Decentralized Multi-Robot Task Allocation and Navigation in Complex Environments (DC-MRTA): Paper Analysis

Sharmad Kalpande (22166), Anirudha Patil (22234), Saurabh Shirke (22297), Rushikesh Bhosale (22078)

1. Introduction and Related Work

1.1. General Overview

The paper proposes a new, coupled approach for solving multi-robot task allocation and decentralized navigation (called DC-MRTA). Instead of assigning tasks before navigation, this method allocates tasks dynamically during robot movement, using reinforcement learning (RL). The objective is to minimize the total travel delay (TTD), i.e., the total time spent traveling without performing tasks, and to improve makespan (total time to complete all tasks).

Key Contributions are as follows:

1. **Two-level Architecture**
   - Combines task allocation and navigation in warehouse environments.
   - Utilizes Markov Decision Processes and Reinforcement Learning for decentralized task assignment.
   - Can be integrated with existing navigation methods.
2. **Scalability & Flexibility**
   - Works with a variable number of robots and tasks.
   - Makes no specific assumptions about the environment.
3. **Reward Design**
   - The high-level RL agent receives feedback (rewards) from the low-level navigation system.
   - Rewards are based on Total Travel Delay (TTD).
   - Compatible with other decentralized methods such as ORCA.
4. **Performance Evaluation**
   - Compared with standard greedy methods like MPDM and RBTS.
   - Tested in various warehouse setups with different numbers of robots.
   - Achieved up to 14% better task completion time for 500 tasks.
   - Reduced inter-agent collisions by 40% in dense environments.
   - Scales to thousands of robots.

1.2. Multi-Robot Decentralized Navigation (Related Work):

Definition: Multi-robot navigation involves finding collision-free paths for multiple robots moving in the same environment.

Categories of Navigation Methods: There are two main types

- **Centralized methods:** A single controller makes decisions for all robots based on complete knowledge.
- **Decentralized methods:** Each robot makes its own decisions based on local information.

The discussed approach is based on decentralized planners. Each robot uses local knowledge (like nearby robots' positions, velocities, and sizes) to navigate independently.

- **Velocity Obstacles (VO):** A decentralized algorithm that helps avoid collisions by identifying unsafe velocities.

- **Reciprocal Velocity Obstacles (RVO):** An improvement over VO. Take into account how both robots react to avoid each other. Helps reduce zig-zagging or oscillations in their paths.
- **Optimal Reciprocal Collision Avoidance (ORCA):** A version of RVO formulated as a linear convex optimization problem. It is more efficient and can adapt to different robot movement models.
- **Buffered Voronoi Cell (BVC):** Another decentralized approach: Uses Voronoi diagrams to define safe zones for each robot to move in. It relies only on position data (no velocity needed).

"The authors' method is general. It can be combined with any decentralized navigation strategy, including VO, RVO, ORCA, BVC, or RL-based methods."

Though the Decentral approach offers an advantage that it can be scalable to many robots. It also comes with limitations that in such an approach it is difficult to ensure optimal path and deadlock-free navigation especially in a complex environment.

1.3. Multi-Robot Task Allocation (MRTA) (Related Work):

Definition: MRTA refers to optimally assigning robots to tasks. It's a version of the Multiple Traveling Salesman Problem (mTSP)—a known NP-hard problem, meaning it's very hard to solve exactly for large instances.

Previous Work:

- Integer Linear Programming (ILP) has been used to model warehouse task allocation.
- Some algorithms address optimal task allocation for heterogeneous agents (robots with different capabilities).
- These approaches assume that all agents are available at the same time to make the best decisions.

Two main approach in MRTA:

1. **Market-Based Approaches**
   - Tasks are treated like commodities, and robots "bid" to do them.
   - Auction-based methods like CBBA (Consensus-Based Bundle Algorithm) and CBAA (Consensus-Based Auction Algorithm) are popular.
   - These use local robot information and then reach a consensus to align with the global objective.
   - Variants like deep-RL-based CBBA are used in environments with limited communication bandwidth.
2. **Optimization-Based Approaches**
   - Use mathematical models (e.g., linear programming) to find the best possible task assignment.
   - Require global knowledge and often assume centralized planning.

The paper uses a regret-based auction method as a baseline for comparison.

"Existing decentralized MRTA solutions are limited in scalability and robustness. The paper aims to develop a more scalable and general solution that can also work with decentralized navigation strategies in realistic warehouse-like environments."

2. Problem Formulation

A system of n robots is considered. The robots are: **Holonomic** (they can move in any direction) or **Disk-shaped** (their geometry is circular, simplifying collision calculations).

They operate in a **2D environment** $W \subset \mathbb{R}^2$.

Each robot is referred to as an agent A_i, its position p_i, velocity v_i, and geometry (size and shape) are represented by A_i itself.

- **Local Awareness (Neighbourhoods):**
  - Every agent has a *sensing radius* to detect nearby robots.
  - Robots within this range form the *neighborhood set* N_i of agent A_i.
  - Agent A_i has local access to:

    * Positions p_j and velocities v_j of its neighbors $j \in N_i$.
    * Positions of **obstacles** (both static and dynamic) around it.
  - Information is gathered using:
    * **Perception sensors** (e.g., LiDAR, camera).
    * **Communication modules** (e.g., wireless exchange of positions).

- **Decentralized Navigation Formulation:**
  - Each robot makes decisions using **only local knowledge**.
  - Decisions involve:
    * What tasks to perform.
    * How to navigate without colliding with others or obstacles.
  - No **central controller** is involved in coordinating actions.

- **Obstacle Avoidance & Path Planning:**
  - For static obstacles, the standard A* algorithm is used.
  - A* is a *graph-based shortest path algorithm* that ensures collision-free paths.
  - Only static obstacles are handled here; dynamic ones are managed via local coordination.

- **Task Representation:**
  - A **task** is defined as a tuple (o_t, d_t) where:
    * o_t: **Origin** location (pickup point).
    * d_t: **Destination** location (drop-off point).
  - Each robot assigned a task has its **goal set** as:

$$G_i = \{o_t, d_t\}$$

3. Metrics

Our goal is to design a decentralized task allocation scheme for a team of robots where each robot executes only a single task at any given time. We consider a dynamic, lifelong setting where new tasks are continuously generated, and task assignment is instantaneous — that is, a task is assigned to an available robot at each decision timestep.

3.1. System Objectives

The system aims to:

- **Minimize Makespan:** The total time taken to complete a specific number of tasks.
- **Minimize TTD (Task Travel Delay):** The time required for a robot to travel from its current location $\mathbf{p}_i$ to the starting location of the allocated task $\mathbf{o}_i$.

3.2. Task Execution Workflow

Given a task assigned to robot *i*, the robot sequentially:

1. Moves from its current position $\mathbf{p}_i$ to the task's start location $\mathbf{o}_i$.
2. Then navigates to the task's destination $\mathbf{d}_i$.

The task is considered complete upon reaching $\mathbf{d}_i$, after which the robot becomes available for new task assignments.

3.3. *Collision-Free Navigation*

At any timestep, a robot agent A_i must remain collision-free, meaning:

$$A_i \cap A_j = \emptyset, \quad \forall j \in \mathcal{N}_i \tag{1}$$

$$A_i \cap O_j = \emptyset, \quad \forall j \in \{1, 2, \ldots, m\} \tag{2}$$

where:

- A_i is the geometry of robot i,
- O_j is the geometry of obstacle j,
- $\mathcal{N}_i$ is the set of neighboring agents of i,
- m is the number of obstacles in the environment.

At each timestep, the robot moves towards its current goal $g_i \in G_i = \{\mathbf{o}_i, \mathbf{d}_i\}$ while avoiding collisions.

3.4. *Optimization Objective*

Our primary objective is to minimize the travel distance:

$$\min \|\mathbf{g}_i - \mathbf{p}_i\| \tag{3}$$

where $\mathbf{g}_i$ is the current goal of agent i.

This optimization is non-trivial because the choice of $\mathbf{g}_i$ depends on task allocation decisions, making joint optimization of task assignment and trajectory planning computationally challenging in practice.

4. Lower Level: Decentralized Navigation

The lower level is responsible for moving the robot safely once it has been assigned a task by the upper-level reinforcement learning (RL) system. Two techniques are used:

1. **A* algorithm**: Used for path planning to avoid static obstacles.
2. **ORCA (Optimal Reciprocal Collision Avoidance)**: Used for real-time collision avoidance to prevent collisions with other moving robots.

4.1. *Velocity Obstacle (VO)*

A set of relative velocities can lead to collisions within a time horizon τ. The velocity obstacle between robot i and robot j is defined as:

$$VO^{\tau}_{i|j} = \left\{ \mathbf{v} \mid t \in [0, \tau],\, t\mathbf{v} \in D(\mathbf{p}_j - \mathbf{p}_i,\ r_i + r_j) \right\}$$

Where:

- $\mathbf{p}_i$, $\mathbf{p}_j$: positions of robots i and j
- r_i, r_j: radii of the disk-shaped robots
- τ: time horizon (how far into the future the system looks)
- $D(\cdot)$: denotes a disk centered at the given relative position with radius equal to the sum of robot radii
- $\mathbf{v}$: relative velocity of robot i with respect to robot j

Assuming agents use velocities v_i^{opt} and v_j^{opt}, they are collision-bound if:

$$v_i^{opt} - v_j^{opt} \in VO^{\tau}_{i|j}$$

4.2. *ORCA Half-Plane Constraint*

ORCA computes a suitable, collision-free velocity. The constraint is defined as:

$$ORCA_{i|j}^{\tau} = \left\{ \mathbf{v} \,\middle|\, \left(\mathbf{v} - \left(\mathbf{v}_i^{opt} + \frac{1}{2}\mathbf{u} \right) \right) \cdot \mathbf{n} \geq 0 \right\}$$

- v_i^{opt}: current velocity of robot i
- u: minimum change in velocity needed to avoid a collision
- $\frac{1}{2}u$: each robot takes half responsibility for collision avoidance
- n: unit normal vector pointing outward from the velocity obstacle boundary
- v: any candidate new velocity for robot i

A safe velocity v is selected such that it lies on the safe side of the velocity obstacle boundary. This defines a half-plane of safe velocities.

The final velocity is chosen from the set $ORCA_{i|j}^{\tau}$ by minimizing the distance to v_i^{pref}, the preferred velocity of agent i toward its goal.

5. Higher Level: RL-based Task Allocation

At the higher level, reinforcement learning is used to allocate tasks to robots that must perform pick-and-drop operations. The task allocation problem is formulated as a Markov Decision Process (MDP):

$$\mathscr{M} = (S, A, R, P, \gamma)$$

Where:

- S: State space
- A: Action space
- R: Reward function
- P: Transition probabilities
- γ: Discount factor

5.1. *State*

At time step t, the state s includes:

- **Robot information:**
  - p_j: position of robot j
  - r_j: task completion time of robot j
- **Task information:**
  - For each task i: (o_i, d_i, k_i, l_i) where:
    - o_i: origin
    - d_i: destination
    - k_i: distance to robot
    - l_i: task length
- j_{sel}: currently available robot for assignment

Thus, the full state is

$$\mathbf{S} = \left\{ (\mathbf{p}_j, r_j)\, \forall j \in \mathscr{R},\ (o_i, \mathbf{d}_i, k_i, l_i)\, \forall i \in \mathscr{P},\, j_{sel} \right\}$$

5.2. Action

An action a is the selection of one task from the current task queue for the available robot.

- The action space A includes all available tasks.
- The policy $\pi(s)$ maps the state to a distribution over actions: $\pi : S \to D(A)$

The policy π is modeled using a deep neural network (DNN):

$$\pi_\theta(\cdot \mid s)$$

Where:

- θ: parameters (weights) of the network
- Input: current state s
- Output: probabilities for each task

Note: A simple feedforward network does not scale well for many robots and tasks. Therefore, an **attention-based architecture** is used to focus on relevant parts of the input (e.g., nearest task, available robot).

- During training: the agent samples an action based on the output distribution.
- During testing: the agent selects the task with the highest probability (greedy selection).

6. Coupling via Reward Design

Reward design is one of the most critical aspects of any reinforcement learning (RL) framework, as it directly influences the learned behavior of the agent. In our DC-MRTA framework, we explore how reward formulation can either couple or decouple high-level task allocation from low-level robot navigation.

6.1. Decoupled Reward (Baseline)

A naive approach would be to define a binary reward:

- Reward = 0 if the task is completed,
- Reward = 1 otherwise.

While simple, this reward structure is agnostic to the process by which the task is completed. It does not consider the navigation effort, time taken, or collisions encountered, leading to a complete decoupling between task allocation and navigation trajectories.

6.2. Coupled Reward (Proposed)

To better align task assignment decisions with navigation outcomes, we define a reward that captures the navigation effort required to reach a task. Specifically, the reward is defined as:

$$\text{Reward} = -\text{Time}(\mathbf{o}_i, \mathbf{p}_i) \tag{4}$$

where:

- $\mathbf{p}_i$ is the current location of robot i,
- $\mathbf{o}_i$ is the start location of the assigned task,
- $\text{Time}(\mathbf{o}_i, \mathbf{p}_i)$ is the time taken to travel from $\mathbf{p}_i$ to $\mathbf{o}_i$,

This reward formulation effectively penalizes longer travel times and encourages efficient task assignments. It also introduces a coupling between task allocation and the navigation module, which is responsible for computing travel time using a decentralized path-planning algorithm. This coupling ensures that the high-level decisions are informed by real-world feasibility.

6.3. *State Transitions*

The RL environment is defined by asynchronous state transitions based on robot availability. At a given state s_t, we have:

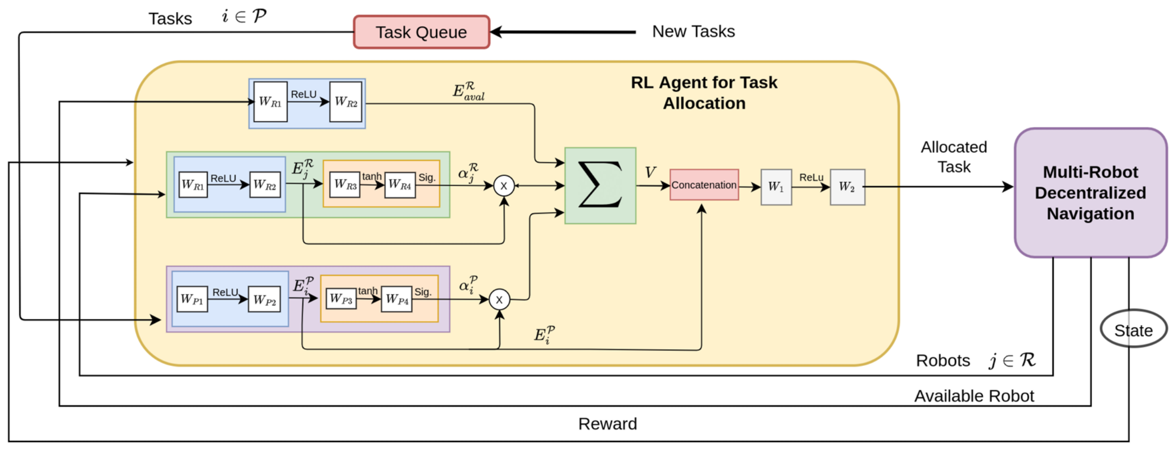
- A task list $\mathscr{P}$ in queue,
- One available robot $i \in \mathscr{R}$.

The policy $\pi(a_t|s_t)$ selects a task to assign to the available robot. Upon assignment:

- The selected task is removed from the queue,
- A new task may be added,
- The robot becomes unavailable until the task is completed.

The next state s_{t+1} is defined when the next robot becomes available. Unlike conventional RL environments, these state transitions do not occur at fixed intervals, making this an event-driven process.

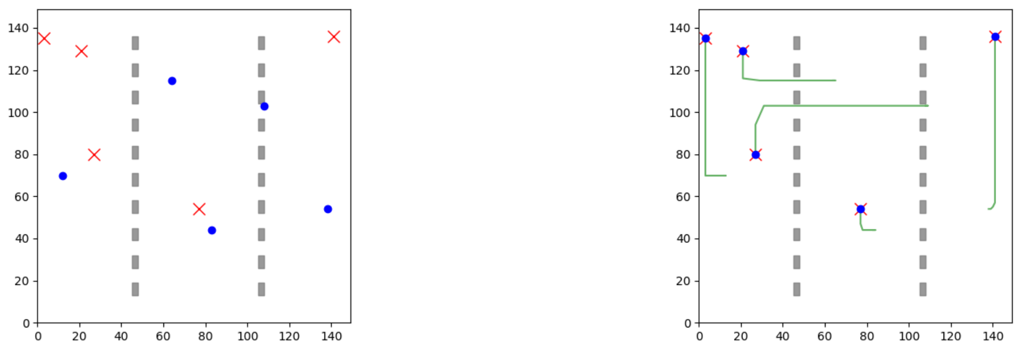
7. Pictorial Representation of Model:



Tasks $i \in \mathcal{P}$
Task Queue
New Tasks
RL Agent for Task Allocation
W_{R1} ReLU W_{R2}
$E^{\mathcal{R}}_{aval}$
W_{R1} ReLU W_{R2} $E^{\mathcal{R}}_j$ W_{R3} tanh W_{R4} Sig. $\alpha^{\mathcal{R}}_j$
W_{P1} ReLU W_{P2} $E^{\mathcal{P}}_i$ W_{P3} tanh W_{P4} Sig. $\alpha^{\mathcal{P}}_i$
$E^{\mathcal{P}}_i$
$\sum$
V
Concatenation
W_1 ReLu W_2
Allocated Task
Multi-Robot Decentralized Navigation
State
Robots $j \in \mathcal{R}$
Available Robot
Reward

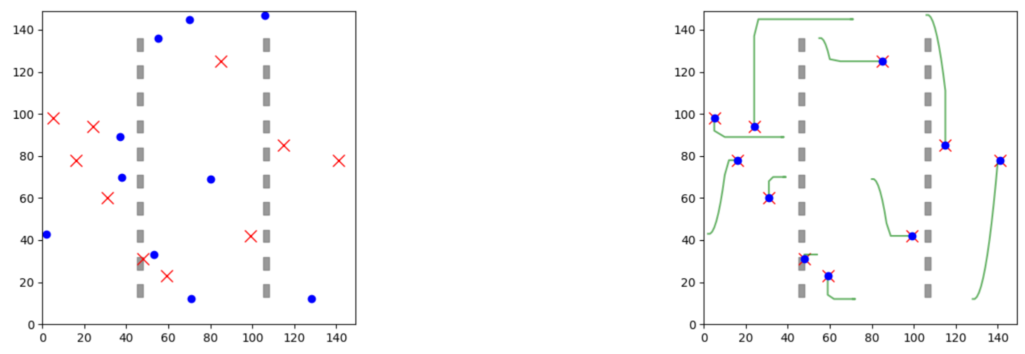
Fig. 1: Reinforcement Learning-based Task Allocation and Navigation Architecture

8. Code/Algorithm Implementation Results



(a) Start State for Robots (b) Goal State reached by robots

Fig. 2: DC-MRTA Based Navigation Results with 5 Robots and 10 Obstacles Setting



(a) Start State for Robots

(b) Goal State reached by robots

Fig. 3: DC-MRTA Based Navigation Results with 10 Robots and 10 Obstacles Setting

9. Conclusion, Limitations and Future Work

- **Proposed Method:** A two-level framework for joint task allocation and decentralized navigation in complex warehouse settings.
  - Lower level: Decentralized, collision-free navigation (e.g., using ORCA).
  - Higher level: RL-based task allocation guided by navigation feedback.
- **Key Contribution:** Coupling navigation and task allocation via shared rewards improves coordination.
- **Performance:** Achieves up to 14% improvement in task completion time over prior approaches.
- **Limitations:**
  - Decentralized paths may not be optimal or always collision-free.
  - Rewards do not yet factor in congestion or agent density.
  - Assumes homogeneous agents and full obstacle awareness.
- **Future Work:**
  - Design improved reward functions for congestion handling.
  - Test performance with dynamic layouts and human-like obstacles.
  - Explore limited local communication among agents for better coordination.

10. References

DC-MRTA Research Paper Link