An End-to-End Deep Reinforcement Learning Based Modular Task Allocation Framework for Autonomous Mobile Systems

Song Ma, *Student Member, IEEE*, Jingqing Ruan, Yali Du, Richard Bucknall, *Member, IEEE*, and Yuanchang Liu, *Member, IEEE*

***Abstract*— Intelligent decision-making systems that can solve task allocation problems are critical for multi-robot systems to conduct industrial applications in a collaborative and automated way, such as warehouse inspection using mobile robots, hydrographic surveying using unmanned surface vehicles, etc. This paper, therefore, aims to address the task allocation problem for multi-agent autonomous mobile systems to autonomously and intelligently allocate multiple tasks to a fleet of robots. Such a problem is normally regarded as an independent decision-making process decoupled from the following task planning for the member robots. To avoid the sub-optimal allocation caused by the decoupling, an end-to-end task allocation framework is proposed to tackle this combinatorial optimisation problem while taking the succeeding task planning into account during the optimisation process. The problem is formulated as a special variant of the multi-depot multiple travelling salesmen problem (mTSP). The proposed end-to-end task allocation framework employs deep reinforcement learning methods to replace the handcrafted heuristics used in previous works. The proposed framework features a modular design of the reinforcement learning agent which can be customised for various applications. Moreover, a real-robot implementation setup based on the Robot Operating System 2 is presented to fulfil the simulation-to-reality gap. A warehouse inspection mission is executed to validate the training outcome of the proposed framework. The framework has been cross-validated via both simulated and real-robot tests with various parameter settings, where adaptability and performance are well demonstrated.**

***Note to Practitioners*—This paper is motivated by the problem of dispatching a fleet of autonomous mobile robots to tackle a mission that can be resolved into multiple waypoint-following tasks. An end-to-end modular framework is proposed, making task allocation decisions based on the given waypoint information. By using the reinforcement learning technique, the deep neural network could learn sophisticated policies for allocating tasks. The policies are trained in a specific pattern which ensures their joint optimisation for a solver that outputs the near optimal task execution sequences in an efficient way. This leads to a multiple travelling salesmen problem (mTSP) solution. Pre-trained policies are tested in several industrial scenarios reflecting the applications of search and rescue, maritime surveying, and warehouse automation, among others. A hardware implementation configuration based on the Robot Operating System 2 is also presented to support the practical deployment the framework.**

***Index Terms*— Deep reinforcement learning, task allocation, multi-agent planning, field robotics.**

Manuscript received 28 September 2023; revised 30 January 2024; accepted 12 February 2024. This article was recommended for publication by Editor P. Rocco upon evaluation of the reviewers' comments. This work was supported in part by the Dean's Prize of University College London Faculty of Engineering Sciences, in part by the China Scholarship Council, in part by Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/Y000862/1, and in part by the Royal Society under Grant RGS\R2\212343 and Grant IEC\NSFC\191633. *(Corresponding author: Yuanchang Liu.)*

Song Ma, Richard Bucknall, and Yuanchang Liu are with the Department of Mechanical Engineering, University College London, WC1E 7JE London, U.K. (e-mail: yuanchang.liu@ucl.ac.uk).

Jingqing Ruan is with the Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China.

Yali Du is with the Department of Informatics, King's College London, WC2R 2LS London, U.K.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TASE.2024.3367237.

Digital Object Identifier 10.1109/TASE.2024.3367237

I. Introduction

SINCE the development of the first generation of autonomous mobile robots (AMR) [1], [2], which dates back to the early 1980s, the AMRs have impressed the community with their flexibility to a wide range of tasks and the huge potential in its capability supported by the rapid progress of the computing hardware. AMRs have two major hardware components, the mobility system and the on-board sensors. The mobility system can be made up with a variety of mechanisms, such as wheeled chassis, legs, and propellers, depending on the requirements for deployment. Sensor systems like lidar and camera help the AMR gather information and perceive the surrounding environment, which is processed by the software hosted on the computing unit of the robot to make decisions and navigate around moving and static obstacles.

Moreover, there is an increasing popularity in the deployment of multi-robot systems, where a group of operating robots can increase efficiency by dividing one main task into several sub-tasks and allocating them to be executed in parallel [3], [4]. The deployment of a multi-robot system can also provide more redundancy to address extreme or emergency conditions, as well as guarantee better coverage of a large area. The multi-robot systems also ensure systematic versatility as it is easy to be configured for different missions, which reduces overall cost and increases the flexibility of the system.

In this work, the task allocation for multi-robot systems is discussed in the context of a common robotic task, i.e., waypoint following, which involves guiding an AMR to visit

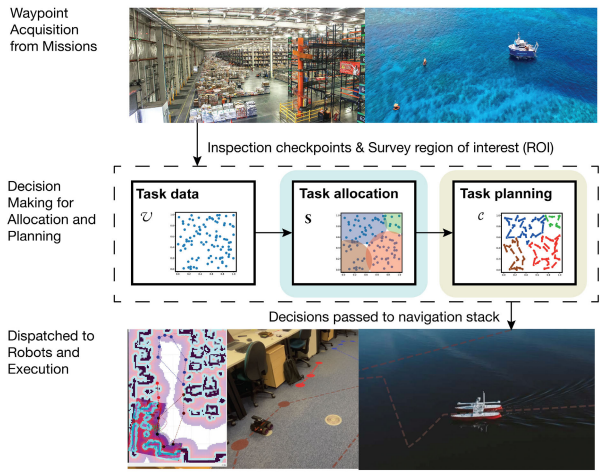


Fig. 1. The decision-making pipeline solves the task allocation and task planning modules in the context of waypoint following tasks for autonomous mobile systems. Addressing the waypoint following, a fundamental functionality module for autonomous mobility, such a decision-making system can be applied in many robotics applications, such as warehouse inspection with ground vehicles and maritime surveying with unmanned surface vehicles (USV).

a series of waypoints in a specified order provided by a dedicated task planning module. The waypoints are normally defined as spatial coordinates in the workspace, and the robot's objective is to reach the given waypoint within a predefined tolerance range. Depending on the density of the waypoint distribution, travelling along a set of sorted waypoints could also determine a sparse path for the robot [5], [6]. Waypoint following is a common but important task in autonomous mobile robot applications [7], [8]. It requires the robot to sense the surrounding environment, estimate its location and orientation, and use this information to plan the desired paths and navigate along the trajectories leading to the waypoints autonomously.

In such context, the waypoint following task should be addressed with a multi-robot system which consists of a fleet of autonomous mobile robots (AMR). Each AMR acts independently visiting a sub-set of the waypoints but also collaborates with each other contributing to the overall waypoint following task. Generally, the multi-agent planning for such systems is composed of two processes (Fig. 1):

- task allocation which assigns a sub-set of waypoints to each individual robot,
- task planning which determines the sequence of execution for each robot visiting the waypoints allocated to it.

An end-to-end DRL framework is proposed in this work to tackle the multi-agent planning for the AMRs. The proposed framework contains a feedback optimisation mechanism which couples the task allocation step and the task planning step. The framework has high adaptability in terms of the number of tasks for allocation. It also features a modular DRL agent where various networks can be implemented for specific purposes. The main contributions of this paper are as follows:

- A comprehensive DRL-based task allocation framework for multi-robot decision-making for waypoint following tasks is proposed. This end-to-end framework refines task allocation policies using subsequent task planning feedback, offering improvements over prior decoupled approaches.
- The modular DRL agent enhances flexibility, allowing various neural network agents. Two different network architectures are implemented, namely the multi-layer perception (MLP) and mixture of experts (MoE), for additional validation.
- The optimality gap and computational speed of the proposed approach have been reduced by more than 40% and 90%, respectively, compared to several state-of-the-art mTSP solutions, which showcases the superior performance achieved. Extensive testing on both simulated and real-world datasets further verifies its practical applicability.
- By configuring and integrating all required software and network modules using ROS 2, the proposed approach is validated on a multi-robot hardware platform. The implementation of the trained policy on hardware has led to the successful completion of the test mission in a fully autonomous way.

This paper is organised as follows. Related literatures are reviewed and discussed in Section II. In Section III, the task allocation problem and the objectives are defined. Then the task allocation agent, task planning solver, and reward of the proposed end-to-end DRL framework are explained in Section IV, and the training results are tested in a series of simulation scenarios in Section V. In Section VI, the experimental setup and a real-robot warehouse automation mission are described to validate the proposed method. Lastly, this work is concluded in Section VII. The source code of the decision-making system in this work is available at https://github.com/ucl-frl/rl_waypoint_mrta.git.

II. Related Works

The multi-robot waypoint following is a task where a fleet of robots is requested to navigate through a predefined set of waypoints while avoiding collisions with obstacles and each other. This task is present in many real-world applications such as warehouse automation [9], [10], [11], search and rescue [12], [13], [14], maritime and bathymetric surveying [15], [16], [17] and precision agriculture [18]. In multi-robot systems, efficient task allocation strategies are vital to ensure that the robots work collaboratively and accomplish the mission goals. Task allocation involves assigning specific tasks to different robots according to the requirements of the task [19], [20]. Efficient task allocation can help reduce completion time, improve overall performance, and increase the adaptability and scalability of the system.

However, task allocation in multi-robot systems is a challenging problem due to the changing environment [21], the uncertainty in the robot's perception and communication reliability, and the complexity of the problem. Common solutions used to address the multi-robot task allocation problem, such as market-based & auction-based approaches [22], [23], and ML-based approaches [24], [25], have common limitations in

terms of sole focus on the allocation process, but decoupled from the succeeding planning.

Therefore, it is important to consider a joint optimisation of both task allocation and its succeeding planning, for instance, planning visiting sequences and routes for each robot. This helps reach solutions which can better estimate the overall global optima, as the allocation of tasks to robots is considered in the context of the later optimisation problems in the system. One promising approach is to use deep reinforcement learning (DRL) algorithms, which can learn to optimise the task allocation policies through trials interacting with the environment.

In this paper, the focus will be on solving the task allocation and associated task planning using a learning-based framework. Considering the technical factors of robotics, this combinatorial optimisation problem is formed as a variant of the classic multiple travelling salesmen problem (mTSP). Research focusing on the mathematical optimisation has developed several heuristic algorithms, often with overly simplified datasets and lack of validation in the real world [26], [27]. In order to generate a better estimate of the optima, these heuristics are usually computationally expensive for robotic systems. On the contrast, similar robotic research tends to implement light-weight decoupled modules and only a small number of hardware implementations are presented [28], [29], [30]. Considering multi-robot cooperative navigation, a joint optimisation of the mTSP and clustering models can help reduce the overlapping among clusters in a simultaneous way, generating the benefits of increased collision avoidance capacity and improved system robustness and efficiency. In addition, when pretrained models are deployed in hardware constrained systems, a generalised learning-based method benefits from reducing computational load and execution time compared to traditional solvers that require large-scale computation at runtime.

III. Problem Formulation

The context of this research is aimed at the coordination of multiple AMRs to undertake multi-task missions. Specifically, the objective of the proposed framework is to cooperatively generate a plan for K robots to visit a set of waypoints, with each robot starting and ending at the same location, i.e. forming closed-loop paths for the robots. The given task can be formulated as consisting of a set of waypoints to traverse as $\mathcal{V} = \{\mathbf{v}_n\}, n = 1, \ldots, N$, where $\mathbf{v}_n \in \mathbb{R}^F$. In the scenarios described in this paper, $F = 2$, which represents the x, y waypoints' spatial coordinates.

The planned paths are formalised as a set of cycle graphs as, $\{\mathcal{C}_k\}$, $k = 1, \ldots, K$, where K is the number of vehicles deployed. For each cycle graph $\mathcal{C}_k$, the graph can be formulated as $\mathcal{C}_k = (\mathcal{V}^k, \mathcal{E}^k)$, where $\mathcal{V}^k = \{\mathbf{v}_i\}_k, i = 1, .., N_k$ denotes the task data set to be assigned to a particular USV, and $\mathcal{E}^k = \{l_{ij}\}_k$ denotes a set of lengths of the linked paths between $\mathbf{v}_i$ and $\mathbf{v}_j$, which topologically forms the edges of the simple cycle graph. Along the cycle graph, the elements of $\mathcal{V}^k$ will subsequently have ordered labels from 1 to N_k, where $N_k = \mid \mathcal{V}^k \mid$ is the number of elements in $\mathcal{V}^k$. The travelling distance of each cycle graph $\mathcal{C}_k$ could be obtained by summing the distances of all compartments within $\mathcal{E}_k$.

Denote $\mathcal{C}_k, k = 1, \ldots, K$ as the cycle graph, and the entire vertex set, $\mathcal{V}$ can be divided into K corresponding partitions, $\mathcal{V}_k, k = 1, \ldots, K$, where $\mathcal{V}_i \cap \mathcal{V}_j = \emptyset, \forall i, j \in \{1, \ldots, K\}$. Denote π_k as a permutation of the nodes in C_k that leads to the shortest path. In terms of the overall objectives, various metrics for performance evaluation can be adopted on a case-by-case basis. In this work, the objective function is set as the total route distance for conciseness. Therefore, the overall objective is written as:

$$\min_{\pi_k, C_k, k=1,\ldots,K} \sum_{k=1}^{K} L(\mathcal{C}_k|\pi_k, \mathcal{V}_k). \tag{1}$$

$L(\mathcal{C}_k|\pi_k, \mathcal{V}_k)$ in (1) is defined as:

$$L(\mathcal{C}_k|\pi_k, \mathcal{V}_k) = \|V^k_{\pi_{N_k}} - V_{\pi_1}\|_2 + \sum_{j=1}^{N_k-1} \|V^k_j - V^k_{j+1}\|_2, \tag{2}$$

where $\|\cdot\|_2$ calculates the Euclidean distance.

To solve (2), a two-stage learning process is introduced as per the aforementioned task allocation and task planning stages: a) partitioning $\mathcal{V}$ into K subsets; b) solving travelling salesman problem (TSP) within the K subsets. Since both stages are NP-hard, it is impossible to obtain exact solutions of the global optima using classical methods. The multi-stage NP-hard problem can be solved using a set of heuristic methods in a cascade procedure, where the heuristics approximate the exact solution in an iterative way. During the approximation process, the heuristic algorithm will sacrifice some optimality for computational speed, which will eventually lead to a systematic disadvantage. Regarding this work, coupling the two stages with specially designed feedback between task allocation and task planning can mitigate this disadvantage. Therefore, this problem can be solved in a coupled manner by adopting an RL structure. More specifically, the evaluation results of the task planning will be taken into account and used as the rewards in the RL structure. This will be further used to guide the optimisation of (1) where the policy gradient strategy is used in this paper.

IV. End-to-End DRL Framework

A. Background of DRL

DRL is a branch of machine learning (ML) methods that train an artificial agent to learn by interacting with an environment through trials. DRL combines reinforcement learning (RL) with deep learning (DL) techniques, allowing the agent to learn to make decisions based on unstructured input like raw sensor data in robotics.

In DRL, deep neural networks (DNNs) are employed to approximate the policy function, which maps an observed state of the environment to an action taken by the agent. The use of DNN enables DRL to learn directly from the perceptual data without hand-crafted feature extractions, which forms the end-to-end learning capability and leads to better generalisation towards different tasks. Policy gradient methods are often used as the optimiser for the policy functions in DRL. These methods take the gradient of the policy function with respect to the parameters of the network using it to update the parameters to optimise the network.

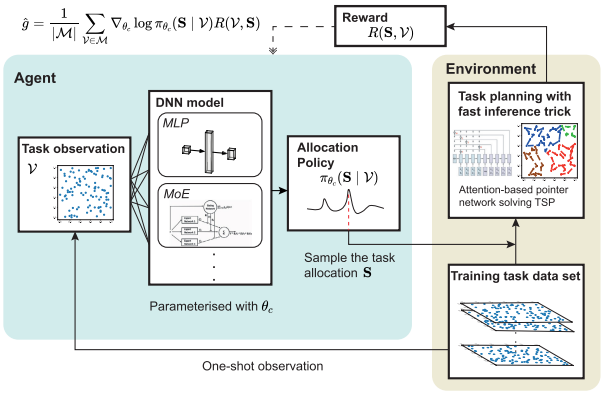


Fig. 2. The diagram of the proposed DRL framework.

DRL has great potential and already impressed the public in a number of fields like robotics, gaming and autonomous driving by allowing agents to learn from their own experience and improve their networks and the way to behave over time, without explicit programming.

B. Overall Framework

The proposed end-to-end framework, which conceptually follows the general structure of DRL, as shown in Fig. 2, comes with the agent module highlighted in blue and the environment module in light yellow. The DRL agent module is responsible for the partition procedure to cluster the given waypoints; whereas, the environment module that the agent interacts with provides the raw data and is also in charge of solving the TSP problem for each allocation cluster. The framework starts by taking in the task data as the input to form a one-shot task observation. The core component of the DRL agent module is a deep neural network (DNN) which represents the task allocation policy. The policy parameterised by the DNN is modelled as a discrete categorical distribution in order to make the task allocation decisions at the output stage. The agent, then, samples the clustering results with respect to the modelled distribution. The input and output of the agent are formatted as a waypoint tuple and task allocation matrix respectively so that any DNN network that can adopt this input/output pattern can be used to parameterise the policy of the modular DRL agent.

After the task allocation decision has been sampled, waypoints ($\mathcal{V}$) will be grouped into K subsets (K represents the sum total of USVs). For each subset, TSP will be solved to generate the task execution sequence with various existing methods available, such as branch and cut [31], dynamic programming [32], and self-organising map [33]. The pre-trained attention-based pointer network [34] is adopted. According to Bello et al. [35], in terms of solving TSP problems with 50 cities and 100 cities, the pre-trained pointer network can be more than 10 times faster than the Concorde solver [36] respectively. Therefore, inferring pre-trained models can guarantee a much faster response speed compared to employing classical TSP solvers, which is critical for the success of the proposed DRL framework as the task planning module is invoked in every iteration step.

It should be noted, in this work, when training the proposed framework, apart from considering the optimal routing to evaluate the policy, the clustering quality should also be well evaluated. Hence, in this paper, a novel clustering reward design inspired by unsupervised graph neural network (GNN) node clustering has been proposed, which consists of two sub-reward, R_0 and R_c, reflecting the routing and the clustering respectively. Additionally, the balance between R_0 and R_c is manipulated via a tunable hyperparameter, λ. The overall reward R will be fed back to the DRL agent for policy optimisation using the policy gradient algorithm. Algorithm 1 briefly sketched the outline of the framework, which will be further described in the following paragraphs.

In general, by creating edges between each node and their h nearest neighbours with respect to Euclidean distances, the task data set $\mathcal{V}$ can be constructed into a graph, which can facilitate the quantitative evaluation by having an unsupervised loss, R_c. Details of this clustering reward, R_c, will be explained in Section IV-E.

C. Modular Deep Learning Based DRL Agent

In this section, the core of the proposed end-to-end DRL framework: the DRL agent, will be elaborated. The agent observes the task dataset and maps the data to an allocation decision. Its adaptable, modular design allows compatibility with diverse network models, enhancing the customisability of the proposed framework. Different applications primarily vary in state and action spaces. Accordingly, an input/output design of the framework is proposed, taking customisation requirements into consideration, which is achieved by training specific models aligned with corresponding input/output dimensions. In addition, the framework's implementation is further described in two different networks, which demonstrates that the framework is unbounded by specific network structures. It further highlights the flexibility of the proposed framework.

In this work, both the MLP and the MoE networks are tested. The basic MLP is tested for validating the framework with only the necessary setup. However, the naive MLP networks cannot capture the underlying complicated properties of the problem and are difficult to generalise to the unseen tasks. Therefore, for achieving higher adaptability, an assembling learning method that utilises an MoE [37], [38] mechanisms is introduced. Researchers who hope to adopt this framework can feel confident to employ other DNN networks that suit their application.

1) Input/Output: In order to build the modular DRL agent with customisable DNN networks to represent the policy, the input/output data structure of the network must be predefined and accepted by potential DNN candidates. The number of neurons of the input layer is set as, the input feature dimension, F, and the size of the input tensor is (M, N, F), where M is the batch size and N is the number of waypoints in one task, also known as the task size. As for the output, the allocation decision made by the agent is in the form of a matrix (allocation matrix), $\mathbf{S} \in \mathbb{1}^{N \times K}$ for implementation,

Algorithm 1 DRL Task Allocation Algorithm

Require: training dataset $\mathcal{X}$, selected neural network (NN), TSP solver (TSP), parameter κ of the nearest neighbour graph (κ-NNG), k, tunable hyperparameter λ
  Initialise NN with parameters θ_c
  for $\mathbf{X} \in \mathbb{R}^{N\times F}$ in $\mathcal{X}$ **do**
    $\mathbf{S} \in \mathbb{R}^{N\times K} \leftarrow \mathrm{NN}_{\theta_c}(\mathbf{X})$
    for $k = 1, 2, \ldots, K$ **do**
      $\pi_k \leftarrow \mathrm{TSP}(\mathbf{X}, \mathbf{S})$
    end for
    $R_0 \leftarrow \sum_k$(travelling distance of π_k)
    $\mathcal{G} \leftarrow \kappa\text{-NNG}(\mathbf{X}, \kappa)$
    $R_c \leftarrow \mathrm{MinCutLoss}(\mathcal{G}, \mathbf{S})$
    $R \leftarrow (\lambda - 1)R_0 - \lambda R_c$
    $\hat{g} \leftarrow \nabla_{\theta_c} \log p_{\theta_c}(\mathbf{S}|\mathbf{X})R$
    $\theta_c \leftarrow \mathrm{ADAM}(\theta_c, \hat{g})$
  end for

where the value of $\mathbb{1}$ is set to 1 when $\mathbf{v}_n$ is allocated to robot k, otherwise, 0. Each row of the allocation matrix is a K dimension vector with each element representing the probability for the corresponding waypoint to be allocated to the k-th robot. The output logits of the last layer are converted into a categorical distribution, regarding the K robots as K options of the decisions. Therefore, each row of the allocation matrix produces an action distribution for the corresponding waypoint. The size of the output tensor representing the allocation matrix is (M, N, K).

2) MLP-Based Architecture: The most straightforward deep learning architecture to generate the policy is a multi-layer perception (MLP) with one hidden layer. The number of neurons in the input layer is set as F, and the size of the input tensor is (M, N, F). The output is a K dimension vector with each element representing the probability for the corresponding waypoint to be allocated to the k-th vehicle. The size of the output tensor is (M, N, K). Since the neuron number of the hidden layer also affects the performance, it is regarded as a hyperparameter. Empirically, the hidden layer with 128 neurons has the best performance during the tests. In summary, this process can be formulated as follows.

$$\mathcal{O}_l = \mathrm{MLP}_{\mathrm{layer}}(\mathcal{V}) = \boldsymbol{W}\mathcal{V} + \boldsymbol{b}, \tag{3}$$

where $\mathcal{V} \in \mathbb{R}^{M\times N\times F}$ denotes the feature of waypoints to traverse, $\boldsymbol{W} \in \mathbb{R}^{F\times 128}$ and $\boldsymbol{b} \in \mathbb{R}^{1\times 128}$ denote the learnable weight matrix and bias vector, respectively. $\mathcal{O}_l \in \mathbb{R}^{M\times N\times K}$ denotes the logits outputted from the MLP layer. Upon applying the *softmax* activation function to the output $\mathcal{O}_l$, it can be converted into a Categorical probability distribution $\mathcal{P}_l$, from which the waypoints allocated to the k-th vehicle is sampled.

3) MoE-Based Architecture: The deep learning network served as the DRL agent in this framework is not limited to the MLP or any other specific networks. Provided the input and output dimensions, different neural network architectures can fit in the framework to adapt to different tasks where specialities are required. In order to test the adaptability of the DRL agent in terms of different neural network architectures, besides the classic MLP, a MoE [39] is also employed in the framework. The MoE agent consists of several independent neural networks, namely the experts. Each expert neural network is supposed to be specialised in one particular task. Above the expert neural networks, there is another managing neural network that will decide which expert to solve the given task. Regarding the research problem of this DRL framework, there are n expert neural networks in the MoE. Each of them specialises in allocating waypoints to one specific vehicle. Meanwhile, the managing neural network decides the weights for the results from each expert. The output is a logit vector that can be computed via a weighted sum model, from which the deep learning-based RL agent makes allocating decisions for the waypoints. In terms of formulas, the aforementioned MoE-based process can be summarised as follows.

(a) *Construction of each expert model*. Suppose there are n_h expert models, each being an MLP layer.

$$\mathrm{expert}_i(\mathcal{V}) = \mathrm{ReLu}(W_i\mathcal{V} + b_i), \tag{4}$$

where $i \in [1, n_h]$ is an integer. $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ denote the learnable weight matrix and bias vector of the i-th expert model, respectively. ReLu is an activation function.

(b) *Construction of gating network*. The gating network g utilises a softmax function to output the weights for each expert:

$$g_i(\mathcal{V}) = \frac{\exp(\boldsymbol{v}_i^T\mathcal{V} + \boldsymbol{c}_i)}{\sum_{j=1}^{n_h}\exp(\boldsymbol{v}_j^T\mathcal{V} + \boldsymbol{c}_j)}, \tag{5}$$

where $\boldsymbol{v}_i$ and $\boldsymbol{c}_i$ are the weights and biases of the gating network.

(c) *Calculation of the overall MoE output*. The output of the entire model is a weighted sum of the outputs from each expert model:

$$f(\mathcal{V}) = \sum_{i=1}^{n_h} g_i(\mathcal{V}) \cdot \mathrm{expert}_i(\mathcal{V}) \tag{6}$$

Like $\mathcal{O}_l$ in (3), $f(\mathcal{V})$ can be used to model a Categorical probability distribution and perform sampling.

The framework with either MLP or MoE can be adaptive to different values of N, enabling the framework to infer tasks with different sizes of the training set.

D. Pointer Network Based Task Planning

After the completion of the task allocation procedure, the allocation decision is passed to the following task planning module in the format of the allocation matrix described in the last section. The task planning module will accordingly plan the execution sequence for each robot in the fleet given their assigned waypoints, which is, mathematically, a TSP problem.

Since TSP is an NP-hard problem, the exact solution of its global optima cannot be solved within polynomial time. There are several rule-based heuristic methods which can obtain approximate solutions within a reasonable time. However, wrapping these iterative heuristics into RL training will substantially increase the time complexity of the algorithm. In order to guarantee the RL agent is trainable, the trained models of attention-based reinforcement learning are

employed [34], [40]. The algorithm generates the solution π_k for the node set $\mathcal{V}_k$, and π_k is formed as the action selected per time step $\{\pi_k^1, \pi_k^2, \ldots, \pi_k^t, \}$. As the reward for training this attention-based RL policy is assigned episodically for a TSP problem, the attention-based model is optimised via policy gradient [41]. The objective $J(\theta_p \mid \mathcal{V})$ is the expected cost, which will be estimated with respect to the parameters θ_p.

$$J(\theta_p \mid \mathcal{V}) = \sum_k J^k(\theta_p \mid \mathcal{V}_k) \tag{7}$$

$$\nabla_{\theta_p} J^k(\theta_p \mid \mathcal{V}_k) = \mathbb{E}_{p_{\theta_p}(\pi_k|\mathcal{V}_k)}[(L(\mathcal{C}_k|\pi_k, \mathcal{V}_k) - b(\mathcal{V})) \nabla \log p_{\theta_p}(\pi_k \mid \mathcal{V}_k)], \tag{8}$$

where $b(\mathcal{V})$ is a value function acting as a baseline to reduce the variance in policy gradients [41], [42], which can stabilise the training process and improve the training efficiency.

It is also worth to ensure the loss of optimality is within an acceptable range when employing the pretrained pointer network models for solving TSP. According to the benchmarking results reported in [34], in terms of TSP with 20, 50 and 100 waypoints, the pointer-network-based solvers suffers from an average optimality gap of 1.74% consuming average 1.09 s. As a baseline, the Google OR-tools solver suffers from an averaged 1.80% optimality gap and the Concorde achieved the optima with an averaged runtime of 9.33 s.

E. Reward Design and Training of the Framework

Another key component of a DRL framework is the reward design which returns the performance of the state-action pairs to the optimisation process. As mentioned before, in order to help the DRL framework have a sophisticated estimation of performance, a reward consisting of two parts is designed: 1) a single-value reward reflecting the objective of task allocation and planning, which, in this case, is the total routing distance, and 2) a clustering reward. For the representative of the overall objective, the total route distance, R_0, of the K vehicles will be returned from the task planning stage as (9), where the TSP solutions will be output as a policy π_k based on partition results. π_k is generated for the node set $\mathcal{V}_k$ and formed as the action selected per time step $\{\pi_k^1, \pi_k^2, \ldots, \pi_k^t, \}$. With both the node sets $\mathcal{V}_k$, and the corresponding sequences π_k available, the cycle graph $\mathcal{C}_k$ can be obtained. $\mathcal{V}_k$, π_k and $\mathcal{C}_k$ will then be passed to the reward module. The total route distance for all autonomous robots, R_0, is calculated as:

$$R_0 = \sum_{k=1}^{K} L(\mathcal{C}_k|\pi_k, \mathcal{V}_k) \tag{9}$$

The R_0 is normalised as in (10), before used as reward component by policy gradient.

$$\bar{R_0} = \frac{R_0 - \mu(R_0)}{\sigma(R_0) + \epsilon} \tag{10}$$

where $\mu(R_0)$ is the average of a batch of R_0, and $\sigma(R_0)$ denotes the standard deviation of a batch of R_0.

Besides, adding the clustering reward for the DRL agent helps generate clustering results. In this paper, MinCut pooling [43] is adopted as the base for the clustering reward. MinCut pooling develops a soft clustering, formulated as an allocation matrix, $\mathbf{S} \in \mathbb{1}^{N \times K}$. A multiple layer perception (MLP) within the MinCut predicts the allocation matrix $\mathbf{S}$, given the waypoints $\mathcal{V}$ of a graph. The original implementation consists of an unsupervised loss which has two components: the basic clustering loss as:

$$R_{cc} = -\frac{\mathrm{Tr}(\mathbf{S}^\top \mathbf{A}\mathbf{S})}{\mathrm{Tr}(\mathbf{S}^\top \mathbf{D}\mathbf{S})} \tag{11}$$

and an orthogonality loss to avoid degeneration clustering:

$$R_{co} = \left\| \frac{\mathbf{S}^\top \mathbf{S}}{\|\mathbf{S}^\top \mathbf{S}\|_F} - \frac{\mathbf{I}_K}{\sqrt{K}} \right\|_F \tag{12}$$

where $\mathbf{A}$ and $\mathbf{D}$ are the adjacency matrix and degree matrix of the encoded graph which will be introduced in the following paragraph. The $\mathrm{Tr}(\cdot)$ denotes the trace of the matrix and $\|\cdot\|_F$ denotes the Frobenius norm.

In order to implement the MinCut pooling as the clustering reward, the task data set $\mathcal{V}$ has to be converted into a graph. This is achieved by taking the undirected κ-nearest neighbour graph (κ-NNG) of $\mathcal{V}$ in its Euclidean distance space, as $\mathcal{G} = (\mathcal{V}, \mathcal{L})$. With the adjacency matrix, $\mathbf{A}$, and degree matrix, $\mathbf{D}$, of the κ-NNG, $\mathcal{G}$, the clustering loss and orthogonal loss can be computed by (11) and (12). Then the unsupervised clustering loss, R_c, is obtained via:

$$R_c = R_{cc} + R_{co} \tag{13}$$

The training objective of the DRL framework is to optimise the partition with external reference from the TSP solutions mentioned in section IV-D. Therefore, by combining the cost reflecting the overall optimisation objective based on TSP results, and the unsupervised clustering loss, the reward is eventually defined as:

$$R = (\lambda - 1)\bar{R_0} - \lambda R_c, \tag{14}$$

where λ is the hyperparameter leveraging the influence of the unsupervised clustering loss within the overall reward.

Degeneration happens when the task allocation policy doesn't assign any waypoint to an available robot in the fleet, which is viewed as an unacceptable case. Therefore, when the algorithm detects degeneration cases during training, a punishment strategy will be triggered, $R_0 = p$, where p is the penalty score which would be dramatically greater than a sensible R_0.

V. Simulation Implementations

This section covers the training of the task allocation policies and a set of simulations testing the framework with the trained models. The test cases involve both naive scenarios with randomly generated data and practical scenarios with real-world data.

A. Experiment Setup and Training

PyTorch [44] is used for the implementation and training of the proposed method. The state is provided to the agent on the go by generating N waypoints from a uniform distribution on the $[0, 1) \times [0, 1)$ unit square for each episode of training,

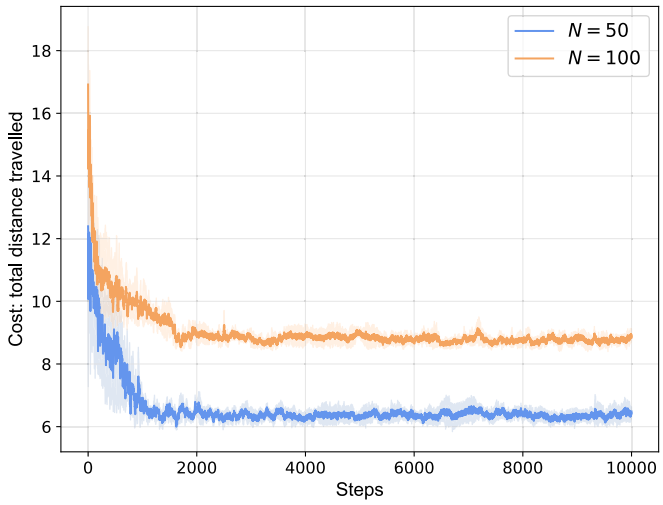


Fig. 3. Averaged training curves of the training sessions on $N = 50$ and $N = 100$ datasets.

so that the trained models could be used in different scales by normalising the input. The agent makes a decision in the form of an allocation matrix $\mathbf{S}$ as described in section IV. For the optimisation objective, the agent focuses on minimising the total route distance of the robots in the entire mission. The closed loop routes for robots to execute are planned by the fast-response TSP solver introduced in IV-D.

An MoE network with 3 component experts is adopted as the agent. Each component expert network is a 3-layer MLP with a hidden layer size of 128, using the ReLu activation function. The training produced two sets of models on $N = 50$ and $N = 100$. For each set of the training sessions, three models are trained with $K = 3, 4$ and 5. During the training, 1 million sets of waypoints were generated. Each set of waypoints is used as a task instance which contains 100 random waypoints distributed within the defined workspace. The batch size of training sessions is set as 32. An Adam optimiser [45] with a learning rate of 0.01, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ is used in the policy gradient method. As for the reward-related hyperparameters, λ is set to 0.5 and p is 10 times the length of the workspace.

The training curves for two model sets are shown in Fig. 3, illustrating the RL agent's performance throughout the training. The vertical axis indicates the cost of total travel distance by all robots, while the horizontal axis denotes the training step. The data, derived from the first 10,000 training iterations of the six open-source models available at our repository,[1] showcase two curves. The blue and orange curves are the averaged training cost of $N = 50$ and $N = 100$, respectively. The shades shown in the figure visualise the bounds defined by average value $\pm$ standard deviation. Initially, there is a noticeable descent in the cost, indicating the rapid acquisition of the easy rewards and the agent's initial exploration of the state space. Subsequently, the curve tends to stabilise, reflecting a period of fine-tuning as the agent refines its policy.

[1]https://github.com/ucl-frl/rl_waypoint_mrta.git

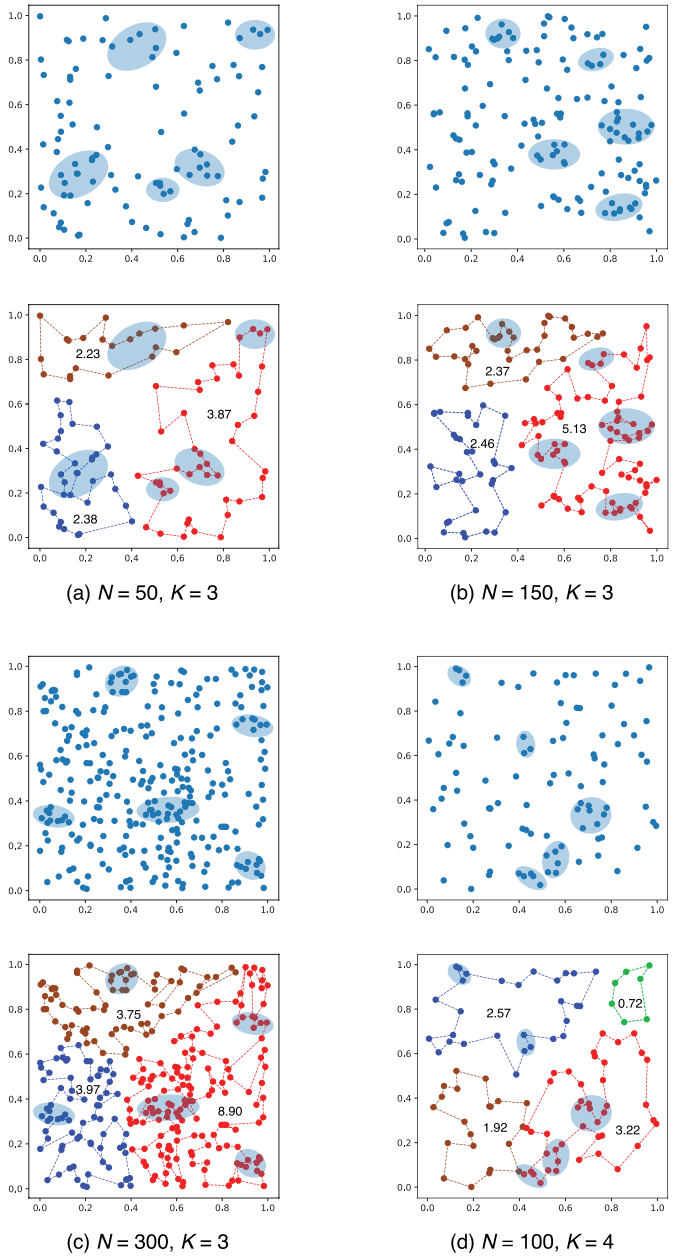
B. Experiment Results and Evaluation

In order to validate the framework and the performance of the pre-trained models, a set of experiments are performed to test the trained models. The experiments consist of missions that require the agent to make decisions with a different number of available robots, $K = 3$ and $K = 4$, where the different pre-trained models described in the experimental setup are utilised respectively. Four generated test missions have a different number of waypoints to be traversed.

The workspace is kept identical to the normalised one used for training where four sets of randomly generated waypoints are located, forming the four missions with a different number of waypoints, $N = 50, 100, 150$ and 300. The pre-trained model with $K = 3$ would be used to solve the missions with $N = 50, 150$ and 300. By solving these three missions, the adaptability of the proposed framework is demonstrated in terms of the size of tasks, N. A further test is carried out by using the pre-trained model with $K = 4$ to solve the missions with $N = 100$ to ensure the framework is able to address task allocation for different sizes of robot fleets.

The results of the four experiments are shown in Fig. 4. For each sub-figure which corresponds to a mission, the upper sub-figures show the distribution of the waypoints in the workspace and the lower sub-figures visualise the allocation decision taken by the agent and the corresponding execution sequences planned by the pointer-network-based TSP task planner. For qualitative analysis and visualisation, a number of areas with densely distributed waypoints are shaded into blue. In the illustrated results, different colours represent different robots being responsible. To showcase the closed loop route generated by the TSP task planner, dashed lines link the consecutive waypoints according to the execution sequence, which indicates the visiting order during traversal instead of the absolute path guiding a robot. As for the absolute paths, the path planning solution will be discussed in section VI in the context of real-robot implementation. Taking Fig. 4 as an example, the 50 waypoints are assigned to the robots in three clusters, where none of the five pre-marked dense groups is separated into two different clusters. Then the task planning solver produced the execution sequences for the three robots with the routing distances of 2.23, 2.38, and 3.87 unit lengths, respectively. Fig. 4b and c illustrated two more complicated cases but the processing patterns are the same. In Fig. 4, the other pre-trained model is tested with $N = 100$ and $K = 4$ situations. Four allocation clusters are assigned to the robots, and four out of five pre-marked dense groups are kept within one cluster.

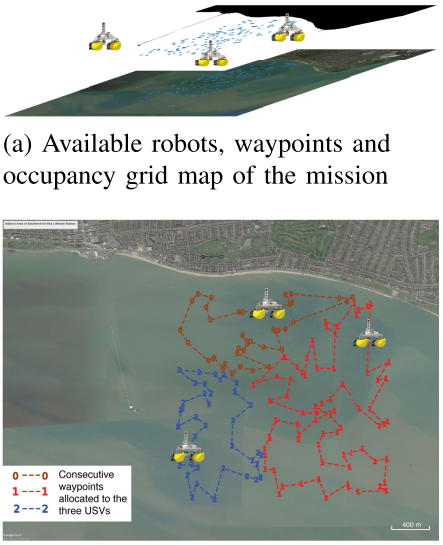
Regarding the inferred decisions made following the trained policy, the overall goal is addressed in an appropriate way where the waypoints are allocated to the available robots. The waypoints assigned to one robot cluster together, tending not to overlap with an area which is occupied by waypoints assigned to other robots. Only one of the aforementioned shaded areas is split by two robots, whereas the others are allocated to one single robot. Overall, the performance of these tests indicated the generalisation of the robot in terms of task size, as well as the capability to tackle the different sizes of robot fleets.



(a) $N = 50$, $K = 3$ (b) $N = 150$, $K = 3$

(c) $N = 300$, $K = 3$ (d) $N = 100$, $K = 4$

Fig. 4. Four sets of test sessions with different waypoint numbers, N, and available robot numbers, K. The waypoints are generated randomly in the 1×1 workspace which are allocated by the DRL agent to K robots based on the inference of the trained networks. Some densely distributed waypoints are manually marked with blue shades. The task allocation and task planning results are also visualised in colours to distinguish the waypoints allocated to different robots, and the execution sequences are indicated by dashed lines connecting the consecutive waypoints. The values of route distances for each robot are also included in the figure.



(a) Available robots, waypoints and occupancy grid map of the mission

(b) Decision-making result for the seach and rescue mission

Fig. 5. The inferred search and rescue plan assigned to 3 USVs along with designated execution sequences for each vehicle.

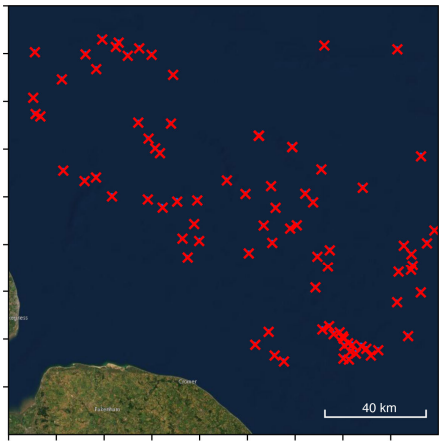


Fig. 6. The offshore infrastructure scenario simulated the task allocation and planning for regular maintenance and inspection of 86 Oil rigs located in the north sea petroleum reservoir.

C. Practical Implementations

To further validate that framework has the capability to be integrated into decision-making systems of autonomy solutions, two more practical scenarios are configured: (a) a search and rescue scenario in a coastal area, and (b) a regular maintenance and inspection mission for offshore infrastructures.

In the first search and rescue scenario, there are three USVs available to carry out this mission in an 2.5 km^2 adjacent area near the Royal National Lifeboat Institution Southend-on-Sea lifeboat station, UK. As shown in Fig. 5, 100 waypoints are randomly scattered into the mission area as the exploration checkpoints, and the system makes the task allocation and associated task planning decisions based on the pre-trained policy.

So far, waypoints in both the training and test tasks are generated based on the random generator, PCG64 [46], implemented in NumPy [47]. Therefore, in the second offshore infrastructure maintenance scenario, the proposed method is verified on real-world data, locations of 86 oil rigs in the North Sea petroleum reservoir, to ensure the DRL agent does not overfit into the random-pattern input data. The GIS data of the oil rigs are sourced from North Sea Transition Authority open data [48]. Eighty-six oil rigs are selected in an area to the east of Lincoln in the North Sea as shown in Fig. 6.

Both pre-trained models mentioned before inferred this simulation so that two different cases can be tested where the

TABLE I
COMPARATIVE TOTAL DISTANCE RESULTS ON SOLVING DIFFERENT SCALE TSPLIB PROBLEMS

K	Algorithms	kroA100 $N = 100$	kroA150 $N = 150$	kroB150 $N = 150$	kroA200 $N = 200$	Avg Opt Gap
3	PSO [52]	127181.21	196907.78	-	274854.17	673.43%
	IPGA [52]	25889.13	44004.46	-	53166.97	52.69%
	IWO [55]	45697.64	57256.01	-	66659.31	123.10%
	PGA [52]	71686.28	161911.94	-	229190.75	462.74%
	Ours	**22728.72**	**30194.03**	**29374.46**	**34762.93**	**13.85%**
	CPLEX [54]	20905	-	25741	29287	-
	Opt Gap of Our Method	**8.72%**	-	**14.12%**	**18.70%**	-
4	PSO [52]	124405.83	199202.46	-	276537.26	671.22%
	IPGA [52]	23312.45	36070.85	-	59225.40	57.18%
	IWO [55]	58404.80	72961.48	-	82029.60	180.38%
	PGA [52]	75062.36	160985.45	-	231062.08	475.07%
	Ours	**22769.33**	**28999.84**	**28859.81**	**33434.20**	**11.99%**
	CPLEX [54]	20831	-	25678	29256	-
	Opt Gap of Our Method	**9.31%**	-	**12.39%**	**14.28%**	-
5	PSO [52]	123017.49	201597.24	-	270835.05	658.94%
	IPGA [52]	24271.69	46004.41	-	58581.94	58.52%
	IWO [55]	72544.38	89122.50	-	103767.16	252.10%
	PGA [52]	77737.34	164183.75	-	244364.07	504.51%
	Ours	**23375.24**	**29621.99**	**28986.89**	**33086.35**	**12.76%**
	CPLEX [54]	20721	-	25749	29305	-
	Opt Gap of Our Method	**12.81%**	-	**12.57%**	**12.90%**	-

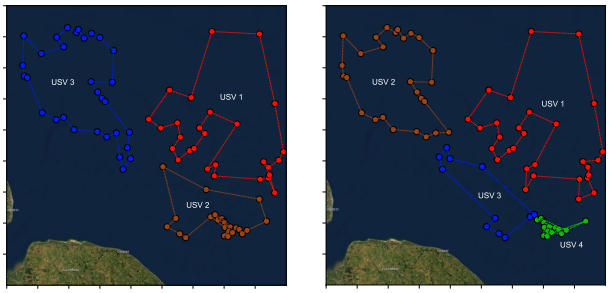


Fig. 7. Regular unmanned inspection plan covering 86 north sea oil infrastructures to be carried out by a fleet with 3 or 4 USVs.

maintenance and inspection missions of the 86 are allocated to two fleets with different sizes, $K = 3$ and 4. The decision-making results can be found in Fig. 7.

D. Comparative Analysis

The proposed task allocation framework for multi-robot systems is regarded as a *minsum* multi-depot mTSP problem, a unique variant of the broader mTSP or vehicle routing problem (VRP). There are two further variants of the multi-depot mTSP problem, differing in depot location determination. This work concentrate on the unpredetermined variant.

In such a problem, K robots will be dispatched without predetermined depots. Any K waypoints can be selected as the depots by the algorithm and the optimisation objective is to minimise the total distance travelled by all robots. Though there are many research covering the general mTSP [49], [50] and the fixed multi-depot problems [26], [51], very limited research has focused on this specific unpretermined variant [27], [52]. Zhou et al. [52] proposed a series of algorithms based on the genetic algorithm (GA) and particle swarm optimisation (PSO) [53] to solve mTSP under the same settings as the proposed framework. The two GA-based algorithms are namely partheno genetic algorithms (PGA) and Improved PGA (IPGA). Another commonly-used [52], [54] baseline method is the invasive weed optimization (IWO) proposed by Venkatesh and Singh [55]. The current known best results under this setup were reported by Karabulut et al. [54], using the IBM CPLEX platform to solve a mixed-integer linear programming (MILP) model with an one-hour runtime, where the results are rounded to integer. These known best results are underlined and reported in the 'CPLEX' rows of Table I.

All the available optimisation results and computational time are acquired from the above work for a comprehensive comparative analysis. The validation problems are from the open-source dataset, TSPLIB[2] which is well established as the standard for the analysis of conventional TSP and VRP problems. The benchmarking results using different TSPLIB instances are provided in Table I. Each problem instance is solved by our framework given $K = 3, 4,$ or 5 robots by inferring a model pre-trained with $N = 100$ dataset. None of the tested instance has been used for the training of our models. For the baselines, all available results are reported in Table I. The chosen problem instances, kroA100, kroA150, kroB150 and kroA200 are of different scale and commonly used in other mTSP research [56], [57], [58].

The values provided in Table I are the best *minsum* costs tested from all methods. The results of our framework outperform all the baseline methods and are also adequately close to the known best. The average optimality gap is also reported in the last column which compares each method

[2]http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

TABLE II
COMPUTATIONAL TIME ON SOLVING PROBLEMS OF DIFFERENT SCALE

Algorithms	$N = 35$, $K = 5$	$N = 15$, $K = 3$
PSO [52]	329.82 s ± 28.48 s	154.80 s ± 4.48 s
IPGA [52]	21.42 s ± 0.15 s	21.68 s ± 3.19 s
IWO [55]	31.72 s ± 2.07s	25.68 s ± 2.52 s
PGA [52]	7.97s ± 0.48 s	4.98 s ± 0.03 s
Ours	**0.23 s ± 0.02 s**	**0.11 s ± 0.03 s**
CPLEX	3600.00 s	3600.00 s

against the CPLEX results. Smaller optimality gap values mean better performance. It can also be found that when N of the test instances is closer to the $N = 100$ of our training dataset, the optimality gap tends to diminish. Therefore, in real applications, high quality solutions could be obtained by selecting proper models given an estimate of the number of waypoints. The computational time taken by the compared methods on solving problems with same scales and settings are provided in Table II in the form of average computational time ± variance. The baseline methods are run on a 2.7 GHz Intel Core i5-5257U (14 nm, 2015). Our hardware is a 2.6 GHz Intel Core i7-3720QM (22 nm, 2012), which, according to the CPU Benchmarks,[3] has a poorer single-thread performance compared to the i5-5257U. Obviously, our framework requires much less computing resources to deploy compared to all the baselines as well as the one-hour runtime of the known best solution, which fits more for edge computing systems in robotics applications, with an acceptable optimality trade-off.

E. Ablation Study

This section analyses the impact of the coupled task allocation framework and the tailored MoE employed in the RL agent using an ablation analysis. Besides the complete baseline model, there are results from four additional ablated models.

There is one model based on a simpler RL-agent but keeping the coupled framework. Two other models ablate the coupled framework, using conventional clustering algorithms as the task allocation modules instead. The one last model is set up with the most basic random allocation. All models are validated on the TSPLIB instances `kroA100`, `kroA150` and `kroA200` which has been introduced in the comparative analysis. While the task allocation module being switched or ablated, the task planning module, i.e. the TSP solver, is kept the same as the complete baseline model for all tests, which is the pointer network based solver introduced in Section IV-D. For the two RL-based models, pretrained parameters from the first 65 steps after the training convergence are used for the inference. For the non-RL-based decoupled models, 65 random seeds are provided for the algorithms to repeat the tests. In Table III, the average and standard deviation (SD) of the costs as well as the computational time of the 65 repetitions are reported.

The complete baseline model of this ablation study utilises the coupled task allocation framework introduced throughout this work. The task allocation module of the baseline is a MoE agent, as mentioned in Section IV-C, with two 128-dimension hidden layers and three 128-dimension expert layers. The results of the baseline task allocation module is reported in the 'MoE' row of Table III in bold font. In the first ablation step, the MoE is substituted with a dynamic scaling MLP which has three 128-dimension hidden layers. This MLP can also be regarded as a single-expert MoE. Using this MLP-based task allocation module could validate the improvement of the expert layers in the complete baseline. Then, the coupled allocation-planning loop is ablated. The ablation forms the conventional decoupled clustering + TSP solver framework to validate the improvement of the RL-based modular task allocation framework. The analysis involves two popular clustering algorithms, the k-means algorithm and a multi-way spectral clustering [59]. The tested spectral clustering employs an 10-NNG as the affinity matrix and it has no tuning parameters which leads to a trivial SD. Lastly, the decoupled clustering-based task allocation is replaced by a random process which samples task allocation labels from a discrete uniform distribution. For the consistency, all test cases reported in this section are based on the aforementioned hardware, 2.6 GHz Intel Core i7-3720QM.

As the results revealed in Table III, the coupled RL-based task allocation framework significantly improves the performance compared to the conventional clustering-based task allocation under all test cases. Meanwhile, the deployment runtime of the proposed framework, though only to a subtle extent, also outperforms the decoupled clustering-based task allocation. Within the proposed framework, it is also obvious that advanced neural networks like MoE could improve the performance against the basic MLP. However, it is also worth mentioning that for all learning based task allocation, extensive training on high performance computing (HPC) platforms is necessary, though the cost of training would not be reflected on the deployment of robotics systems.

F. Summary

In this section, the training setup of the framework and several experimental demonstrations based on both generated and real-world data are introduced. The task allocation framework solved all the test missions using the given resources correctly. Meanwhile, its adaptability to the size of tasks, as well as the capability to incorporate different sizes of the fleet are also validated. A comprehensive comparative analysis is also carried out to validate the performance of the proposed methods in the context of state of the art. Furthermore, an ablation study is reported to analyse the performance improvement of the designed components of the framework.

VI. REAL-ROBOT IMPLEMENTATION

In this section, the hardware implementation setup is explained. Meanwhile, a test run on the Husarion ROSbot robot platform carrying out a warehouse automation mission is performed.

A. Experiment Setup

Implementing the DRL-based task allocation framework on real robots is also necessary for testing the performance,

[3]https://www.cpubenchmark.net

TABLE III
PERFORMANCE OF DIFFERENT TASK ALLOCATION MODULES ON TSPLIB INSTANCES

K	Task Allocation	kroA100 $N = 100$				kroA150 $N = 150$				kroA200 $N = 200$			
		Avg Cost	Cost SD	Avg Time	Time SD	Avg Cost	Cost SD	Avg Time	Time SD	Avg Cost	Cost SD	Avg Time	Time SD
3	Random	154805.10	9854.29	0.18	0.01	228338.40	11753.46	0.23	0.03	301515.02	16178.03	0.25	0.01
	Kmeans	89353.41	3334.05	0.26	0.02	127764.85	6236.20	0.29	0.02	172433.02	6831.01	0.31	0.02
	Spectral	84459.00	0.00	0.22	0.01	136071.74	0.00	0.25	0.02	167565.03	0.00	0.29	0.02
	MLP	27325.61	1263.37	0.20	0.01	35681.88	1488.86	0.24	0.01	41102.14	1846.82	0.29	0.02
	MoE	**25337.92**	**875.48**	**0.19**	**0.02**	**32298.74**	**1181.60**	**0.23**	**0.02**	**36914.51**	**1321.88**	**0.27**	**0.02**
4	Random	152533.32	7064.78	0.22	0.01	227066.06	13339.53	0.27	0.01	299348.52	15486.63	0.29	0.01
	Kmeans	75672.43	3498.95	0.29	0.02	111540.46	4402.47	0.32	0.02	145454.79	7913.57	0.36	0.02
	Spectral	83320.14	0.00	0.24	0.01	113775.44	0.00	0.28	0.01	156019.54	0.00	0.37	0.04
	MLP	26556.19	869.08	0.22	0.03	34106.62	866.40	0.25	0.02	38778.69	679.56	0.29	0.02
	MoE	**24561.65**	**1088.36**	**0.23**	**0.03**	**31570.47**	**1094.55**	**0.26**	**0.03**	**36541.09**	**1392.50**	**0.31**	**0.03**
5	Random	154907.45	8644.58	0.27	0.02	230319.96	12118.32	0.30	0.01	300972.51	14771.41	0.34	0.02
	Kmeans	65792.64	2682.97	0.32	0.03	97470.17	5698.84	0.36	0.03	126604.89	6697.60	0.39	0.02
	Spectral	67363.25	0.00	0.30	0.02	86913.51	0.00	0.34	0.02	134810.52	0.00	0.38	0.02
	MLP	33515.77	2830.05	0.28	0.02	41051.49	1696.10	0.33	0.01	45952.61	1955.38	0.36	0.02
	MoE	**25399.37**	**974.00**	**0.25**	**0.04**	**31447.06**	**1043.10**	**0.29**	**0.04**	**35574.70**	**1117.64**	**0.33**	**0.04**

robustness, and reliability of the system in real-world scenarios and addressing the sim-to-real gap. Therefore, a real-robot experiment using ROS 2 as the test middleware is configured. In this section, a concise real-world robotic test setup based on pure open-source software is firstly presented, and then, the task allocation and task planning systems is implemented on real robots based on this test framework to carry out an inspection task.

The experimental setup consists of three major hardware components, the workstation, robot fleet, and a router. The workstation hosts the occupancy grid map for robot localisation and navigation and the task allocation and task planning modules for decision-making. The workstation can be a normal PC as there are no computationally expensive algorithms relying on it. The robot fleet consists of K robots, each of which is equipped with an onboard computer to execute ROS nodes, sensors to observe the surrounding environment and estimate the robot state, and actuators to drive the mobile robot or commit any specific task. The router that bridges the other two components, as shown in Fig. 8, is in charge of data distribution, passing information and messages between robots in the fleet and the workstation.

The setup is implemented with ROS 2 Foxy [60] and its default data distribution service, Fast DDS. To deal with substandard network environments and gain flexible internet connectivity, the Linux-based embedded operating system OpenWrt is utilised, especially its repeater and wireless access point features. The connectivity interface between the workstation and the router is IEEE 802.3 [61] Ethernet connection for maximum reliability and data throughput speed. Wireless solution based on IEEE 802.11 [62], however, is used for mobile robots to ensure mobility. Autonomous mobility modules implemented on the robots are developed based on Nav2 [63], which is the default navigation package of ROS 2. The mobility module deployed on the onboard computer of the robots is composed of three major parts:

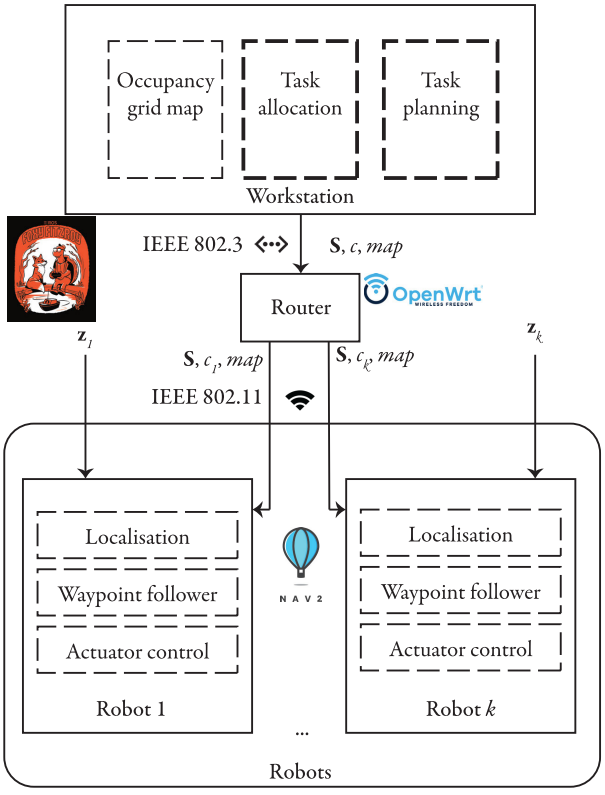


Fig. 8. The experimental setup for the real-robot tests based on ROS 2. The networking among the ground control workstation and the autonomous mobile robots is managed by a router through both Ethernet and wireless connection. Localisation and navigation algorithms are implemented using the navigation stack of ROS 2, which helps the robot follow the waypoints in the given map using only onboard sensor observations. The robots subscribe to decisions made by the task allocation agent and task planning module that determines the responsibility distribution and execution sequences.

- localisation algorithm that calculates the frame transformation between the odometry and mapping origins,
- waypoint following algorithm that plans the paths for the robot to navigate to the given waypoint and avoid obstacles,

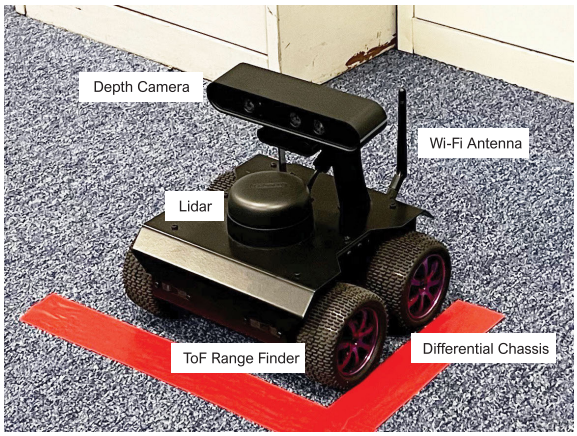
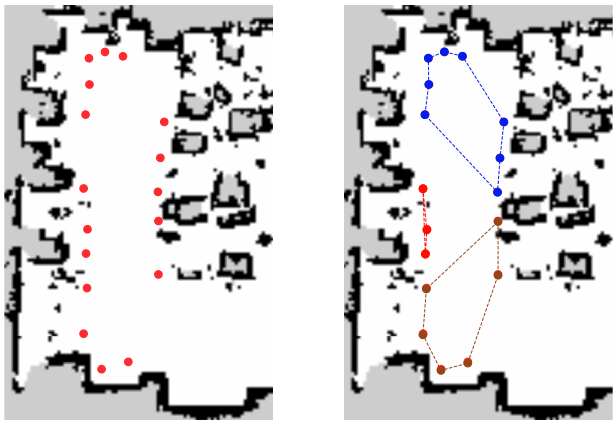


Fig. 9. Husarion ROSbot 2 pro platform used in the experiment.



(a) Waypoint distribution of the configured task (b) Inferred decision plan

Fig. 10. The experimental setup for the real-robot tests based on ROS 2. The networking between the ground control workstation and the autonomous mobile robots.

- actuator control algorithm that communicates with the actuator driver board to manoeuvre the robot and follow the planned paths.

The mobile robot platform used in this setup is Husarion ROSbot 2 Pro (Fig. 9). ROSbot 2 Pro is an educational AMR mounted with a series of sensors: a 360° lidar scanner, a structured light RGB depth camera, an integrated inertial measurement unit (IMU) and four time of flight (ToF) range finders. The main perceptual sensor is the omnidirectional lidar scanner with a detection radius of 25 m and a sampling rate of 16000 Hz. The computing unit that hosts the onboard ROS nodes is an UP Board with a 1.92GHz Intel® ATOM™ x5-Z8350 Processor. The actuators of the mobility module are 4 DC motors with encoders mounting on a chassis with differential dynamics. Other ROS-compatible AMRs may also be used in the fleet, provided that they are equipped with required sensors, power supply and actuators. The ROS package deployed on the onboard computing unit is available at https://github.com/ucl-frl/frl_rosbot_onboard.git.

B. Hardware Implementation: A Warehouse Inspection Scenario

This test task is configured as a warehouse inspection scenario in an office environment, assuming each desk is a warehouse racking bay. The task consists of 17 waypoints in the workspace (Fig. 10a), 2 inspection waypoints for each racking bay and an additional inspection waypoint close to the entrance of the warehouse. In a real warehouse mission, the robot would normally be required to execute further tasks such as taking pictures or scanning QR codes. Therefore, in this task, users would request the robot to turn to face the racking bay after arriving at the racking bay inspection waypoints and to face North for the entrance inspection waypoint.

Besides the orientation on the arrival of waypoints, other criteria of success for the implementation includes: traversing all the allocated waypoints, following the designated execution sequence unless replanned by the decision-making system, and no collision against other robots or obstacles.

In this inspection scenario, a fleet consisting of three robots is dispatched. The task allocation policy divided the inspection waypoints into three groups containing 6, 3, and 8 waypoints respectively. Based on this allocation decision, the task planning module also provides the execution sequences shown in Fig. 10b.

In the configured test environment, GPS is not applicable for indoor setup. To make the case closer to applications in practice, it is also assumed that there is no external indoor localisation system such as ultra-wide band (UWB) anchors or motion capture system mounted. Consequently, this test can only rely on the onboard perception system of the robots for localising themselves on the given map. The *a prior* occupancy grid map, as in Fig. 10, is obtained with simultaneous localisation and mapping (SLAM) carried out by the Husarion ROSbot 2 Pro in advance. The adaptive Monte Carlo localisation (AMCL) [64], [65] is adopted in the implementation. As for path planning, a global path planner based on the Dijkstra algorithm computes the route from the current state of the robot to the goal state at the waypoint taking obstacle avoidance into account. The obstacle information is formatted into a global cost map given the occupancy grid map hosted in the workstation. The path is then subscribed by the DWB local motion planner which is an extended version of the classic dynamic window approach (DWA) [66] algorithm for path following. The motion planner actively observes the vicinity of the robot using the onboard sensors, lidar in this case. If moving occupancy cells are observed, the planner will assume that they are dynamic obstacles and try to predict their trajectories a few time intervals ahead. Based on the obstacle information, the DWB planner works out the desired speed in a search space limited by the current speed and acceleration limit of the robot by maximising an objective which aims to align the robot to the goal, avoid obstacles and speed up the manoeuvre.

Two of the three robots (Robot 0 and Robot 2) dispatched are deployed in the test. The velocity command for the actuator of the differential chassis consists of two components, the linear velocity $\dot{x}$ and the angular velocity $\dot{\theta}$. The maximum velocities are limited to $\max(\dot{x}) = 0.26$ m/s, $\max(\dot{\theta}) = 1$ rad/s, and the corresponding acceleration maxima

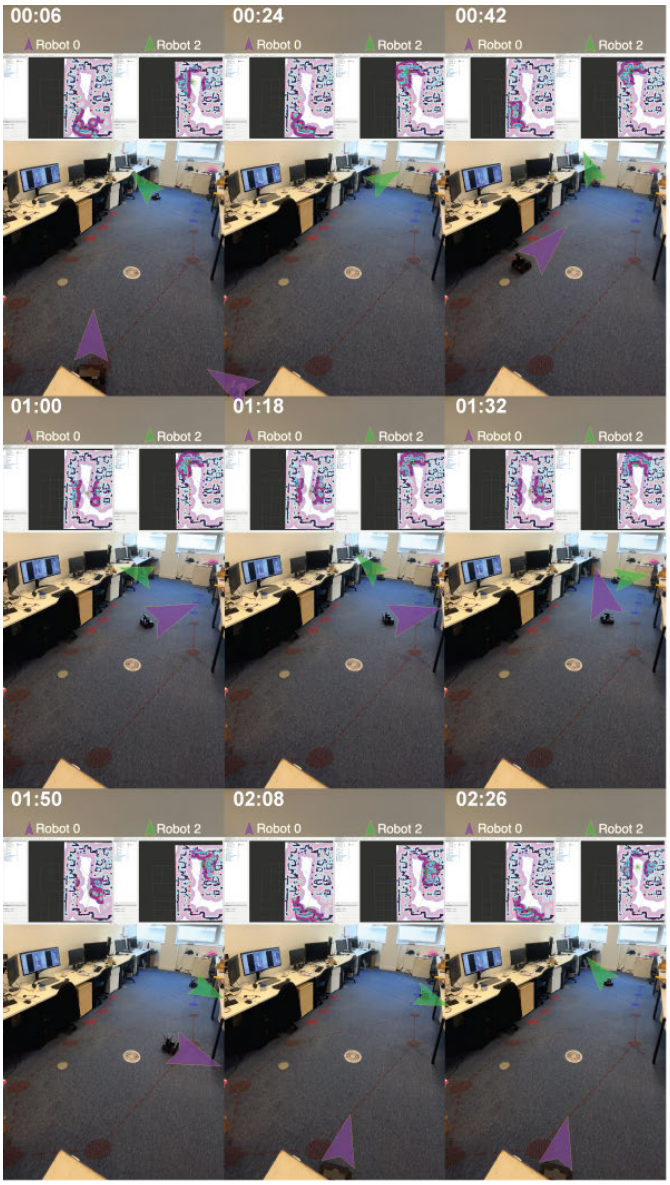


Fig. 11. The warehouse inspection scenario implementation.

are $\max(\ddot{x}) = 2.5\ \mathrm{m} \cdot \mathrm{s}^{-2}$, $\max(\ddot{\theta}) = 3.2\ \mathrm{rad} \cdot \mathrm{s}^{-2}$. In the warehouse inspection scenario test, the robots successfully executed the brown route at the south end of the map and the blue route at the north end as shown in Fig. 11, which meets all defined criteria under the guidance of the task allocation and task planning modules without collisions. At the beginning of the task, the robot sets off from the initial waypoints. In a real warehouse, the robots may need to navigate from their docks to the nearest assigned waypoint in the first place. In Fig. 11, nine pictures of the test are taken at a constant time interval of 18s. Due to the short distance from detected obstacles within the local planner's dynamic window, there is some offset from the actual path to the ideal lines between two consecutive waypoints. However, the robots reached the designated waypoints and the correct orientation which faces the warehouse racks (desks in the photo) to inspect. The average time consumption of the robot from one waypoint to the consequent one is about 15 seconds, where the robots will turn from the inspection attitude back to follow the planned path and turn to the rack again after reaching the succeeded waypoint. Additionally, there is no human intervention or any auxiliary instruction provided other than the ones described in this work throughout the hardware implementation. More details can be found at https://youtube.com/watch?v=x_Vipmf4ph0.

VII. Conclusion

In this work, the task allocation problem for multi-AMR systems is discussed in the context of the waypoint following tasks. An end-to-end DRL framework is proposed to address the decoupled decision-making of the task allocation and the succeeding task planning module. The modular design of the DRL agent and the objective-oriented component reward makes it possible to apply the framework in tasks of various sizes as well as to extend this framework for more autonomous applications. Experimental results have validated the good scalability which ensures pretrained models can be reused when the waypoint number changes. Formulating the research problem as a variant of mTSP, a comparative study against other state-of-the-art solvers revealed the superior performance of our framework. Moreover, a real-robot implementation solution based on ROS 2 is provided, and an autonomous test run with no external localisation and human interventions is deployed to validate the trained policies.

Limitations have also been identified in this study. Currently, the framework cannot generalise to a robot fleet of different number of robots from those used in training. Skewness in some allocation plans is also observed, a reasonable occurrence given that the reward is based on total distance. In future research, the exploration of recurrent neural networks and a more sophisticated reward design could potentially mitigate these known issues. Notably, the models generated by the proposed framework require minimal computing capacity, rendering it ideal for hardware constrained robotic systems.

References

[1] N. J. Nilsson, "Shakey the robot," SRI Int., Menlo Park, CA, USA, Tech. Note 323, 1984.

[2] J. Crowley, "Navigation for an intelligent mobile robot," *IEEE J. Robot. Autom.*, vol. RA-1, no. 1, pp. 31–41, Jul. 1985.

[3] M. Afrin, J. Jin, A. Rahman, A. Rahman, J. Wan, and E. Hossain, "Resource allocation and service provisioning in multi-agent cloud robotics: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 842–870, 2nd Quart., 2021.

[4] Y. Rizk, M. Awad, and E. W. Tunstel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Comput. Surveys*, vol. 52, no. 2, pp. 1–31, Apr. 2019, doi: 10.1145/3303848.

[5] E. Krell, A. Sheta, A. P. R. Balasubramanian, and S. A. King, "Collision-free autonomous robot navigation in unknown environments utilizing PSO for path planning," *J. Artif. Intell. Soft Comput. Res.*, vol. 9, no. 4, pp. 267–282, Oct. 2019, doi: 10.2478/jaiscr-2019-0008.

[6] T. Manderson et al., "Vision-based goal-conditioned policies for underwater navigation in the presence of obstacles," 2020, *arXiv:2006.16235*.

[7] A. Felder, D. Van Buskirk, and C. Bobda, "Automatic generation of waypoint graphs from distributed ceiling-mounted smart cameras for decentralized multi-robot indoor navigation," in *Proc. 13th Int. Conf. Distrib. Smart Cameras*. New York, NY, USA: Association for Computing Machinery, Sep. 2019, doi: 10.1145/3349801.3349814.

[8] J. Krantz, A. Gokaslan, D. Batra, S. Lee, and O. Maksymets, "Waypoint models for instruction-guided navigation in continuous environments," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 15142–15151.

[9] K. Sharma and R. Doriya, "Coordination of multi-robot path planning for warehouse application using smart approach for identifying destinations," *Intell. Service Robot.*, vol. 14, no. 2, pp. 313–325, Apr. 2021.

[10] M. Beul, D. Droeschel, M. Nieuwenhuisen, J. Quenzel, S. Houben, and S. Behnke, "Fast autonomous flight in warehouses for inventory applications," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 3121–3128, Oct. 2018.

[11] A. Bhat, N. Kai, T. Suzuki, T. Shiroshima, and H. Yoshida, "Safe, efficient waypoint manipulation for path planning of non-holonomic robots," in *Proc. 27th Int. Conf. Autom. Comput. (ICAC)*, Sep. 2022, pp. 1–6.

[12] T. Yang and X. S. Shen, *Mission-Critical Search and Rescue Networking Based on Multi-Agent Cooperative Communication*. Singapore: Springer, 2020, pp. 55–76, doi: 10.1007/978-981-15-4412-5_5.

[13] M. Atif, R. Ahmad, W. Ahmad, L. Zhao, and J. J. P. C. Rodrigues, "UAV-assisted wireless localization for search and rescue," *IEEE Syst. J.*, vol. 15, no. 3, pp. 3261–3272, Sep. 2021.

[14] M. F. Ozkan, L. R. G. Carrillo, and S. A. King, "Rescue boat path planning in flooded urban environments," in *Proc. IEEE Int. Symp. Meas. Control Robot. (ISMCR)*, Sep. 2019, pp. B2-2-1–B2-2-9.

[15] G. A. Di Caro and A. W. Z. Yousaf, "Multi-robot informative path planning using a leader-follower architecture," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 10045–10051.

[16] N. Karapetyan, J. Moulton, J. S. Lewis, A. Quattrini Li, J. M. O'Kane, and I. Rekleitis, "Multi-robot Dubins coverage with autonomous surface vehicles," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 2373–2379.

[17] J. McConnell, Y. Huang, P. Szenher, I. Collado-Gonzalez, and B. Englot, "DRACo-SLAM: Distributed robust acoustic communication-efficient SLAM for imaging sonar equipped underwater robot teams," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2022, pp. 8457–8464.

[18] I. A. Hameed, "A coverage planner for multi-robot systems in agriculture," in *Proc. IEEE Int. Conf. Real-Time Comput. Robot. (RCAR)*, Aug. 2018, pp. 698–704.

[19] Y. Emam, S. Mayya, G. Notomista, A. Bohannon, and M. Egerstedt, "Adaptive task allocation for heterogeneous multi-robot teams with evolving and unknown robot capabilities," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2020, pp. 7719–7725.

[20] X. Chen, P. Zhang, G. Du, and F. Li, "A distributed method for dynamic multi-robot task allocation problems with critical time constraints," *Robot. Auton. Syst.*, vol. 118, pp. 31–46, Aug. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/

[21] J. C. Amorim, V. Alves, and E. P. de Freitas, "Assessing a swarm-GAP based solution for the task allocation problem in dynamic scenarios," *Exp. Syst. Appl.*, vol. 152, Aug. 2020, Art. no. 113437. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095741742030261X

[22] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *IEEE Access*, vol. 6, pp. 28573–28593, 2018.

[23] J. Shi, Z. Yang, and J. Zhu, "An auction-based rescue task allocation approach for heterogeneous multi-robot system," *Multimed. Tools Appl.*, vol. 79, nos. 21–22, pp. 14529–14538, 2020.

[24] B. Park, C. Kang, and J. Choi, "Cooperative multi-robot task allocation with reinforcement learning," *Appl. Sci.*, vol. 12, no. 1, p. 272, Dec. 2021. [Online]. Available: https://www.mdpi.com/2076-3417/12/1/272

[25] J. Yang, X. You, G. Wu, M. M. Hassan, A. Almogren, and J. Guna, "Application of reinforcement learning in UAV cluster task scheduling," *Future Gener. Comput. Syst.*, vol. 95, pp. 140–148, Jun. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18325299

[26] S. Karakatič and V. Podgorelec, "A survey of genetic algorithms for solving multi depot vehicle routing problem," *Appl. Soft Comput.*, vol. 27, pp. 519–532, Feb. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494614005572

[27] S. Trigui, O. Cheikhrouhou, A. Koubaa, U. Baroudi, and H. Youssef, "FL-MTSP: A fuzzy logic approach to solve the multi-objective multiple traveling salesman problem for multi-robot systems," *Soft Comput.*, vol. 21, no. 24, pp. 7351–7362, Dec. 2017.

[28] S. Ma, W. Guo, R. Song, and Y. Liu, "Unsupervised learning based coordinated multi-task allocation for unmanned surface vehicles," *Neurocomputing*, vol. 420, pp. 227–245, Jan. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220314399

[29] S. Wang et al., "Cooperative task allocation for multi-robot systems based on multi-objective ant colony system," *IEEE Access*, vol. 10, pp. 56375–56387, 2022.

[30] X. Chen, P. Zhang, G. Du, and F. Li, "Ant colony optimization based memetic algorithm to solve bi-objective multiple traveling salesmen problem for multi-robot systems," *IEEE Access*, vol. 6, pp. 21745–21757, 2018.

[31] M. Padberg and G. Rinaldi, "Optimization of a 532-city symmetric traveling salesman problem by branch and cut," *Oper. Res. Lett.*, vol. 6, no. 1, pp. 1–7, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0167637787900022

[32] C. Malandraki and R. B. Dial, "A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem," *Eur. J. Oper. Res.*, vol. 90, no. 1, pp. 45–55, Apr. 1996. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377221794002991

[33] B. Angéniol, G. de La Croix Vaubois, and J.-Y. Le Texier, "Self-organizing feature maps and the travelling salesman problem," *Neural Netw.*, vol. 1, no. 4, pp. 289–293, Jan. 1988. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608088900020

[34] Y. Xu, M. Fang, L. Chen, G. Xu, Y. Du, and C. Zhang, "Reinforcement learning with multiple relational attention for solving vehicle routing problems," *IEEE Trans. Cybern.*, vol. 52, no. 10, pp. 11107–11120, Oct. 2022.

[35] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2017, *arXiv:1611.09940*.

[36] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton, NJ, USA: Princeton Univ. Press, 2007, doi: 10.1515/9781400841103.

[37] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Comput.*, vol. 3, no. 1, pp. 79–87, 1991.

[38] S. R. Waterhouse, "Classification and regression using mixtures of experts," Ph.D. dissertation, Dept. Eng., Univ. Cambridge, Cambridge, U.K., 1998.

[39] D. J. Miller and H. Uyar, "A mixture of experts classifier with learning based on both labelled and unlabelled data," in *Advances in Neural Information Processing Systems*, vol. 9, M. Mozer, M. Jordan, and T. Petsche, Eds. Cambridge, MA, USA: MIT Press, 1996. [Online]. Available: https://proceedings.neurips.cc/paper/1996/file/a58149d355f02887dfbe55ebb2b64ba3-Paper.pdf

[40] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–25. [Online]. Available: https://openreview.net/forum?id=ByxBFsRqYm

[41] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. 31st Int. Conf. Mach. Learn.*, vol. 32, no. 1, E. P. Xing and T. Jebara, Eds. Bejing, China: PMLR, Jun. 2014, pp. 387–395. [Online]. Available: https://proceedings.mlr.press/v32/silver14.html

[42] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *J. Mach. Learn. Res.*, vol. 5, no. 9, pp. 1–60, 2004.

[43] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," 2019, *arXiv:1907.00481*.

[44] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[46] M. E. O'Neill, "PCG: A family of simple fast space-efficient statistically good algorithms for random number generation," Harvey Mudd College, Claremont, CA, Tech. Rep. HMC-CS-2014-0905, Sep. 2014. [Online]. Available: https://www.cs.hmc.edu/tr/hmc-cs-2014-0905.pdf

[47] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020, doi: 10.1038/s41586-020-2649-2.

[48] NSTA. (2018). *Offshore Oil and Gas Activity*. Accessed: Mar. 6, 2023. [Online]. Available: https://www.nstauthority.co.uk/data-centre/

[49] J.-Y. Potvin, G. Lapalme, and J.-M. Rousseau, "A generalized K-Opt exchange procedure for the MTSP," *INFOR, Inf. Syst. Oper. Res.*, vol. 27, no. 4, pp. 474–481, Nov. 1989.

[50] G. Nagy and S. Salhi, "Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries," *Eur. J. Oper. Res.*, vol. 162, no. 1, pp. 126–141, Apr. 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221703008361

[51] B. M. Baker and M. A. Ayechew, "A genetic algorithm for the vehicle routing problem," *Comput. Oper. Res.*, vol. 30, no. 5, pp. 787–800, Apr. 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054802000515

[52] H. Zhou, M. Song, and W. Pedrycz, "A comparative study of improved GA and PSO in solving multiple traveling salesmen problem," *Appl. Soft Comput.*, vol. 64, pp. 564–580, Mar. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494617307561

[53] F. Marini and B. Walczak, "Particle swarm optimization (PSO). A tutorial," *Chemometric Intell. Lab. Syst.*, vol. 149, pp. 153–165, Dec. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169743915002117

[54] K. Karabulut, H. Öztop, L. Kandiller, and M. F. Tasgetiren, "Modeling and optimization of multiple traveling salesmen problems: An evolution strategy approach," *Comput. Oper. Res.*, vol. 129, May 2021, Art. no. 105192. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054820303099

[55] P. Venkatesh and A. Singh, "Two metaheuristic approaches for the multiple traveling salesperson problem," *Appl. Soft Comput.*, vol. 26, pp. 74–89, Jan. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494614004827

[56] C. Wei, Z. Ji, and B. Cai, "Particle swarm optimization for cooperative multi-robot task allocation: A multi-objective approach," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 2530–2537, Apr. 2020.

[57] X.-X. Liu, D. Liu, Q. Yang, X.-F. Liu, W.-J. Yu, and J. Zhang, "Comparative analysis of five local search operators on visiting constrained multiple traveling salesmen problem," in *Proc. IEEE Symp. Series Computational Intell. (SSCI)*, Dec. 2021, pp. 1–8.

[58] B. Y. Yilmaz and S. N. Denizer, "Multi UAV based traffic control in smart cities," in *Proc. 11th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, Jul. 2020, pp. 1–7.

[59] A. Damle, V. Minden, and L. Ying, "Simple, direct and efficient multi-way spectral clustering," *Inf. Inference, A J. IMA*, vol. 8, no. 1, pp. 181–203, Mar. 2019, doi: 10.1093/imaiai/iay008.

[60] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Sci. Robot.*, vol. 7, no. 66, May 2022, Art. no. eabm6074. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[61] IEEE Standards Association et al., *IEEE Standard for Ethernet*, IEEE Standard IEEE 802.3-2018, 2018.

[62] 802.11 WG–Wireless LAN Working Group, *IEEE Standard for Information Technology–Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks–Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Standard Std 802.11-2020, 2021, pp. 1–4379.

[63] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The Marathon 2: A navigation system," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 2718–2725.

[64] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo localization for mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 2, 1999, pp. 1322–1328.

[65] C. Kwok, D. Fox, and M. Meila, "Adaptive real-time particle filters for robot localization," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 2, 2003, pp. 2836–2841.

[66] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, Mar. 1997.

Song Ma (Student Member, IEEE) received the M.Sc. degree in mechanical engineering from University College London in 2019, where he is currently pursuing the Ph.D. degree in robotics with the Department of Mechanical Engineering. His main research interests pertaining to the autonomous system are in multi-agent planning and robotic exploration.

Jingqing Ruan is currently pursuing the Ph.D. degree in pattern recognition and intelligent systems with the Institute of Automation, Chinese Academy of Sciences. Her research focuses on multi-agent reinforcement learning, multi-agent planning on game AI, and traffic light control with multi-agent coordination. Her research output has been published in AAMAS and ICAPS.

Yali Du is currently a Lecturer with the King's College London. Her research interests include enable machines to exhibit cooperative and trusted behavior in intelligent decision-making tasks, reinforcement learning, and multi-agent systems and applications in game AI and data science. Her research output has been widely published in prestigious venues, including *AI* journal, ICML, NeurIPS, ICLR, and AAMAS.

Richard Bucknall (Member, IEEE) received the B.Eng. degree (Hons.) from the University of Plymouth, Plymouth, U.K., in 1988, and the Ph.D. degree from the Royal Naval Engineering College, Devon, U.K., in 1995. He is currently a Professor in marine systems engineering and the Head of the Department of Mechanical Engineering, University College London (UCL), London, U.K. Having gained experience working in both the shipping and rail industries as a Practicing Engineer across the world, he joined UCL as a Senior Research Associate in 1995 to follow an academic career. He has obtained research funding in excess of 10M, supervised more than 25 Ph.D. students, and authored or coauthored more than 250 articles in engineering science journals, conferences, and press. His research interests include electrical power systems, marine propulsion, and low-carbon technology.

Yuanchang Liu (Member, IEEE) received the M.Sc. degree in power systems engineering and the Ph.D. degree in marine control engineering from University College London, London, U.K., in 2011 and 2016, respectively. He is currently an Associate Professor with the Department of Mechanical Engineering, University College London, and the Programme Director of M.Sc. Power Systems Engineering. He has authored/coauthored over 70 peer-reviewed articles, including 60 publications in high-impact journals. His research interests include automation and autonomy, mainly focusing on the exploration of technologies for sensing and perception and guidance and control of intelligent and autonomous vehicles.