt{\frac{ln\, N(s)}{N(s,a)+1}}]$
Progressive Widening	Limits the number of expanded action
Expansion	When widening allows, a previously untried joint action is added to the tree and initialised with Q = 0
Simulation/Rollout	From the expanded leaf, the algorithm performs a stochastic rollout for up to 60 steps using an ε-greedy policy, with probability 0.2 choose a random action, otherwise choose among k = 8 sampled actions with the highest immediate reward.

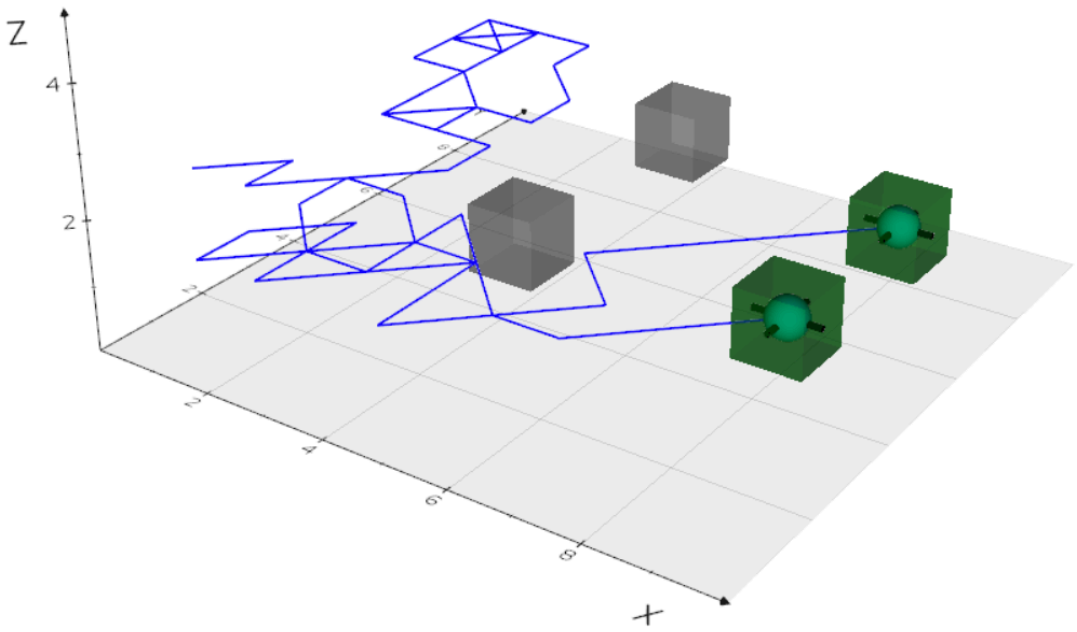
Backup	The total discounted return G = $r_0 + \gamma r_1$ +… is propagated back along the visited path to update Q(s,a) via incremental mean updates.
Action Choice	After the per-step time budget expires, the action with the highest mean Q(s,a) at the root is executed.
Hyperparameters	$c_{uct} = 1.5,\ C_{pw} = 1.5,\ \alpha_{pw} = 0.5, \gamma$ from YAML config, rollout horizon = 60.

Integration with Provided Framework:

- The planner is implemented in myplanner.py as the class UCTPlanner.
- It interfaces with run_planner.py exactly through the expected API:
    - plan(current_state, planning_time_per_step)
    - Uses env.simulate(state, action) to sample transitions.
- No external dependencies beyond NumPy. Stochastic transitions and rewards are fully handled by the environment.

Execution Results:

- Configuration: two drones on a 10 × 10 × 5 grid with wind stochasticity (α = 0.5), XY-only movement (8-neighbour).
- Run settings: planning_time_per_step = 1.0 s.
- Outcome: success: True, Total discounted reward: 54.17
- The 3D trajectory plot (Figure below) shows both drones (green) successfully reaching their respective goal regions while avoiding obstacles (grey cubes).
  One drone reached the target earlier, while the other took a longer, safer route. Expected behaviour due to stochastic action effects and limited search depth.



Summary:

The implemented UCTPlanner successfully performs online planning in a stochastic multi-drone environment.

It demonstrates collision free trajectories and convergence to goal states, validating the theoretical design from Q2.

4. Scalability with Increasing Action Space

Experimental Setup:

I fixed drone motion to 8-neighbour XY-plane movement (change_altitude = False) so that each drone had 8 possible actions per step. The joint action space therefore grows exponentially with the number of drones ($|A| = 8^N$).

I used the map and parameters, set up in the same file (step_cost = -1.0, goal_reward = 100.0, collision_penalty = -50.0, discount = 0.98), and a constant per-step planning budget of 1 second.

For each $N \in \{1,2,3,4\}$, the environment was run 20 times with different random seeds, and obtained the mean ± 95 % confidence interval (CI) for total discounted reward, steps to termination, and success rate.

Results:

No. of Drones	Total Discounted Reward (± 95 % CI)	Steps (± 95 % CI)	Steps (± 95 % CI)
1	43.7 (34.3 - 53.1)	25.4 (19.8 - 31.0)	100 % (≈100 - 100 %)
2	-9.0 (-32.1 - 14.1)	48.7 (38.7 - 58.6)	55 % (33 - 77 %)
3	-53.7 (-78.6 - -28.7)	31.8 (20.5 - 43.1)	15 % (0 - 30 %)
4	-83.3 (-98.2 - -68.4)	23.7 (13.7 - 33.6)	0 % (0 - ≈0 %)

(n = 20 trials per setting, planning_time_per_step = 1 s) (results obtained are in q4_xyonly_results.json)

Analysis:

- 1 Drone: The planner consistently achieved the goal, yielding a high positive discounted reward and a tight confidence interval. UCT easily explores all 8 actions.
- 2 Drones: The joint action space expands to $8^2 = 64$. With the same time budget, UCT explores only a small subset of joint actions, leading to lower average reward, nearly doubled path length, and reduced success (≈ 55 %).

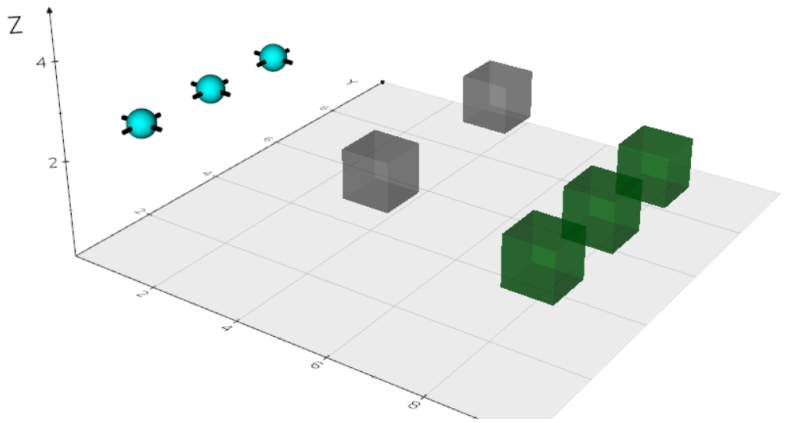
- 3 to 4 Drones: The branching factor ($8^3 = 512$, $8^4 = 4096$) overwhelms the limited simulations per second. The planner's rollouts seldom reach successful joint trajectories, causing strongly negative rewards and near zero success rates.

Summary:

The experiment demonstrates the curse of dimensionality in joint action planning. As the number of drones increases, the effective sample density per joint action decreases exponentially, and the UCT estimates become increasingly noisy.

Although progressive widening restricts expansion to $|A_z| \leq C_{pw} N(s)^{\alpha_{pw}}$, it only mitigates the branching growth, as it does not eliminate it. With a fixed per-step time limit, the planner prioritises a few early actions, often yielding sub optimal coordination (some drones stagnate or collide).

Performance of the UCT based solver degrades sharply as the joint action space increases, confirming that fixed budget online planning scales poorly with the number of cooperating drones.



5. Effect of Planning Time per Step

Experimental Setup:

I used the same two drone XY-only environments as in Q4, so each drone had 8 possible actions and the joint action space contained 64 combinations per step. The per-step planning budget T was varied to examine how longer computation times affect performance: $T \in \{0.25, 0.5, 1.0, 2.0, 3.0, 4.0\}$. Each configuration was run 20 times with different random seeds.

Results:

Planning Time (s)	Discounted Reward (± 95 % CI)	Steps (± 95 % CI)	Success Rate (± 95 % CI)
0.25	-61.8 (-82.1 - 41.5)	31.2 (22.5 - 39.9)	15% (0 - 31%)
0.5	-33.0 (-56 - -9)	42.8 (30.9 - 54.7)	35% (14 - 56%)
1.0	-20.5 (-41.5 - 0.5)	48.3 (36.7 - 59.8)	45% (23 - 67%)
2.0	20.0 (-8.2 - 48.2)	33.1 (25.0 - 41.1)	70% (50 - 90%)
3.0	13.0(-18.1 - 44.2)	25.6 (19.0 - 32.1)	50% (28 - 72%)
4.0	33.1 (8.6 - 57.6)	31.6 (26.1 - 37.1)	65% (44 - 86%)

(20 trials per setting, discount factor = 0.98) (results obtained are in q5_timebudget_results.json)

Analysis:

- Reward increases with planning time:
  Mean discounted reward increases steadily from around -62 (at 0.25 s) to ≈ +33 (at 4 s).
  Longer budgets let UCT perform more rollouts leading to better action value estimates, further leading to more goal directed decisions.
- Success rate increase:
  Success rises from about 15 % to 65 %.
  With more samples, progressive widening explores more actions and the planner avoids local failures or collisions.

- Steps decreased and stabilised:
  Average episode length shortens (≈ 49 -> 31 steps) as plans become more efficient.

Initial Increase (0.25s to 1.0s): The number of steps initially increases from 31.2 to 48.3. This suggests that with minimal planning time, episodes often terminate early due to a collision failure (which has a high negative penalty) rather than a successful long-term path.
Significant Decrease (1.0s to 4.0s): As the planning time increases from 1.0s to 2.0s and 4.0s, the average episode length drops sharply (from 48.3 to 31.6 steps). This decrease indicates that the planner is:

- Achieving Success: It is now successfully reaching the goal state, which typically requires fewer steps than an uncoordinated, failed path.
- Finding Optimal Paths: The increased computation time allows UCT to explore deeper and identify more efficient, coordinated trajectories, resulting in shorter average paths to the goal

UCT's performance is bounded by how many simulations per decision it can afford.
Increasing planning_time_per_step linearly increases the number of rollouts, which improves both the breadth of explored actions and the depth of lookahead before rollout. This reduces value variance and yields higher-quality policies.

However, gains saturate beyond ≈ 2 s per step: after the most promising actions have been well explored, additional samples offer diminishing returns.
Although the empirical success rate shows minor fluctuations (e.g., a dip at 3 s), the overall trend demonstrates that increasing the planning budget substantially improves performance.
Mean discounted reward increases steadily from -61 to +33, and success rate rises from 15 % at 0.25 s to around 65 to 70 % at longer time budgets.
The small non monotonic variation is within the 95 % confidence intervals and likely due to stochastic variance in the environment.

Conclusion:

The experiment confirms that more computation time directly improves online planning quality. While Q4 showed performance degrading with larger joint action spaces, Q5 demonstrates that this degradation can be partially offset by longer planning budgets, validating the expected trade off between solution quality and real-time constraints in stochastic multidrone MDPs.

6. Limitations

A. Computational scalability:

- The most significant limitation of the current solver is its poor scalability with joint action space size. Because the planner treats all drones jointly, the branching factor grows exponentially (8^N for XY-only motion).
- Even with progressive widening, the number of actions that can be explored within a fixed per step time budget grows sub linearly with visits, meaning that as N increases, a smaller fraction of actions are ever evaluated.
- This explains the rapid drop in success rate in Q4 as drones increase and the failure of four drone scenarios.

B. Limited planning budget:

- The planner is strictly time bounded, so performance depends heavily on planning_time_per_step.
- In Q5 we observed that increasing the budget from 0.25s to 4s improved results substantially. However, real time multi robot systems often cannot afford such long compute times per decision.
- The algorithm therefore struggles to maintain performance under tight real-time constraints.

C. Shallow rollouts and lack of heuristics:

- Rollouts use an ε-greedy random policy based only on one step reward, with no heuristic guidance toward the goal.
- This limits the effective search depth of UCT and increases variance in value estimates.
- In larger or more cluttered maps, rollouts often terminate before reaching a goal, producing poor or noisy backups.
- Adding heuristic rollouts (e.g., distance to goal shaping or learned value estimates) would greatly improve sample efficiency.

D. Centralised control assumption:

- The planner assumes a fully centralised decision maker that observes all drones simultaneously and selects a single joint action.
- In realistic multi robot systems, this would require high bandwidth communication and synchronisation, and is computationally expensive.
- Distributed or hierarchical planning (e.g., per-drone UCT with coordination constraints) would be more practical.

E. Stochastic transition simplification:

- The environment models uncertainty as an independent per drone noise parameter alpha, but does not model correlated wind effects or partial observability.
- In the real world, disturbances would affect all drones jointly and sensor noise would make states only partially observable.
- The current planner ignores this, so performance may not transfer directly to real systems without extensions such as belief space planning.

F. Evaluation and Implementation limitations:

- Small sample size (20 trials) means large confidence intervals. More repetitions would improve statistical reliability.
- Only XY-plane motion was tested for most experiments. True 3D motion would further increase complexity and expose additional scalability issues.
- The planner does not reuse parts of its tree between time steps (no tree carry over), losing valuable information after each move.

Acknowledgements:

I used OpenAI's ChatGPT to (i) clarify some motion-planning concepts from lectures, (ii) draft and refine parts of the report (methods and results write-ups), and (iii) general purpose debugging. All final decisions on methods, analysis, and conclusions were my own

References:

[1] Hanna Kurniawati 2025, COMP6320 Advanced Topics in AI: Lecture Slides, The Australian National University, Canberra.

[2] Sutton, R. S., and Barto, A. G. 2018. Reinforcement Learning: An Introduction (2nd ed.). MIT Press.

[3] Puterman, M. L. 1994. Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley.

[4] Bellingham, J., Tillerson, M., Richards, A., and How, J. P. 2003. Multi-task allocation and path planning for cooperating UAVs. Proceedings of the IEEE CDC.

[5] Ng, A. Y., Harada, D., and Russell, S. J. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. ICML 1999.