An Online Planning Framework for Multi-Robot Systems with LTL Specification

Rohit Singh
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur, India
kun@cse.iitk.ac.in

Indranil Saha
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur, India
isaha@cse.iitk.ac.in

***Abstract*—We present a framework for deploying a multi-robot system in a dynamic environment where the robots have to react to external events. The specification for the system is given in a sub-class of Linear Temporal Logic (LTL), a widely used logical language for robot motion planning. The LTL specifications capture how the robots should react to different environmental events. We provide a framework for managing the robots through persistent sensing, planning, and monitoring their execution. We formally prove that, under certain assumptions, our framework enables the robots to always satisfy the LTL specifications. Furthermore, we evaluate our technique on two complex use cases using a multi-robot system involving unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs) — one on persistent surveillance of critical infrastructure and the other on production management in a factory. Experimental results establish that our technique is scalable and has the potential to be applicable to diverse applications of multi-robot systems in challenging dynamic environments.**

***Index Terms*—multi-robot systems, formal specification, LTL, online planning**

I. Introduction

Multi-robot systems have the potential to find applications in various domains, such as disaster response, security and surveillance, object transportation, and mapping, to name a few. There has been a plethora of research for designing reliable multi-robot systems. However, for the multi-robot systems to be applicable to more complex missions, it is imperative to develop frameworks that deal with more complex specifications involving various spatial and temporal constraints.

Recently, temporal logic has been popular in specifying requirements for multi-robot systems. Given a specification in temporal logic, several papers address the problem of finding a plan for the multi-robot system offline [1]–[6]. A significant limitation of this approach is that the robots cannot cope with the change in environment and respond to incoming tasks.

To enable a multi-robot system to react to the dynamism in the environment effectively, a major approach adopted by many researchers is *reactive synthesis* [7]–[11]. In reactive synthesis, the specification for the robots is captured in a special subclass GR(1) of Linear Temporal Logic. Given a specification in this restricted class of LTL, the synthesis algorithm aims to find a reactive controller that enables the robots to perform appropriate actions in their current state to respond to environmental events. The reactive synthesis promises to provide a correct-by-construction high-level controller for a robot to satisfy the task specification, provided the environment satisfies the predefined assumptions. However, such synthesis methodology suffers severely from the lack of scalability in terms of the complexity of the specification, the size of the workspace, and the number of robots.

We propose an alternative approach to address the problem of synthesizing plans for the robots in a multi-robot system to satisfy complex task specifications perpetually by responding to environmental events. We introduce a restricted subclass of LTL to capture the requirements for such a system formally. In this class of LTL, the environmental conditions to which the robots need to respond are captured as Boolean formulae, assuming that the system is equipped with sensors that provide binary inputs. The desired activities of the robots in response to various environmental events are captured as a general LTL formula. The specification of this form can capture a wide range of requirements for reactive multi-robot systems, as will be evident by case studies conducted in this paper.

We design a system to manage multiple robots and provide them with correct and optimal plans to respond to various environmental events according to the formally specified requirements. Our system is based on the *Sense-Plan-Act (SPA)* paradigm inspired by [12], where the system keeps track of the current state of the environment using a set of available sensors. Whenever an environmental change renders the current plan invalid, the system re-plans in light of the new information. A major challenge in implementing such a system with many robots is that the multi-robot planner generally takes a prohibitively large amount of time to solve the planning problem for all the robots when some environmental change appears. We provide a sound scheme for partitioning the robots into mutually exclusive groups based on the given specification, which ensures that the plan for a robot in one partition can be computed independently from any robot in any other group. This enables the system to solve the planning problem for each group separately, leading to scalability.

We prove that under certain assumptions on the behavior of the environment, the proposed system ensures the satisfaction of the LTL specification by the multi-robot system. We implement our framework on ROS and evaluate it in two case studies: persistent surveillance and warehouse management. Both case studies involve complex specifications and demonstrate the broad applicability of our method. Through a scalability experiment, we demonstrate that our method can work for large workspaces and many robots. We also evaluate our method through a real experiment with two ground robots and a quadcopter on a limited version of the persistent surveillance case study. This experiment establishes the practical feasibility of our proposed technique.

In summary, we make the following contributions:

- We introduce a restricted subclass of LTL to capture the specifications for multi-robot systems deployed to perform specific tasks in a dynamic environment where a change in the state of the environment leads to new tasks that need to be completed by the robots.
- We present a framework that, given an LTL specification of the above-mentioned form, enables a multi-robot system to move and perform various tasks so that the LTL specification is satisfied.
- We provide a ROS [13] implementation of our framework and apply it to two real-life case studies. Moreover, we evaluate our method through a real robotic experiment, establishing its practical feasibility.

II. Problem

A. Workspace, Robots and Environment

1) Workspace: We assume that n robots operate in a 3-D discrete workspace $\mathcal{W}$, which we represent as a grid map. The grid divides the workspace into cube-shaped cells. Each grid cell is referenced by its (x, y, z) coordinates. We ignore the z coordinate for the ground robots. Obstacles may occupy some of the cells in the workspace, and the robots cannot visit such cells. We denote the set of obstacles using $\mathcal{O}$.

2) Motion Model for Robots: We capture the motion of a robot i using a set of actions Act^i. The robot changes its state in the workspace by performing the actions from Act^i. An action $act \in Act^i$ is associated with a `cost`, which captures the energy consumption or time delay (based on the need) to execute it. A robot can move to satisfy a given specification by executing a sequence of actions in Act^i.

Formally, we model the motion of the i-th robot in the workspace $\mathcal{W}$ as a *weighted transition system* defined as $T^i = (S^i, s^i_0, E^i, \Pi^i, L^i, w^i)$. Here (i) S^i is the set of states denoting obstacle-free cells in $\mathcal{W}$, (ii) $s^i_0 \in S^i$ is the initial state of the robot i, (iii) $E^i \subseteq S^i \times S^i$ is the transitions/edges allowed to be taken by robot i, $(s^i_1, s^i_2) \in E^i$ iff $s^i_1, s^i_2 \in S^i$ and $s^i_1 \xrightarrow{act} s^i_2$, where $act \in Act^i$, (iv) Π^i is the set of atomic propositions defined for robot i, (v) $L^i : S^i \longrightarrow 2^{\Pi^i}$ is a map that provides the set of atomic propositions `true` in state $s^i \in S^i$, (vi) $w^i : E^i \longrightarrow \mathbb{N}_{>0}$ is a weight function capturing the cost of the action on an edge $e^i \in E^i$.

A *joint transition system* T is a transition system that captures the collective motion of a team of m robots in a workspace $\mathcal{W}$, where the i-th robot executes one action from the set of actions Act^i available to it. We define a joint transition system as $T := (S_T, s_0, E_T, \Pi_T, L_T, w_T)$, where (i) S_T is the set of vertices/states in a joint transition system, where each vertex is of the form $\langle s^1, s^2, \dots, s^m \rangle$, s^i represents the state of robot i in transition system T^i, (ii) $s_0 := \langle s^1_0, s^2_0, \dots, s^m_0 \rangle \in S_T$ is the joint initial state of the team of m robots, (iii) $E_T \subseteq S_T \times S_T$ is the set of edges, $(s_1, s_2) \in E_T$ iff $s_1, s_2 \in S_T$ and $(s^i_1, s^i_2) \in E^i$ for all $i \in \{1, 2, \dots, m\}$, (iv) $\Pi_T := \bigcup_{i=1}^{m} \Pi^i$ is the set of atomic propositions, (v) $L_T : S_T \longrightarrow 2^{\Pi_T}$, and $L_T(s_j) := \bigcup_{i=1}^{m} L^i(s^i_j)$ gives us the set of propositions `true` at state s_j, and (vi) $w_T : E^T \longrightarrow \mathbb{N}_{>0}$, and $w_T(s_j, s_k) := \sum_{i=1}^{m} w^i(s^i_j, s^i_k)$ is a weight function. We can also consider the transition system as a weighted, directed graph with vertices, edges, and a weight function. Whenever we use a graph algorithm over a transition system, we mean to apply it over its equivalent graph.

We define robots' trajectories ξ^r as a discrete sequence of multi-robot states $\xi^r_1 \xi^r_2 \dots$, where ξ^r_t denotes the combined state $\langle s^1_t, s^2_t, \dots, s^m_t \rangle$ of all the robots at the t-th time step. Here s^i_t represents the state of the i-th robot at the t-th step.

3) Environment Model: A robot interacts with its environment using various sensors, which are sampled periodically. We place the following high-level assumptions on the possible behaviour of the sensor variables.

- There exists a sensor data processing system that processes the analog and noisy sensor signals and generates correct values for the binary sensor variables representing the accurate state of the environment.
- Once a sensor variable is changed, it will stay so until the robots perform some actions in response to the change in the sensor value. For instance, if the battery of a robot is down, it will continue to stay so until the corresponding robot moves to a charging location to charge its battery.

Our goal is to construct a planning and control mechanism that enables the multi-robot system to satisfy the given specifications for all possible admissible environments satisfying these assumptions. The l binary sensor variables $K = \{\kappa^1, \kappa^2, \dots, \kappa^l\}$ have their own dynamics. We define environment trajectory ξ^e as a discrete sequence of environment states $\xi^e_1 \xi^e_2 \dots$ generated through periodic sampling of the sensors, where $\xi^e_i = \langle \kappa^1_i, \kappa^2_i, \dots, \kappa^l_i \rangle$. Here κ^j_i denotes the value obtained from the j-th sensor at the i-th time step. A change in environment is observed at time step t, if $\xi^e_{t-1} \neq \xi^e_t$.

Example II.1. *Given two sensor variables* $K = \{b, h\}$*, with initial state* $\langle b = \texttt{true}, h = \texttt{false} \rangle$*, a possible environment trajectory* $\xi^e = \langle \texttt{true}, \texttt{false} \rangle \langle \texttt{true}, \texttt{false} \rangle \langle \texttt{true}, \texttt{false} \rangle \langle \texttt{true}, \texttt{true} \rangle \langle \texttt{true}, \texttt{true} \rangle \dots$*, where the sensor variable* h *changes to* `true` *at the* 4^{th} *time step, but the sensor variable* b *never changes.*

B. Specification Language

1) Linear Temporal Logic: We capture the specification of the multi-robot system using *Linear Temporal Logic* (LTL). LTL formulae over the set of atomic propositions Π_T are formed according to the following grammar [14]:

$$\Phi ::= \texttt{true} \mid a \mid \phi_1 \vee \phi_2 \mid \neg \phi \mid \bigcirc \phi \mid \phi_1 U \phi_2.$$

The basic ingredients of an LTL formula are the atomic propositions a $\in \Pi_T$, the Boolean connectors ($\neg$ (Negation) and $\vee$ (Disjunction), and two temporal operators $\bigcirc$ (Next) and U (Until). The Boolean connectors have their usual meaning. Given negation ($\neg$) and disjunction ($\vee$) operators, we can derive the conjunction ($\wedge$) and implication ($\Rightarrow$) operators.

The semantics of an LTL formula is defined over an infinite sequence of sets of atomic propositions called *trace*. The trace σ satisfies a formula ϕ, if the first state of σ satisfies ϕ. The Next operator $\bigcirc$ is a unary operator and is followed by a formula, which has to be satisfied in the next time step. The Until operator U is a binary operator between two formulas. The formula $\phi_1 \, U \, \phi_2$ says that ϕ_2 should be observed at some step t, and for all steps t', $0 \leq t' < t$, ϕ_1 must be observed. There are two other widely-used temporal operators, namely $\Diamond$ (Eventually) and $\Box$ (Always), which can be derived from the basic logical and temporal operators as follows: $\Diamond\phi := T \, U \, \phi$, and $\Box\phi := \neg\Diamond\neg\phi$. Here, $\Diamond\phi$ says that ϕ has to be observed at some time step eventually, and $\Box\phi$ says that ϕ must be observed at all the steps, i.e., it is not the case that $\neg\phi$ will be observed eventually. There is yet another operator, namely W (Weak Until), which is defined as: $\phi_1 \, W \, \phi_2 := (\phi_1 \, U \, \phi_2) \vee \Box\phi_1$. As opposed to U (Until), $\phi_1 \, W \, \phi_2$ does not require that ϕ_2 eventually becomes `true`.

A robot trajectory σ satisfying an LTL specification Φ can be represented as $\sigma = \sigma_{pref}.(\sigma_{suff})^{\omega}$, where σ_{pref} is a prefix path starting from the initial location of the robot and is traversed once to reach the suffix cycle σ_{suff}, which is then traversed repetitively (See [14] for more details).

C. Problem Statement

To capture the specification of a multi-robot system dealing with a dynamic environment, we introduce a sub-class of LTL, which is presented below.

Proposed Specification for Reactive System. We capture the specification Φ of the system as the conjunction of several sub-specifications Φ_i as follows:

$$\Phi \equiv \bigwedge_{i=1}^{n} \Phi_i. \tag{II.1}$$

Each sub-specification Φ_i is of the following form:

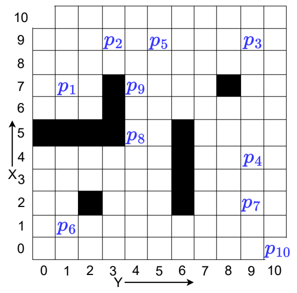
$$\Phi_i \equiv \Box(\psi_i^e \Rightarrow \Diamond(\phi_i^r \, W \, \neg\psi_i^e)). \tag{II.2}$$

Here, ψ_i^e is a Boolean condition on the environment variables K, and ϕ_i^r is an LTL formula on the propositions defined on the states of the robots S_T in the transition system.

The formal specification intuitively says that whenever a Boolean condition ψ_i^e on the environmental variables becomes `true`, eventually the system will reach a state from where, in each subsequent state, the system will satisfy the LTL formula ϕ_i^r till the Boolean condition ψ_i^e becomes `false`. Moreover, it is not mandatory that ψ_i^e will eventually become `false`. In that case, the robots need to keep satisfying ϕ_i^r all the time in the future.

Now, we formally define the problem. Given environment trajectory ξ^e and robots trajectory ξ^r, we define the system trajectory ξ as $\xi_1\xi_2\ldots$, where ξ_i denotes the tuple $\langle \xi_i^e, \xi_i^r \rangle$ at the i-th time step.

(a) Real workspace



(b) Grid representation

Fig. 1: Workspace of size 11×11 for a surveillance application.

Problem 1. *Consider a multi-robot system with m robots. The environment of this multi-robot system is sensed using l sensors. The specification of the multi-robot system Φ is of the form given in equation II.1. Generate trajectories ξ^r for the robots in response to the environment trajectory ξ^e such that the resulting system trajectory ξ satisfies the formula Φ.*

D. Illustrative Example

Throughout this paper, we use a surveillance example for illustration purposes. Figure 1a shows the real workspace, and Figure 1b shows its grid representation $\mathcal{W}$. In Figure 1b, the cells in the black colour represent obstacles ($\mathcal{O}$). There are 3 robots: r_1, r_2, and r_3. Robots r_1 and r_2 are unmanned ground vehicles (UGVs) that can move to one of the four neighbouring cells with cost 1 and stay in their current cell with cost 0. The robot r_3 is an unmanned aerial vehicle (UAV) that can move to one of its eight neighbouring cells or stay in its current cell with cost 1. The robots start at locations p_8, p_9 and p_{10}, respectively.

Here $\Pi^1 = \{p_1, p_2, p_3, p_4, p_8, \mathit{follow}\}$, $\Pi^2 = \{p_1, p_2, p_3, p_4, p_9, \mathit{follow}\}$, and $\Pi^3 = \{p_5, p_6, p_7, p_{10}\}$. The propositions $p_1, p_2, \ldots, p_7$ represent patrolling points, whereas p_8, p_9, p_{10} the charging locations. These propositions are satisfied if the corresponding robot visits the location marked in Figure 1b. The proposition *follow* is satisfied when the robot visits a neighbouring cell of the cell where the intruder was detected.

We use four sensor variables $K = \langle b_1, b_2, b_3, \mathit{intrusion}\rangle$. Sensor variable $\mathit{intrusion}$ refers to detecting a human in the workspace. The UAV is equipped with camera sensors to detect any intruder (human) in the workspace (the intruder is detected along with its coordinates). For simplicity, we assume that there cannot be more than one intruder (if any) in the workspace at any time instant, and detection of a human by the UAV will set the sensor variable $\mathit{intrusion}$ to `true`. The absence of boundaries at points $(0, 10)$ and $(10, 0)$ in Figure 1b serves as entry/exit points for an intruder. Sensor variable b_i refers to the battery status of robot r_i. Initially, it is set to `true`. It is set to `false` when the corresponding robot's battery voltage drops below a lower threshold voltage, meaning that the battery is about to discharge and requires charging. These sensor variables will be set to `true` when the

TABLE I: Queries for Example II-D

i	ψ_i^e	ϕ_i^r
1	$((b_1 \wedge b_2) \vee ((b_1 \vee b_2) \wedge \neg intrusion))$	$(\lozenge p_1 \wedge \lozenge p_2 \wedge \lozenge p_3 \wedge \lozenge p_4)$
2	b_3	$(\lozenge p_5 \wedge \lozenge p_6 \wedge \lozenge p_7)$
3	$((b_1 \vee b_2) \wedge intrusion)$	$\lozenge(follow)$
4	$\neg b_1$	$\lozenge p_8$
5	$\neg b_2$	$\lozenge p_9$
6	$\neg b_3$	$\lozenge p_{10}$

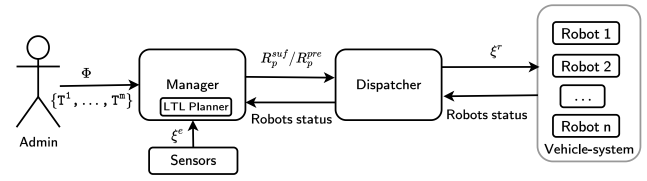


Fig. 2: The proposed online planning framework for multi-robot systems with LTL specifications

corresponding robot's battery reaches above an upper threshold voltage, meaning the battery is fully charged.

We define six queries/tasks of the form given in equation II.1 for our example problem in Table I. In natural language, the specification states that robots r_1 and r_2, whenever available, should continuously patrol the locations p_1, p_2, p_3, and p_4 (Specification Φ_1). The robot r_3, which is a UAV, should patrol the locations p_5, p_6, and p_7 whenever it has sufficient battery charge (Specification Φ_2). While patrolling the locations p_5, p_6, p_7, the UAV looks for an intruder and sets the sensor variable $intrusion$ to `true` whenever it detects an intruder. When that happens and one of the ground robots is available with sufficient battery charge, it pursues the intruder by satisfying the $follow$ proposition eventually (specification Φ_3). When the battery of a robot is low, the robot should visit its charging location and stay there until the battery is fully charged (specification Φ_4, Φ_5, and Φ_6). We assume that once a robot visits its respective charging location, there is some mechanism to charge its battery.

III. Framework

We aim to solve a planning problem in which the multi-robot system must react to environmental changes. As a result, offline planning is not applicable because the plan may become invalid with any environmental changes. Because of this, we employ a planning technique based on the *Sense-Plan-Act (SPA)* paradigm inspired by [12], where we generate plans for an initial environment state and distribute the plan to the appropriate robots incrementally for execution. Every time an environmental change renders our plan invalid, we amend or re-plan in light of the new information. In what follows, we propose a framework that, following the principle mentioned above, generates continuous robot trajectories as per the system's goals given in Φ.

Figure 2 outlines the framework's basic structure and data flow among various modules. The framework consists of five different modules. The admin is a human being who configures the manager module with the number of robots, the transition systems T^i for each robot, and a set of queries Φ_i (i.e., goals to be achieved by the system) each of the form in equation II.2. The manager module decides the high-level directives to satisfy the system's operational goals as per the specification Φ and sends the same to the dispatcher module. The objective of the module is to generate plans for the robots (by employing a planner) satisfying Φ that are consistent with the changes in the environment. We will deliberate on the functionalities of the manager module in Section III-A. The dispatcher module receives a high-level plan from the manager module and manages its execution on the robots. It refines the high-level plan into low-level *collision-free* trajectories and incrementally sends the refined trajectories to the vehicle-system module to be executed by the robots.

The robotic platforms are mounted with various sensors and actuators for sensing and interacting with the environment. The vehicle-system module provides the manager and dispatcher modules with the required information about the robots and the environment. It also performs the atomic tasks received from the dispatcher module by generating the required low-level command using a feedback controller. The actuators connected to the robots directly receive the executable commands.

The sensors module receives raw data from different sensors, processes the data, and provides the manager module with updated values for the environment sensor variables K. The sensors may be installed in the robots or may be placed in any location of interest in the workspace.

A. Manager

The manager module uses a static Multi-Agent LTL Planner to plan for a reduced version of Φ (let us call it Φ^{redc}) based on the current environmental condition. It then sends the plan to the dispatcher module for execution on the robots, which results in robot trajectories (ξ^r). The robots execute their respective plan until Φ^{redc} becomes invalid, i.e., the environment condition changes. On detecting any such change, the manager recomputes Φ^{redc} accordingly and re-plan for it.

We present the operation of the manager module in Algorithm 1 and explain the concepts of major steps in detail. Given the transition system of m robots $\{T^1, \ldots, T^m\}$ and an LTL specification Φ of the form in Equation II.2, the goal is to generate continuous trajectories ξ^r for the robots in response to the environment trace ξ^e such that the resulting system trace ξ satisfies the formula Φ.

We first group the m queries of Φ into v groups (line 1) using the procedure described in section III-A1. The goal of this step is to partition the robots into different groups so that the behavior of a robot in one group is independent of the behavior of a robot in any other group. For each group ω_p, we maintain a Boolean vector $status_p$ of size $|\omega_p|$ and two Boolean variables $ready_p$, $replan_p$ as group variables. These variables help us keep track of each query $\omega_{p,k}$, the k-th specification in the group ω_p, and thereby help us in generating and sending the required plans for the robots to the dispatcher module. In $status_p[k]$, we store the last known

Algorithm 1: manager

```
Input: Transition systems for $m$ robots
    $\{T^1, \dots, T^m\}$ and the LTL specification $\Phi$.
Output: Continuous trajectories $\xi^r$ for the robots.
1  $\Omega = \{\omega_1, \dots, \omega_v\} \leftarrow$ creategroup($\Phi$)
2  initialize_flags ($\Omega$)
3  parallel for $1 \leq p \leq |\Omega|$ do
4      while true do
5          update_flags($\omega_p$)
6          if $ready_p$ then
7              if $replan_p$ then
8                  $\Phi_p^{redc} \leftarrow$
                     compute_spec_for_planning($\omega_p$,
                     $status_p$)
9                  $\langle \sigma_p^{pref}, \sigma_p^{suf} \rangle \leftarrow$
                     Planner(($T^1, \dots, T^q$), $\Phi_p^{redc}$)
10                 $replan_p \leftarrow$ false
11                 dispatcher($\sigma_p^{pref}$)
12             dispatcher($\sigma_p^{suf}$)
13             $ready_p \leftarrow$ false
14 procedure initialize_flags($\Omega$)
15     for $1 \leq p \leq |\Omega|$ do
16         for $k \leftarrow 1$ to $|\omega_p|$ do
17             $status_p[k] \leftarrow$ false
18         $ready_p \leftarrow$ false
19         $replan_p \leftarrow$ false
20 procedure update_flags($\omega_p$)
21     read_sensor_values()
22     for $k \leftarrow 1$ to $|\omega_p|$ do
23         if $\neg\omega_{p,k}^e$ then
24             if $status_p[k]$ then
25                 $status_p[k] \leftarrow$ false, $replan_p \leftarrow$ true
26         else if $\neg status_p[k]$ then
27             $status_p[k] \leftarrow$ true, $replan_p \leftarrow$ true
28     if dispatcher.$free(\omega_p)$ then
29         $ready_p \leftarrow$ true
30 procedure compute_spec_for_planning($\omega_p$,
     $status_p$)
31     $\Phi_p^{redc} \leftarrow$ true
32     for $k \leftarrow 1$ to $|\omega_p|$ do
33         if $status_p[k]$ then
34             $\Phi_p^{redc} \leftarrow \Phi_p^{redc} \wedge \Box\omega_{p,k}^r$
35     return($\Phi_p^{redc}$)
```

information about the validity of $\omega_{p,k}^e$, which denotes the Boolean environment condition associated with specification $\omega_{p,k}$. The information about the need for replanning for group ω_p is stored in $replan_p$, whereas $ready_p$ denotes that the dispatcher is waiting for any new plans from the manager module for the robots in group ω_p.

At the beginning of the operation, the group variables are set to false, as the manager currently has no information about the environment by calling the function initialize_flags() at line number 2. Next, it runs an infinite loop for each group ω_p in parallel (lines 4-13), which generates continuous trajectories ξ^r for the robots. The first task in the loop is to update the group variables with the latest information about the environment by calling the function update_flags() at line number 5. In the function, for each query $\omega_{p,k}$, we check if the current value of $\omega_{p,k}^e$ and $status_p[k]$ are the same. If there is any change, we set $status_p[k]$ to $\omega_{p,k}^e$, and also set $replan_p$ to true. We also set the variable $ready_p$ to true if the robots belonging to queries in ω_p are ready and waiting for new trajectories to execute. The function is continuously called until all the robots belonging to queries in ω_p are ready, i.e., variable $ready_p$ is set to true. Now, if the earlier generated plan is valid ($replan_p$ is false), the manager sends the previous plan to the dispatcher module and waits for the robots to execute their task (by setting $ready_p$ to false), and repeat the steps in the loop. If the earlier generated plan becomes invalid (variable $replan_p$ is true), the manager performs the steps in line numbers 7-11. For re-planning, it first generates Φ_p^{redc} using the procedure described in section III-A2. The LTL formula Φ_p^{redc} captures the specification for which the planner needs to generate a plan. Next, the manager invokes a multi-robot planner with the transition systems for the robots and Φ_p^{redc} as the LTL specification. After the plan generation, the variable $replan_p$ is set to false, and the plan's prefix part (σ_p^{pre}) is sent to the dispatcher module for execution on robots. Next, the manager sends the suffix part (σ_p^{suf}) of the plan to the dispatcher module and waits for the robots to execute their trajectories by setting variable $ready_p$ to false (line number 13) while continuously evaluating the group variables. This way, by updating generated trajectories for the robots as per the environment trace, the manager aids in generating continuous trajectories ξ^r for the robots in response to the environment trace ξ^e such that the resulting system trace ξ^s satisfies the formula Φ.

1) Grouping of queries: The formula Φ is a conjunction of n queries. We denote the set of these n queries by $\Omega = \{\Phi_1, \dots, \Phi_n\}$. The queries assign tasks to the robots. A query may assign tasks to a single robot or a set of robots. Two or more queries may assign different tasks to the same set of robots. We combine these queries into the same group to generate the correct plan for the multi-robot system. The LTL formulae Φ_i and Φ_j belong to the same group if there exists at least one robot that is responsible for the satisfaction of both formulae. To formalize this notion, let us denote by Σ_i the set of robots that can contribute to the satisfaction of the specification Φ_i. Furthermore, we denote by Λ_i the set of atomic propositions involved in Φ_i. Note that only the robots in Σ_i are responsible for satisfying the propositions in Φ_i. Mathematically,

$$\Sigma_i = \{r^k \mid \exists \pi_i \in \Pi^k \text{ such that } \pi_i \in \Lambda_i\}.$$

Now, to divide the LTL specifications into groups, we form a directed graph G by considering each Σ_i as a node in a graph. A directed edge between two distinct nodes Σ_i and Σ_j is added to the graph if $\Sigma_i \cap \Sigma_j \neq \emptyset$. Subsequently, we find the strongly connected components in G. For every connected component, we create a group including each Φ_i for which Σ_i is a node in the corresponding component. The set of all these specification groups is denoted by Ω, where $1 \leq |\Omega| \leq n$, and the p-th group, $1 \leq p \leq |\Omega|$, is denoted by ω_p, where $|\omega_p| \geq 1$. We denote the k-th query of the p-th group by

$$\omega_{p,k} \equiv \Box(\omega^e_{p,k} \Rightarrow \Diamond(\omega^r_{p,k} W \neg\omega^e_{p,k})).$$

So a query Φ_i is now represented as $\omega_{p,k}$, where ψ^e_i is represented as $\omega^e_{p,k}$ and ϕ^r_i as $\omega^r_{p,k}$. The groups are exhaustive and pairwise mutually exclusive, i.e.,

$$\bigcup_{p=1}^{|\Omega|} \omega_p = \Omega,$$

$$\forall p, q.\ 1 \leq p, q \leq |\Omega| \ \wedge\ p \neq q : \ \omega_p \cap \omega_q = \emptyset.$$

Now, the formula Φ can be written as:

$$\Phi \equiv \bigwedge_{p=1}^{|\Omega|} \zeta_p, \text{ where } \zeta_p \equiv \bigwedge_{k=1}^{|\omega_p|} \omega_{p,k}.$$

In this way, the specification Φ, which is a conjunction of n sub-specifications Φ_i, gets distributed into $|\Omega|$ groups. Thus, to satisfy Φ, each ζ_p must be satisfied by the robots.

Example III.1. *For our example problem in II-D, we get $\Sigma_1 = \{r_1, r_2\}$, $\Sigma_2 = \{r_3\}$, $\Sigma_3 = \{r_1, r_2\}$, $\Sigma_4 = \{r_1\}$, $\Sigma_5 = \{r_2\}$ and $\Sigma_6 = \{r_3\}$. Thus, two groups $\omega_1 = \{\Phi_1, \Phi_3, \Phi_4, \Phi_5\}$ and $\omega_2 = \{\Phi_2, \Phi_6\}$ are formed using the procedure described above.*

2) Generation of Φ^{redc}_p: Each ζ_p is a conjunction of $|\omega_p|$ sub-specifications. Thus, for ζ_p to hold, each $\omega_{p,k}$ must be satisfied by the robots. Now, since sub-specification $\omega_{p,k}$ is of the form in Equation II.2, it holds at every time instant if $\omega^e_{p,k} = \texttt{false}$. Otherwise, if at any time instant $\omega^e_{p,k} = \texttt{true}$, then $\Diamond(\omega^r_{p,k} W \neg\omega^e_{p,k})$ should hold by the trace starting at that time instant. Now, on expanding $\Diamond(\omega^r_{p,k} W \neg\omega^e_{p,k})$ we get $\Diamond((\omega^r_{p,k}\ U\ \neg\ \omega^e_{p,k})\ \vee\ \Box\ \omega^r_{p,k})$, which says that eventually either $(\omega^r_{p,k}\ U\ \neg\ \omega^e_{p,k})$ or $\Box\ \omega^r_{p,k}$ should hold true. The first expression is dependent on $\omega^e_{p,k}$, which consists of a Boolean expression over sensor variables K, on which the planner does not have any control. Thus, we consider the second expression $\Box\ \omega^r_{p,k}$ for planning. However, once the Boolean condition $\neg\ \omega^e_{p,k}$ holds $\texttt{true}$, the change in the flags $status_p[k]$ and $replan_p$ invalidate the current plan. The robot stops executing the plan generated for $\Box\ \omega^r_{p,k}$, without violating the specification $\omega_{p,k}$ as $(\omega^r_{p,k}\ U\ \neg\ \omega^e_{p,k})$ gets satisfied.

Let ω'_p represent the set of queries in group ω_p for which $\omega^e_{p,k}$ is $\texttt{true}$ at the current time instant. Thus,

$$\Phi^{redc}_p = \bigwedge_{k=1}^{|\omega'_p|} \Diamond\Box(\omega'^r_{p,k}).$$

The LTL specification Φ^{redc}_p represents the LTL query to be satisfied on the current environment instance for group ω_p. Here, we satisfy $\Diamond\Box\omega^r_{p,k}$ till $\neg\ \omega^e_{p,k}$ becomes $\texttt{true}$, then we update the formula accordingly.

Example III.2. *Revisiting the example in section II-D, Figure 3 depicts a possible environment trace ξ^e, which is $\xi^e_1 = \langle \texttt{T}, \texttt{T}, \texttt{T}, \texttt{F} \rangle \ldots \xi^e_{47} = \langle \texttt{T}, \texttt{T}, \texttt{T}, \texttt{T} \rangle \ldots \xi^e_{72} = \langle \texttt{T}, \texttt{T}, \texttt{T}, \texttt{F} \rangle \ldots$. For this trace, at time instant $1 \leq t < 47$, the reduced query for group ω_1 will be $\Phi^{redc}_1 = \Diamond\Box(\Diamond p_1 \wedge \Diamond p_2 \wedge \Diamond p_3 \wedge \Diamond p_4)$ and for group ω_2, $\Phi^{redc}_2 = \Diamond\Box(\Diamond p_5 \wedge \Diamond p_6 \wedge \Diamond p_7)$. And for time instant $47 \leq t < 72$, the reduced query for group ω_1 is updated to $\Phi^{redc}_1 = \Diamond\Box(\Diamond p_1 \wedge \Diamond p_2 \wedge \Diamond p_3 \wedge \Diamond p_4) \wedge \Diamond\Box(follow)$ whereas for group ω_2 the query remains unchanged.*

3) Multi-Robot Planner: The manager module requires a static multi-agent LTL path planner to generate plans for a subset of robots to satisfy Φ^{redc}_p. The standard procedure for solving this planning problem is based on the automata-theoretic model-checking approach. In this approach, the manager first identifies the subset of robots Σ that are involved in the satisfaction of the specifications in ω' as follows:

$$\Sigma = \{r^k \mid \exists \pi_i \in \Pi^k \text{ such that } \pi_i \in \Lambda\},$$

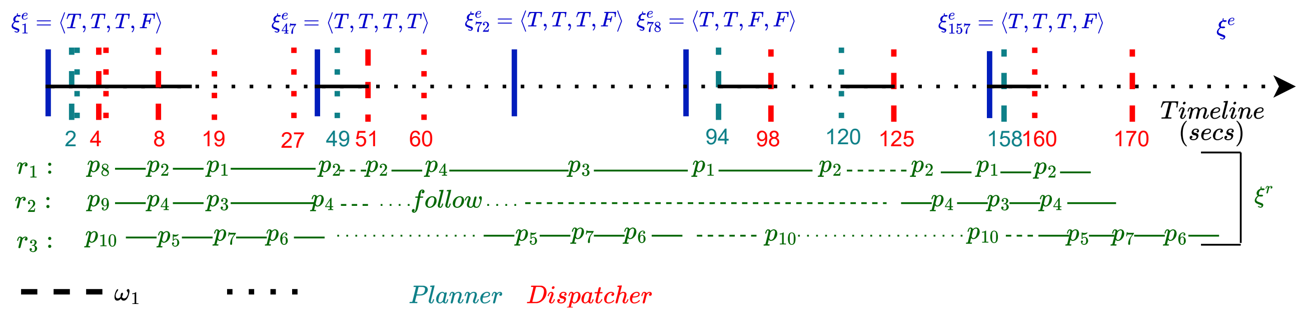
where Λ denotes the set of atomic propositions involved in Φ^{redc}_p. Subsequently, it creates the joint transition system T based on the definition introduced in Section II-A2 and converts Φ^{redc}_p to an equivalent Büchi Automaton B. Now, a product automaton P out of the joint transition system T and the Büchi Automaton B is computed. Please see Appendix A for the definitions of Büchi Automaton and Product Automaton and [14] for more details. The product automata P captures all possible ways the group of robots Σ can satisfy the LTL specification Φ^{redc}_p. Finally, the manager uses Dijkstra's shortest path algorithm on P to compute the required minimum cost suffix run having a valid prefix. Thus, the plan generated is of the form $R^{pref} \cdot (R^{suf})^\omega$. The manager module maps each state in R^{pre} and R^{suf}, which is a state in the product automaton P, to the corresponding states in the joint transition system T to generate robot trajectories σ^{pre} and σ^{suf}. The manager module sends $\sigma^{pre} \cdot \sigma^{suf}$ as the high-level plan to the dispatcher module. Subsequently, the suffix part σ^{suf} is repeatedly sent until a change is observed in the environment.

B. Theoretical Guarantee

The soundness guarantee provided by our framework is captured in the following theorem.

Theorem III.3. *Given the transition systems $\{T_1, \ldots, T_m\}$ of m robots and an LTL specification Φ of the form given in Equation II.1. For any environment trace ξ^e satisfying the assumptions listed in the Environment Model, the robot trace ξ^r generated by the manager and the dispatcher ensures that the system trace $\xi = \langle \xi^e, \xi^r \rangle$ satisfies the LTL specification Φ.*

Proof. To prove that ξ satisfies Φ, we have to show that each Φ_i, $i \in \{1, \ldots, n\}$, is satisfied by ξ. To satisfy Φ_i by ξ, the manager has to evaluate the corresponding ψ_i at all times due to the '$\Box$' operator. If ψ_i turns out to be $\texttt{false}$ in a time step,



$\xi^e_1 = \langle T, T, T, F \rangle$
$\xi^e_{47} = \langle T, T, T, T \rangle$
$\xi^e_{72} = \langle T, T, T, F \rangle$
$\xi^e_{78} = \langle T, T, F, F \rangle$
$\xi^e_{157} = \langle T, T, T, F \rangle$
ξ^e
2 4 8 19 27 49 51 60 94 98 120 125 158 160 170
Timeline (secs)
r_1 : p_8 — p_2 — p_1 — p_2 - - p_2 — p_4 — p_3 — p_1 — p_2 - - - p_2 — p_1 — p_2 —
r_2 : p_9 — p_4 — p_3 — p_4 - - - follow - - - p_4 — p_3 — p_4 —
r_3 : p_{10} — p_5 — p_7 — p_6 — p_5 — p_7 — p_6 — - - - p_{10} p_{10} - - - p_5 — p_7 — p_6 —
ξ^r
ω_1
Planner Dispatcher

Fig. 3: A timeline showing the key activities of the framework for persistent surveillance: The solid blue vertical lines denote the reading of sensor variables. The green and red vertical lines represent the invocation of the planner by the manager and the operation of the dispatcher, respectively. The dashed and dotted vertical lines are used for robot groups ω_1 and ω_2, respectively. The trajectories for the robots are shown in green below the timeline: Continuous lines correspond to the path execution, dotted lines the repetition of the same behavior, and dashed lines the waiting of the robots when the plan is being computed.

further evaluation of Φ_i is not required. However, if ψ_i turns out to be `true` in a time step, the infinite trace starting at that step has to satisfy $\Diamond(\phi^r_i W \neg\psi^e_i)$. At that time step, the planner computes the plan for the specification $\Diamond\Box\phi^r_i$, which provides the plan in the form of $\sigma^{pre} \cdot (\sigma^{suf})^\omega$. Depending on the environment condition, there could now be two cases: (i) ψ^e_i never becomes `false`, and (ii) ψ^e_i becomes `false` eventually, making $\neg\psi^e_i$ `true`.

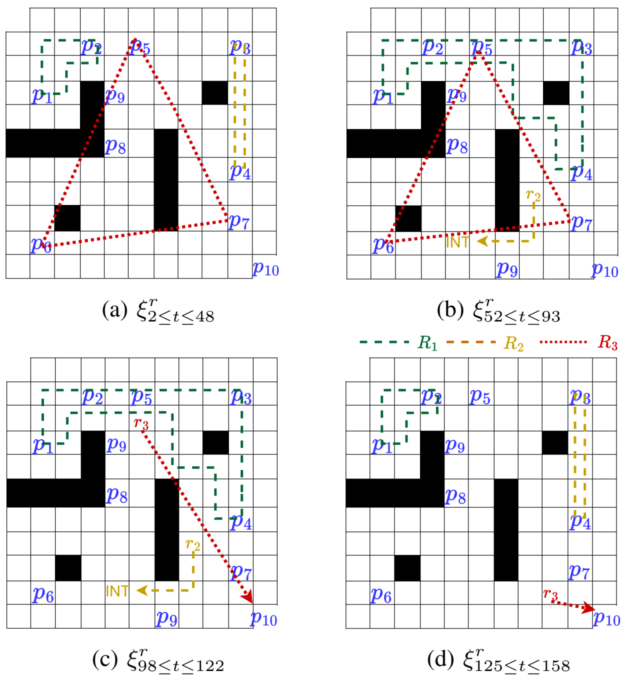
Case (i): In this case, the dispatcher keeps on repeating the R^{suf} for the robots responsible for satisfying the specification. Based on the semantics of weak until ('W'), this is a right behavior of an infinite trace to satisfy $\phi^r_i W \neg\psi^e_i$ as the satisfaction of the property when $\neg\psi^e_i$ is `false` requires $\Box\phi^r_i$ to be satisfied by the trace. The prefix R^{suf} ensures the satisfaction of the specification $\Diamond(\phi^r_i W \neg\psi^e_i)$.

Case (ii): When $\neg\psi^e_i$ becomes `true`, the dispatcher stops sending the current suffix path σ^{suf} to the robots. Any infinite extension of this trace is correct as based on the semantics of 'W', ϕ^r_i has to remain *true* only till $\neg\psi^e_i$ becomes `true`. The continuation of the repetition of R^{suf} is also a correct behavior according to the semantics of 'W', but that may lead to wrong behavior of the robots as a `false` value for ψ^e_i implies that some other environment condition ψ^e_j, $j \in \{1, \ldots, n\}$, $j \neq i$ may become `true`, and the robots need to be available for such a situation. □

IV. Evaluation

In this section, we present several results to demonstrate the efficacy of our framework. We implement our framework's manager, dispatcher, vehicle-system, and sensors modules as ROS [13] packages in C++. In the manager module, we use recently developed scalable multi-agent LTL planner MT* [5], which uses the LTL2TGBA tool [15] as the LTL query to Büchi automaton converter. In the dispatcher module, we apply generic collision avoidance techniques from [16] and generate a collision-free path for each robot.

We provide Gazebo simulations [17] for two use cases described below. Our experiments use Husky robots [18] for



p_1 p_2 p_3 p_4 p_5 p_6 p_7 p_8 p_9 p_{10}
r_1 r_2 r_3
INT
(a) $\xi^r_{2\le t\le 48}$
(b) $\xi^r_{52\le t\le 93}$
R_1 R_2 R_3
(c) $\xi^r_{98\le t\le 122}$
(d) $\xi^r_{125\le t\le 158}$

Fig. 4: Robot trajectories ξ^r generated based on ξ^e for the persistent surveillance use case

unmanned ground vehicles (UGVs) and generic hex-copters for unmanned aerial vehicles (UAVs). For UGVs, we use the ROS Navigation Stack[1] for point-to-point navigation with TEB [19] as the local planner and model predictive control (MPC) on UAV for point-to-point navigation.

The results have been obtained on a desktop computer with a 3.4 GHz quad-core processor with 32 GB of RAM, running Linux OS Version 20.04.6 LTS.

A. Case Study I: Persistent Surveillance

We use the example problem described in section II-D as our first case study. Figure 3 represents a global timeline (units

[1]https://github.com/ros-planning/navigation

in seconds) describing a possible execution of the framework generating robot trajectories (ξ^r) in response to environment trace (ξ^e) such that the resulting system trace satisfies Φ. Figure 4 shows the suffix of the trajectories of the robots ξ^r generated by the framework considering the environment trace ξ^e at different time steps.

The environment starts with $\xi^e_1 = \langle \mathtt{T}, \mathtt{T}, \mathtt{T}, \mathtt{F} \rangle$, denoting that the batteries of all robots are charged and no intruder has been detected. All the variables for both groups are initialized. At this time point, for group ω_1, the variables $status_1[1]$, $replan_1$, $ready_1$ are set to `true` by the `update_flags()` function. Similarly, for group ω_2 the function sets the variables $status_2[1]$, $replan_2$, and $ready_2$ to `true`. Thus, the planner is invoked for both groups at time $t = 2$ (represented in Figure 3), and Figure 4a depicts the suffix (R^{suf}) part of the computed plan for both groups $\{r_1, r_2\}$ and $\{r_3\}$. For robot r_1, its prefix R_1^{pre} moves it from its starting location p_8 to the starting point of suffix p_2. Similarly, for robot r_2 and r_3, their R^{pref} move them from their starting locations p_9 and p_{10} to the starting point of their suffixes p_4 and p_5, respectively. The prefix paths are sent to the robots of both groups for execution at time $t = 4$. For group ω_1, the robots take 15 s to complete their R_1^{pref} and start executing R_1^{suf} at time $t = 19$, whereas for group ω_2 the robot takes 4 s to complete execution for its R_2^{pref} and starts executing R_2^{suf} at time $t = 8$. At time $t = 27$, group ω_2's robot completes executing its R_2^{suf} and executes the same suffix as there is no change in ξ^e trace.

Robot r_3 detects an intruder at time $t = 27$, and the environment state changes to ξ^e_{47}=$\langle \mathtt{T}, \mathtt{T}, \mathtt{T}, \mathtt{T} \rangle$. This leads to $\omega^e_{1,2}$ being `true`. Accordingly, the flags $status_1[2]$ and $replan_1$ are set to `true` for group ω_1 by the function `update_flags()`, but since the robots are already executing their previous task, the manager module waits for $ready_1$ flag to become `true`. It is set to `true` at time $t = 49$ when robots r_1 and r_2 complete execution of their earlier R_1^{suf} and the re-planning is performed for group ω_1. Figure 4b shows the suffixes of the new plan. As per the updated plan, robot r_2 moves from its current location to the neighbouring cell of the intruder's last known location as part of R_1^{pre} and follows the intruder as part of R_1^{suf} whereas robot r_1 patrols the proposition locations p_1, p_4, p_3, p_2 as part of R_1^{suf}. The plan has no R_1^{pre} for robot r_1 as the robot starts at the suffix starting point. The robots starts executing the plan at time $t = 51$. Currently, for the robot in group ω_2, there is no change in its flags, and it keeps executing its R_2^{suf}. It completes the execution at time $t = 60$ and re-executes the same R_2^{suf} since there is no change in the group's variables.

At time $t = 72$, robot r_2 assumes that the intruder is lost as it cannot find it, and thus the environment trace changes to ξ^e_{72}=$\langle \mathtt{T}, \mathtt{T}, \mathtt{T}, \mathtt{F} \rangle$. The `update_flags()` function updates the variable $status_1[2]$ to `false` and variable $replan_1$ to `true`, and the manager module waits for $ready_1$ to become `true` for group ω_1. The voltage of robot r_3's battery dips below the threshold at time $t = 78$, and the environment trace changes to ξ^e_{78}=$\langle \mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{F} \rangle$. Accordingly, the flags $status_2[2]$ and $replan_2$ are set to `true` and $status_2[1]$ is set to `false` by the function `update_flags()` and the manager module waits for $ready_2$ to become `true` for group ω_2. At time $t = 94$, $ready_2$ is set to `true` as the robot r_3 has executed its R_2^{suf}. Now, re-planning is done for group ω_2, and as per the new plan, the robot r_3 visits p_{10} and stays there. Figure 4c shows the R^{suf} of the plans for the robots after the re-planning. Similarly, at time $t = 122$, $ready_1$ becomes `true`, and re-planning is performed for group ω_1, and the R^{suf} of the updated plans for the robots are shown in Figure 4d. This way, the execution moves forward, generating ξ^r in response to ξ^e.

TABLE II: LTL Specifications for Use-Case 2

i	ψ_i^e	ϕ_i^r
1	$b_1 \wedge i_1$	$((\lozenge p_1 \wedge \lozenge s_{d_1}) \wedge \square((p_1 \rightarrow \bigcirc(\neg p_1 \cup s_{d_1})) \wedge (s_{d_1} \rightarrow \bigcirc(\neg s_{d_1} \cup p_1))))$
2	$b_1 \wedge i_2 \wedge \neg i_1$	$((\lozenge p_2 \wedge \lozenge s_{d_2}) \wedge \square((p_2 \rightarrow \bigcirc(\neg p_2 \cup s_{d_2})) \wedge (s_{d_2} \rightarrow \bigcirc(\neg s_{d_2} \cup p_2))))$
3	$b_1 \wedge i_3 \wedge \neg i_1 \wedge \neg i_2$	$((\lozenge p_3 \wedge \lozenge s_{d_3}) \wedge \square((p_3 \rightarrow \bigcirc(\neg p_3 \cup s_{d_3})) \wedge (s_{d_3} \rightarrow \bigcirc(\neg s_{d_3} \cup p_3))))$
4	$\neg\ b_1$	$\lozenge\ g_1$
5	$b_2 \wedge d_1$	$((\lozenge s_{p_1} \wedge \lozenge d) \wedge \square((sp_1 \rightarrow \bigcirc(\neg sp_1 \cup d)) \wedge (d \rightarrow \bigcirc(\neg d \cup sp_1))))$
6	$b_2 \wedge d_2 \wedge \neg d_1$	$((\lozenge sp_2 \wedge \lozenge d) \wedge \square((sp_2 \rightarrow \bigcirc(\neg sp_2 \cup d)) \wedge (d \rightarrow \bigcirc(\neg d \cup sp_2))))$
7	$b_2 \wedge d_3 \wedge \neg d_1 \wedge \neg d_2$	$((\lozenge sp_3 \wedge \lozenge d) \wedge \square((sp_3 \rightarrow \bigcirc(\neg sp_3 \cup d)) \wedge (d \rightarrow \bigcirc(\neg d \cup s_{p3}))))$
8	$\neg b_2 \wedge b_1$	$\lozenge g_2$
9	b_3	$(\lozenge p_1 \wedge \lozenge p_2 \wedge \lozenge p_3 \wedge \lozenge d)$
10	$\neg b_3$	$\lozenge a$

A video of the Gazebo Simulation is available at https://youtu.be/9Yg_GgSUvlg.

B. Case Study II: Warehouse Management

In this case study, we apply our methodology to solve the logistics problem in a warehouse where UAVs and UGVs are used to pick up and drop items. The workspace W is shown in Figure 5. It deals with three different types of items. Item i is produced at production zone p_i and stored at the storage zone s_i. The produced item i is dropped at s_{d_i} for storage, and the same item is picked from s_{p_i} for delivery whenever required. Moreover, d represents the delivery location, a represents the UAV charging location, and g_1 and g_2 represent charging locations for UGVs.

We illustrate this use case using three robots: r_1(UGV), r_2(UGV), and r_3(UAV). Here $\Pi^1 = \{p_1, p_2, p_3, s_{d_1}, s_{d_2}, s_{d_3}, g_1\}$, $\Pi^2 = \{s_{p_1}, s_{p_2}, s_{p_3}, d, g_2\}$, and $\Pi^3 = \{p_1, p_2, p_3, a\}$. Robot r_1 links between the production and storage zone, r_2 links between storage and delivery location, and r_3 monitors the production zones. We use nine sensor variables $K = \{b_1, b_2, b_3, i_1, i_2, i_3, d_1, d_2, d_3\}$. Sensor variables $b_{j=1,2,3}$ represent the battery state of robot r_j, $i_{j=1,2,3}$ represents the presence of items at the j^{th} production zone p_j, whereas $d_{j=1,2,3}$ represents the delivery required of items from j-th storage zone. The functionality of the batteries is the same as described in subsection II-D. Sensor variable i_j set to `true` represents that item j is to be moved from the production zone to the storage zone, and d_j set to `true` represents that item j is to be moved from the storage zone to the delivery location.

The queries in Table II represent the tasks to be achieved by the system. As per queries ϕ_1, ϕ_2, and ϕ_3, robot r_1 should pick item j from location p_j and drop the item to the storage drop-up area (denoted by s_{d_j}) if it has sufficient battery and sensor variable i_j is set to `true`. Similarly, queries ϕ_5, ϕ_6, and ϕ_7 say that robot r_2 should pick items from storage drop-up area s_{p_j} depending on sensor variables s_j and drop the item to the dispatch location if it has sufficient battery voltage. Here, priority is given to the item with the lower index for both storage as well as delivery. According to query ϕ_9, robot r_3 should continuously visit production zones and report if production is available for the j^{th} item by updating the corresponding sensor variable if it has sufficient battery voltage. Queries ϕ_4, ϕ_8, and ϕ_{10} ask the robots to move to their corresponding charging locations when their battery sensor indicates a low battery charge.

We present a detailed description of this use case in Appendix B. A video of the Gazebo Simulation is available at https://youtu.be/HQn30q98ROA

C. Scalability Experiments

We conduct a series of scalability experiments using our framework, employing scenarios that closely resemble those discussed in Section II-D. In all the experiments, we logically divided the workspace into 4 quadrants, where a set of UGVs are responsible for patrolling and following intruders in a quadrant, whereas the UAVs are responsible for patrolling the whole workspace. These experiments involve varying the dimensions of the workspace and the number of robots. The results of these experiments are presented in Table III, providing valuable insights into our framework's scalability and performance under various conditions.

In Table III, WS stands for workspace size, #R represents the total number of robots, including UGVs and UAVs, #K represents the number of environmental sensor variables, #AP represents the count of atomic propositions within Φ, and #Events denote the average number of sensor events observed during an experiment run. For example, in the first experiment, #K includes 5 sensor variables representing the battery status (1 per robot) and 4 sensor variables (1 per quadrant) representing the quadcopter's intrusion detection status. The set of atomic propositions AP comprises 5 charging locations (1 per robot), 5 atomic propositions per quadrant (4 patrolling points and a "follow" proposition) for the UGVs, and 6 additional propositions denoting the patrolling locations for the UAVs. During the experiments, 5 sensor events were observed on average. Moreover, PT (Planning Time) refers to the total planning time (in seconds), and RT (Response Time) measures the average time between a sensor event and the time instant when the computation of the plan corresponding to the sensor event gets completed (in seconds). Thus, the response time includes the planning time and the time required to complete the execution of the previous suffix loop.

For each scenario, we repeat the experiment 10 times and report the average and standard deviation in Table III. For the 20×20 and 40×40 workspaces, each experimental trial spans for 600s and 900s, respectively. From Table III, we observe that the planning time, which increases with both the size of the workspace and the number of robots, is small compared to the response time (RT) thanks to the splitting of the specifications into disjoint groups as presented in Section III-A1. We have also attempted to compute the plan without splitting the specifications into independent groups. However, for this monolithic approach, the generation of the Büchi automata took 19min, showing the necessity of grouping the specifications. In smaller workspaces, response time tends to be faster, with relatively swift responses. It increases with high variation if workspace size is increased. For a fixed workspace, with an increase in the number of robots, the response time decreases. This observation highlights that the complexity of the environment and the robot count play crucial roles in influencing the system's response time.

TABLE III: Efficiency of the framework

WS	#R	#K	#AP	#Events	PT (s)	RT (s)
20×20	$4+1$	9	31	5 ± 1	0.12 ± 0.01	41.89 ± 17.62
	$8+2$	14	36	5 ± 1	0.27 ± 0.02	34.50 ± 11.60
40×40	$4+1$	9	41	5 ± 2	0.11 ± 0.04	153.9 ± 45.00
	$8+2$	14	46	6 ± 2	1.11 ± 0.06	114.3 ± 25.59
	$12+4$	20	52	7 ± 1	15.26 ± 4.35	90.0 ± 50.79

D. Real Experiment

We conduct a real outdoor experiment with two ground robots and a quadcopter. In our experiment, we use the first three LTL specifications in Table I, excluding the specifications involving battery charges. Figure 1a presents the top view of the experimental arena. The details of the robots have been provided in Appendix C. A video of the experiment is available at https://youtu.be/ZaPIQ6qd57U.

V. Related Work

Linear Temporal Logic has been widely used to capture the specification for task and path planning for multi-robot systems(e.g., [1]–[6], [20]). These works use linear temporal logic to capture the task specifications, and the goal is to synthesize the trajectories for the robots so that they satisfy the LTL specifications. However, these papers consider a static environment and the absence of any events appearing online.

To deal with the dynamism in the environment, researchers have adopted the reactive synthesis technique [21] to synthesize a high-level controller from a subset of LTL called GR(1), which captures the specification in terms of environmental assumptions and the tasks of the robots to be performed against different environmental events. The first work on applying reactive synthesis for synthesizing a high-level robot controller from the GR(1) subset of LTL is due to Kress-Gazit et al. [7]. To deal with the scalability limitation of the reactive synthesis method, Wongpiromsarn et al. [9] propose a receding horizon approach where the planning task is split into multiple smaller planning tasks. Livingston et al. [10] address the problem of using plans generated by reactive synthesis in a dynamic environment where the synthesized plan may become invalid.

They propose μ-calculus based local planning strategy to deal with new observations in the environment that do not fit with the original assumptions on the environmental behavior. DeCastro et al. [11] extend the reactive synthesis methodology to multiple robots. Subsequently, the idea of reactive synthesis has also been applied to robot swarms to synthesize high-level plans for the robots from temporal logic specifications [22], [23]. However, the major limitation of reactive synthesis is that this approach scales poorly with both the size of the workspace and the number of robots.

To circumvent the scalability limitation of reactive synthesis, the Satisfiability Modulo Theory (SMT) based approach has been adopted in several prior works to deal with the dynamism in the environment. For example, SMT-based frameworks were introduced for switching to another specification satisfying paths to avoid collisions with dynamic obstacles in the workspace [24] and to deal with the recharge requirement of a robot [25]. Unlike our framework, these papers deal with a very specific environmental event. The closest to our work is Antlab [26], which is a framework for managing a group of robots as resources and utilizing them to satisfy incoming LTL requirements. However, unlike our framework, Antlab does not support sensing events and considers incoming LTL specifications to be independent of each other. Recently, Kalluraya et al. [27] present an adaptive sampling-based methodology for optimal path planning for multi-robot systems to satisfy LTL specifications, capturing requirements for completing tasks collaboratively amidst uncertain semantic targets governed by stochastic dynamics.

VI. Conclusion

We have presented a task and path planning framework for multi-robot systems where the robots need to respond to environmental events by adapting their behavior dynamically. To capture the specifications of the desirable reactive behavior of the multi-robot system, we have introduced a restricted fragment of LTL, which has the potential to be applicable to a wide range of practical problems requiring automation to deal with the dynamic nature of the environment, as evident from the case studies and the real experiment we have presented in the paper. In the future, we plan to explore the possibility of using large language models (LLM) [28] to create a natural language interface for our framework.

Acknowledgment

The authors thankfully acknowledge the Defence Research & Development Organisation (DRDO), India, for funding the project through JCBCAT, Kolkata.

References

[1] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *I. J. Robotic Res.*, vol. 32, no. 8, pp. 889–911, 2013.

[2] Y. E. Sahin, P. Nilsson, and N. Ozay, "Multirobot coordination with counting temporal logics," *IEEE Trans. Robotics*, vol. 36, no. 4, pp. 1189–1206, 2020.

[3] X. Luo, Y. Kantaros, and M. M. Zavlanos, "An abstraction-free method for multirobot temporal logic optimal control synthesis," *IEEE Trans. Robotics*, vol. 37, no. 5, pp. 1487–1507, 2021.

[4] L. Lindemann, J. Nowak, L. Schönbächler, M. Guo, J. Tumova, and D. V. Dimarogonas, "Coupled multi-robot systems under linear temporal logic and signal temporal logic tasks," *IEEE Trans. Control. Syst. Technol.*, vol. 29, no. 2, pp. 858–865, 2021.

[5] D. Gujarathi and I. Saha, "MT*: Multi-robot path planning for temporal logic specifications," in *IROS*, 2022, pp. 13 692–13 699.

[6] S. Hustiu, D. V. Dimarogonas, C. Mahulea, and M. Kloetzer, "Multi-robot motion planning under MITL specifications based on time petri nets," in *European Control Conference, ECC 2023, Bucharest, Romania, June 13-16, 2023*. IEEE, 2023, pp. 1–8.

[7] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[8] J. Alonso-Mora, J. A. Decastro, V. Raman, D. Rus, and H. Kress-Gazit, "Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles," vol. 42, no. 4, 2018.

[9] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Trans. Automat. Contr.*, vol. 57, no. 11, pp. 2817–2830, 2012.

[10] S. C. Livingston, P. Prabhakar, A. B. Jose, and R. M. Murray, "Patching task-level robot controllers based on a local μ-calculus formula," in *ICRA*, 2013, pp. 4588–4595.

[11] J. A. DeCastro, J. Alonso-Mora, V. Raman, D. Rus, and H. Kress-Gazit, "Collision-free reactive mission and motion planning for multi-robot systems," in *ISRR*, 2015, pp. 459–476.

[12] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen, "A deliberative architecture for auv control," in *ICRA*, 2008, pp. 1049–1054.

[13] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[14] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[15] A. Duret-Lutz, "LTL translation improvements in Spot 1.0," *International Journal on Critical Computer-Based Systems*, vol. 5, no. 1/2, pp. 31–54, 2014.

[16] D. Silver, "Cooperative pathfinding," in *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference*, 2005, pp. 117–122.

[17] Open Robotics, "Gazebo Simulation Environment," http://gazebosim.org.

[18] [Online]. Available: https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/

[19] C. Roesmann, W. Feiten, T. Woesch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," in *ROBOTIK 2012; 7th German Conference on Robotics*, 2012, pp. 1–6.

[20] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Automated composition of motion primitives for multi-robot systems from safe LTL specifications," in *IROS*, 2014, pp. 1525–1532.

[21] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," *J. Comput. Syst. Sci.*, vol. 78, no. 3, pp. 911–938, 2012.

[22] S. Moarref and H. Kress-Gazit, "Automated synthesis of decentralized controllers for robot swarms from high-level temporal logic specifications," *Auton. Robots*, vol. 44, no. 3-4, pp. 585–600, 2020.

[23] J. Chen, R. Sun, and H. Kress-Gazit, "Distributed control of robotic swarms from reactive high-level specifications," in *17th IEEE International Conference on Automation Science and Engineering, CASE 2021, Lyon, France, August 23-27, 2021*. IEEE, 2021, pp. 1247–1254.

[24] P. Purohit and I. Saha, "DT*: Temporal logic path planning in a dynamic environment," in *IROS*, 2021, pp. 3627–3634.

[25] T. Kundu and I. Saha, "Energy-aware temporal logic motion planning for mobile robots," in *ICRA*. IEEE, 2019, pp. 8599–8605.

[26] I. Gavran, R. Majumdar, and I. Saha, "Antlab: A multi-robot task server," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5, pp. 190:1–190:19, 2017.

[27] S. Kalluraya, G. J. Pappas, and Y. Kantaros, "Multi-robot mission planning in dynamic semantic environments," in *ICRA*, 2023, pp. 1630–1637.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017, pp. 5998–6008.

APPENDIX

A. Büchi Automaton and Product Automaton

In this section, we provide the formal definition of a Büchi Automaton B_ϕ and the Product Automata P for a transition system T and a Büchi Automaton B_ϕ.

1) LTL to Büchi Automaton: Given an LTL specification ϕ, a Büchi Automaton B_ϕ models ϕ. A Büchi automaton is represented as a tuple $B_\phi = (Q_B, q_0, A, \delta, F)$, where (i) Q_B is a finite set of states, (ii) $q_0 \in Q_B$ is the initial state, (iii) A is the set of input alphabets, (iv) $\delta : Q_B \times A \longrightarrow Q_B$ is a transition function, and (v) $F \subseteq Q_B$ is the set of accepting (final) states. A run ω over an infinite input sequence $a_0 a_1 \ldots a_i \in A$, is a sequence of automaton states $\rho = q_0 q_1 \ldots$, $q_i \in Q_B$ with $q_0 = q_{B_0}$ and $q_1 = \delta(q_0, a_0), q_2 = \delta(q_1, a_1)$ and so on, where $a_i \in A$. An infinite input sequence η is said to be accepted by Büchi Automaton B_ϕ iff there exists a run in which at least one state in F is visited infinitely often.

2) Product Automaton: The Product Automaton P between the joint transition system $T := (S_T, s_0, E_T, \Pi_T, L_T, w_T)$ and the Büchi automaton $B_\phi = (Q_B, q_0, A, \delta, F)$ is defined as $P := (S_P, S_{P,0}, E_P, F_P, \omega_p)$, where (i) $S_P = S_T \times Q_B$ (ii) $S_{P,0} = (s_0, q_0)$ is an initial state, (iii) $E_P \subseteq S_P \times S_P$, where $((s_i, q_k), (s_j, q_l)) \subseteq E_P$ if and only if $(s_i, s_j) \subseteq E_T$ and $(q_k, L_T(s_j), q_l) \subseteq \delta_B$, (iv) $F_P = S_T \times Q_f$ set of final states, and (v) $\omega_P : E_P \longrightarrow N_{>0}$ such that $\omega_P((s_i, q_k), (s_j, q_l)) = \omega_T(s_i, s_j)$. To generate a trajectory in T, which satisfies the LTL query ϕ, we can refer to P.

The task of satisfying ϕ in $\mathcal{W}$ can be accomplished by finding a cycle consisting of any final state $f \in F_P$ in the product graph P. The path from the initial state to one of the final states of the graph is called the *prefix path* and is denoted by R_{pref}. A non-trivial cycle from the final location to itself is called the *suffix cycle* and is denoted by R_{suff}. Thus, an infinite run R over P can then be written as $R = R_{pref}.(R_{suff})^\omega$, where R_{pref} is traversed once, and R_{suff} is traversed infinitely.

B. Description of Warehouse Management Case Study

In this section, we describe how our framework enables the robots to accomplish the tasks in Warehouse Management. To explain the working of Algorithm 1 on this use case, we take a possible initial environment trace ξ^e given by ξ^e_1= $\langle \mathtt{T}, \mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{F}, \mathtt{F}, \mathtt{F}, \mathtt{F} \rangle \ldots \xi^e_{10}$=$\langle \mathtt{T}, \mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{F}, \mathtt{F} \rangle \ldots \langle \xi^e_{20}$= $\langle \mathtt{T}, \mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}, \mathtt{F}, \mathtt{T}, \mathtt{F}, \mathtt{F} \rangle \ldots \xi^e_{79}$=$\langle \mathtt{T}, \mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}, \mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{F} \rangle \ldots \langle \xi^e_{96}$ =$\langle \mathtt{T}, \mathtt{T}, \mathtt{T}, \mathtt{T}, \mathtt{T}, \mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{F} \rangle \ldots$. As per this trace, initially, the robots start with full battery charge, and none of the items is present in the production zone, neither is delivery of any item required. At time $t = 10$, item i_2 becomes available at production zone p_2. Next, at time $t = 20$, delivery is required for items from storage zone s_1. Similarly, at time $t = 79$, item i_3 becomes available at production zone p_3, and at time $t = 96$, i_1 becomes available at production zone p_1.

From Φ, we get $\Sigma_1 = \Sigma_2 = \Sigma_3 = \Sigma_4$=$\{r_1\}$, $\Sigma_5 = \Sigma_6 = \Sigma_7 = \Sigma_8 = \{r_2\}$ and $\Sigma_9 = \Sigma_{10} = \{r_3\}$. So three groups ω_1 =$\{\Phi_1, \Phi_2, \Phi_3, \Phi_4\}, \omega_2$ =$\{\Phi_5, \Phi_6, \Phi_7, \Phi_8\}$ and ω_3=$\{\Phi_9, \Phi_{10}\}$ are formed in line number 1 in Algorithm 1 using the procedure described in Section III-A1. Figure 5 depicts the R^{suf} of the plan generated at different time instances by the manager module. The environment starts with ξ^e_1, and all the variables for the three groups are initialized. For group ω_3, the variables $status_3[1]$, $replan_3$, $ready_3$ are set to `true` by the `update_flags()` function, whereas for groups ω_1 and ω_2, the variables $ready_1$ and $ready_2$ are set to `true` respectively. Thus, we get Φ_3^{redc} as $\Diamond\Box(\Diamond p_1 \wedge \Diamond p_2 \wedge \Diamond p_3 \wedge \Diamond d)$ whereas Φ_1^{redc} and Φ_2^{redc} is empty (i.e., `true`), and the plan generated is shown in Figure 5a. Here, no plan is generated for robots r_1 and r_2, while R_3^{suf} is to patrol the proposition locations p_1, p_2, and p_3. At time $t = 10$, the instance of ξ^e changes to ξ^e_{10}, and the variables $status_1[2]$, $replan_1$, $ready_1$ are set to `true` by the `update_flags()` function for group ω_1. We get Φ_1^{redc} as $\Diamond\Box((\Diamond p_2 \wedge \Diamond s_{d_2}) \wedge \Box((p_2 \rightarrow \bigcirc(\neg p_2 \cup s_{d_2})) \wedge (s_{d_2} \rightarrow \bigcirc(\neg s_{d_2} \cup p_2))))$ and the generated plan is shown in Figure 5b, where robot r_1 picks item i_2 from p_2 and drops it at s_{d_2}. Next, when the instance of ξ^e changes to ξ^e_{20}, the variables $status_2[1]$, $replan_1$, $ready_1$ are set to `true` by the `update_flags()` function for group ω_2 and Φ_2^{redc} is computed as $\Diamond\Box((\Diamond s_{p_1} \wedge \Diamond d) \wedge \Box((sp_1 \rightarrow \bigcirc(\neg sp_1 \cup d)) \wedge (d \rightarrow \bigcirc(\neg d \cup sp_1))))$. Thereby planning is done for Φ_2^{redc}, and the plan is shown in Figure 5c where robot r_2 picks item i_1 from s_{p_1} and drops it at d. At time $t = 79$, when the instance of ξ^e changes to ξ^e_{79}, no change is done to R^{suf} as R^{suf} is still valid as per Φ. Next when the instance of ξ^e changes to ξ^e_{96}, and the variables $status_1[1]$, $replan_1$, $ready_1$ are set to `true` and $status_1[2]$ is set to `false` by the `update_flags()` function for group ω_1. Φ_1^{redc} is now computed as $\Diamond\Box((\Diamond p_1 \wedge \Diamond s_{d_1}) \wedge \Box((p_1 \rightarrow \bigcirc(\neg p_1 \cup s_{d_1})) \wedge (s_{d_1} \rightarrow \bigcirc(\neg s_{d_1} \cup p_1))))$. Accordingly, planning is done, and the plan is shown in Figure 5d where robot r_1 now picks item i_1 from p_1 and drops it at s_{d_1}. This way, the execution moves forward, generating ξ^r in response to ξ^e.

C. Hardware Details for the Real Experiments

Here, we provide the details of the hardware used in our outdoor experiment. We use 0xDelta robots [2] as UGVs and an assembled quadcopter based on the TAROT[3] Ironman 650 Multi-rotor airframe with Orange-cube[4] autopilot running PX4 firmware [5]. Each UGV is attached with a Zed2 [6] camera and two 360° Lidars [7] as sensors with Intel NUC[8] and Jetson Xavier NX [9] as compute modules. On the quadcopter, we

[2]http://www.nex-robotics.com/products/0x-series-robots/0x-delta-rugged-all-terrain-robot.html

[3]http://www.tarotrc.com/?lang=en

[4]https://docs.cubepilot.org/user-guides/autopilot/the-cube-module-overview

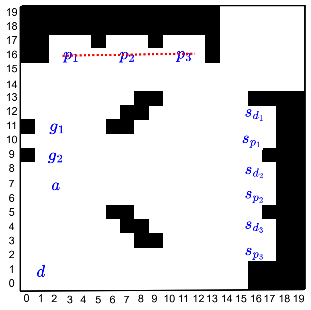
[5]https://docs.px4.io/main/en/flight_controller/cubepilot_cube_orange.html

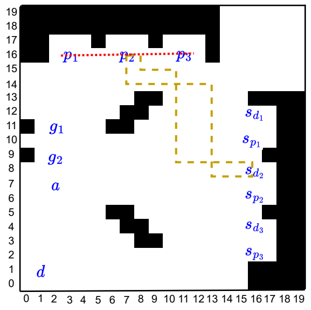
[6]https://www.stereolabs.com/zed-2/

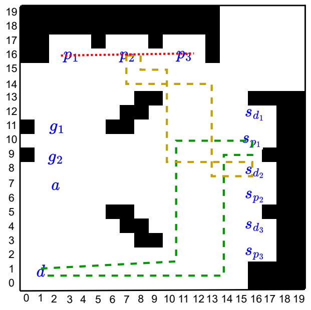
[7]http://bucket.download.slamtec.com/f19ea8efcc2bb55dbfd5839f1d307e34aa4a6ca0/LD601_SLAMTEC_rplidar_datasheet_S1_v1.4_en.pdf

[8]https://www.intel.in/content/www/in/en/products/details/nuc.html

[9]https://www.nvidia.com/en-in/autonomous-machines/embedded-systems/jetson-xavier-nx/


(a) $\xi^r_{1 \leq t \leq 10}$


(b) $\xi^r_{11 \leq t \leq 20}$


(c) $\xi^r_{21 \leq t \leq 79}$

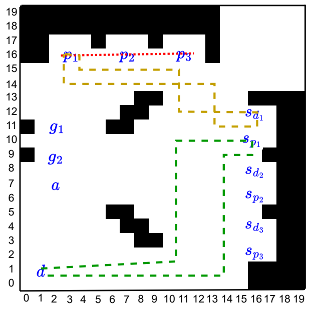
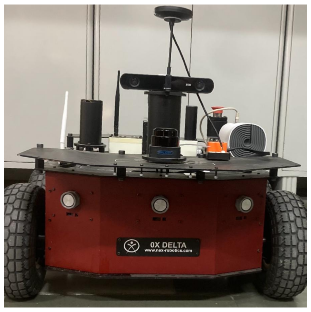

(d) $\xi^r_{80 \leq t \leq 104}$

Fig. 5: Trajectories generated based on ξ^e for Warehouse Management use case


(a) UAV

(b) UGV

Fig. 6: Robots used for the real experiment

use Here 3 GPS[10], Go-pro Hero 7 camera [11], Benawake TF-02 distance sensor [12], and ODROID XU4 [13] as a compute module.

[10]https://docs.cubepilot.org/user-guides/here-3/here-3-manual

[11]https://gopro.com/content/dam/help/hero7-black/manuals/HERO7Black_UM_ENG_REVA.pdf

[12]http://wiki.amperka.ru/_media/products:lidar-tf02-pro:lidar-tf02-pro-product-manual.pdf

[13]https://wiki.odroid.com/odroid-xu4/odroid-xu4