ChatGPT

Warehouse Robot Multi-Agent Path Planning Project

Introduction

Multiple mobile robots in automated warehouses must **navigate efficiently without collisions** to transport goods between stations. This course project explores a theoretical solution for multi-robot path planning in a warehouse setting. We integrate **classical search algorithms** (like A for path finding), symbolic planning (using a PDDL domain model for high-level task planning), and reinforcement learning (RL) techniques (DQN and PPO) to coordinate robots. The focus is on a conceptual design demonstrating understanding of path planning, planning domain modeling, and learning-based approaches from class, rather than a full implementation. We also discuss Responsible AI considerations* (fair task allocation, safety, accountability) relevant to deploying warehouse robots.

Team Roles and Responsibilities

Our team of three has divided the project tasks according to each member’ s expertise, while preserving the overall technical scope:

- **Member 1 – Environment & Simulation:** Set up the warehouse environment representation and simulation. This includes designing the **PDDL domain model** for the warehouse (defining objects, predicates, and actions) and configuring a simulator (grid layout, obstacles, robots) to visualize and validate plans.
- **Member 2 – Path Planning Algorithms:** Develop and test **path planning algorithms** for robot navigation. This involves implementing A search for shortest paths on the warehouse grid and extending it to multi-robot scenarios with conflict avoidance* (ensuring robots do not collide by scheduling or adjusting paths).
- **Member 3 – Reinforcement Learning & Evaluation:** Explore a learning-based approach using **deep reinforcement learning** (e.g. DQN and PPO) for robot navigation or task decisions. This includes training policies in simulation, comparing their performance to the classical planner, and evaluating results (success rates, path efficiency, collision frequency).

This structure ensures each aspect – environment and planning logic, algorithmic pathfinding, and learning/evaluation – is addressed by a dedicated team member, while all parts integrate into a coherent multi-robot planning system.

Environment Modeling and PDDL Planning Logic

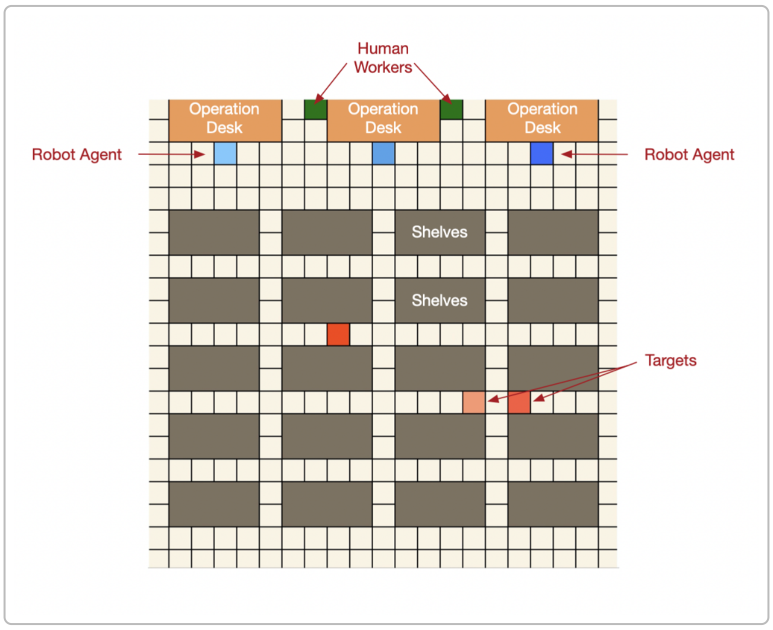

Figure: Example grid-based warehouse environment with multiple robot agents (blue) navigating around storage shelves (brown). Robots pick up goods from **Operation Desks** (orange, top) where human workers (green) prepare orders, and deliver them to target drop-off locations (red). This grid layout is used both in simulation and in the planning model.

We represent the warehouse as a discrete grid with designated areas for shelves, workstations, and goal drop-off points. Each **robot** and **item (box)** is an object in this environment. To plan high-level tasks like "robot A deliver box X to location Y," we use a symbolic AI planning model in **Planning Domain Definition Language (PDDL)**. In PDDL, the **domain file** defines object types, predicates, and actions available to the robots, and a **problem file** specifies the initial state (e.g. locations of each robot and box) and goal state (e.g. each box delivered to its destination).

The PDDL domain uses predicates to describe world state properties that can change. For example, we define predicates such as `robot-at(r, loc)` to denote a robot's position, `box-at(b, loc)` for a box location, `robot-has(r, b)` meaning a robot is carrying a box, or `delivered(b)` if a box is at its goal. Actions are defined with **parameters, preconditions, and effects** that update these predicates [1]. For instance, a simplified **"pick-up-box"** action might be defined as:

- **Parameters:** a robot `?r`, a box `?b`, and a location `?c` (cell).
- **Preconditions:** `(robot-at ?r ?c)`, `(box-at ?b ?c)`, and `(robot-empty ?r)` – the robot is at the box's location, the robot isn't carrying anything, and the box is present there.
- **Effects:** the robot is no longer empty and now holds the box, and the box is no longer at that location (the box's location predicate is removed) [2].

Similarly, a "move" action would allow a robot to go from one grid cell to an adjacent cell if there's no obstacle, updating its `robot-at` predicate. A "drop-off" action would let a robot put a box down at a target location (marking `delivered(b)` true and freeing the robot's hands). The **PDDL planner** uses these definitions to find a **sequence of actions** that achieve the goal conditions from the initial state [3]. By including all robots in the domain model, the planner can even decide which robot should perform each pickup or delivery action – effectively **assigning tasks** to robots as part of the plan. For example, if two boxes need delivering, the planner might output a plan where Robot1 picks up boxA while Robot2 picks up boxB in parallel, if the domain and planner support parallel actions, or intermix their actions in a single plan. The resulting plan might look like: Robot1 moves to boxA, picks it up, Robot2 moves to boxB, picks it up, Robot1 moves to drop-off, etc., ensuring all goals (deliveries) are satisfied. This symbolic planning layer provides high-level **task planning logic** that complements the low-level path planning, by deciding what each robot should do and in what order.

We will use a standard PDDL solver (e.g. Fast Downward or other classical planner) to generate such plans. The environment simulation will then execute these plans: for each action like "move R1 to cell X" or "pick-up R1, box3 at cell Y," we invoke the corresponding behavior in the simulator. This validates that the PDDL model correctly represents the warehouse tasks. The PDDL model thus serves as the **brain for task planning**, encoding the warehouse rules and robot capabilities, while our path planning algorithms (next section) ensure each move action can be realized without collisions in the grid.

Path Planning Algorithms (A* Search and Conflict Avoidance)

To navigate the grid efficiently, each robot needs a pathfinding algorithm. We chose **A** search as our primary path planner for individual robots, due to its optimality and efficiency on grid graphs. A will compute the shortest path from a robot's current location to its target location (e.g. the next waypoint like a box or drop-off) while treating other robots as dynamic obstacles. The heuristic is typically the Manhattan distance on the grid (since robots move in four directions), which is admissible for shortest path. This guarantees that if a single robot needs to go from point A to B around shelves, A* will find an optimal route avoiding the shelf obstacles.

The challenge arises in **multi-robot path planning**, where multiple robots move simultaneously. Naively planning each route independently can lead to **collisions** – for example, two robots might plan paths that cross the same grid cell at the same time. Our project addresses this with a **conflict avoidance** strategy. One rigorous approach we considered is **Conflict-Based Search (CBS)** [4], an optimal multi-agent pathfinding algorithm. CBS operates on two levels: a high-level search adds constraints to resolve collisions between robots, and low-level searches recompute individual robot paths under those constraints [4]. This ensures no two robots occupy the same cell at the same time, without brute-forcing a joint state space.

However, implementing full CBS can be complex for a course project, so we also explore a simpler rule-based coordination scheme (a prioritized planning approach). In this scheme, each robot initially plans a route with A. If a conflict is detected (e.g., Robot1 and Robot2 are set to enter the same cell simultaneously), we resolve it by assigning priority to one robot and adjusting the other's path. A simple priority rule is to allow the robot closer to its goal* (by remaining distance) to have the right of way, while the other robot waits briefly or takes an alternate step [5]. This heuristic is intuitive – the robot that is nearly finished gets to go first, reducing overall wait times. For example, if Robot1 is one step away from delivering its box and Robot2 is far from its goal, Robot2 will yield if they conflict, letting Robot1 finish quickly. We will implement such collision detection and resolution at each time step of the simulation:

1. **Collision Detection:** Continuously monitor robot positions and planned next moves. If two robots plan to enter the same cell (or swap cells) in the next step, that's a potential collision.
2. **Coordination:** Apply the priority rule – compare their distances to their respective goals (Manhattan distance on the grid). The robot with the shorter remaining distance (i.e., closer to goal) gets priority [5].
3. **Conflict Resolution:** The non-priority robot will wait in its current cell for a small time step or re-route if waiting would cause a backup. After the other robot passes, the waiting robot can resume its path (or a newly computed path to avoid lingering behind too long).

We will also account for **static obstacles** (shelves, walls) by treating those grid cells as blocked for A and for dynamic obstacles such as temporary blockages or human workers moving through. In the case of a human crossing a robot' s path, the robot will pause until the path is clear (with perhaps a timeout threshold for safety) [6] . The outcome of our path planning module is that each robot follows a collision-free trajectory* from start to goal. We will test this by simulating multiple robots going to various targets simultaneously and verifying they do not collide and that the total travel time is reasonably efficient.

By combining this path planning module with the PDDL task planner, we have a two-layer planning system: the high-level planner decides which locations each robot must visit (and in what order) to achieve all tasks, and then the low-level A planner decides how* to traverse the grid to those locations without collisions. This mimics real warehouse robot systems that separate task assignment/scheduling from actual motion planning. It also allows a clear connection to course material: graph search algorithms for path planning and constraint reasoning for multi-agent coordination.

Reinforcement Learning Approach (DQN & PPO)

In addition to classical planning, we investigate **reinforcement learning** as an alternative or complementary approach for multi-robot path planning. RL can potentially allow robots to **learn navigation policies** through trial-and-error and adapt to dynamic conditions or unseen scenarios. We focus on two popular algorithms from class: **Deep Q-Network (DQN)** and **Proximal Policy Optimization (PPO)**.

Problem Formulation: We model each robot' s navigation as an episodic MDP. At each time step, a robot observes the state (which could include its own position, the goal location, and possibly the positions of obstacles or other robots within a certain range), and takes an action (move up/down/left/right, or wait). It receives a reward for progress – for example, a large positive reward for reaching its goal successfully, and negative penalties for undesirable events like collisions or moving into dead-ends. The goal of RL is to learn a policy that maximizes the cumulative reward, which aligns with efficient, collision-free navigation.

DQN: Using DQN, we discretize the state space (e.g. the grid positions, maybe combined with a limited view of other robots). A neural network approximates the Q-value function Q(s,a) for state-action pairs. We train the network by letting the robot (or multiple robots in parallel) **explore the warehouse**. Initially, the agent moves randomly, discovering which actions lead to goal completion vs. collisions. Through many episodes, the Q-network is updated via the Bellman equation using experience replay. The result should be a Q-network that, given a state (robot at X, goal at Y, perhaps other robots at Z), outputs estimated values for moving in each direction. The robot can then choose the action with highest Q-value (greedy) to follow a path that it has learned leads to the goal efficiently. Prior work has shown that Q-learning can indeed learn shortest paths on grid mazes and avoid obstacles, given appropriate reward structure [7] [8] .

PPO: We also consider PPO, a policy-gradient method known for stable training in continuous or large discrete spaces. PPO directly learns a policy $\pi_\theta(a|s)$ that outputs a probability distribution over actions for each state, by optimizing expected reward with constraints to keep updates moderate. In our context, we would use PPO to train a robot' s navigation policy in a similar environment. PPO has the advantage of strong performance in dynamic environments and better sample efficiency in some cases [9] [10] . Our project is inspired by recent research combining classical path planning with RL: for example, a PPO + Dijkstra (PP-D) method was proposed where PPO handles real-time adjustments and Dijkstra (similar to A) provides a baseline route [8] . In our design, PPO could allow the robot to adjust its path on the fly if a new obstacle appears, while still aiming for the optimal path discovered by A.

We will train the RL agents in simulation. To keep things tractable, we might first train a **single robot** in a static environment (no moving obstacles) to reach a goal using DQN/PPO, verifying it learns the shortest path. Next, we can introduce a second robot or dynamic obstacles and train policies to handle avoidance (this can be framed as a multi-agent RL problem or by including other robot's positions in the state). Training multi-agent RL from scratch is challenging, so we may leverage transfer learning or curriculum learning as done in similar projects [11] [12]. For instance, first train robots to navigate the empty warehouse (no collisions, just avoid walls), then gradually introduce interactions between robots where they learn to negotiate right-of-way. We can also initialize the policy with **imitation learning** from the A* planner (see next section), which gives the RL a strong starting point and speeds up convergence.

Evaluation: Once trained, we will evaluate the learned policies against the classical planner. Metrics include: success rate (percent of episodes where the robot(s) reach goals without collision), average path length or time (compared to optimal A path length), and the ability to handle dynamic changes (e.g., if a new obstacle is introduced mid-run, does the RL policy naturally adapt while A would require re-planning?). We expect the classical A to give near-optimal paths in static settings, but a well-trained RL agent might show more robustness* in dynamic scenarios or unforeseen situations by virtue of its training experience [13] [14]. On the other hand, RL policies are not guaranteed optimal, so part of our analysis is understanding the trade-offs – RL might sacrifice some path optimality to avoid complicated maneuvers, or it might learn to favor safety (which could be desirable). Overall, incorporating RL in the project allows us to demonstrate understanding of how learning-based methods can complement or substitute traditional planning, tying into class topics on exploration vs. exploitation, reward design, and policy optimization algorithms.

Responsible AI Considerations

Deploying multi-agent robots in a real warehouse not only requires technical efficiency but also adherence to Responsible AI principles:

- **Fairness (Task Load Balancing):** We must ensure that the system assigns tasks to robots in a fair and efficient manner. No single robot should be overburdened with all deliveries while others remain idle without justification. Our task planning logic (PDDL planner or scheduling policy) can include fairness by rotating assignments or considering each robot's current load and battery level when allocating a new task. Fair load balancing prevents situations where one robot wears out faster or becomes a single point of failure due to bias in task assignment.

- **Safety (Collision Avoidance and Fail-safes):** Safety is paramount in a warehouse with multiple robots (and possibly human workers). Our path planning module is explicitly designed to avoid collisions by construction, and we include buffers (like waiting protocols and emergency stops) to handle unexpected obstacles or robot malfunctions. We also consider **speed limits and safe distances** – for example, if two robots come too close, even if not on a direct collision course, they might slow down as a precaution. In simulation, we will test worst-case scenarios (like many robots converging in one area) to ensure the avoidance logic scales. Additionally, if using RL, we must carefully shape rewards to never encourage unsafe behavior; safety rules (like "never collide") are baked in as hard constraints or large negative rewards. This way, even a learned policy remains within safe operational bounds. Ensuring safety also extends to human-robot interaction: if a human worker crosses the path, robots should yield and only resume when it's safe [6], as per warehouse safety protocols.

- **Accountability and Transparency:** It should be clear **which decisions the robots are making and why**, to both the operators and the system developers. Using a PDDL planner for high-level tasks provides a human-readable plan (a sequence of actions) that can be inspected for correctness. This transparency is useful for auditing and debugging – we can trace which robot was assigned to which task and the rationale (encoded in cost functions or rules). For learned components like DQN/PPO policies, interpretability is more challenging; however, we log their decisions and outcomes, and we can enforce that any decision made by RL still respects the overall plan constraints. In case of incidents (e.g., a collision or a missed delivery), we design the system to log relevant data (robot states, decisions taken) so that we can analyze the cause and improve the policy or planner accordingly. Accountability also means each robot' s actions are predictable to an extent – for instance, if two robots approach an intersection, the priority rule ensures it' s predictable which one will yield. This predictability is important for **trust** in autonomous systems: warehouse staff and managers should be able to understand and trust the robots' behavior, knowing they follow set rules or learned policies that have been thoroughly tested for reliability.

By embedding these Responsible AI considerations into our project design, we ensure that our theoretical system is not only effective but also aligned with ethical and practical standards for real-world deployment.

Enhancements with Advanced Planning Techniques

Beyond the base methods, we propose several advanced techniques that could **enhance our system** or serve as extensions in future work. These draw on decision-time planning and learning from demonstrations, as covered in advanced topics:

- **Monte Carlo Tree Search (MCTS) for Decision-Time Planning:** MCTS is a heuristic search algorithm that can be used online (at decision time) to find good actions by simulating many random future action sequences. At each decision point, MCTS builds a search tree where nodes are states (configurations of all robots) and edges are possible joint actions, then conducts repeated simulations (playouts) to evaluate the long-term reward of different action choices [15]. The tree is selectively expanded and updated based on these rollouts (using selection and backpropagation steps) to focus on promising action branches. In our context, one could integrate MCTS to handle particularly tricky multi-robot coordination decisions. For example, if our rule-based conflict resolution is not sufficient in a congested scenario, the system could invoke an MCTS planner that considers various combinations of moves (like robot1 waits while robot2 goes first vs. vice versa, or both reroute in different ways) for a few steps into the future. The MCTS would simulate these and choose the action set that likely leads to the best outcome (e.g., minimal delay and no collisions). This kind of online planning is useful if the environment is highly dynamic or unpredictable – rather than sticking to a pre-computed plan, robots can think on the fly and re-plan locally using MCTS whenever a new decision is needed. MCTS can also incorporate learned heuristics: for instance, if we have a trained policy (from RL or imitation), we can use that policy to bias the rollouts (to simulate more realistic or knowledgeable moves) [15]. This hybrid of search + learned policy can yield better decisions than either alone. While we do not implement MCTS within the scope of the course project, we acknowledge it as a powerful technique that could complement our approach by improving decision quality in complex multi-agent interactions.

- **Imitation Learning (Behavioral Cloning and DAgger):** Imitation learning offers a way to leverage expert demonstrations to train a policy, which can greatly accelerate learning compared to pure RL. In our project, the **expert** could be our classical planner itself (the combination of PDDL task planner and A path planner). We can generate a dataset of state-action examples by recording the trajectories that the planner produces for various scenarios (e.g., states could be a robot' s local observation and the expert action is the move A would take next). Using **Behavioral Cloning (BC)**, we train a neural network policy to imitate these expert actions via supervised learning [16] [17]. This gives an initial policy that roughly mimics optimal paths without needing any trial-and-error. However, a known issue is distributional shift – the learned policy might encounter states that were not in the training data (especially if it deviates even slightly from expert paths), leading to compounding errors [18]. This is where **DAgger (Dataset Aggregation)** comes in [19]. With DAgger, we iteratively improve the cloned policy: let the policy run in the environment, and whenever it visits a state, query the expert (planner) for the correct action for that state, adding this to the training dataset [19]. Over a few iterations, the policy' s dataset grows to include states from its own behavior, and it learns to recover from its own mistakes by incorporating the expert' s guidance for those situations. Applying DAgger in our project, we would: 1) start with the planner' s trajectories as initial data, train a policy network; 2) have the robot use this policy in simulation, collect any states where its action differed from the planner' s action, 3) query the planner for the best action at those states and add to data, retrain the policy. This process yields a **robust navigation policy** that closely matches A performance on typical scenarios, but also can handle off-nominal states because it has been trained on them with expert feedback [19]. The end result could be a policy that navigates almost as well as A but executes much faster at runtime (just a neural network forward pass per step, no graph search) and can generalize to similar new maps or configurations after training on a variety of warehouse layouts.

- **Integration of IL/MCTS with RL:** These techniques are not exclusive. For example, the policy obtained via imitation learning can serve as an excellent initialization for reinforcement learning, a process sometimes called **warm-starting** the RL. Instead of learning from scratch, the RL agent begins near an expert' s policy and then **fine-tunes** it through further trial-and-error to maybe exceed the expert in areas where the expert was suboptimal (or to adjust to changes the expert wasn' t trained on). Conversely, MCTS at decision time could use the value estimates from an RL critic network to guide its simulations (as done in AlphaGo). Our project' s modular design means we could swap in these advanced components if time permitted: e.g., replace the DQN agent' s ε-greedy decisions with an MCTS that uses the DQN' s Q-values for rollouts, or train an imitation policy and then use PPO to refine it on a slightly different warehouse configuration. These additions would demonstrate a grasp of cutting-edge strategies in planning and learning – showing how **demonstration and search** can enhance the base algorithms to achieve better performance and reliability.

By discussing these optional enhancements, we connect our project to broader themes in AI planning: the synergy between search and learning. Even if not fully implemented in our coursework, understanding these concepts enriches the solution and shows awareness of how one might tackle the problem of multi-agent path planning with even more sophisticated AI tools.

Conclusion

In this project, we designed a comprehensive approach for multi-agent path planning in a warehouse robot system, integrating principles from classical planning, heuristic search, and reinforcement learning. We created a PDDL domain model to capture the warehouse operations (robots, items, actions) and used it to plan high-level task assignments and sequences of actions, demonstrating knowledge of AI planning formalisms. For the actual navigation of robots, we implemented A*-based path finding augmented with strategies to prevent collisions among multiple robots, illustrating core path planning algorithms and multi-agent coordination techniques from the curriculum. We then explored deep reinforcement learning (DQN and PPO) as a modern approach to learn navigation policies, which provided insight into how learning algorithms can adapt to dynamic environments and potentially work in tandem with classical methods (for example, using an RL policy to adjust on the fly or as a fallback in unforeseen scenarios). Throughout the project, we remained cognizant of Responsible AI issues, ensuring that our theoretical design would prioritize fairness, safety, and accountability if deployed in the real world of warehouse automation.

Overall, this project allowed us to apply and connect class material in a realistic scenario: from search algorithms and planning domains to policy learning and ethical considerations. The result is a **conceptual system architecture** for warehouse robots that could be presented in a 15-minute talk, highlighting how each component – **simulation environment, planning logic, path planning, learning, and decision-making enhancements** – contributes to enabling a fleet of robots to work together efficiently and safely. Through this work, we not only reinforced our understanding of each individual technique but also learned how combining them can overcome their individual limitations, which is a key lesson in AI system design. In future extensions, implementing the described modules and testing them on a simulated warehouse would be the next step, moving from theory to practice, and potentially yielding a robust solution applicable to real automated warehouses. The interdisciplinary nature of the project, touching on algorithms, modeling, machine learning, and AI ethics, has been a valuable exercise in holistic problem-solving at the undergraduate level. We believe the insights gained here provide a strong foundation for tackling more complex multi-robot planning problems and adapting to the rapidly evolving landscape of AI in robotics.

Sources: The project concepts and methods were informed by classical AI planning literature and modern research. Notably, PDDL domain modeling for robotics was guided by examples like the warehouse domain in PDSim [2] [3], multi-robot path finding drew on the Conflict-Based Search paradigm [4] and practical coordination heuristics [5], reinforcement learning strategies were inspired by prior works integrating RL with path planning [8], and our discussion of imitation learning followed the DAgger algorithm for safe policy cloning [19]. These references underpin the theoretical framework of our project.

---

[1] [2] [3] icaps23.icaps-conference.org
https://icaps23.icaps-conference.org/program/workshops/keps/KEPS-23_paper_2186.pdf

[4] www.movingai.com
https://www.movingai.com/papers/sharon2015cbsjournal.html

[5] [6] [7] [11] [12] GitHub - LyapunovJingci/Warehouse_Robot_Path_Planning: A multi agent path planning solution under a warehouse scenario using Q learning and transfer learning.
https://github.com/LyapunovJingci/Warehouse_Robot_Path_Planning

[8] [9] [10] [13] [14] Paper Title (use style: paper title)
https://www.arxiv.org/pdf/2411.06128

[15] Lecture-adv-classic v3.4.pdf
file://file-1quNqLvSMSE2VpodzZ8raV

[16] [17] [18] web.stanford.edu
http://web.stanford.edu/class/cs237b/pdfs/lecture/lecture_10111213.pdf

[19] DAgger - imitation
https://imitation.readthedocs.io/en/latest/algorithms/dagger.html