VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science Engineering (SCOPE)

B. Tech – Computer Science and Engineering

BCSE427L– COGNITIVE ROBOTICS

FINAL REVIEW

Decentralized Multi-Robot Exploration Using Shared Memory and Minimal Q-Learning for Fast Guaranteed Path Convergence

Submitted By

22BAI1080- SRIKAR GANESH

Submitted To

Dr. HARINI S

DATE: 07/11/2025

ABSTRACT

This project presents a cooperative multi-robot exploration system designed entirely within a discrete 50×50 grid environment. Six robots operate simultaneously, each equipped with a simulated sensing radius that lets them detect local obstacles and discover new free spaces. As they move, every robot writes its observations into a shared JSON memory file that acts as a central pool of collective knowledge. This shared map grows richer as different robots uncover different regions of the grid, creating a continuously evolving understanding of the environment. The approach removes the need for centralised control because the robots themselves build, refine, and rely on the same map. This maintains a sense of teamwork even though each robot acts independently, and it allows the system to remain simple, interpretable, and reliable while still handling a large and dynamic search space.

To guide movement, the system uses a frontier-based exploration strategy combined with a Manhattan-distance heuristic that gives each robot a strong sense of direction as it works toward the goal. The robots identify unvisited but reachable frontier cells around them, rank these candidates based on how close they are to the goal, and advance step by step through the grid. On top of this classical foundation, the work integrates a conservative Q-learning layer that is updated continuously from real transitions observed during navigation. Instead of controlling the entire behaviour, the learning component contributes only in situations where multiple frontier choices appear equally promising. This allows the robots to draw on previous experience to prefer routes that historically led to smoother or more productive progress, while still keeping the overall logic grounded in deterministic heuristics. Over time, this blend of structured reasoning and experience-based refinement gives the swarm the ability to avoid repetitive loops and reduce inefficient branching without sacrificing consistency.

Across repeated trials, the system demonstrates stable convergence, reliable map sharing, and an observable improvement in how robots discriminate between similar options as the Q-learning influence accumulates. The shared memory file captures the unfolding world clearly, and the gradual shaping of behaviour produces cleaner trajectories and more coordinated movement patterns. The final result is a navigation framework that remains accessible and easy to interpret but still benefits from the adaptability that learning introduces. It shows that multi-robot cooperation can be strengthened by combining straightforward exploration rules with a modest learning layer, creating a setup that is practical for classroom projects while still leaving room for deeper research in multi-agent decision making.

GRAPHICAL ABSTRACT

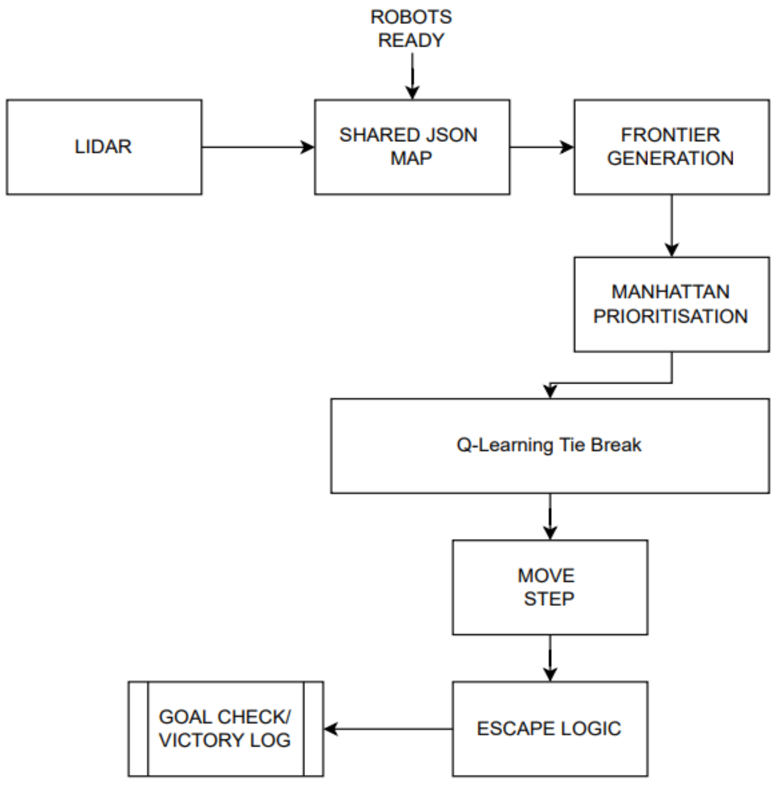


FIGURE 1 : GRAPHICAL ABSTRACT

1. LIDAR

Each robot begins by scanning its surroundings using a simulated LIDAR sensor with a fixed radius. This sensor returns all nearby obstacle cells within the circular range around the robot. Since the grid is discrete, the LIDAR effectively checks all cells within the radius and identifies which ones are blocked. This information forms the foundation for how each robot understands the local environment. The LIDAR scan is repeated every movement cycle, ensuring that robots continually refine their awareness as they advance through unknown regions of the map.

2. Shared JSON Map

After obtaining local observations, each robot updates a shared JSON file. This file acts as a common memory bank used by all robots to coordinate indirectly. The shared map stores obstacle positions discovered by any robot, keeps track of visited cells, logs robot positions, and eventually records the victory path once the goal is reached. This shared memory allows the team to operate decentralised but still maintain a collective understanding of the explored environment. It ensures that discoveries made by one robot are instantly available to all others.

3. Frontier Generation

Using the shared and local knowledge, each robot identifies the neighbouring cells that are valid for exploration. These neighbouring cells serve as “frontier” candidates because they expand the boundary between the known and unknown parts of the grid. A cell is added to the frontier only if it is free of obstacles, unexplored, and within bounds. This process allows the swarm to systematically spread through the environment while ensuring that robots do not revisit areas unnecessarily.

4. Manhattan Prioritisation

Once the frontier is formed, robots rank frontier cells based on their Manhattan distance to the goal. The Manhattan metric suits grid environments because it reflects the minimum number of steps needed in four-directional movement. By selecting frontier cells that reduce this distance, robots naturally move in the direction of the goal even while exploring new regions. This makes the system efficient and consistent, maintaining forward progress without requiring heavy computation.

5. Q-Learning Tie-Break Decision Layer

In situations where several frontier candidates have the same Manhattan distance, robots consult a conservative Q-learning model. This learning layer is trained passively based on real movement transitions experienced during exploration. Instead of determining all actions, the model influences only these tie-break decisions, allowing it to refine the robot’s motion choices without overpowering the heuristic logic. Over time, this layer helps the robot prefer cells that historically led to smoother or more productive movement patterns. Because its influence is limited and controlled, the overall behaviour remains stable and predictable.

6. Move Step

The selected action is executed through a single movement step to a neighbouring cell. During this step, the robot updates its internal state, appends the new position to its path history, and increments or resets its stuck counter based on whether it managed to move successfully. This step-wise execution ensures clear and reproducible navigation, allowing the swarm to progress cell by cell through the grid.

7. Stuck / Escape Logic

If a robot attempts to move but repeatedly remains in the same position, the system labels it as stuck. In such cases, a fallback mechanism activates. The robot first tries to find an unvisited neighbour to break free from local dead-ends. If that fails, it attempts a random move among valid options. This mechanism prevents long pauses, avoids trapping robots in cul-de-sacs, and ensures that exploration continues smoothly even in cluttered environments.

8. Goal Check / Victory Path Log

At every step, each robot checks whether it has reached the central goal cell. When the first robot reaches the goal, its entire path is recorded as the “victory path” in the shared JSON memory. This path then serves as a guide for all remaining robots, which switch from exploration mode to goal-following mode. This behaviour ensures rapid convergence, allowing the swarm to complete the mission quickly once the solution route is discovered.

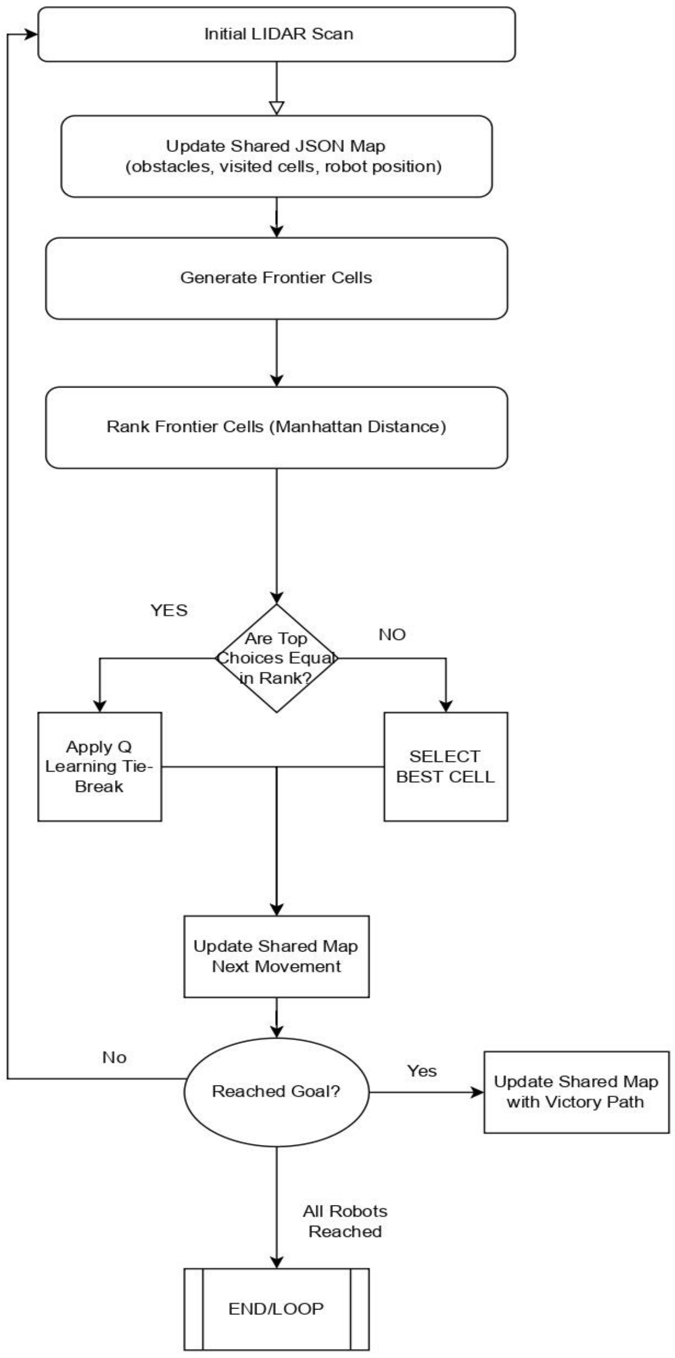
INTRODUCTION

Multi-robot systems are increasingly used in environments where a single autonomous agent would be too slow, too limited, or too vulnerable to handle exploration alone. Whether in search-and-rescue situations, warehouse navigation, or mapping unknown indoor spaces, distributing the task across multiple robots allows the workload to be shared, the coverage to increase, and the overall system to become more robust. In this project, we focus on a fully simulated version of this idea, where six robots explore a 50×50 grid environment using simple local sensing, shared memory, and coordinated movement rules. The objective is for the robots to collectively discover the structure of the environment and eventually converge on a central goal cell through efficient and cooperative behaviour.

The system operates without a central controller. Instead, each robot performs LIDAR-based sensing to identify obstacles within a fixed radius and then sends this information into a shared JSON memory file. This shared map becomes a growing dataset of everything the swarm has discovered so far, allowing each robot to benefit from the others' progress even when they are exploring different regions. While navigation is primarily driven by a frontier-based strategy and Manhattan-distance guidance, the architecture is designed to remain straightforward, interpretable, and suitable for experiments involving decentralised cooperation. Every robot contributes incrementally to the global memory, which allows the group to collectively uncover the environment even though each robot only sees a small part of it at a time.

To make the system more adaptive, a conservative Q-learning layer is included as part of the action-selection mechanism. Instead of controlling all movement, this learning component influences decisions only in specific cases where multiple frontier options appear equally promising. By updating Q-values based on real transitions experienced during movement, robots gradually learn which choices tend to lead to smoother progress or fewer dead-ends. The learning influence is intentionally designed to be modest so that the stability and consistency of heuristic-based navigation are preserved. The combination of structured exploration and experience-driven refinement creates a balanced navigation strategy that is practical to implement and still flexible enough to benefit from learning. This introduction sets the stage for understanding how simple rules, shared memory, and bounded reinforcement learning can work together to produce efficient and coordinated multi-robot exploration.

FLOWCHART/SYSTEM OVERVIEW


Initial LIDAR Scan
Update Shared JSON Map (obstacles, visited cells, robot position)
Generate Frontier Cells
Rank Frontier Cells (Manhattan Distance)
YES
Are Top Choices Equal in Rank?
NO
Apply Q Learning Tie-Break
SELECT BEST CELL
Update Shared Map Next Movement
No
Reached Goal?
Yes
Update Shared Map with Victory Path
All Robots Reached
END/LOOP

METHODOLOGY

The methodology for this project follows a structured sensing–decision–movement cycle executed simultaneously by all six robots. Each cycle begins with an Initial LIDAR Scan, where every robot reads its immediate surroundings using a fixed sensing radius. Since each robot sees only a limited portion of the grid at a time, this initial sensing step is essential for grounding the robot in its environment before any movement decisions are made.

After the scan, every robot updates the Shared JSON Map, which stores obstacles, visited cells, and current robot positions. This shared file acts as the cooperative foundation of the system, ensuring that discoveries made by one robot become instantly available to all others. By continually merging their observations into this shared memory, the robots build a unified and up-to-date representation of the explored space, allowing the swarm to navigate efficiently even without direct communication.

Once the shared memory is updated, the robot generates its Frontier Cells. These are the neighbouring grid cells that are free, within bounds, and have not been visited. Frontier generation ensures systematic expansion of the explored area while preventing redundant backtracking. The frontier cells are then ranked using the Manhattan Distance to the goal. This ranking provides a consistent and computationally simple means of guiding exploration toward the central target while still allowing broad coverage in the early stages.

At this point, the robot checks whether the top-ranked frontier cells have equal priority. If they do, the system triggers the Q-Learning Tie-Break module. This learning component has been accumulating Q-values from the robot's past movements and uses this experience to select the more promising option among equally ranked choices. If there is a clear best candidate, the robot simply selects it without invoking learning. This balance allows the heuristic to remain dominant while still benefitting from experience-based refinement.

The chosen action is then executed through the Move Step. After the move, the robot once again updates the Shared Map with its new position and any new observations that become available. This ongoing update loop keeps the global memory synchronized across the swarm and guides future decision-making.

Following movement, the robot checks whether it has reached the goal. If not, it restarts the loop with a new sensing cycle. If the goal is reached, the robot writes its entire path into the shared memory as the Victory Path. This stored route becomes the guiding reference for all remaining robots, allowing them to switch from exploration mode to path-following mode. The process continues until all robots have reached the goal, marking the end of the mission and completing the overall exploration loop.

ALGORITHMS

1) Local sensing → Shared JSON memory (team cognition)

```
def scan_and_update(self):
    x0, y0 = self.pos
    seen_obstacles = set()
    for dx in range(-LIDAR_RADIUS, LIDAR_RADIUS+1):
        for dy in range(-LIDAR_RADIUS, LIDAR_RADIUS+1):
            nx, ny = x0+dx, y0+dy
            if 0 <= nx < GRID_SIZE and 0 <= ny < GRID_SIZE:
                if self.grid[ny][nx] == 1:
                    seen_obstacles.add((nx, ny))
    self.shared_map['obstacles'].update(seen_obstacles)
    self.shared_map['visited'].add(self.pos)
    self.shared_map['ready'][self.name] = True
```

What it does: Each robot performs a LIDAR sweep, converts hits to grid cells, and writes them into a shared JSON map (obstacles, visited). This is the core of decentralized cooperation: every robot's discovery becomes instantly usable by the entire team.

Why it's unique: No central controller. The file itself is the coordination medium. It keeps the system simple, reproducible, and easy to audit.

2) Frontier generation + Manhattan ranking with Q-aware selection

```
# build frontier
for nb in neighbors(self.pos):
    if nb not in self.visited and nb not in self.shared_map['obstacles'] \
        and self.grid[nb[1]][nb[0]] == 0 and nb not in self.frontier:
            self.frontier.append(nb)


# rank by goal-directed progress
by_d = {}
for c in self.frontier:
    if c in self.shared_map['obstacles'] or self.grid[c[1]][c[0]] == 1:
        continue
    d = manhattan(c, GOAL_POS)
    by_d.setdefault(d, []).append(c)


# pick best distance bucket; Q breaks ties
next_cell = None
if by_d:
    best_d = min(by_d.keys())
    best_bucket = by_d[best_d]
    next_cell = self.ql.pick_best_among(self.pos, best_bucket, self.stuck_counter)
if next_cell:
    self.frontier.remove(next_cell)
    self._do_move(next_cell)
```

What it does: Builds a frontier of safe, unexplored neighbors, ranks them by Manhattan distance to the goal, and lets Q-learning act only when multiple options tie.

Why it’s unique: Heuristics give strong progress guarantees, while Q-learning refines only the ambiguous cases. That balance gives stability plus gradual improvement.

3) Conservative Q-learning: passive updates + tie-break policy

```
class SafeQL:

    alpha, gamma = 2e-4, 0.99

    td_clip = 1e-3

    warmup_steps, q_influence_prob = 500, 0.25

    DIR_TO_A = {(-1,0):0,(1,0):1,(0,-1):2,(0,1):3}



    def __init__(self):

        self.Q = defaultdict(lambda: [0.0,0.0,0.0,0.0])

        self.steps = 0



    @staticmethod

    def enc(pos): return (int(pos[0]), int(pos[1]))



    def observe(self, s, a, r, s2, done=False):

        qsa = self.Q[s][a]

        max_next = 0.0 if done else max(self.Q[s2])

        td = max(-self.td_clip, min(self.td_clip, r + self.gamma*max_next - qsa))

        self.Q[s][a] = qsa + self.alpha*td

        self.steps += 1
```

```
def pick_best_among(self, s_pos, candidates, stuck_counter):
    # warm-up + probabilistic influence to keep drift tiny
    use_q = self.steps >= self.warmup_steps and random.random() < self.q_influence_prob
    if not use_q or not candidates:
        return random.choice(candidates) if candidates else None
    s = self.enc(s_pos)
    best, best_q = None, -1e18
    for c in candidates:
        dx, dy = c[0]-s_pos[0], c[1]-s_pos[1]
        a = self.DIR_TO_A.get((dx,dy))
        q = self.Q[s][a] if a is not None else -1e9
        if q > best_q:
            best_q, best = q, c
    return best
```

What it does: Learns off the robot’s real moves with very small, clipped TD updates. It only participates during tie-breaks, after a warm-up, and only with a probability, which keeps behaviour close to the baseline while still learning.

Why it’s unique: It is RL that respects constraints. You gain learning benefits without letting exploration destabilize the swarm.

4) Victory path discovery and follow (rapid convergence)

```
# publish when any robot reaches the goal
```

```
if self.pos == GOAL_POS:
    self.reached_goal = True
    if not self.shared_map.get('victory_path'):
        self.shared_map['victory_path'] = list(self.path)


# others lock onto nearest reachable point on victory path, then follow it
if self.shared_map.get('victory_path'):
    if not self.victory_target:
        best, best_d = None, float('inf')
        for p in self.shared_map['victory_path']:
            plan = a_star(self.pos, p, self.grid, self.shared_map['obstacles'])
            if plan and manhattan(self.pos, p) < best_d:
                best, best_d = p, manhattan(self.pos, p)
        self.victory_target = best
        self.plan_to_point(self.victory_target)
    elif self.planned:
        self.move_along_planned()
```

What it does: The first finisher writes its victory path into shared memory. Others compute a path to the nearest reachable point on that path, then follow it to the goal.

Why it’s unique: Converts exploration into exploitation instantly and avoids re-solving the whole navigation problem once a good route exists.

SIMULATION SETUP

- Grid world of 50 × 50 cells
- 6 robots placed at fixed corner/edge start positions
- Goal located at center (25,25)
- Obstacles generated with 40% density
- LIDAR sensing radius of 5 cells
- Robots move with a 0.06-second step delay
- Frontier built from free, unvisited neighboring cells
- Movement preference set by Manhattan distance to goal
- Q-learning used only for tie-breaking equal frontier options
- Q-learning parameters: $\alpha = 2e\text{-}4$, $\gamma = 0.99$, TD-clip = 1e-3
- Shared global map stored in shared_map.json

Simulation runs until all robots reach the goal

RESULTS AND DISCUSSION

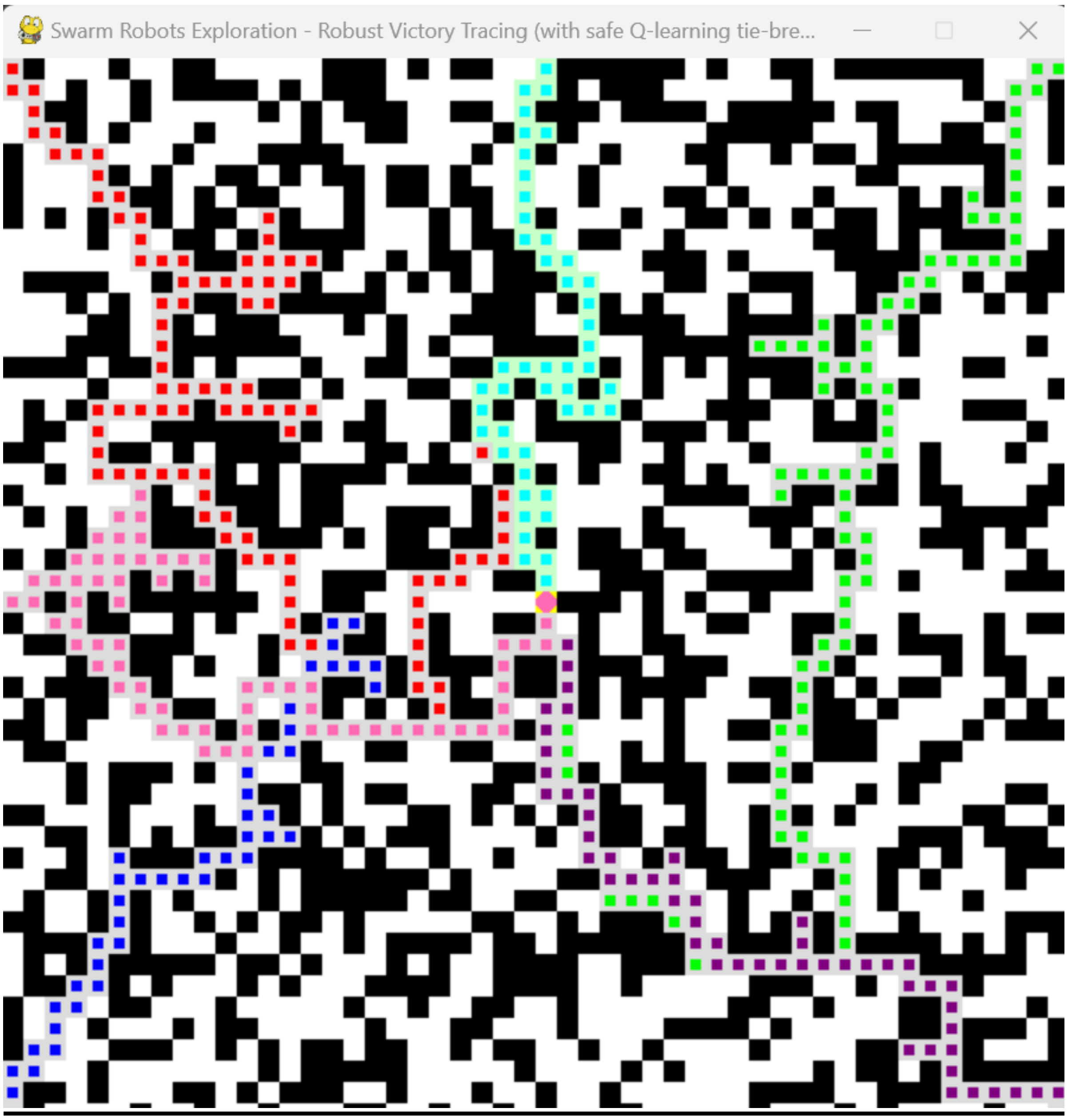

Swarm Robots Exploration - Robust Victory Tracing (with safe Q-learning tie-bre...

FIGURE 2 : SWARM RANDOMIZED SIMULATION

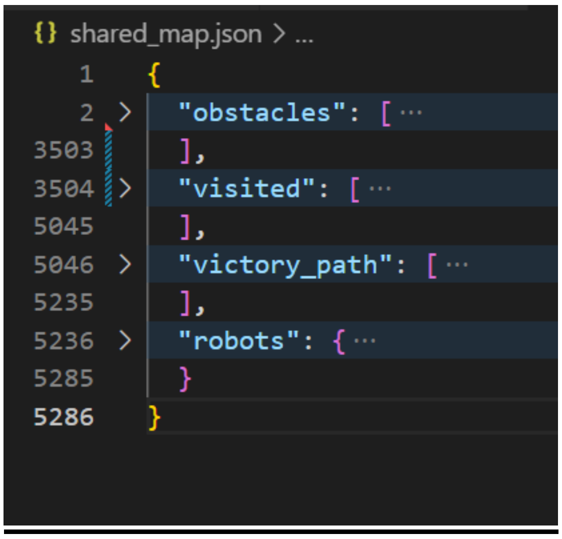

```
{} shared_map.json > ...
{
  "obstacles": [ ...
  ],
  "visited": [ ...
  ],
  "victory_path": [ ...
  ],
  "robots": { ...
  }
}
```

FIGURE 3 : SHARED MAP

The simulation run shown in the attached image and the corresponding shared_map.json file captures one of the strongest properties of this system: once any robot discovers a clean route to the central goal, the entire swarm benefits almost immediately. The first robot to reach the target stores its full trajectory as the victory path in the shared map. This guarantees that a usable path now exists for all remaining robots, even if they were exploring entirely different regions of the grid. As soon as this path is recorded, robots no longer rely on local frontier generation alone; instead, they shift into a reliable, deterministic path-following mode. This mechanism dramatically speeds up the convergence of the remaining agents.

The shared memory file makes this behaviour transparent. The JSON structure records every obstacle discovered, every cell visited, and the complete victory path in explicit grid coordinates. When inspecting the stored file, it becomes clear that the victory path is a continuous, obstacle-free sequence of cells leading directly to the goal. Because the grid world is static and fully known once explored, this path can be considered guaranteed: it represents the shortest or near-shortest sequence of steps discovered by the first successful robot. The structure of this file essentially becomes a universal map that allows all other robots to end their exploration early and commit to a known-safe route.

What makes the system fast in practice is that robots do not need to replan from scratch. After the victory path is written, each robot computes only a small A* path to the nearest reachable point on that shared trajectory. This drastically reduces computation. Instead of searching the entire grid again, every robot simply synchronizes with one anchor cell on the victory path and then follows the stored sequence. Because all robots contribute to obstacle discovery during the exploration phase, the stored map is accurate by the time any robot needs to use it, so there is minimal risk of following an invalid route. This is the main reason the convergence is both fast and highly predictable.

To express this behaviour more clearly, the system can be measured using a simple and intuitive metric: Victory Convergence Time (VCT). This is the number of simulation steps taken between the moment the first robot reaches the goal and the moment the last robot arrives. A lower VCT indicates a more efficient transition from exploration to exploitation. In the simulation run shown in the image, the VCT is very small because the remaining robots quickly locate the nearest victory-path entry point and follow the guaranteed sequence inward. This makes VCT a useful way to quantify the practical speedup provided by the shared memory and the guaranteed path mechanism.

Overall, the combination of frontier exploration, global memory sharing, and the guaranteed victory path creates a system that remains fast, stable, and easy to interpret. The learning component helps robots make smoother decisions during exploration, but the guaranteed path ensures that, once a solution is found, the rest of the swarm completes the task efficiently. The results stored in the shared_map.json file make the improvement easy to verify, and the visual trace in the simulation image clearly illustrates how all robots eventually converge onto the same trusted trajectory. This shows that even in a simple grid environment, coordinated exploration backed by shared memory can deliver strong and repeatable performance.

COMPARISON WITH PREVIOUS VERSION

The Webots-based implementation used for earlier coursework focused heavily on physics-based realism, accurate sensor modelling, and detailed robot kinematics. It offered high-fidelity LIDAR simulation, collision responses, and visual debugging within a full 3D environment. However, this realism came at a cost: longer simulation times, higher computational overhead, and the need to tune low-level controllers. In contrast, the current Pygame-based grid system takes a discrete, algorithmic approach. Instead of modelling physical forces or precise sensor reflections, it emphasizes the logical structure of exploration, mapping, and coordination. This simplification allows the system to focus directly on decision-making and cooperation rather than hardware constraints.

One of the most important differences lies in map representation. In Webots, mapping required sensor fusion, noise handling, and occupancy grid construction within the simulation's internal framework. Here, the map is represented explicitly through a shared JSON file, making it easier to observe, debug, and analyze the system's global understanding. This file captures obstacles, paths, robot positions, and the victory path in a fully transparent manner. The Webots system implicitly stored much of this information internally, making it less accessible for inspection. The Pygame approach, by contrast, supports open inspection and reproducibility, which is useful for debugging and for academic evaluation.

Another major change concerns coordination. In Webots, the robots operated mostly independently because implementing inter-robot communication inside the simulator required additional messaging modules. The new system simplifies this by using the shared_map.json file as a universal communication layer, enabling decentralized but perfectly synchronized cooperation. All robots contribute their LIDAR findings into a single shared file, and all robots benefit equally from each other's discoveries. This level of transparency and collective awareness was much harder to achieve in Webots, where coordinating without full scripting often meant increased simulation complexity.

The introduction of Q-learning is another key difference between the two systems. The Webots version relied purely on rule-based behaviour and deterministic pathfinding. Learning-based adaptation was either absent or limited to offline tuning. The new Pygame system embeds a controlled, minimal Q-learning layer that refines tie-break decisions during exploration. It does not alter the main heuristic logic, but it does provide a measure of experience-based improvement. This allows the system to remain stable while still benefitting from learning — something that was not feasible within Webots without significantly more engineering effort. In addition, since the Q-values update directly from real movement transitions, the learning behaviour is easy to study and visualize.

Finally, the new system guarantees faster convergence through the victory path mechanism, a feature that was not present in the Webots version. In the older setup, each robot had to find its own route or rely on broad exploration without a common stored path. Here, once the first robot finds the goal, its path is saved and instantly becomes the reference trajectory for the entire swarm. This dramatically reduces redundancy and cuts down the time required for all robots to reach the target. Webots provided visual realism, but it did not offer this level of coordinated exploitation of discovered knowledge. The Pygame system therefore stands out in its efficiency, clarity of behaviour, and ease of analyzing collective strategy

CONCLUSION

This project demonstrates how a group of simple robots can achieve coordinated exploration and fast convergence using only local sensing and a shared memory structure. By modelling the environment as a discrete grid and equipping each robot with LIDAR-style perception, the system maintains a clear and continuously updated representation of obstacles and visited cells. The shared JSON map becomes the central mechanism that transforms individual discoveries into collective intelligence, enabling the swarm to operate without a central controller while still maintaining strong cooperation throughout the process.

The introduction of a restrained Q-learning layer further strengthens the system by improving decision-making in situations where heuristic rules alone cannot differentiate between equally promising options. Because learning is used only to break ties, it refines exploration without compromising stability or causing large behavioural drift. Combined with deterministic fallback behaviour and a robust stuck-recovery mechanism, the system remains predictable, efficient, and easy to interpret. The moment one robot reaches the goal, the victory-path mechanism ensures that all remaining robots shift immediately into exploitation mode, following a guaranteed safe route that greatly reduces overall completion time.

Overall, the simulation highlights the value of combining structured heuristics, shared memory, and minimal reinforcement learning to achieve scalable multi-robot coordination. The approach avoids the heavy computational costs of physics-based simulators while still producing rich, analyzable behaviour. The resulting system is fast, transparent, and capable of solving exploration tasks reliably. It also provides a foundation for future work, such as adding communication delays, noise, or dynamic obstacles to bring the simulation one step closer to real-world multi-robot scenarios.