A Literature Review: Prioritized Trajectory Planning and Optimization for a system of Non-holonomic Mobile Robots

Abhijay Singh[1, *] and Tharun Puthanveetil[1, †]

[1] *Maryland Applied Graduate Engineering, University of Maryland, College Park, Maryland, USA*

(Dated: November 25, 2022)

This report is intended to provide a detailed overview of the work done by Juncheng, Maopeng, Lihua in [1]. In [1], a trajectory optimization algorithm is proposed to generate optimal and guaranteed trajectories for the multi-robot system in obstacle-rich environments. The first stage of the algorithm involves the generation of discrete paths for the individual robots in the system followed by a constraint-based optimization to refine the paths. Collision-free traversal is achieved through the construction of Safe Corridors for each robot in the system. The novel prioritized trajectory optimization strategy designed in [1] enables trajectory planning for large-scale robotic systems.

We have focused on understanding the novel trajectory planning technique proposed in [1]. Further, we have analyzed and discussed the methodology of the solution. The simulation results are replicated and the significant inferences stated by the authors are validated. Finally, the hyper-parameters of the solution are altered to extract additional inferences.

I Introduction

The main goal of [1] is to develop an efficient collision-free multi-robot trajectory planning strategy for a large group of non-holonomic mobile robots. A priority-based trajectory optimization technique is proposed to obtain computationally efficient trajectories for multi-robot systems.

The efficacy of most modern applications like robot-based material handling in warehouses and drone-based package delivery systems improves multi-fold with the deployment of multi-robot systems. As this ensures quicker and resource-efficient task completions. For the successful operation of these multi-robot systems, path-planning algorithms that can generate guaranteed collision-free trajectories in any obstacle-rich environment are an unavoidable requisite. The generation of optimal paths in single robot systems has been successfully handled with the formulation of novel search algorithms. Extending these trajectory generation algorithms to multi-robot systems becomes a convoluted problem that demands attention to significant details like inter-robot collision, mutually supportive decision-making, and resource-constrained deployment.

Typically, multi-robot planners are based on coupled trajectory-generation methods where the trajectories of all robots are jointly optimized. Despite these solutions being able to provide guaranteed solutions, they tend to fail in cases where the number of robots in the system and obstacles in the environment are large. In comparison, in decoupled multi-robot trajectory planning techniques where planning for each robot is performed in a decoupled sequential fashion, the trajectory for each robot is planned while keeping the path plans of previously planned robots as a constraint.

Conventionally, trajectory generation problems for single robot systems are solved with a two-stage pipeline. The first stage is path-finding and the subsequent is the optimization of the generated path. The same is also found to provide efficient and guaranteed solutions for multi-robot trajectory optimization problems. But these solutions are focused on robots with linear dynamics. With most modern mobile robots being differential drive based, the consideration of non-linear dynamics that govern the functioning of these robots is significant in the trajectory-planning process. To ensure that trajectories are feasible, the two-step method used in [1] directly takes into account the nonlinear motion model of non-holonomic mobile robots.

Distributed path planning methods are also a prominent alternative to solving trajectory planning problems. However, despite being computationally efficient, these methods are not well-suited for complex environments. Hence in [1], the authors have opted for a centralized trajectory planning method which performs planning in a centralized fashion for all the robots in the system prior to the start of the operation.

In this literature review, we focus on understanding the novel trajectory planning technique proposed in [1]. Further, we analyze and discuss the methodology of the solution. The simulation results are replicated and the significant inferences stated by the authors are validated. Finally, the hyper-parameters of the solution are altered to extract additional inferences.

A Background Information

Non-holonomic mobile robots:
Robots can be broadly classified into two categories: mobile and stationery. Although stationary robots are fixed, they exhibit motion confined to a specific bounded space. Based on motion, robots can be further classified as either holonomic or non-holonomic robots. Non-holonomic mobile robots are robots whose control-

* abhijay@umd.edu UID : 118592619
† tvpian@umd.edu UID : 119069516

lable degree of freedom is less than the total degrees of freedom. A ground mobile robot with differential drive has three degrees of freedom; i.e. its position in two axes and its orientation. However, there are only two controllable degrees of freedom which are acceleration (or braking) and the turning angle of the steering wheel. In [1], the authors propose an optimization strategy for 4-wheeled-mobile robots. Since the robots in consideration in [1] do not have castor or Omni wheels that would allow movement in any direction, these robots can also be categorized as non-holonomic.

Collision Models:
Collision models are standard geometries used to represent the physical space occupied by a robot or obstacles in an environment. With the use of standard geometries, the mathematical computations involved in quantifying interactions can be minimized without compromising on the details[2]. In order to define the interactions of the robots with each other or the obstacles in the environment it is important to define their collision models. For instance, a collision model for a mobile robot can be as simple as a circle in 2D space, and for that, the link of the manipulator can be a cylinder. The objective while defining collision models is to choose one or a combination of a few standard geometries to represent the physical volume of the robot or obstacles as closely as possible.

Unicycle model:
The simplest kinematic model of a robot is the unicycle model, which relies on the idea that the robot can be represented by a single wheel[3]. The kinematic restrictions in this situation are the same as those that apply to a wheel rolling on a plane. Fig. 1 provides a condensed diagram outlining the generalized coordinates of this model.

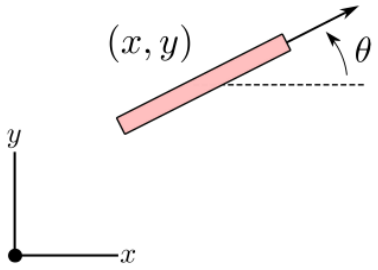


FIG. 1. Unicycle model[3]

The figure represents a mobile robot with a unicycle model. The Center of Mass of the robot is positioned at (x,y). It is turned by an angle θ along the z-axis(axis of rotation.)

The kinematic model for the configuration is as given below:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{1}$$

where v is the forward speed of the unicycle and is the rate of rotation. The unicycle model has the advantage of being straightforward and being able to model one of the most fundamental behaviors of wheeled robots. Such a model would be appropriate for more complex motion planning tasks, such as laying out geometric routes for a robot to follow.

CBS:
Constrained Based Search(CBS) as the name suggests is a constraint-based path-planning algorithm for solving Multi-Agent Path Finding (MAPF) problems It works in two levels. At high-level constraints are generated for each individual agent in the system[4]. At the low-level paths for the individual agents are found considering their implicit constraints.

The high level: At the high level, CBS searches the constraint tree (CT) which is a binary tree.

Each node N in the CT as shown in Fig. 2 contains:
(1) A set of constraints (N.constraints), imposed on each agent.
(2) A solution (N.solution). A single solution (i.e) a path from source to goal for each agent in the system.
(3) The total cost(N.cost): Total cost of the current solution.

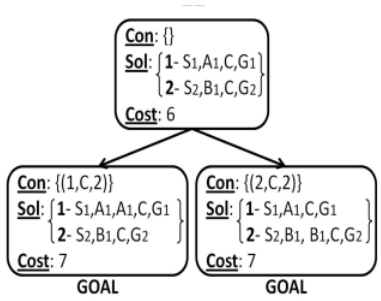


FIG. 2. Constraint Tree (CT)[4]

Constraint tree having 3 nodes. Each node except the root node has a set of constraints and a total cost.

The initial node or the root of the CT consists of an empty set of constraints. Each successor of a node in the CT inherits the constraints of its respective parent node. Consequently, it adds a new constraint for a single agent. N.solution is obtained by performing a local search using any single-agent path-finding algorithm like A*. In order to verify the validity of the present node as a goal node, the planned path for all the agents given this solution is checked for a no-conflict condition. At the high level, a best-first search based on cost is executed to extract the most optimal set of paths for the agents.

II Literature Overview

A Problem Formulation

In [1], the scenario considered is that of a multi-robot system comprising of N mobile robots whose 2D workspace is defined as $\mathcal{W} \subseteq \mathbb{R}^2$. The space in the real world where the robot can perform the free movement to undertake the required actions is termed its workspace. The obstacles in the environment are considered to be known and denoted as $\mathcal{O}$. The free workspace of the robot is defined by $\mathcal{F} = \mathcal{W} \backslash \mathcal{O}$. A circular collision model of radius R is defined around each of the robots. The subset of the workspace $\mathcal{W}$ occupied by the robot body denoted by $\mathcal{R}(x)$ where x denotes a position such that $x \in \mathbb{R}^2$. The task of every robot in the system is to move from start position $s_i \in \mathcal{F}$ to goal position $g_i \in \mathcal{F}$. Both $s_i \in \mathcal{F}$ and $g_i \in \mathcal{F}$. In order to guarantee there exist no collision at the start and end positions, each task is defined with conditions $\mathcal{R}(s_j) \cap \mathcal{R}(s_k) = \emptyset$ and $\mathcal{R}(g_j) \cap \mathcal{R}(g_k) = \emptyset$ for all $j \neq k$.

The start and end point of a path associated with the task of a robot is connected by a sequence of waypoints. The path of waypoints is denoted by $p = \left\{r^k\right\}_0^{N_p}$ where $r^k = \left[x^k, y^k, \theta^k\right]^{\mathrm{T}}$ denotes the kth waypoint on the path and N_p denotes the total number of waypoints in a path. The trajectory v_i between two points is defined as path p_i parameterized by time t. The 2D position of the robot at waypoint r is defined as $\mathrm{pos}(r)$. To ensure there exist no collision between robots with the obstacles in the environment, each robot i at all waypoints in its trajectory should be at a free workspace ,i.e, $\mathcal{R}(\mathrm{pos}(r_i(t))) \subseteq \mathcal{F}$. Similarly, for avoiding inter-robot collision, no two robots should be at the same waypoint at the same time while tracing its trajectory,i.e, $\mathcal{R}(\mathrm{pos}(r_i(t))) \cap \mathcal{R}(\mathrm{pos}(r_j(t))) = \emptyset, \forall i \neq j$.

The kinematic model for each robot is defined as

$$\dot{z} = f(z, u), u \in \mathcal{U}. \tag{2}$$

It is important to define the kinematic model to understand and analyze whether the planned trajectory is dynamically feasible with respect to the physical aspects of the robots under consideration. The authors in [1] use a unicycle model for differential drive robots to model the kinematics of robots used for the experiments. The z is defined as $[x, y, \theta]^{\mathrm{T}}$ with x, y, and θ representing the position and orientation respectively. The motion equations that define how the linear velocity in the $x-y$ direction and the angular velocity about the z-axis for the robots are affected by the velocity control input $u = [v, \omega]^{\mathrm{T}}$ are given by:

$$\dot{x} = v\cos(\theta), \quad \dot{y} = v\sin(\theta), \quad \dot{\theta} = \omega \tag{3}$$

where v is the linear velocity and ω is the angular velocity from eqn.1.

A maximum velocity constraint is also imposed on each of the robots such that $\mathcal{U}_i = \left\{[v_i, \omega_i]^{\mathrm{T}} : |v_i| < v_i^{\max}, |\omega_i| < \omega_i^{\max}\right\}$. The final trajectories to be generated for all the N robots which are dynamically feasible and optimal are defined as $\nu_1, \ldots, \nu_N$.

B Methodology

The end-to-end architecture for the multi-robot trajectory planning approach proposed in [1] is as shown in Fig. 3. The entire pipeline can be divided into three important stages. Firstly, a variant of CBS is used to obtain the shortest collision-free paths for all the non-holonomic constraint-based agents in the system. Secondly, geometrical safe corridors in the shape of convex polyhedrons that represents safe space are constructed around the path planned for each robot. The planned paths and the associated safe corridors are further used to formulate a constrained non-linear optimization problem. Finally, the optimization problem is solved using the proposed prioritized trajectory optimization to generate safe and near-optimal trajectories for the robots in the system. The significance and details of each stage are elaborated in the following sections.

1 Discrete Path Planning

The space in the real world where the robot is enabled to perform free locomotion to perform the required actions is termed its workspace[2]. For the robots under consideration in this paper, the workspace is defined as an undirected graph denoted as $\mathcal{G} = \mathcal{V}, \mathcal{E}$. Each vertex v belonging to the set V and each edge $(v_i, v_j) \in \mathcal{E}$ represents the vertices and edges in the graph that the robots can occupy and traverse respectively. The 2D grid graph mentioned above is directly obtained from the occupancy map of the environment in which the experiment is conducted. Occupancy maps [5] are meant to probabilistically quantify the presence and absence of an object or an obstacle in a 3D space onto a 2D grid representation.

The traditional grid graphs consider only the position and fail to account for the orientation information of the robots. As a result of which it is not suitable to localize non-holonomic robots in 2D space. Therefore, the authors have come up with a novel graph representation as shown in Fig. 4. As per the new representation, the position, as well as the orientation of the robot, is noted at each vertex v of graph $\mathcal{G}$. The position at each vertex is represented as $P(v)$, and orientation or heading is chosen from four main directions which are north, south, east, and west. Here, each vertex in $P(v)$ should satisfy $\mathcal{R}(P(v)) \in \mathcal{F}$ in order to prevent collisions.

In order to understand how the robots make a state transition, it is essential to define the motion primitives [6] of the robots. Motion primitives are the pre-defined motions, a robot can take at a given state. In this case, the robots are constrained to take only one or a combina-

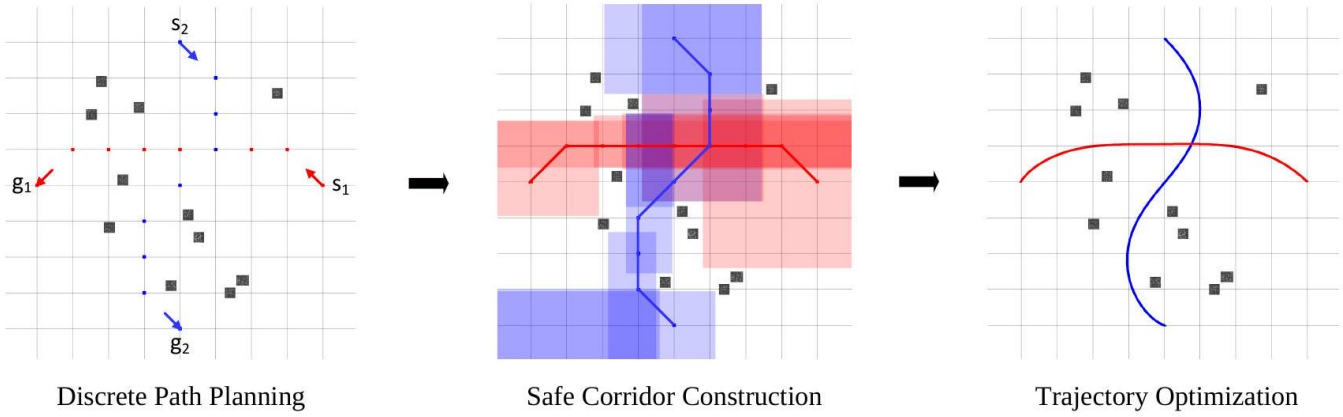


FIG. 3. An overview of the architecture of the proposed multi-robot trajectory planning approach. The gray squares represent the static obstacles in the multi-agent environment. In this example, the start and goal positions of agents 1 and 2 are assigned as $s_1 = (4, 0), g_1 = (4, 0), s_2 = (0, 4), g_2 = (0, 4)$. Firstly, the shortest collision-free paths for the robots need to be found. Secondly, the construction of the safe corridors (red and blue blocks) along the planned paths has to be done. Finally, a constrained non-linear optimization problem is solved to generate smooth and dynamically feasible trajectories for the robots[1].

tion of the three defined actions: moving forward, moving backward, and turning 90 degrees. These together constitute the motion primitives of the robots and are denoted by $\mathcal{A}$.

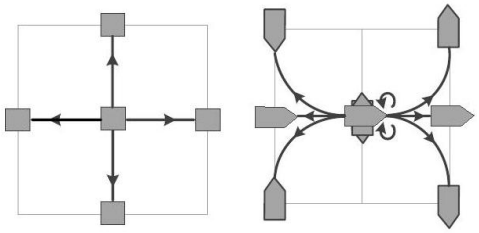


FIG. 4. Edges in the proposed grid graph. Figure on the left shows the traditional 4-connected edges. The edges in the graph are defined by the motion primitives, as shown in the figure on the right[1].

A state transition made with a motion primitive in $\mathcal{A}$ results in the formation of an edge in the grid graph $\mathcal{G}$. However, as stated in subsection I A of background information, each edge or the transition that represents the edge is considered valid only if there arises no inter-robot collision along the edge during the transition.

As the graph construction process carefully adheres to the kinematics defined for the robots it is essential to validate the feasibility of the motion primitives. Given the grid size of graph $\mathcal{G}$ to be D and ΔT to be the time required to traverse one edge of the graph, the motion primitives are considered feasible only if the following condition for a one-step diagonal transition is guaranteed to be feasible.

$$v_i^{\max}\Delta T \geq \frac{\pi}{2}D, \quad w_i^{\max}\Delta T \geq \frac{\pi}{2}, \quad \forall i \in \{1, \cdots, N\} \tag{4}$$

If the above conditions are satisfied, all actions within $\mathcal{A}$ can be considered possible. Therefore it can be incurred that the user-defined parameters such as ΔT and D can be chosen to construct graphs representation for environments of different scales.

Considering the path planning for each robot is done separately, the chances for conflicts are to be considered. Two probable conflicts that could arise in graph $\mathcal{G}$ as a result of the above are vertex conflict and edge conflict. A vertex conflict denoted by $\langle i, j, k\rangle$ occurs when two robots i and j occupy two vertexes that are too close to each other at any timestep k. Similarly, if two robots are too close to each other when traversing their own edge between timesteps k_1 and k_2, it represents an edge conflict and is denoted by $\langle i, j, k_1, k_2\rangle$.

As per the notations used above, the vertex conflict set at timestep k and edge conflict set at timestep (k_1, k_2) is described as:

$$\begin{aligned}\mathrm{VCon}(k) = \{\langle i, j\rangle \mid \\ \mathcal{R}\left(\mathrm{pos}\left(r_i^k\right)\right) \cap \mathcal{R}\left(\mathrm{pos}\left(r_j^k\right)\right) \neq \emptyset\}\end{aligned} \tag{5}$$

$$\begin{aligned}&\mathrm{ECon}\left(k_1, k_2\right) = \{\langle i, j\rangle \mid \\ &\mathcal{R}\left(\mathrm{pos}\left(r_i(t)\right)\right) \cap \mathcal{R}\left(\mathrm{pos}\left(r_j(t)\right)\right) \neq \emptyset, \forall t \in (k_1\Delta T, k_2\Delta T)\}\end{aligned} \tag{6}$$

With the basic conditions described, the problem to be solved is that of finding conflict-free paths for multiple agents in a defined graph. The formulated multi-agent path-finding problem is solved using a variant of CBS called Enhanced Conflict Based Search(ECBS). ECBS as a discrete path planner is capable of providing bounded sub-optimal solutions with guaranteed completeness. Similar to CBS, ECBS works on two levels where a binary constraint tree(CT) is constructed to resolve conflicts at the high level, and at the low level,

optimal paths for individual agents are planned which are consistent with their own constraints. In comparison to other CBS techniques, ECBS applies a focal search with conflict heuristics at both high-level and low-level making them efficient for solving MAPF problems.

A conflict scenario is handled by ECBS as follows: 1) Initially, the low-level planner assigns the root node of the tree with an initial solution as there to exist no inter-robot constraints to be handled. 2) When a vertex conflict $\langle i, j, k\rangle$ is spotted, two child nodes are added to the CT to resolve the conflict. 3) the First node adds a constraint for agent i to avoid staying at r_i^k at timestep k. Similarly, the second node adds a constraint for agent j to avoid staying at r_j^k at timestep k. 4) An edge conflict is also handled in a similar manner (i.e) whenever an edge conflict $\langle i, j, k_1, k_2\rangle$ is encountered, two child nodes are created with each having a separate constraint added to prevent either of the agents i and j from traversing the same edge. The CT expansion continues till a fully conflict-free solution, i.e a path is obtained for all the agents in the system.

As it is important to check the relative distance between the robots at each timestep in the trajectory optimization process, it is necessary to keep the length of the discrete path generated for each agent to be same. Hence, if the length of the longest path is M, paths p_i of each agent i respectively are appended with goal position g_i until their paths are also of length M.

2 Safe Corridor Construction

In order to ensure that each robot has a collision-free passage from the start to the goal position, a valid corridor that indicates a safe region around the planned discrete path needs to be created. The discrete path of each robot obtained from the previous step is a collection of waypoints in that robot's free working space. This collection of waypoints is denoted by $p = \left\{r^k\right\}_0^M$ where each waypoint r^k is in the free space $\mathcal{F}$. Line segment $I^k = \left\langle r^{k-1} \rightarrow r^k\right\rangle$, denotes the kth line segment in the path. A convex polyhedron is generated around each line segment k of the discrete path to construct a safe corridor. This safe corridor indicates a safe region of passage for each agent in the multi-robot system.

The convex polyhedron constructed around every line segment I^k in the path p is denoted as $\mathcal{S}^k$. The following constraint is applied to each convex polyhedron to ensure that an agent in any given polyhedron $\mathcal{S}^k$ of the discrete path does not collide with an obstacle in the environment,

$$\mathcal{R}(a) \cap \mathcal{O} = \emptyset, \forall a \in \mathcal{S}^k. \tag{7}$$

The collection of all such convex polyhedra denoted by $\mathrm{SC}(p) = \left\{\mathcal{S}^k \mid k = 1, \ldots M\right\}$. The safe corridor is sequentially connected and satisfies the condition:

$$\mathcal{S}^k \cap \mathcal{S}^{k+1} \neq \emptyset, \quad \forall k \in \{1, \cdots, M-1\}. \tag{8}$$

Algorithm 1 Safe Corridor Construction

Require: discrete path $p = \{r^k\}_0^H$
  function SAFE CORRIDOR(p)
    for $k \leftarrow 1$ to H **do**
      if $r^k \in \mathcal{S}^{k-1}$ **then**
        $\mathcal{S}^k \leftarrow \mathcal{S}^{k-1}$
      else
        $\mathcal{S}^k \leftarrow r^k$
        $\mathcal{D} \leftarrow \{+x, -x, +y, -y\}$
        while $D \neq \varnothing$ **do**
          for d in D **do**
            $\mathcal{S}^k \leftarrow$ expand $\mathcal{S}^k$ in the direction d
            if $\mathcal{S}^k \subseteq F$ **then**
              $\mathcal{S}^k \leftarrow \tilde{S}^k$
            else
              $\mathcal{D} \leftarrow D \backslash d$
            end if
          end for
        end while
      end if
    end for
    return $\mathrm{SC}(p) = \{\mathcal{S}^k | k = 1, \ldots, H\}$

The safe corridor construction is achieved by employing an axis-search method to build the safe corridor[7]. Each convex polyhedron constructed around a line segment in the path is expanded along the x and y axes to cover the maximum possible free space. As described in the introduction, the motion primitives of the robots are included in the discrete path planning stage. Hence, the planned paths consist of horizontal, vertical, and diagonal line segments that connect the start and goal positions for each agent in the system. However, as shown in Fig. 5, the initial safe corridor around diagonal line segments may not lie completely within the valid free space of a robot.

In [1], it was found that sub-sampling each line segment in the discrete path would solve this problem. The discrete path is modified by dividing each line segment

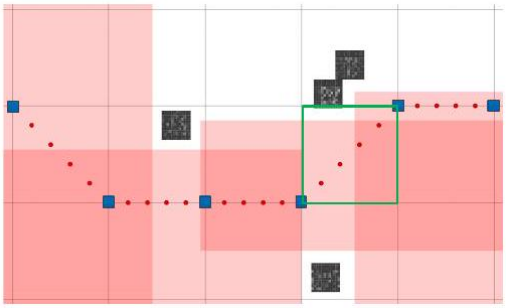


FIG. 5. Safe corridor construction. Due to a failed scenario depicted by the green box, the original discrete path, shown in blue squares, cannot be used to directly build the safe corridor. The safe corridor is constructed by expanding each point in the discrete path (represented by red dots) in the x and y axes[1].

into h equal parts. The new sampled path consists of $H = 1 + h(M-1)$ waypoints. Initially, the safe corridor S^k is set to be the point r^k and is expanded in the x and y axes to cover the maximum possible free space. The continuity in the safe corridor is guaranteed by ensuring that a minimum safe region overlaps between consecutive waypoints. The following condition enforces continuity in the safe corridor:

$$\frac{\sqrt{2}D}{h} < 2R_i, \quad \forall i \in \{1, \cdots, N\}, \tag{9}$$

where R_i is the radius of the collision model of the robot i. In addition to this, if a waypoint r^k exists in the previous safe region $\mathcal{S}^{k-1}$, the safe corridor expansion is not carried out and the convex polyhedron $\mathcal{S}^k$ is the same as $\mathcal{S}^{k-1}1$. The safe corridor construction process of each robot's path is summarized in Algorithm 1.

3 Trajectory Optimization Problem

In this stage, the discrete paths generated for each agent in the path-planning phase are converted into smooth, feasible, and traversable trajectories. The paths for each agent are converted to trajectories by assigning a time $t^k = k\Delta t$ to reach each waypoint in their sampled path $p = \left\{r^k\right\}_0^H$. As a result, the reference trajectory generated can be denoted as $\nu^r = \left\{r^k, t^k\right\}_0^H$. The paths between each waypoint are marked by line segments and each of these segments is enclosed by safe corridors to indicate the free and safe space the robots can occupy. As mentioned in the previous section, in order to prevent the created safe corridor space from accidentally including obstacles in the environment, each segment between waypoints is further sub-sampled into h equal parts. With the new $\Delta t = \Delta T/h$, the optimal trajectory generated from the reference trajectory ν^r and the constructed safe corridors $\mathrm{SC}(p)$, can be denoted as $\nu = \left\{z^k, t^k\right\}_0^H$. The optimal trajectory obtained for each robot is such that each of the agents can reach its goals and avoid any collision.

The nonlinear optimization problem is formulated as:

$$minimize \sum_{i=1}^{N} \left(\sum_{j=1}^{H-1} \Delta u_i^{j^{\mathrm{T}}} P \Delta u_i^j + \sum_{j=1}^{H} \widehat{z}_i^{j\mathrm{T}} Q \widehat{z}_i^j \right) \tag{10a}$$

$$\text{s.t.} \quad z_i^0 = s_i, z_i^H = g_i \tag{10b}$$

$$z_i^{j+1} = f\left(z_i^j, u_i^j\right) \tag{10c}$$

$$\mathrm{pos}\left(z_i^j\right) \in \mathcal{S}_i^j, u_i^j \in \mathcal{U}_i \tag{10d}$$

$$u_i^j \in \mathcal{U}_i \tag{10e}$$

$$\left\| \mathrm{pos}\left(z_{i_1}^j\right) - \mathrm{pos}\left(z_{i_2}^j\right) \right\| \geq R_{i_1} + R_{i_2}, \forall i_1, i_2, j, i_2 \neq i_1 \tag{10f}$$

where $\Delta u_i^j = u_i^j - u_i^{j-1}$, $\widehat{z}_i^j = z_i^j - r_i^j$, and $P \in \mathbb{R}^{2\times 2}$ and $Q \in \mathbb{R}^{3\times 3}$ are two positive definite weighting matrices.

The objective function has two main constituents. The first part quantifies differences between successive control inputs and proportionally impose penalties. A larger difference in the control inputs implies more uneven trajectories and vice versa. The second part aims to minimize the difference between the reference trajectory and the optimal trajectory obtained from it. The idea is that the reference trajectory by itself is a feasible solution and hence a deviated optimal trajectory implies extra computations. Therefore with the minimization of the first part, a smoother trajectory is obtained and with that of the second part, a reference trajectory as close as possible to the optimal trajectory is generated. Equation 10c describes the state transition from state j to state $j+1$ for an agent i with the control input u_{ij}. Condition 10d ensures that the position of every waypoint in the generated trajectory lies within the constructed safe corridor and 10e limits the input control signals to be within the physically admissible values. Finally, 10f states that the minimum possible safe distance between robots should be the sum of the radii of their respective collision objects.

4 Prioritized Trajectory Optimization

The efficiency of the solution of the nonlinear optimization problem is inversely proportional to the number of robots that make up the multi-agent system. The authors in [1] introduce a priority-based trajectory optimization method to improve the drop in efficiency in large multi-agent systems. In the proposed method, the multi-agent system is first subdivided into groups of robots based on unique characteristics, then a priority level is assigned to each of these groups. Based on the assigned priority level, the trajectory optimization problem is solved sequentially on the robots from the highest-priority to the lowest-priority groups. This sequential solution involves the iterative optimization of the trajectories of each robot in a group based on the constraint that they must avoid collisions with all the robots in higher-priority groups. The decoupling of the optimization framework into priority assignments and then optimizing improves the computational efficiency in scenarios with large-scale robot teams. However, the grouping of robots based on unique properties is scenario-specific. This leads to infeasible optimization subproblems for lower-priority robots due to a lack of consideration by higher-priority robots. To ensure that an optimized solution is feasible, [1] proposes a novel grouping and priority assignment strategy that groups robots based on their relative positions in the grid map.

The grouping of robots in prioritized trajectory optimization is based on the inter-robot constraint described in the previous section. As the safe corridor describes the safe region that a robot can take to reach the goal position, all trajectories have to lie in the safe corridor. Hence, the reference trajectory describing the discrete

path is close to the optimal trajectory. Since the optimal trajectories are similar to the reference ones, the inter-robot constraint of the optimization problem can be analyzed on the reference trajectories $\nu_1^r, \ldots, \nu_N^r$. At each timestep t, the grouping of robots into triplets $(i_1, i_2, i_3)^t$ satisfies the condition,

$$\left\|\text{pos}\left(r_a^t\right) - \text{pos}\left(r_b^t\right)\right\| \leq D_{\text{th}}, \forall a, b \in \{i_1, i_2, i_3\}, b \neq a \tag{11}$$

where $D_{\text{th}} = \sqrt{2}D$ is a threshold. The constraint ensures that each triplet describes a scenario where three robots are close to each other in the graph at a specific timestep. Although the inter-robot collision analysis in the discrete path planning stage requires that four robots be present at the vertices of a grid cell at a given time, it is found that the constraint can be satisfied even with three robots. However, scenarios may arise where each robot in a triple is assigned a different priority and this enforces a hard constraint on the lower-priority robots in the same group. To ensure that the feasibility of the optimization problem is not lost, it is required that the three robots in a triple form a group and be assigned the same priority level. The list of all triples that satisfy the inter-robot constraint is given by L.

As the number of robots in a multi-agent system increases, two outcomes are likely. First, as the elements in L becomes large, some robots may be included in different triples of L. In this case, the robots cannot be grouped directly. Second, an element in L may occur more than once. In this case, it is deduced that the trajectories of the robots that comprise this repeated element of L are highly coupled. To ensure that repeated robots and highly coupled trajectories are optimized first, a higher priority is assigned to the repeated elements of the list. Hence, the priority assignment of the triples is based on the number of occurrences of each triple. Algorithm 2 describes the grouping and priority assignment process.

Algorithm 2 Grouping and priority assignment

Require: reference trajectories $\nu_1^r, \ldots, \nu_N^r$
  function PRIORITY ASSIGNMENT($\nu_1^r, \ldots, \nu_N^r$)
    for $t \leftarrow 1$ to H **do**
      $(i_1, i_2, i_3)^t \leftarrow$ find triple satisfies **11**
      Insert (i_1, i_2, i_3) into L
    end for
    while $L \neq \varnothing$ **do**
      $e \leftarrow$ most common element in L
      Append e to the group list G
      for l in L **do**
        for robot m in e **do**
          if robot m in l **then**
            Remove m from l
          end if
        end for
      end for
    end while
    for $n \leftarrow 1$ to N **do**
      if n not in G **then**
        Append n to G
      end if
    end for
    return G

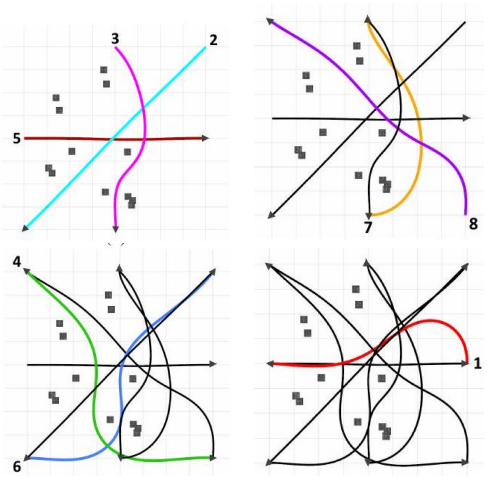


FIG. 6. An illustration of the proposed prioritized trajectory optimization approach. As described, lower priority groups must stay clear of all higher-priority robots. The colored lines in each image represent the optimized path of each robot. The black lines represent the path of higher-priority robots that the current set of robots needs to avoid.[1]

Initially, the mode of the list is selected and assigned the highest priority (first-priority group). As described earlier, it is possible for robots in this first-priority element to occur in other elements of L. Hence, the robots in this new group are removed from L and a new list $L^{'}$ is obtained. As a result of this operation, the element in $L^{'}$ may contain either one, two, or three robots. Following the removal of the repeated elements, the most occurring element in the updated list $L^{'}$ is selected as the second-priority group. Again, the repeated robots in the second priority group are removed from other elements of $L^{'}$. This process is continued until the list L is empty. It is possible that a robot in the multi-agent system may not be assigned a group due to its failure to satisfy the inter-robot constraint. In such a case, the completeness of the problem is lost. To ensure that robots with low coupling also get grouped and assigned a priority, these robots are treated as single-robot groups and are assigned the lowest priority. Finally, the optimization problem from the previous section is formulated for each group and solved sequentially from high-priority to low-priority. To ensure no collisions occur between robots of low-priority and high-priority groups, it is required that in addition to the inter-robot constraints between robots of the current group, inter-robot constraints between robots of optimized higher-priority groups also need to be considered.

An illustration of the overall prioritized trajectory optimization procedure is shown in Fig. 6. First, Algorithm 2 is used to group the eight robots. The groupings are given in order of priority, i.e. [(2, 3, 5), (7, 8), (4, 6),

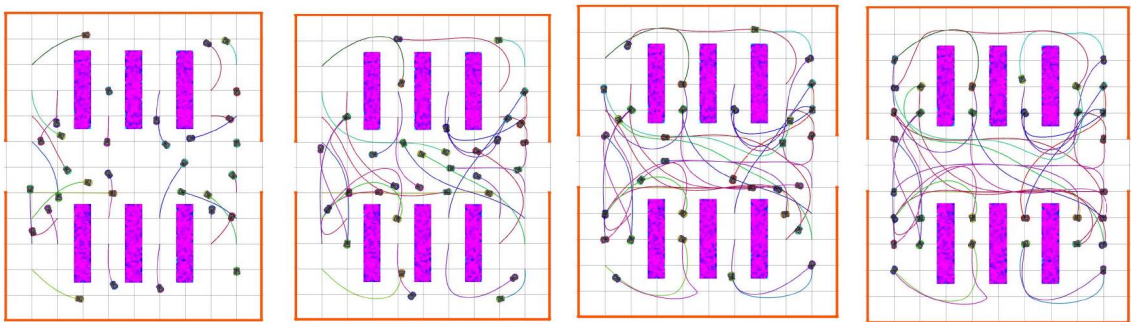


FIG. 7. A warehouse environment with a 32-agents system[1].

(1)] from high priority to low priority. To prevent collisions, inter-robot limitations between the present group of robots and the higher-priority robots that have been optimized should also be taken into account in each iteration of the prioritized trajectory optimization[1].

III Experimental Setup

A Experimentation Scenario

In [1], the proposed solution's primary application is material handling in warehouses. Hence, a warehouse scenario as shown in Fig. 7 is created in the simulation environment to assess the efficacy of the proposed scheme. The simulation world is 10 m x 12 m in size and has 6 shelves that are 3 m x 0.6 m in size. Each robot's start and goal positions are chosen at random at the environment's edge or at the pickup spots close to the shelves.

B Simulation

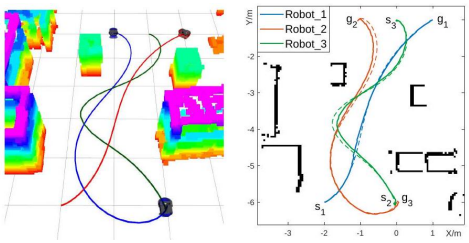
The simulation experiments of [1] are implemented in C++ and tested on a laptop with an Intel i5-6300HQ@2.30GHz CPU and 12GB of RAM running Ubuntu 16.04 OS. Octomaps are used to encode the occupancy map[28] of the environment. The trajectory optimization problem is solved using the interior-point nonlinear programming solver IPOPT[29]. IPOPT, which stands for "Interior Point Optimizer" is a software library written in Fortran and C, used for large-scale nonlinear optimization of continuous systems. Primal-dual interior point approach is used by IPOPT, and line searches based on Filter methods are also used (Fletcher and Leyffer). Numerous modeling platforms, including C, support calling IPOPT.

For conducting the simulations, in [1], the radius of the collision model of each was taken as $R = 0.15$m and the maximum velocities for each were limited such that v_{max} = 1m/s and ω_{max} = 1rad/s. Grid size of the graphs was taken to be $D = 1$m and the time required to traverse an edge of the graph was set as $\Delta T = 1.6$s ensuring that 4 is obeyed. Further, for the efficient construction of the safe corridors, each line segment between waypoints was divided into $h = 5$ equal parts. To meet the demand for computing efficiency, we set the suboptimal bound of ECBS at 1.5 based on the environment variables.

C Hardware

The hardware experiments were conducted with 3 Pioneer 3-AT robots in a practical multi-robot navigation experiment, which was carried out in an indoor testing space of 7 m by 8 m. The Velodyne VLP-16 3D lidar is mounted on one robot, while the Hokuyo 2D lidars, each with a maximum range of 30m, are mounted on the other two robots. LeGO-LOAM, a 3D lidar odometry, and mapping technology is first used to create a map of the test environment. The 3D maps are created as the trajectory planning algorithm must consider barriers of various heights to ensure safety. Each robot's collision model's radius is set to $R = 0.3$m, and its maximum speeds are constrained by $v_{max} = 0.6$m/s and $\omega_{max} = 0.6$rad/s. In addition, the experiment's time period T is set to 2.65 seconds, and all other parameters are left unchanged from III B description. Fig.8(a) displays a 3D map of the hardware experimental setup and Fig.8(b) shows the deviation between the generated trajectory and the path taken by the robot.

The AMCL ROS package was used to determine the robots' location in real-time. The optimal trajectory obtained was traced using a Model Predictive Controller(MPC). The hardware setup for the experiment is as shown in Fig. 9. Fig. 8(a) shows the 3D map of the experiment environment and Fig.8(b) shows the deviation between the planned path and the path traced by the robot in real-time with the help of the MPC. **Note: The hardware implementation and analysis are not considered within the scope of this report. The hardware experiment details of the [1] are included for giving the reader a better overall view of the proposed solution.**



(a) (b)

FIG. 8. Experiment results in [1] on the robot hardware. First figure describes the 3D map of the environment created using the mentioned SLAM algorithm. The images also illustrates the planned trajectory of each robot. Second figure shows the trajectory tracking results, where dash and solid lines represent the planned and real trajectories, respectively.

FIG. 9. Demonstration of the proposed approach on three robots in a real environment.[1]

IV Results and Discussions

The aforementioned simulation environment was set up for combinations of 4, 8, 16, 24, and 32 agents, with their start and goal positions specified, in order to thoroughly examine the time required for each step of the proposed strategy. The Table I summarizes the length of time needed to complete each step and the overall runtime for each combination. In a total of 6.1s, the suggested technique finished trajectory planning and optimization for the 32 agents combination. The time was calculated as the mean of the times that were recorded throughout the course of 40 simulation iterations. The authors in [1] have made a comparative analysis between the Coupled trajectory optimization and the Prioritized trajectory optimization techniques. The analysis is done on the basis of run time and cost. Couple trajectory optimization in this context refers to the proposed technique with the absence of any grouping strategy. The results of the observations are as tabulated in Table I, IV and V.

It was discovered that the coupled optimization technique took longer to complete and produce the outcomes of the trajectory optimization slower than the prioritized optimization technique. Additionally, the latter's runtime increases nearly linearly with the number of agents, whereas the former's runtime increases significantly with

Number of agents	Discrete path planning (s)	Safe corridor construction (s)	Trajectory optimization (s)	Total (s)
4	0.001	0.008	0.142	0.151
8	0.002	0.019	0.360	0.381
16	0.006	0.031	1.042	1.079
24	0.326	0.046	2.526	2.898
32	2.175	0.062	3.853	6.090

TABLE I. Computation time in each stage for the Prioritized Trajectory Optimization strategy in [1]

Grouping Strategy	Number of agents				
	4	8	16	24	32
Proposed Method	100	100	100	100	85
Random Grouping	100	100	100	85	75

TABLE II. Success rate of priority-based and random grouping strategies was calculated from 20 experiment iterations for different agent configurations.

the number of agents. This suggests that the former method is not well suited to managing large-scale systems. Given that the prioritized optimization employs a decoupled framework, it inevitably produces inferior results than the coupled optimization. According to Fig. 10, the total cost of the suggested approach is, on average, 3.7% higher than that of the coupled optimization, a rather insignificant and practical difference considering the guarantee provided by the approach. Although using II B 3 by itself could provide guaranteed solutions, in the end, to the trajectory optimization problem, there are numerous infeasible solutions that are to be handled as sub-problems in the process. This in turn could also negatively affect the runtime of the solution. Hence, as mentioned in section II B 4, the priority-based grouping strategy was introduced in contrast to the original random grouping approach. As expected, although both approaches were able to provide a 100% success rate with a smaller number of agents in the system, as the number of agents increased, the former failed less in comparison to the latter. In support of the above statement, it can be seen in Fig. 11 that the success rate of the coupled optimization strategy has dropped from 100% to 60% as the number of agents increased from 8 to 32. As stated by the authors it can be seen from the Fig. 11 that with the proposed grouping technique they were able to obtain almost 90% success rate even with the 32 agent configuration.

V Adaptation

A Code

We have used the original open source code made available by the authors of [1] and made appropriate changes for better implementation (code). The source files for the

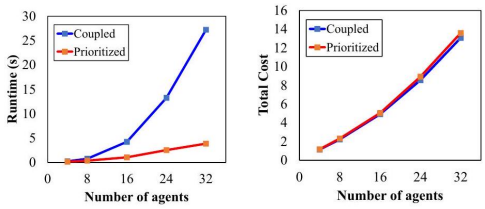


FIG. 10. Comparison of the proposed approach and the coupled trajectory optimization.[1]

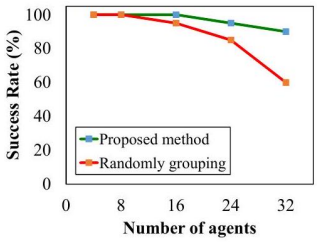


FIG. 11. Success rate of the trajectory planning using different grouping strategies.[1]

implementation of the respective sections are as given:

1) Discrete path planning: ECBS path planning algorithm is included in the *ecbs_planner.hpp* file.

2) Safe corridor construction: Safe corridor construction technique is included in the *safe_corridor.hpp* file.

3) Trajectory optimization: Priority grouping-based and random grouping-based trajectory optimization algorithms are included in *prioritized_optimization.hpp* & *coupled_optimization.hpp* files respectively.

The code used for miscellaneous functions like reading data and plotting are as given below:

1) For plotting the results in Fig.12 and Fig.13 - *runtime_vs_agents.py*.

2) For plotting the results in Fig.15 - *successRate_vs_agents*.

B Inferences

In this section, we have attempted to replicate the results and validate the inferences of the authors in [1]. The conducted simulation experiments were designed as per the details and hyperparameters given in Section III. The experiments were performed on a laptop with intel i7-9750H CPU@2.60GHz CPU and 16GB of RAM running Ubuntu 20.04 OS.

Table III elucidates the time taken for the completion of each stage of the prioritized trajectory optimization technique proposed in [1]. For each configuration (i.e) 4, 8, 16, 24, and 32, a total of 20 iterations of the experiments were performed and the mean was recorded as the final results of Table III. It can be seen that the total time for each configuration in our results is slightly less in comparison to what the authors of [1] have obtained.

Number of agents	Discrete path planning (s)	Safe corridor construction (s)	Trajectory optimization (s)	Total (s)
4	0.001	0.007	0.118	0.127
8	0.001	0.013	0.310	0.324
16	0.004	0.024	0.806	0.835
24	0.486	0.036	1.868	2.392
32	7.469	0.049	3.172	10.694

TABLE III. Computation time in each stage for the Prioritized Trajectory Optimization strategy in our adaptation

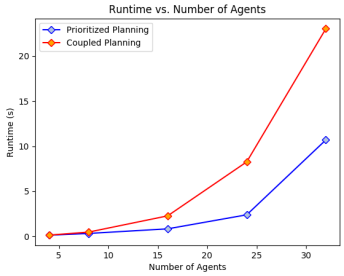
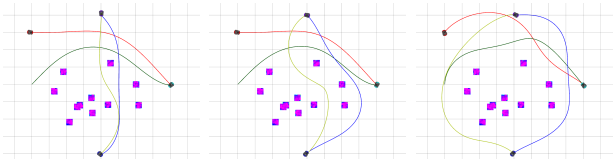


FIG. 12. Computed runtime versus the number of agents for both prioritized trajectory and couple trajectory optimization strategies

FIG. 13. Computed total cost versus the number of agents for both prioritized trajectory and couple trajectory optimization strategies

This can be credited to the superior specifications of the system in which the experiments were conducted. From Fig. 12, it could be inferred that the total time by coupled trajectory optimization is considerably higher when compared to the prioritized trajectory optimization strategy. However, an overall decrease in the time for both strategies when compared to that obtained by the authors of [1] is again because of the higher specifications of the system in which we conducted our experiments.

Similar to Fig. 10(a), Fig. 13 describes the relation between the cost and number of agents for both coupled and prioritized optimization strategies. It can be seen that in Fig. 13 also, the cost for prioritized optimization is approximately 7.9% greater than that of the coupled optimization strategy. As the authors of [1] have already justified this can be due to the fact that the prioritized trajectory optimization has to handle more infeasible solutions in the process of obtaining the most optimal one.



Radius = 0.15 Radius = 0.30 Radius = 0.50

FIG. 14. Deviation in the trajectory traced by the agents with variation in the Collision model radius

FIG. 15. Computed success rate versus the number of agents for both priority-based and random grouping for 20 iterations

Fig. 14 shows how the choice of collision radius for the agents affects the trajectory obtained for the given start and goal positions of the agents. The constraint 10f in the optimization equation 10a is what triggers the observed trajectory deviation in this case.

The success rate for both the priority-based grouping and random grouping strategy was evaluated for each of the agent configurations (i.e) 4, 8, 16, 24, and 32. For this, the experiments were performed 20 times by varying the start and end goal position of the agents in each configuration. Fig. 16 shows 5 out of 20 samples considered for the evaluation of the success rate for each of the configurations. The obtained results are shown in Fig. 15. In alignment with [1], the priority-based grouping is seen to have a greater success rate compared to random grouping strategies for configurations with a large number of agents. The two reasons why we have not obtained an exact numerical match of the authors' results are 1) Lesser number(20 in place of 40) of experiment iterations and 2) Randomness in the selection of the start and goal position for agents in each configuration. Besides tolerable variations, all the results we have obtained are consistent with that obtained in [1] and hence validate the claims of the authors.

VI Conclusion

In [1], an efficient priority-based trajectory planning strategy for multiple non-holonomic robots is proposed. The strategy is found to provide a guaranteed solution irrespective of the number of agents in the systems at a marginally higher but acceptable cost. All the results

Number of agents	Trajectory optimization (s)	Total (s)
4	0.126	0.137
8	0.453	0.471
16	2.234	2.274
24	8.170	8.264
32	20.325	23.025

TABLE IV. Computation time in each stage for the Coupled Trajectory Optimization strategy in our adaptation

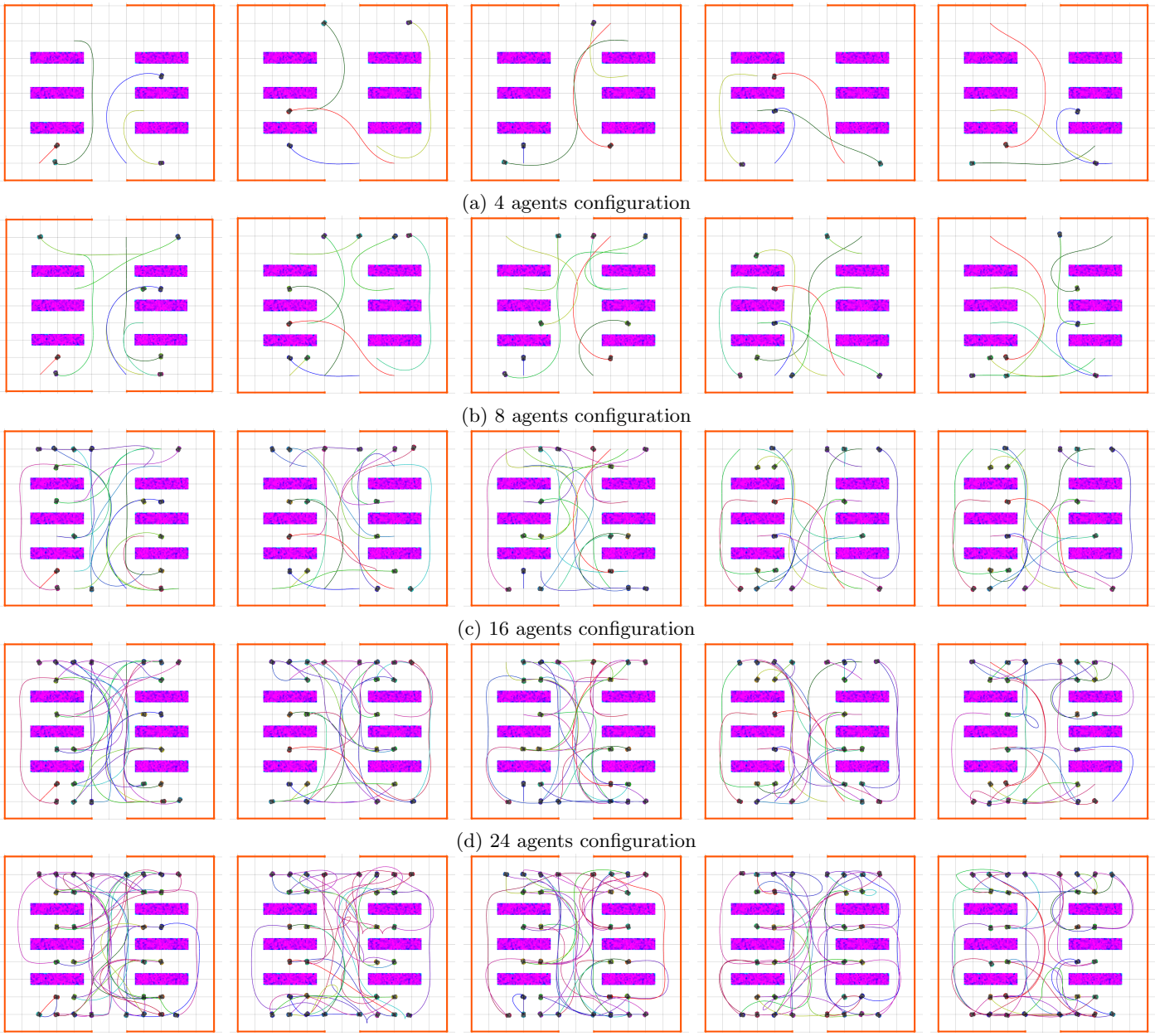
	Trajectory Optimization Strategy	
Number of agents	Prioritized Optimization	Coupled Optimization
4	1.2750	1.1460
8	2.5500	2.1720
16	5.5850	5.1240
24	10.5690	9.9810
32	16.8980	15.5260

TABLE V. Cost computed for both of the optimization strategies against five agent configurations.

obtained by the authors in the reference paper were validated with the help of a simulation setup as designed in [1]. The runtime of the proposed strategy is found to follow a linear relationship with the number of agents in the system implying that the strategy is well-suited for large-scale systems. The computed runtime for our experiments is seen to be lower than that of the authors in [1] and is due to the superior specifications of the system we used to perform our experiments. Further, the cost of computation of the solution for the proposed technique is discovered to be approximately 7.9% higher when compared to that of the coupled trajectory optimization strategy. The addition of the novel, priority-based grouping strategy to the original trajectory optimization algorithm is seen to have increased the success rate by 15% and 10% for the 24 agents and 32 agents configuration respectively. The margin of comparison in our experiments doesn't exactly match that of the reference paper due to the fact that we performed our experiments for 20 iterations in place 0f 40. It is also observed that the hyperparameters like the collision model radius of the agents played a significant role in the length of the trajectories planned for the agents to reach the goal. In conclusion, it can be stated that the proposed priority-based trajectory optimization strategy proposed in [1] is an efficient technique for solving real-time multi-agent path-finding problems, especially for scenarios like in warehouses where the positions of the static objects are known prior and there is no any time constraints.

VII Future Work

Due to time constraints and practical limitations the experiments of this review were limited to simulations



(a) 4 agents configuration

(b) 8 agents configuration

(c) 16 agents configuration

(d) 24 agents configuration

(e) 32 agents configuration

FIG. 16. Five samples from the 20 simulation experiments performed for each of the agent configurations (i.e) 4, 8, 16, 24, and 32. The start and goal position was randomly chosen for all 20 iterations of the experiments for each configuration.

however in the future, we intend to test the proposed strategies in [1] on real robot systems and analyze the results.

VIII Acknowledgement

We would like to thank Dr.Waseem Malik for giving us this opportunity. Also, we greatly appreciate the efforts put in by the Teaching Assistants of the course ENPM667 for reviewing our submission on time and giving us their valuable feedback.

References

[1] Juncheng Li, Maopeng Ran, and Lihua Xie. Efficient trajectory planning for multiple non-holonomic mobile robots via prioritized trajectory optimization. *IEEE Robotics and Automation Letters*, 6(2):405–412, 2020.

[2] Mark W Spong, Seth Hutchinson, and Mathukumalli Vidyasagar. *Robot modeling and control*. John Wiley & Sons, 2020.

[3] Roland Y. Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. Introduction to autonomous mobile robots, second edition. In *Intelligent robotics and autonomous*

agents, 2011.

[4] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Seventh Annual Symposium on Combinatorial Search*, 2014.

[5] Drew Bagnell. Statistical techniques in robotics (16-831, f10) lecture05, mapping, September 2014.

[6] Mihail Pivtoraiko and Alonzo Kelly. Kinodynamic motion planning with state lattice motion primitives. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2172–2179. IEEE, 2011.

[7] Jungwon Park, Junha Kim, Inkyu Jang, and H Jin Kim. Efficient multi-agent trajectory planning with feasibility guarantee using relative bernstein polynomial. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 434–440. IEEE, 2020.