2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 27 - October 1, 2021. Prague, Czech Republic

2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-1714-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/IROS51168.2021.9636088

Decentralized, Unlabeled Multi-Agent Navigation in Obstacle-Rich Environments using Graph Neural Networks

Xuebo Ji[1], He Li[1], Zherong Pan[2], Xifeng Gao[3], and Changhe Tu[1]

***Abstract*— We propose a decentralized, learning-based solution to the challenging problem of unlabeled multi-agent navigation among obstacles, where robots need to simultaneously tackle the problems of goal assignment, local collision avoidance, and navigation. Our method has each robot infer their desired action by communicating with each other as well as a set of position-fixed routers. The inference is carried out on a graph neural network (GNN) with both robot and router nodes. We train our GNN using imitation learning on a small group of robots, where we modify the centralized version of the concurrent goal assignment and planning algorithm (CAPT) as our expert. By sharing weights among all robots and routers, our model can scale to unseen environments with any number of possibly kinodynamic agents during test time. We have achieved a success rate of 91.2% and 85.6% for point and car-like robots, respectively. Source code will be publicly available upon the publication of the work.**

I. Introduction

Multiple ground and aerial robots can be deployed to perform a row of tasks in unknown environments, including surveillance, coverage, house cleaning, infection control, search & rescue, and warehouse automation. The emerging low-cost camera, lidar, sonar, and thermal sensors make these robots more accessible and affordable, but coordinating the robots can become a single point of failure, which is the crucial limitation of centralized methods [1, 2]. Decentralized algorithms [3, 4] are favorable in these settings, where both robot communications and motions are asynchronous. However, environments with rich obstacles, and the limited capability of sensors, make the design of a satisfactory decentralized method a great challenge to solve.

Navigation is a common (sub-)task in the above mentioned application scenarios. Trajectories should be generated for each robot to reach their goal positions through obstacle-rich environments in a collision-free manner. In this work, we consider the problem of unlabeled navigation where the robot-to-goal assignment is one-to-one but the matching is arbitrary. Unlabeled navigation meets the requirement of most surveillance, coverage, search and rescue tasks. It has been shown in [5, 6] that unlabeled navigation tasks are theoretically easier to solve than their labeled counterparts, which is intuitive due to the additional degrees of freedom to assign robots to their nearest goals and such assignments must be collision-free [7]. However, theoretical results for unlabeled navigation are derived for centralized algorithms, while few results are available for decentralized settings. This is presumably due to the intricate coordination of assigned goals among robots. The prior work Turpin *et al.* [7] introduces two versions of CAPT, a centralized method, C-CAPT, and a decentralized one, D-CAPT. However, D-CAPT works by assuming no obstacles in environments.

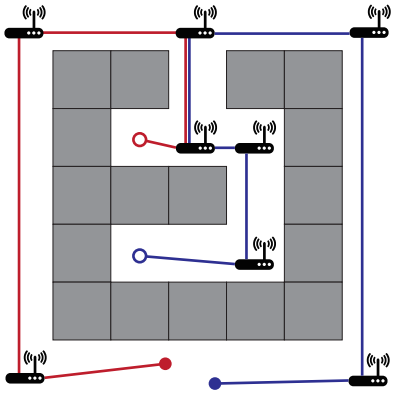


Fig. 1: Our method routing two robots (solid red / blue circles) along the solid line to target positions (hollow circles) in an obstacle-rich environments (gray). Each robot can communicate with nearby routers within its sensor range. Router-router communication can only happen along designated edges. A possible outcome could be the red and blue paths.

Our work is inspired by prior efforts in applying machine learning to solve labeled robot navigation problems [8, 9, 10, 11]. Methods in [8, 9] assume environments and robots are discretized on a grid and centralized conflict-based search algorithms [1, 2] are used to computed groundtruth data. Methods in [10, 11] can be applied to environments of irregular geometric shapes and they use groundtruth-free deep reinforcement learning. However, they only tackle local collision avoidance and do not handle obstacle-rich environments. Independently, computer vision researchers have recently applied GNNs to track or predict the trajectories of social agents [12, 13, 14]. By allowing agents to communicate for multiple rounds, GNNs exhibit consistently better results than their no-communication baselines. This strategy has been used in [8] to solve labeled navigation tasks.

Main Result: We present the first learning-based approach to solve the unlabeled and multi robot navigation problem in obstacle-rich environments, as illustrated in Figure 1. Our main idea is to use GNNs with two types of nodes: robots and routers. Each node is equipped with an inference module that processes hidden information. By communicating the hidden information with neighboring robots or router nodes, the goal assignment problem is approximately solved. The placement and communication with routers also allow us

[1]Xuebo Ji, He Li, and Changhe Tu are with Department of Computer Science, Shandong University. {jixuebo@mail.sdu.edu.cn, 202015075@mail.sdu.edu.cn, chtu@sdu.edu.cn} [2]Zherong Pan is with Department of Computer Science, University of Illinois at Urbana Champaign. {zherong@illinois.edu} [3]Xifeng Gao is with Department of Computer Science, Florida State University. {gao@cs.fsu.edu}

to handle obstacle-rich environments where local sensing information or peer-to-peer communications are insufficient for discovering hidden goal positions or traveling through detours. We use an adapted C-CAPT [7] as the expert to allow robots travel along router-graphs, while the inference modules are trained using imitation learning. By sharing weights among all routers and robots, our method can further scale to unseen environments, arbitrary-sized groups of robots, and different robot modalities during test time.

We have evaluated our method in maze-like environments with point or car-like robots. After training on a desktop machine for ~ 100hr with a group of $5-15$ robots and $7-18$ routers, we achieve a success rate of 91.2% for point robots and 85.6% for car-like robots.

II. Related Work

Our work is closely related to prior research on (de)centralized multi-agent navigation and learning-based multi-agent navigation.

(De)centralized multi-agent navigation finds collision-free agent paths to reach their goals, where the robot-to-goal assignments are arbitrary in unlabeled problems and fixed otherwise. Centralized navigation algorithms assume that the planner has the complete information of agent and obstacle positions, so that agents can be coordinated via recursive graph search. For labeled problems, classical results show that feasibility checks for some graph search algorithms are polynomial-time solvable [15], while finding optimal solutions are intractable in general [16]. Unlabeled problems endow a larger solution space, for which finding distance optimal solutions is tractable [17, 18] using centralized algorithms. It is note-worthy that all these algorithms require a discretization of environments into a graph. By comparison, decentralized navigation algorithms assume robots make (a)synchronous decisions based on information of their local neighborhood. Some works [19] allow robots to communicate with each other, while others [3, 4] solely rely on sensor inputs. Monte Carlo Tree Search has also been used for a multi-agent task allocation problem [20]. Turpin *et al.* [7] proposed a decentralized and a centralized multi-agent navigation algorithm respectively, where the decentralized one is to solve the unlabeled problem by allowing nearby robots to exchange their assigned goals. Decentralized algorithms have superior flexibility and convenience for system integration and can be applied to irregular-shaped environments. On the downside, however, robots are not coordinated and can get stuck in locking configurations, especially in obstacle-rich environments. To side-step this problem, we introduce router nodes to guide robots through obstacles.

Learning-based multi-agent navigation algorithms have drawn significant attention thanks to the recent advances in multi-agent reinforcement learning (MARL [21]) and GNNs. MARL focuses on scalable, simultaneous learning of many policies deployed on a large group of agents using consensus optimization [22, 23]. The agents' goals can be cooperative [24], competitive, or mixed [25, 26]. These algorithms have been applied to solve labeled multi-agent navigation problems [10, 11, 27, 28] where all robots share the same objective type of reaching their respective goal positions, in which case a same policy can be shared among all robots. This strategy significantly reduces the number of learnable parameters and side-steps the scalability challenge of MARL. It is note-worthy that all these methods learn local controllers that can stuck in obstacle-rich environments, so they are only evaluated in (nearly) obstacle-free settings. Robots must learn to communicate in order to avoid locking configurations, for which GNN has become a standardized parametrize model [8, 29]. GNN extends the notion of convolution to unstructured settings via spatial [30] or spectral [31] graph-convolution operators. In particular, the method in [8] is very similar to our approach. By training GNNs for robot coordination using imitation learning, Li *et al.* [8] showed that robots can navigate through obstacles in grid-discretized environments to reach labeled goals. Instead, our method use two types of GNN nodes for robots and routers to simultaneously solve goal-assignment and obstacle avoidance problems, and our method does not rely on a grid-based discretization.

TABLE I: Symbol Table.

Variable	Definition
N	#robots
M	#routers
H	horizon
H_0	communication interval
E, E_f	work/freespace
a, r, t_i	ith robot/router/target
x	position
c_k	config at kth step
f	discrete transition
$u_{i,k}, u^*_{i,k}$	analytic/learned control
$\mathbb{I}_{ij}$	goal assignment
d_l	communication range
d_s	sensing range
T_i	ith triangle
$\mathcal{L}$	loss function
$\mathcal{G}_r =< \mathcal{V}_r, \mathcal{E}_r >$	skeleton-graph
$\bar{\mathcal{G}}_r =< \bar{\mathcal{V}}_r, \bar{\mathcal{E}}_r >$	router-graph
$\mathcal{G}_{r,a,t} =< \mathcal{V}_{r,a,t}, \mathcal{E}_{r,a,t} >$	augmented router-graph
$e_k, \bar{e}_k$	Euclidean/graph error
$\bar{e}_{ijk}$	path on graph
$\mathcal{M}$	control modulator
$\mathcal{N}, \theta_{a,r,u,\mathcal{M}}$	neural-net/parameters
h_k	hidden state at kth step
Δx	relative position
$\mathbb{T}$	one-hot entity type
$\mathbb{I}^i_{reach}$	robot/target reach indicator
β, γ	mixing ratio/attenuation
$\mathbb{D}$	dataset
D, D_0	#datum (supervised/imitation)
$\mathbb{L}$	LiDAR sensor data

III. Unlabeled Robot Navigation Problem

The problem we address in this work is the unlabeled multi-agent navigation problem, with the assumption that the number of robots and targets are equal. Our goal is to

compute a path for each robot to reach a target region without any collisions in a decentralized manner. Specifically, we assume a 2D workspace E with freespace E_f specified by cluttered obstacles. Our goal is for a set of N robots $a_{1,\cdots,N}$ to reach another set of N target positions $t_{1,\cdots,N}$.

We denote the configuration of a_i at the kth time instance as $c_k(a_i)$ and center-of-mass position as $x(c_k(a_i))$. In order for routing the robots through obstacle-rich environments, we use another set of M stationary routers denoted as $r_{1,\cdots,M}$ with positions of r_j being $x(r_j)$. Note that the position and number of routers are not provided but computed using our algorithm. Following the setting of reciprocal velocity obstacles (RVO) based navigation method [3, 10, 11], which guarantees safe and oscillation-free motions of each agent for real-time multi-agent navigation, we do not assume a discrete environments so that $c_k(a_i)$ and $x(r_i)$ can take any continuous value within E_f. As compared with grid-based routing algorithms [1, 8], continuous motions allow robot to be easily deployed in real, cluttered environments and even for kinodynamic robots. The robot dynamics are defined as a discrete equation of motion $c_{k+1}(a_i) = f(c_k(a_i), u_{i,k})$, where $u_{i,k}$ is the control input and f is the discrete transition function. The goal of unlabeled multi-agent navigation is to find $u_{i,k}$ for $i = 1, \cdots, N$ and $k = 1, \cdots, H$, where H is the control horizon, such that the positioning error $e_k = \sum_{i=1}^{N} \sum_{j=1}^{N} \mathbb{I}_{ij} \|x(c_k(a_i)) - t_j\|^2$ is zero when $k = H + 1$ and robot motions are collision-free. Here $\mathbb{I}_{ij}$ is the indicator function for the ith robot to be assign to reach t_j. Table I summarizes all the symbols used in our paper.

Pivotal to long-distance navigation and target assignment is our communication module between agents, routers, and targets. We assume that robots are equipped with two capabilities: A long-range communication can be realized by WiFi and used for exchanging information between robots and routers for goal-assignment and waypoint routing; A short-range sensing is realized by LiDAR and used for local collision-avoidance. The distance limits for long-range communication and short-range sensing, are denoted as d_l and d_s, such that communication or sensing can happen if and only if $\|x_k(a_i) - x_k(a_j)\| \leq d_{l,s}$. We further assume that communications between routers can be established without distance limits.

Our learning-based navigation algorithm consists of three components. First, we propose a method to compute the number and position of routers with the help of the medial axis of E_f (Section IV). The router-graph is then constructed by connecting routers along the medial axis. Second, we propose a modified C-CAPT algorithm with graph-distance to solve unlabeled robot navigation problems in a centralized manner (Section V). Finally, a GNN is trained to imitate the modified C-CAPT, achieving decentralized navigation (Section VI).

IV. Router-Graph Construction

Constructing the Euclidean, router-graph $\mathcal{G}_r =< \mathcal{V}_r, \mathcal{E}_r >$ is an essential step to guide local-communication robots through obstacle-rich environments, where $\mathcal{V}_r$ is the vertices of $\mathcal{G}_r$ corresponding to the router positions $x(r_i)$ and $\mathcal{E}_r$ is the set of edges and we require $\mathcal{E}_r \subset E_f$. We consider a router-graph cost-efficient if the number of routers is kept to a minimal level, and a router-graph is robust if any point $x \in E_f$ is visible from at least one routers on the graph.

We assume that the 2D environment boundaries are discretized using piecewise linear elements. Such discretization allows robust algorithms to compute a medial axis transform [32] as well as a Delaunay triangulation [33]. The medial axis transform returns a super-graph denoted as $\bar{\mathcal{G}}_r =< \bar{\mathcal{V}}_r, \bar{\mathcal{E}}_r >$ that represents the "skeleton" of E_f. We then compute a constrained Delaunay triangulation that segment E_f into a set of R triangular regions $E_f = \bigcup_{i=1}^{R} T_i$, such that each edge $\bar{\mathcal{E}}_r$ is an edge of some T_i, as illustrated in Figure 2. We claim that T_i is visible from $x \in \mathcal{V}_r$ if the convex hull of T_i and x belongs to E_f. We then greedily simplify $\bar{\mathcal{G}}_r$. We remove $x \in \bar{\mathcal{G}}_r$ if all the visible triangles remain visible after the removal. If no further simplification is possible, we set $\mathcal{V}_r \leftarrow \bar{\mathcal{V}}_r$. Next, we connect all pairs of $x, x' \in \bar{\mathcal{G}}_r$ using shortest paths in $\bar{\mathcal{G}}_r$ to form $\mathcal{G}_r$. This procedure is summarized in Algorithm 1. We only allow router-to-router communication along $\mathcal{E}_r$.

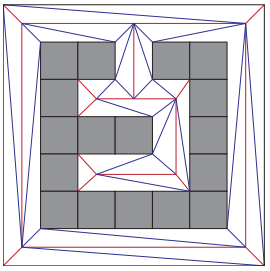


Fig. 2: We illustrate the edge set $\bar{\mathcal{V}}_r$ for skeleton graph (red), and edges are inserted by constrained Delaunay triangulation, segmenting the environment into triangles (blue). After simplification, we derive the router-graph in Figure 1.

Algorithm 1: Construct Router Graph

Input: E_f
Output: $\mathcal{G}_r$
compute medial axis transform $\bar{\mathcal{G}}_r$
compute Delaunay transformation $E_f = \bigcup_{i=1}^{R} T_i$
$\mathcal{V}_r \leftarrow \bar{\mathcal{V}}_r$, simplify$\leftarrow$true
while simplify **do**
    simplify$\leftarrow$false
    for $x \in \bar{\mathcal{V}}_r$ and simplify=false **do**
        simplify$\leftarrow$true
        for $i = 1, \cdots, R$ **do**
            if T_i invisible to $\bar{\mathcal{V}}_r / \{x\}$ **then**
                simplify$\leftarrow$false
        if simplify **then**
            $\mathcal{V}_r \leftarrow \mathcal{V}_r / \{x\}$
$\mathcal{E} \leftarrow \varnothing$
for $x, x' \in \mathcal{V}_r$ **do**
    $e \leftarrow$shortest-path-on-$\bar{\mathcal{E}}_r(x, x')$
    if $e \cap \bar{\mathcal{V}}_r / \{x, x'\} = \varnothing$ **then**
        $\mathcal{E} \leftarrow \mathcal{E} \cap \{e\}$

V. CAPT with Graph-Distance

C-CAPT is an efficient, centralized algorithm to solve unlabeled navigation problems in environments without obstacles. At any time step k, this algorithm first assign goals

to robots by solving:

$$\mathbb{I}_{ij} = \underset{\mathbb{I}_{ij}\in\{0,1\}}{\text{argmin}}\ e_k \quad \text{s.t.} \sum_{i=1}^{N}\mathbb{I}_{ij} = \sum_{j=1}^{N}\mathbb{I}_{ij} = 1,$$

for $\mathbb{I}_{ij}$, and then use goal-directed control algorithms such as [3, 4] to move a_i to $\sum_{j=1}^{N} t_j\mathbb{I}_{ij}$. If no obstacle exists, Turpin *et al.* [7] showed that the straight lines between $x(c_k(a_i))$ and $\sum_{j=1}^{N} t_j\mathbb{I}_{ij}$ are intersection-free, which does not hold in obstacle-rich cases. To side-step this problem, we propose to use graph-distance instead of Euclidean distance. At time step k, we first construct an augmented graph $\mathcal{G}_{r,a,t}$ by inserting edges connecting all visible pairs of $x(c_k(a_i))$ and $x \in \mathcal{G}_r$, as well as all visible pairs of t_j and $x \in \mathcal{G}_r$, as illustrated in Figure 1. Since every point in E_f is visible to some $x \in \mathcal{V}_r$, all robots are connected to all target points, and we can compute the shortest path between a_i and t_j restricted to $\mathcal{G}_{r,a,t}$, the length of this path is denoted as $\bar{e}_{ijk}$. We then define the error function under graph-distance as $\bar{e}_k = \sum_{i=1}^{N}\sum_{j=1}^{N}\mathbb{I}_{ij}\bar{e}_{ijk}$ and solve:

$$\mathbb{I}_{ij} = \underset{\mathbb{I}_{ij}\in\{0,1\}}{\text{argmin}}\ \bar{e}_k \quad \text{s.t.} \sum_{i=1}^{N}\mathbb{I}_{ij} = \sum_{j=1}^{N}\mathbb{I}_{ij} = 1, \tag{1}$$

for $\mathbb{I}_{ij}$. We then use goal-directed control algorithms to generate $u_{i,k}$ that moves a_i to the next discrete position on the shortest path, yielding $c_{k+1}(a_i)$. Note that these shortest paths can still intersect with each other and we need to modulating $u_{i,k}$ into $\mathcal{M}(u_{i,k})$ to avoid local collisions, which can be done using reciprocal penalty functions [3, 4, 34]. And finally, robots' configurations are updated as $c_{k+1}(a_i) = f(c_k(a_i), \mathcal{M}(u_{i,k}))$. We summarize the modified C-CAPT in Algorithm 2. As compared with grid-based algorithms [9, 1] and discrete graph search [2], our modified C-CAPT has no completeness guarantee but it works reasonably well in obstacle-rich environments with general geometric shapes.

Algorithm 2: CAPT with Graph-Distance (Graph-CAPT)

Input: $E_f, \mathcal{G}_r, c_k(a_{1,\cdots,N}), t_{1,\cdots,N}$
Output: $u_{1,k}, \cdots, u_{N,k}$
$\mathcal{G}_{r,a,t} \leftarrow \mathcal{G}_r$
for $i = 1, \cdots, N$ **do**
  for $v \in \mathcal{V}_r$ **do**
    if $x(c_k(a_i))$ visible from v **then**
      $\mathcal{E}_{r,a,t} \leftarrow \mathcal{E}_{r,a,t} \cup < x(c_k(a_i)), v >$
    if t_i visible from v **then**
      $\mathcal{E}_{r,a,t} \leftarrow \mathcal{E}_{r,a,t} \cup < t_i, v >$
for $i = 1, \cdots, N$ **do**
  for $j = 1, \cdots, N$ **do**
    $e_{ijk} \leftarrow$shortest-path-on-$\mathcal{E}_{r,a,t}(x(c_k(a_i)), t_j)$
solve Equation 1 for $\mathbb{I}_{ij}$
$u_{i,k} \leftarrow$ next position on path $\sum_{i,j=1}^{N}\mathbb{I}_{ij}e_{ijk}$

VI. Graph Neural Network (GNN)

Our learning method consists of two stages. In the first stage, the GNN learns to update hidden states $h(a_i), h(r_i)$ for each robot and router, respectively. We train the network to approximate the solution of Graph-CAPT, i.e. $u_{i,k}$. In the second stage, agents are routed along $u_{i,k}$ to reach distinct goal positions. The modulation of $u_{i,k}$ to avoid local collisions can be performed using analytical algorithms. Learning-based second stage is also doable, for example, using reinforcement learning [10, 11] or supervised learning (Section VI-B), leading to an end-to-end learning-based solution.

Algorithm 3: Training GNN

Input: $E_f^{1,\cdots,D+ID'}$
Output: $\theta_a, \theta_r, \theta_u$
$\mathbb{D} \leftarrow \varnothing$, $\beta \leftarrow 1$
for $r = 0, \cdots, I$ **do**
  for $i = 1, \cdots, D\mathbb{I}(r=0) + D'(1-\mathbb{I}(r=0))$ **do**
    construct $\mathcal{G}_r$ for randomly sampled E_r
    randomly sample $c_1(a_i)$
    for $k = 2, \cdots, H+1$ **do**
      compute $u_{i,k}, c_k(a_i)$ by mixed policy
    for $k = 1, \cdots, \lfloor H/H_0 \rfloor$ **do**
      $\mathbb{D} \leftarrow \mathbb{D} \cup \{c_k(a_i), u_{i,kH_0+1}\}$
  optimize $\sum_{i=1}^{D+rD'} \mathcal{L}(\theta_a, \theta_r, \theta_u, E_f^i)$
  $\beta \leftarrow \beta\gamma$

Our GNN is parameterized as two fully connected recurrent blocks $\mathcal{N}_a(h, \Delta x, \mathbb{T}, \theta_a)$, $\mathcal{N}_r(h, \Delta x, \mathbb{T}, \theta_r) : \mathbb{R}^{|h|+5} \mapsto \mathbb{R}^{|h|}$, where $|h|$ is the hidden state size. We assume $\mathcal{N}_a$ is deployed on robots and $\mathcal{N}_r$ is deployed on routers. We further assume that each target position t_i is equipped with a processor with same communication capabilities as routers, deployed with $\mathcal{N}_r$. The weights of these two blocks, $\theta_{a,r}$, are shared among all robots, routers, and targets to enable scaling to arbitrary robot swarm sizes and environment complexities during test time. The two additional inputs to $\mathcal{N}$ are relative position between communicating entities Δx and one-hot node type representation $\mathbb{T}$. $\mathbb{T}$ can take five different values. We let $\mathbb{T} = e_1$ if the entity is a robot and has reach some target, and $\mathbb{T} = -e_1$ if the entity is a robot but has not reach any target. $\mathbb{T} = e_2$ if the entity is a router. $\mathbb{T} = \pm e_3$ if the entity is a target ($\mathbb{T} = +e_3$ if the target has been reached by a robot and $-e_3$ otherwise). The communication has each robot or router collect information from nearby entities. As illustrated in Figure 4, if a robot, router, or target is located at x, then the collected information $h(x)$ is defined as:

$$\begin{aligned} h_{k+1}(x) \triangleq & \sum_{\substack{i:\|t_i-x\|<d_l \\ (t_i,x)\in\mathcal{E}_{r,a,t}}} \mathcal{N}_r(h_k(t_i), t_i - x, e_3(2\mathbb{I}^i_{reach}-1)) + \\ & \sum_{\substack{i:\|x(r_i)-x\|<d_l \\ (x(r_i),x)\in\mathcal{E}_{r,a,t}}} \mathcal{N}_r(h_k(r_i), x(r_i) - x, e_2) + \\ & \sum_{\substack{i:\|x(c_k(a_i))-x\|<d_l \\ (x(c_k(a_i)),x)\in\mathcal{E}_{r,a,t}}} \mathcal{N}_a(h_k(a_i), x(c_k(a_i)) - x, e_1(2\mathbb{I}^i_{reach}-1)), \end{aligned} \tag{2}$$

where $\mathbb{I}_{reach}$ is an indicator of whether a_i has reached some target. This is defined as $\mathbb{I}^i_{reach} = \mathbb{I}(\min_{j=1,\cdots,N}\|x(c_k(a_i)) - t_j\| < d_s/8)$ where we assume the robot reaching a target when the distance between a robot-target pair is smaller than one-eighth the sensing range d_s. Similarly, we use the following indicator of whether a target t_j has been occupied: $\mathbb{I}^j_{reach} =$

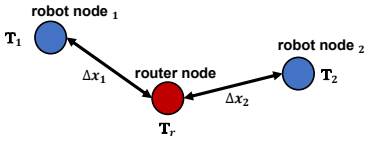
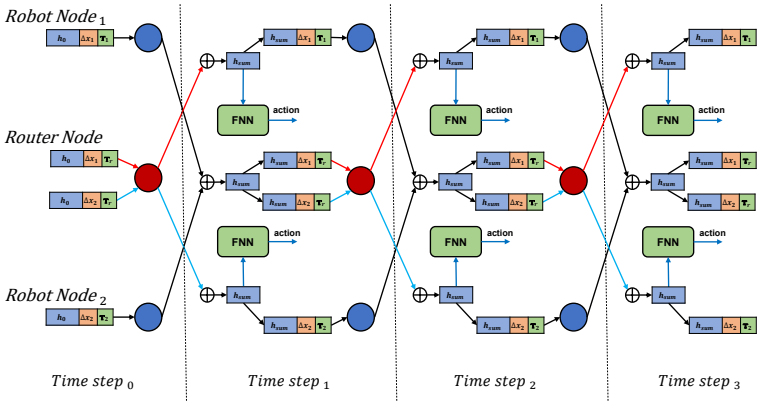


Fig. 3: A simple case with two robot nodes and one router node (top), and its corresponding GNN structure (right). During each time step t, each node will calculate h_{sum}, Δx and $\mathbb{T}$. h_{sum} is obtained by summarizing hidden states sent from its neighbors in time step $t-1$. Δx is the relative position between the node itself and the neighbor it will send information to. $\mathbb{T}$ is a one-hot vector which represents the node's state and type. Each node encodes h_{sum}, Δx and $\mathbb{T}$ into the hidden state for communication of time step $t+1$. If the node represents a robot, it will input h_{sum} into a fully connected neural network to obtain its action for the current time step.



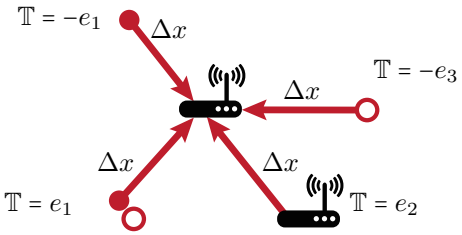


Fig. 4: We illustrate the communication by plotting the information collected by a router from two robots (solid red), one other router, and one target (hollow red). One of the robot is of type 1 as it is close enough to a target.

$\mathbb{I}(\min_{i=1,\cdots,N}\|x(c_k(a_i)) - t_j\| < d_s/8)$. Empirically, we found that using $\mathbb{I}_{reach}$ to explicitly notify that a target has been occupied could avoid competing over goals when agents are close. Figure 3 gives an illustration of our GNN structure.

A. Training GNN

Our training procedure follows the framework of data aggregation [35]. We collect a dataset of D environments. For each environment, we randomly sample robots' initial collision-free configurations $c_1(a_i)$. We then run Graph-CAPT and velocity modulation algorithm (i.e. [3] for point robots and [4] for kinodynamic robots) for H timesteps to generate $c_k(a_i)$ and $u_{i,k}$ with $k = 1,\cdots,H+1$ and $i = 1,\cdots,N$. The communication is mainly used for goal-assignments and waypoint routing, where the solutions to these problems do not change frequently. Therefore, we down-sample the data $c_k(a_i), u_{i,k}$ temporally with a sample interval of H_0 and assume that communication can happen every H_0 iterations. Note that the down-sampled data is control signal generated by Graph-CAPT without modulation. This is because the modulation is used to handle local collision avoidance, which can undergo frequent changes. On the down-sampled data, we can emulate $\lfloor H/H_0 \rfloor$ rounds of communications using recurrent relationship Equation 2 and we use a single-term loss function:

$$\mathcal{L}(\theta_a,\theta_r,\theta_u,E_f^i) \triangleq \sum_{i=1}^{N}\sum_{k=1}^{\lfloor H/H_0\rfloor} \|u^*_{i,kH_0+1} - u_{i,kH_0+1}\|^2 \quad (3)$$
$$u^*_{i,kH_0+1} \triangleq \mathcal{N}_u(h_{kH_0+1}(x(a_i)),\theta_u),$$

where we have introduce a last neural network $\mathcal{N}_u$ that is deployed only on robots, converting hidden states to control signals. We include E_f^i as a last parameter of $\mathcal{L}$ to indicate datum. After pre-training in a supervised manner using D randomized environments, we perform imitation learning via I rounds of data-aggregation (DAgger) [35]. During the rth round, we sample an additional set of D' environments, on which we execute mixed control policy $\mathcal{M}((1-\beta)u^*_{i,k} + \beta u_{i,k})$ to derive the aggregated training data, where β is the mixing parameter that is progressively attenuated at a rate of γ. The complete learning pipeline is summarized in Algorithm 3.

B. Local Collision Avoidance

In addition to globally routing robots through environments, local collisions need to be handled. These local collisions can be handled using either learning-based methods or analytic methods ($\mathcal{M}$) such as [3] for point robots and [4] for kinodynamic robots. With a learning-based local collision handler, our method becomes an end-to-end learning-based solution. Specifically, we can replace $\mathcal{M}$ with another neural network $\mathcal{N}_{\mathcal{M}}(u_{i,k},\mathbb{L},\theta_{\mathcal{M}}) \mapsto u^*_{i,k}$, where $\mathbb{L}$ is the LiDAR sensor data with sensing range being d_s. Fan *et al.* [10] proposed to train $\theta_{\mathcal{M}}$ using on-policy reinforcement learning. Instead, we find that supervised learning exhibits more stable performance. We sample a large number of randomized robot configurations and target velocities, and then use L_2-loss for training $\mathcal{N}_{\mathcal{M}}$ to mimic the groundtruth $\mathcal{M}$.

VII. Experiments

In this Section, we describe the detailed experimental setup, metrics, and quantitative results to demonstrate the effectiveness and generality of our approach. All our experiments are performed on a machine with 48-core, 2.50Ghz Xeon E5-2678 CPU, and an Nvidia TITAN Xp GPU with 12GB of memory and we implement the algorithm using Python and PyTorch. By default, we choose $\gamma = 0.94$, $d_l = 700$, $d_s = 80$, $D = 500$, $D' = 10$, $I = 100$, $H = 500$, $H_0 = 20$, unless otherwise specified. We train the neural networks for 5 epochs in each DAgger [35] iteration, and all neural networks are optimized with a learning rate of 0.001.

We create a dataset of 1000 grid-like scenes of size 700×700, where 500 are used for training and 500 for testing. For

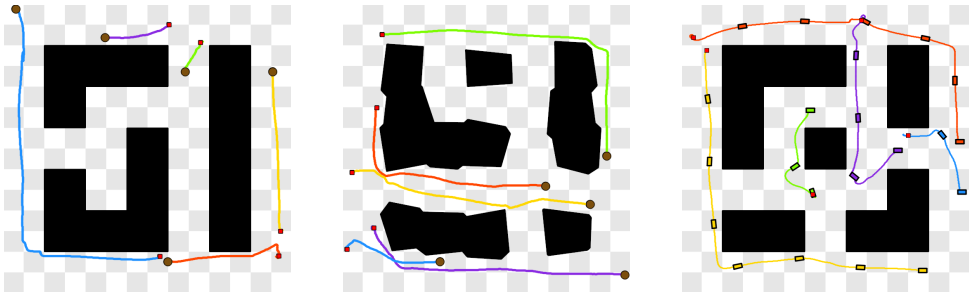


Fig. 5: Three different test cases, from left to right: disk-shaped robots with axis-aligned obstacles (black), disk-shaped robots (solid circles) with irregular obstacles, and car-like robots (rectangles) with regular obstacles. The small red squares are target locations.

each scene, we randomly generate axis-aligned obstacles for training, but we also test irregular obstacle shapes during testing. The robots we tested could have two different types: holonomic disk-shaped robots and non-holonomic car-like robots. We illustrate three exemplary settings in Figure 5.

Three metrics are employed to quantitatively evaluate our method. We define **Success-Rate (SR)** as the number of times where all robots successfully reach some target (according to the definition of $\mathbb{I}^i_{reach}$) within the control horizon (H = 500 during training and H = 600 during test time), averaged over all test cases. We define **Reach-Rate (RR)** as the fraction of robots that have reached some target, averaged over all collision-free test cases. Finally, we define **Trajectory-Optimality (TO)** as the average trajectory length generated by the neural network divided by the average trajectory length generated by the expert Algorithm 2, averaged over all successful test cases.

#Robot	3	5	7	9	11	13	15
SR end-to-end	0.856	0.815	0.764	0.678	0.676	0.801	0.721
RR end-to-end	0.970	0.978	0.977	0.973	0.982	0.992	0.993
SR partial-analytic	0.908	0.912	0.873	0.759	0.840	0.908	0.922
RR partial-analytic	0.965	0.979	0.978	0.965	0.983	0.990	0.993

TABLE II: Performance comparisons of the two modes of our approach tested using different robot swarm sizes, with axis-aligned obstacles.

#Robot	3	5	7	9	11	13	15
SR	0.912	0.912	0.905	0.883	0.864	0.815	0.773
RR	0.965	0.979	0.982	0.983	0.983	0.978	0.969
TO	1.149	1.250	1.334	1.441	1.543	1.690	1.843

TABLE III: We use 5 robots during training in regular environments. We then use **regular, seen** environments for testing. We profile the Success-Rate, Reach-Rate and Trajectory-Optimality when generalizing to different robot swarm sizes.

#Robot	3	5	7	9	11	13	15
SR	0.894	0.888	0.889	0.892	0.871	0.803	0.753
RR	0.958	0.974	0.980	0.983	0.982	0.974	0.963
TO	1.183	1.298	1.425	1.516	1.641	1.775	1.973

TABLE IV: Same as Table III but using **regular, unseen** environments for testing.

#Robot	3	5	7	9	11	13	15
SR	0.838	0.830	0.832	0.831	0.791	0.752	0.642
RR	0.938	0.956	0.967	0.974	0.970	0.964	0.948
TO	1.190	1.314	1.443	1.559	1.712	1.836	2.118

TABLE V: Same as Table III but using **irregular, unseen** environments for testing.

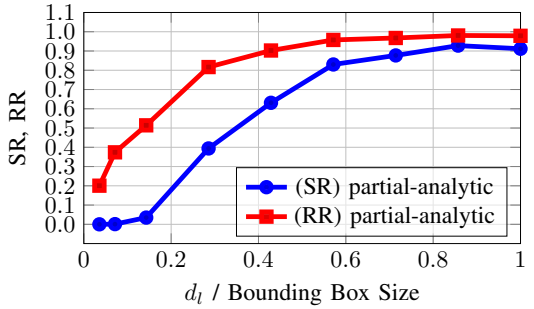


Fig. 6: Influence of sensing ranges on the performances.

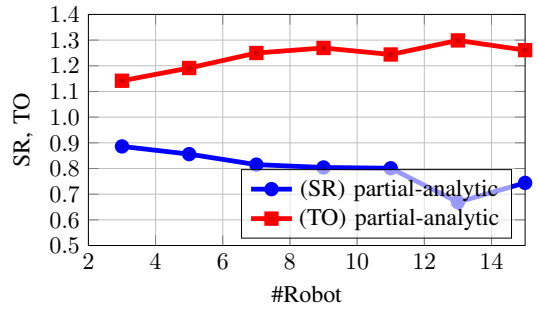


Fig. 7: Performances with car-like robots.

A. *End-to-End Versus Partial-Analytic Learning*

As described in Section VI, the neural-network can be trained in partial-analytic mode or end-to-end mode. In partial-analytic mode, the neural-network is only responsible for routing, while local collision avoidance is performed using [3] or [4]. In end-to-end mode, a learning-based local collision handler implemented by supervised learning is concatenated to the router network. In partial-analytic mode, the ground-truth u_{i,kH_0+1} is the output of Algorithm 2. In end-to-end mode, the ground-truth is generated by [3] or [4] with the output of Algorithm 2 used as target velocities. The velocity modulator $\mathcal{N}_{\mathcal{M}}$ is trained during the first phase using a dataset of 1.5M randomized robot poses and ground-truth modulated velocities. The graph network $\mathcal{N}_a, \mathcal{N}_r$ is then trained during the second phase. We compare the Success-Rate and Reach-Rate of two modes in Table II over the 1000 test cases, where we use same number of robots and axis-aligned obstacles for training and testing. As expected, the analytical collision avoidance component helps to improve the performance of our approach. But end-to-end mode only suffers from a loss of 11.6% in terms of Success-Rate and 1% in terms Reach-Rate.

B. Effect of Sensing Range

The sensing range of WiFi, i.e. d_l, plays an important role in our approach. We investigated the effects of d_l on the performance of our approach using the same experimental setup as Section VII-A, where the number of robots being 5. By increasing the value of d_l, we compare the changes of the performance of our approach in Figure 6. While increasing d_l does positively affect SR and RR, which demonstrates the necessity of communications during routing, our approach is robust to sensing range changes when d_l is larger than half the bounding box size of the environment, where we observe both the Success-Rate and Reach-Rate being larger than 80%.

C. Generalization to Unseen Scenarios

Being decentralized, our GNN can inherently generalize to different robot swarm sizes and irregular obstacles. Table III, Table IV, and Table V summarize the performances of our method in different scenarios where the number of robots during testing is larger than that during training. Our method performs consistently well in terms of Reach-Rate. The Success-Rate in both seen and unseen, regular environments are almost identical. But the Success-Rate in irregular environments can drop by at most 11%. We expect that such drop can be mitigated by adding irregular environments to the training data. However, the Trajectory-Optimality can deteriorate with more robots. This can be explained by the need for more communications before move towards the targets. The worst performance is around TO=2.1 in unseen, irregular environments, i.e. a travel distance twice as long as the expert solution.

D. Performance Using Car-Like Robots

Our algorithm works for robots with different kinematics and we experimented with both point-robot and car-like robot, where we use the same governing equation as [36] for the car-like robot. Figure 7 shows the performance when training and testing our GNN with the same number of car-like robots in regular environments. Note that, the higher value of SR the better, while an as low as possible TO value is preferred. Interestingly, there is only marginal performance differences between disk and car robots, which demonstrates that our pipeline is universally effective.

E. Real-world Experiments

To demonstrate the practical usage of our method, we built a real-world multi-robot system to verify the effectiveness of the trained network. As shown in Figure 8, we used 5 NanoRobot-ROS model with combined IMU and UWB position sensing. To simplify the internet communication, we simulate the decentralization process on a master computer. The master computer sends movement commands to each individual robots, receives positions from the positioning systems, and recomputes new commands for robots on-the-fly. See the attached video for more details.

VIII. Conclusion & Discussion

We present a learning-based, unlabeled robot navigation method in obstacle-rich environments. Our major innovation is a GNN with two types of nodes: robots and routers. By sharing neural network weights among robots and routers, our GNN inherits the desirable property of scaling to arbitrary robot swarm sizes and number of routers, as prior learning-based navigation algorithms [10, 8]. Our algorithm resembles [10] in the use of continuous configuration spaces without grid-based discretization. As a result, arbitrary robot dynamic models can be used. We have evaluated our algorithms on various settings, such as changing robot swarm sizes, environment sizes, robot dynamic models, and communication/sensing ranges.

A major limitation of learning-based solution is the lack of completeness guarantee. Indeed, we have only used plain, fully-connected neural network architectures as GNN building blocks and we expect GNN performance to be further improved by using memory units or attention modules over targets. We have also experimented with multi-agent reinforcement learning at an early stage of this research, but we found that the performance of learned GNN is highly sensitive to the parameters for action exploration, which requires case-by-case tuning. An additional limitation lies in our analytical method to compute the router-graph. Some applications require the robot swarm to adapt to an arbitrary, pre-defined network of routers. The number of routers computed using our method can be far from optimal and more aggressive router placement optimizers might be considered [37].

References

[1] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[2] R. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *IJCAI*, 2011, pp. 294–300.

[3] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE International Conference on Robotics and Automation*, IEEE, 2008, pp. 1928–1935.

[4] J. Alonso-Mora, A. Breitenmoser, P. Beardsley, and R. Siegwart, "Reciprocal collision avoidance for multiple car-like robots," in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012, pp. 360–366.

[5] M. Katsev, J. Yu, and S. M. LaValle, "Efficient formation path planning on large graphs," in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 3606–3611.

[6] K. Solovey, J. Yu, O. Zamir, and D. Halperin, "Motion planning for unlabeled discs with optimality guarantees," in *Robotics: Sciences and Systems*, 2015.

[7] M. Turpin, N. Michael, and V. Kumar, "Capt: Concurrent assignment and planning of trajectories for multiple robots," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 98–112, 2014.

[8] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020.

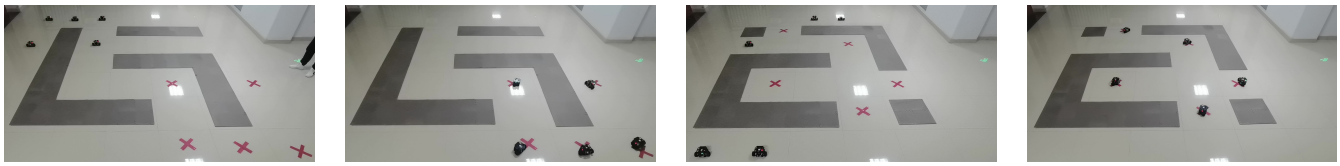


Fig. 8: Five robots start from random positions, move through the guidance of our method, and successfully reach the targets (red crosses) without any collisions during the travelling. The two pictures on the left show the start and goal robot positions in one experiment scenario and the two pictures on the right show another, where the gray carpets are obstacles. White squares are reflection of lights that should be ignored when observing the images.

[9] S. D. Han and J. Yu, "Ddm: Fast near-optimal multi-robot path planning using diversified-path and optimal sub-problem solution database heuristics," *IEEE Robotics and Automation Letters*, 2020.

[10] T. Fan, P. Long, W. Liu, and J. Pan, "Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 856–892, 2020.

[11] Q. Tan, T. Fan, J. Pan, and D. Manocha, "Deepmnavigate: Deep reinforced multi-robot navigation unifying local & global collision," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020.

[12] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 961–971.

[13] Y. Ma, X. Zhu, S. Zhang, R. Yang, W. Wang, and D. Manocha, "Trafficpredict: Trajectory prediction for heterogeneous traffic-agents," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 6120–6127.

[14] R. Chandra, U. Bhattacharya, A. Bera, and D. Manocha, "Traphic: Trajectory prediction in dense and heterogeneous traffic using weighted interactions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8483–8492.

[15] V. Auletta, A. Monti, M. Parente, and P. Persiano, "A linear-time algorithm for the feasibility of pebble motion on trees," *Algorithmica*, vol. 23, no. 3, pp. 223–245, 1999.

[16] J. Yu and S. M. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *The Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013, pp. 1444–1449.

[17] ——, "Multi-agent path planning and network flow," in *Algorithmic Foundations of Robotics X, Springer Tracts in Advanced Robotics (STAR)*, vol. 86, Springer Berlin/Heidelberg, 2013, pp. 157–173.

[18] ——, "Distance optimal formation control on graphs with a tight convergence time guarantee," in *The 51st IEEE Conference on Decision and Control*, 2012, pp. 4023–4028.

[19] V. R. Desaraju and J. P. How, "Decentralized path planning for multi-agent teams with complex constraints," *Autonomous Robots*, vol. 32, no. 4, pp. 385–403, 2012.

[20] D. Claes, F. A. Oliehoek, H. Baier, and K. Tuyls, "Decentralised online planning for multi-robot warehouse commissioning," May 2017, pp. 492–500.

[21] R. E. Wang, M. Everett, and J. P. How, "R-maddpg for partially observable environments and limited communication," 2019.

[22] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully decentralized multi-agent reinforcement learning with networked agents," in *International Conference on Machine Learning*, 2018, pp. 5872–5881.

[23] H.-T. Wai, Z. Yang, Z. Wang, and M. Hong, "Multi-agent reinforcement learning via double averaging primal-dual optimization," in *Advances in Neural Information Processing Systems*, 2018, pp. 9649–9660.

[24] R. E. Wang, C. Kew, D. Lee, E. Lee, B. A. Ichter, T. Zhang, J. Tan, and A. Faust, "Model-based reinforcement learning for decentralized multiagent rendezvous," 2020.

[25] M. Tan, "Multi-agent reinforcement learning: Independent versus cooperative agents," in *ICML*, 1993.

[26] R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, Curran Associates, Inc., 2017.

[27] A. Khan, E. Tolstaya, A. Ribeiro, and V. Kumar, "Graph policy gradients for large scale robot control," in *Conference on Robot Learning*, PMLR, 2020, pp. 823–834.

[28] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro, "Learning decentralized controllers for robot swarms with graph neural networks," in *Conference on Robot Learning*, 2020, pp. 671–682.

[29] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in neural information processing systems*, 2016, pp. 2137–2145.

[30] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in neural information processing systems*, 2015, pp. 2224–2232.

[31] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

[32] P. Felkel and S. Obdrzalek, "Straight skeleton implementation," in *Proceedings of spring conference on computer graphics*, Citeseer, 1998.

[33] O. Devillers, "Improved incremental randomized delaunay triangulation," in *Proceedings of the fourteenth annual symposium on Computational geometry*, 1998, pp. 106–115.

[34] I. Karamouzas, N. Sohre, R. Narain, and S. J. Guy, "Implicit crowds: Optimization integrator for robust crowd simulation," *ACM Transactions on Graphics*, vol. 36, no. 4, Jul. 2017.

[35] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.

[36] A. De Luca, G. Oriolo, and C. Samson, "Feedback control of a nonholonomic car-like robot," in *Robot motion planning and control*, Springer, 1998, pp. 171–253.

[37] F. Xhafa, C. Sanchez, L. Barolli, and R. Miho, "An annealing approach to router nodes placement problem in wireless mesh networks," in *2010 International Conference on Complex, Intelligent and Software Intensive Systems*, IEEE, 2010, pp. 245–252.