Guaranteeing High-Level Behaviors while Exploring Partially Known Maps

Shahar Sarid Bingxin Xu Hadas Kress-Gazit
Sibley School of Mechanical and Aerospace Engineering
Cornell University
Ithaca, New York 14853, USA
Email: {sarid, bx38, hadaskg}@cornell.edu

***Abstract*—This paper presents an approach for automatically synthesizing and re-synthesizing a hybrid controller that guarantees a robot will exhibit a user-defined high-level behavior while exploring a partially known workspace (map).**

The approach includes dynamically adjusting the discrete abstraction of the workspace as new regions are detected by the robot's sensors, automatically rewriting the specification (formally defined using Linear Temporal Logic) and re-synthesizing the control while preserving the robot state and its history of task completion. The approach is implemented within the LTLMoP toolkit and is demonstrated using a Pioneer 3-DX in the lab.

I. Introduction

Consider the case of a "Search and Rescue" scenario where a mobile robot is patrolling inside a collapsed building with unknown rooms. The robot is required to explore all the accessible rooms and search for survivors. The robot's high-level behavior must be guaranteed to ensure safety and mission completion while expanding the map to include unknown regions.

Recently, several approaches have been suggested for generating correct high-level robot behavior [14, 12, 8, 15, 11, 2, 20] from an abstract description of a task; these approaches tackle the inherent continuous problem of robot sensing, motion and action by creating a discrete abstraction of the task, generating a provably-correct discrete solution and implementing the solution by composing a set of continuous low-level controllers such that the overall continuous behavior of the robot is the desired one.

The specification formalisms are usually a variant of temporal logic, with Linear Temporal Logic (LTL) being the most commonly used. The creation of the discrete solution is based on ideas from formal methods, mainly model checking [3] and synthesis [17] when assuming no noise on sensors and actuators (e.g. [14, 12, 20, 2]), and probabilistic model checking and policy synthesis for Markov Decision Processed (MDPs) when uncertainty is taken into account (e.g.[15, 10]). When implementing the continuous behavior, researchers have considered potential field type motion planners (e.g. [14, 12]) as well as sampling-based approaches (e.g. [11, 2]).

This work was supported by NSF CNS-0931686 and NSF CAREER CNS-0953365

What is common to all aforementioned approaches is that the workspace is assumed to be known; that is, the robot is required to perform its high-level task within a known map. In the process of generating the control, the map is abstracted to a graph where the nodes are regions in the map and edges in the graph indicate that the robot can move between the two regions without going through a third region. In general, when considering a convex partition of a polygonal map, the edges represent the adjacency of the regions (an edge exists between nodes representing two regions that share a common face) since there are several low-level controllers that have been developed over the years that are guaranteed to drive a robot from any point within the convex polygon to one of its faces, and thus to the adjacent region (e.g. [4, 1, 16]).

This paper relaxes the assumption of an a priori completely known map; instead, an initial map is assumed which may contain as little as one region, and as the robot is performing its task, new regions are discovered and the task is adjusted accordingly in an automatic and provable-correct manner. The new regions are detected using an occupancy grid [7] and then added to the known map, gradually expanding and creating a larger map.

This work builds on the approach of [14] where a fragment of LTL is the formalism used to captures tasks that are reactive, meaning that the robot behavior depends on the occurrence of environmental events, for example "continually patrol all the rooms, stop if you see a person and ask them if they need help". Given a formula in the logic, a tractable synthesis technique is used to generate an automaton such that all its executions satisfy the LTL formula. The complexity of synthesizing an arbitrary LTL formula is double exponential in the size of the formula, but if the specification is restricted to a specific fragment of LTL, the complexity becomes polynomial in the state space [17].

The contribution of this paper is in providing guaranteed high-level behavior of robots operating in unknown workspaces. Specifically, this paper describes: (i) The set of user-specified high-level tasks which is enriched by allowing quantifiers ('all', 'any') over regions. This allows a user to say "visit all offices". (ii) The discrete abstraction of the workspace that is updated on-the-fly based on sensor information, and (iii) A re-synthesis algorithm. The algorithm preserves the task

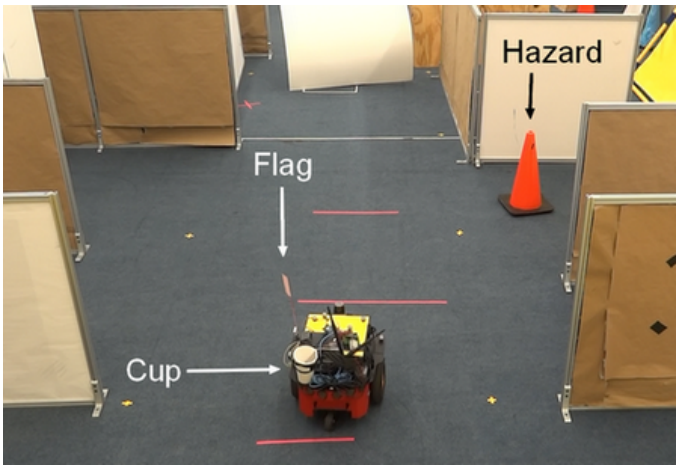


Fig. 1: The robot is visiting the classrooms. The cone represents a hazardous item in front of a classroom and the raised flag was the robot's response to detecting the cone. The cup contained a touch sensor and was used to collect assignments.

history and automatically update the instructions of the robot behaviors, according to the new workspace.

Testing the re-synthesis method and algorithms in real environments with a physical robot equipped with range sensors, gives rise to questions regarding when and how to decide a new region is detected. As a first step toward a generalized solution to the problem, assumptions regarding the environment are made (Section III).

The work described in this paper was implemented on a physical robot in the lab, where the onboard laser range scanner was used to detect previously unknown regions while the robot was performing a high-level tasks. The task described in Section VI required the robot to collect assignments from students in classrooms and hand them to professors in their offices. When the robot saw a hazardous item in front of a classroom (a cone), it avoided entering that classroom and raised a flag of warning (Fig. 1). The robot detected new rooms, distinguished between classrooms and offices according to their size, and continued behaving according to the specification in the a priori unknown workspace.

The paper is structured as follows: Section II presents background information regarding the approach and formalisms for generating high-level correct robot control. Section III defines the problem this paper is focusing on and the assumptions that are made. Section IV discusses the enriched grammar containing region quantifiers, and Section V describes the re-synthesis process. Section VI describes the experiments and the paper concludes in Section VII.

II. Background

This section presents the background information needed for the remainder of the paper. It includes the definition of the underlaying logical formalism, an overview of the process of generating provably-correct robot control from logical formulas and a description of LTLMoP [9], the toolbox that is used to generate the control and perform experiments.

A. Linear Temporal Logic (LTL) specifications

Linear Temporal Logic (LTL) is a modal logic that includes in addition to Boolean operators (such as 'not', 'and', etc.), temporal operators, thus allowing formulas to capture truth values of atomic propositions (π) as they evolve over time.

LTL formulas are constructed from atomic propositions $\pi \in AP$ according to the following recursive rules

$$\varphi ::= \pi | \neg\varphi | \varphi \vee \varphi | \bigcirc \varphi | \Diamond\varphi | \Box\varphi \quad (1)$$

where $\bigcirc\varphi$ is 'Next', $\Diamond\varphi$ is 'Eventually', and $\Box\varphi$ is 'Always'.

This work considers robot specifications that are defined over a discrete abstraction of the robot motion and action and are captured in a fragment of LTL. Specifically, the atomic propositions comprise of a set $X = \{x_1, ..., x_m\}$ of environment propositions corresponding to abstract sensor information (e.g. "object detected"), and a set $Y = \{r_1, ...r_n, a_1, ..., a_k\}$ of robot propositions that correspond to the location of the robot (if r_i is true then the robot is currently in region i) and its actions a_j (e.g."flag is raised").

The fragment of LTL considered in this work follows [17, 14] where formulas are of the form $\varphi = (\varphi_e \Rightarrow \varphi_s)$; φ_e is an assumption about the sensor propositions, and thus about the behavior of the environment, and φ_s represents the desired behavior of the robot. Both φ_e and φ_s have the following structure: $\varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e; \varphi_s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$, where:

- φ_i^e,φ_i^s are non-temporal Boolean formulas constraining the initial value(s) for the environment and robot respectively. They are of the form B_i, where B_i is a boolean formula over the set $X \cup Y$.
- φ_t^e,φ_t^s represents safety assumptions on the environment and safety requirements on the robot's behavior. Safety includes all behaviors that the environment or the robot must always satisfy. φ_t^e constrains the next environment state based on the current environment state and current robot state, and φ_t^s constrains the possible next robot state based on the current environment, current robot state, and the next environment state. The formulas are of the form $\Box B_i$ where B_i is a boolean formula over the set $X \cup Y \cup \bigcirc X$ for φ_t^e and $X \cup Y \cup \bigcirc X \cup \bigcirc Y$ for φ_t^s.
- φ_g^e, φ_g^s represent liveness assumptions for the environment and liveness requirements for the robot. The liveness includes goals the environment or the robot should always eventually satisfy. The formulas are of the form $\Box \diamond B_i$ where B_i is a boolean formula over the set $X \cup Y$.

One property of the synthesis algorithm is that the order of the formulas in φ_g^s determines the sequence in which the robot will fulfill its liveness requirements. For a liveness requirement φ_i, the goal number $gNum(\varphi_i) = i$ indicates that φ_i is in the i-th position in the sequence of goals. For example, in $\varphi_g^s = \Box\Diamond r_a \wedge \Box\Diamond(r_b \vee flag)$, the requirement that the robot eventually go to r_a is of *gNum = 0* (the first liveness) and the requirement of going to r_b or raising the flag is of *gNum = 1*.

B. Discrete abstraction of the workspace

The robot's workspace is assumed to be a 2 dimensional polygonal environment. The motion of the robot in the

workspace is abstracted by a graph where each node represents a region and the edges represent adjacency relations between the regions. Leveraging controllers such as those of [4, 1, 16], given a set of convex polygons, a robot can move between any adjacent regions if there are no obstacles.

For example, from region r_2 shown in Figure 2 the robot can move to adjacent regions r_0, r_1, r_3 or stay in r_2. The motion constraints are captured by:

$$\square(r_2 \Rightarrow (\bigcirc r_2 \vee \bigcirc r_0 \vee \bigcirc r_1 \vee \bigcirc r_3))$$

C. Control generation

Given a robot task as an LTL formula belonging to the fragment described above, if the task is synthesizable [18] an automaton whose behaviors satisfy the formula will be automatically generated (see [17, 14] for details).

The hybrid controller used to continuously control the robot is based on the execution of the automaton. An admissible input sequence is a sequence $X_1, X_2, ...$ s.t. $X_j \in 2^X$ is the set of environment propositions that are true at time step j, that satisfies φ_e. A run of the automaton under an admissible input sequence is a sequence of states $q_0, q_1, ...$, which starts at a possible initial state of the automaton: $q_0 \in \mathcal{Q}_0$. At each time step, the robot sensor information is used to determine the truth values of the environment propositions X, and together with the current state q the next state q' is determined following the transition relation δ, i.e., $q' = \delta(q, X)$. γ is the state labeling function where $\gamma(q) = y$ and $y \subseteq Y$ is the set of robot proposition that are true in state q. Based on the labels of q', the next region and the next actions are performed and the appropriate low-level controllers are executed. The reader is referred to [14] for more details.

Every state $q \in \mathcal{Q}$ has an associated goal number, which indicates the current goal (liveness requirement) the robot is heading toward. This goal number is denoted by $\gamma_r(q)$.

D. LTL MissiOn Planner (LTLMoP)

Linear Temporal Logic MissiOn Planning (LTLMoP) [9] is a Python-based, open-source toolkit that allows users to control physical and simulated robots by specifying high-level instructions in structured English. Furthermore, if a specification contains behaviors that cannot be guaranteed (they may be inconsistent or there may be an environment in which the robot fails), no automaton will be synthesized and LTLMoP facilitates the process of understanding the problem in the given specification [18]. The experiments described in this paper were executed using LTLMoP as further discussed in Section VI.

III. Problem Formulation and Assumptions

This paper focuses on guaranteeing the execution of high-level tasks by a mobile robot operating in an a priori unknown environment. While executing it's mission, the robot detects regions that are not defined in its current map. The basic assumptions regarding the unknown regions are as follows. First, the robot starts the execution with a partially known map. The initially known map can be arbitrary small, on the order of the size of the robot. This assumption is essential, since the robot must be positioned in a free area first. Second, the robot must have the capabilities to detect new regions, i.e., it must have adequate sensors that can sense obstacles or lack thereof.

The robot is assumed to have the appropriate sensors and actuators to perform the high-level tasks it is instructed to do. These sensors and actuators are assumed to be binary, as described in Section II.

Definition 1: Robot model. The robot is assumed to be of size D, where size can be the diameter of a circular robot, or the longest dimension in the plane of movement. We assume that the robot's structure and dynamic properties constrain its movement such that it has a minimal operating circular area of radius sD (for example for turning), where $s \geq 1$ is a safety factor which depends on the robot's structure and dynamic properties. The robot's position and orientation relative to a fixed frame are assumed to be known.

Definition 2: Region. A region p with index i, denoted p^i, is a polygon in the plane, defined as a circuit of j line segments, sides [5], or edges [6], $p^i_1p^i_2, p^i_2p^i_3, \ldots, p^i_jp^i_1$ joining $j = \|p^i\|$ points, or vertices, $p^i_1, p^i_2, p^i_3, \ldots, p^i_j$. The edges of p^i must not cross each other. The polygon is regarded as consisting of its vertices, edges, and the interior area bounded by it. Each region is simply connected, meaning there are no holes within it. The region p^i can be non-convex. A region is free from obstacles, and it contains a bounded circle r_i whose diameter d_{r_i} satisfies $d_{r_i} \geq sD$ to comply with Definition 1.

Definition 3: Initial region: The robot starts from initial region p', which conforms to Definition 2.

Definition 4: Workspace. The workspace is the map the robot uses, in which its operation can be guaranteed and is composed of a set of regions as defined above. The current workspace is assumed to be known in each step i, and is denoted P^i. A step is defined as a stage of execution, for which the map does not change. If the map changes, as explained later, the step is incremented. Each workspace has a boundary defined as follows.

Definition 5: Boundary. The boundary B^i of a workspaces P^i is defined as a polygon, or a set of polygons, composed of all the edges of the regions of P^i which are not shared between two regions. If the workspace is a simple polygon, its boundary is composed from only one polygon. However, if the workspace contains holes, the boundary is composed from an outer polygon, and another polygon for each hole. Within each step, the workspace is assumed to be static inside its boundary, but dynamic in the sense that a previously occupied boundary edge can become unoccupied and serve as the edge of a new detected region.

Definition 6: Expansion. The workspace can be dynamically expanded in each step. The robot starts in step 0 with initial workspace P^0, where P^0 contains at least the initial region p', and is possibly composed of a total of g regions, such that $P^0 = p'^1 \cup p'^2 \cup p'^3 \ldots \cup p'^g$. In each step i, a new region p^i is added. Each new added region p^i conforms to

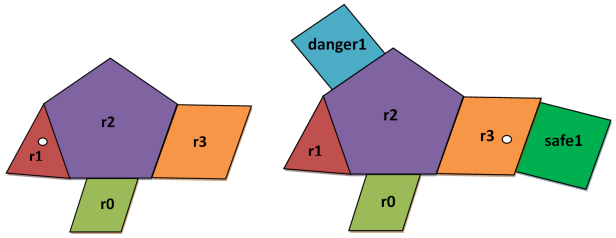


Fig. 2: Left: Initial known map. The robot position is marked with a white circle. Right: Two new, unknown regions, *safe1, danger1*, are updated when the robot visits *r2, r3*

Definition 2. Moreover, each p^i must be adjacent to at least one known region ,p^h, $h = 1 \dots i-1$ or $p^h \in P^0$. The workspace is recursively defined as follows, $P^0 = \bigcup_{d=1}^{g} p'^d$, $P^i = P^{i-1} \cup p^i$, $\forall i > 0$.

New-region sensor: The robot is assumed to have a new-region sensor, which can detect new regions. The new-region sensor is capable of distinguishing, up to its operating range, between free and occupied space[1]. According to Definition 2, the dimensions of a new region must have minimal values, therefore, the sensor's operating range must be greater than sD.

Problem 1: (Operating in an unknown environment) Given an initial known workspace, P^0, an initial region $p' \in P^0$, a mobile robot equipped with a new-region sensor, initial conditions and task specification given in structured English, construct a controller such that the robot's behavior satisfies the user specification, in any admissible, possibly unknown environment.

The problem formulation is illustrated in the following example which demonstrates a simple high-level task defined over an unknown workspace and executed using *LTLMoP*.

Example 1: Search and Rescue mission.
Consider the case of a "Search and Rescue" scenario where a mobile robot is placed in a building which collapsed due to an earthquake. The robot must explore the inner parts of what is left from the building and search for survivors. The building is partially ruined, such that the original blueprint cannot be used. The robot is placed in a partially clear area, shown in Figure 2 (Left). There are two groups of regions: dangerous and safe. The robot is required to visit all the dangerous regions: *r2,r3*, and to search for survivors (people). If the robot finds people, it will guide them back to one of the safe places: *r0* or *r1*. If a new region is detected, it must be identified as a safe or as a dangerous region, and then be added to the map.

IV. Grammar

The grammar [13, 9] used by LTLMoP has been enriched with quantifiers and reactions to new regions needed for defining tasks in unknown maps.

[1]The new-region sensor should be some sort of a range sensor (could be infrared, ultrasonic, laser scanner or camera using vision techniques)

A. Quantifiers

Consider the scenario in Example 1, the same task is assigned over multiple regions. When a new region is detected, no matter safe or dangerous, extra specification is needed for defining the tasks over the new region. Therefore an automatic process for assigning tasks over multiple regions is necessary.

To deal with such conditions, quantifiers are introduced in this section. To apply quantifiers over multiple regions, one defines groups as follows:

- Group *groupName* is *region1, region2, region3...*

If the *groupName* is used together with the quantifiers "all" or "any", the sentences are automatically converted into LTL formulas over the regions. The translation process is as follows:
if quantifier 'any':
result = join $region_i[:]$ with $\vee$
substitute *groupName* with *result*
if quantifier 'all'
$result_i$ = substitute *groupName* with $region_i$
join $result_i[:]$ with $\wedge$
Revisiting Example 1, the new specification would be:

- Group *Safe* is r_0, r_1
- Group *Dangerous* is r_2, r_3
- If you are not activating *guide* then visit all *Dangerous*
- If you are activating *guide* then visit any *Safe*

is translated into

$$\wedge_{i \in \{2,3\}} \square\diamond((\neg a^{guide}) \Rightarrow r_i)$$
$$\wedge \square\diamond((a^{guide}) \Rightarrow (r_0 \vee r_1))$$

These quantifiers facilitate writing specifications for unknown workspaces as they do not require the explicit enumeration of all regions. When a new region is detected, the user is not required to manually write the additional specification and the robot is able to automatically re-synthesize the controller and resume the execution. The grammar for re-synthesis is introduced in Section IV-B, and the algorithm is introduced in Section V.

B. Specification for re-synthesis

Now assume, in Example 1, when the robot enters *r2*, it detects a new dangerous region *danger1* adjacent to *r2*. Then, when it enters *r3*, it detects a new safe region *safe1* adjacent to *r3*. The new workspace is shown in Figure 2 (Right).

We define a special robot action "re-synthesize", which terminates the execution of the hybrid controller and re-generates the controller. "re-synthesize" is used in the same manner as a robot proposition in the requirements specification:

- If you are sensing *regionSensor* then do re-synthesize.

The re-synthesize action is activated when the *regionSensor* returns true.

If the user explicitly indicates which group to add the new region to, the extra indication is allowed as follows:

- If *regionSensor* then do re-synthesize and add into *groupName*.

The following relation is defined: *regionSensor* $\rightarrow$ *groupName*. Later, during the re-synthesis process, if *regionSensor* = *True*,

the corresponding *groupName* is returned to ensure that the correct group is updated with the new region. Furthermore, only specifications applied to this group will need to be rewritten as new LTL formulas. If the robot is capable of distinguishing between different features of the new regions, the detected regions can be added into different groups accordingly. One sensor proposition for each of the groups is necessary, as in Example 1:

- If you are sensing safe-new-region then do re-synthesize and add it into Safe.
- If you are sensing dangerous-new-region then do re-synthesize and add it into Dangerous.

V. Controller re-synthesis during execution

This section describes the algorithms and automated process for automatically re-synthesizing the high-level controller when new areas are found. It addresses the special robot action *re-synthesize*, the need for capturing the current robot state and goal, the detection of a new region using sensors, the process of modifying the discrete abstraction of the workspace and the algorithm for creating and synthesizing a modified LTL formula.

A. The robot action "re-synthesize"

Detection of new region is captured by a Boolean environment proposition, *regionSensor*. The re-synthesis action is captured by a robot proposition. The two propositions are denoted as $s^{new-region}$ and $a^{re-synthesize}$ respectively. The specification "If you are sensing *regionSensor* then do re-synthesize" is captured by the LTL formula:

$$\square(\bigcirc s^{new-region} \Rightarrow \bigcirc a^{re-synthesize})) \tag{2}$$

The "re-synthesize" action is a special action since the low-level controller associated with it terminates the execution of the automaton, and calls the module to rewrite the LTL formula and re-synthesize an appropriate automaton.

The re-synthesis process is shown in Algorithm 1. In line 1, the execution of the current automaton is terminated. The following two propositions are reset: $s^{new-region} = False$ and $a^{re-synthesize} = False$. In lines 2-3 the environment propositions and robot propositions of the current state are recorded, in order to describe the initial state of the robot when resuming the execution. In line 4, the initial condition for the new automaton is obtained and the LTL formulas φ_i^s, φ_i^e for initial conditions are updated. In line 5, the new region proposition is added as: $\hat{\mathcal{Y}} = \mathcal{Y} \bigcup r_{new}$. The process of modifying the workspace and rewriting the LTL formula φ_t^s is discussed in Section V-B. In line 6, the goal number of the liveness requirement toward which the robot is currently moving is recorded. In line 7, re-ordering of the goal requirements is achieved by rewriting the LTL formula φ_g^s as described in Algorithm 2 and Section V-C. In lines 8-9, the updated LTL formula is synthesized and the new automaton is executed.

Algorithm 1 Low-level controller for the *re-synthesize* action

1: Break execution, reset $s^{new-region}$, $a^{re-synthesize}$
2: CurrRobotState $\Leftarrow \gamma(q)$
3: CurrEnvState $\Leftarrow$ values of sensor propositions
4: NewInitState: $\hat{q}_0 \Leftarrow$ CurrRobotState $\wedge$ CurrEnvState
5: Modify discrete abstraction in workspace, get $\hat{\mathcal{Y}}$ (see Section V-B)
6: CurrGoalNum $\Leftarrow \gamma_r(q)$
7: Modify liveness conditions in LTL (see Algorithm 2)
8: Re-synthesize the automaton
9: Load new automaton and resume execution

B. Modifying the discrete abstraction

As defined in Section III, the robot maintains a map of the workspace and expands it on-the-fly. The map maintained by the robot is composed of a list of polygonal regions. At step i, the robot is equipped with a map of the already known area, P^i, as well as its outer boundary, B^i.

The new region sensor repeatedly checks each edge of the current region p^c, the region where the robot is currently located in. If the edge also belongs to the boundary, the new region sensor tests whether there is a new region emanating from it. Defining a new region is possible only if the new regions fulfill the following conditions: the length and the width of the new region, and the width of the edge the new region is emanating from, must be greater than sD.

Since the new region is adjacent to the known map, there must exist a transition boundary between the new region and the known regions. The motion constraint graph is re-generated from the new adjacency relationships between the regions, which is written into the robot safety assumption φ_t^s.

Revisiting Example 1, the change in the known workspace results in an updated φ_t^s:

$$\begin{cases} \square(r_0 \Rightarrow (\bigcirc r_0 \vee \bigcirc r_2)) \\ \wedge \square(r_1 \Rightarrow (\bigcirc r_1 \vee \bigcirc r_2)) \\ \wedge \square(r_2 \Rightarrow (\bigcirc r_2 \vee \bigcirc r_0 \vee \bigcirc r_1 \vee \bigcirc r_3 \vee \bigcirc danger_1)) \\ \wedge \square(r_3 \Rightarrow (\bigcirc r_3 \vee \bigcirc r_2 \vee \bigcirc safe_1)) \\ \wedge \square(danger_1 \Rightarrow (\bigcirc danger_1 \vee \bigcirc r_2)) \\ \wedge \square(safe_1 \Rightarrow (\bigcirc safe_1 \vee \bigcirc r_3)) \end{cases}$$

C. Rearranging the liveness conditions

Given the current goal number, the robot is able to determine which liveness requirement it was pursuing. Therefore, it is able to distinguish between the complete and incomplete goals. The history of the completed high-level tasks is captured by re-assigning the order of the goals. The robot is allowed to first address incomplete liveness requirements, and in addition, to choose the exploration strategies by inserting the new goal at different positions, either first, thereby creating a depth first strategy, or afterwards, creating a breadth first strategy.

The detailed process is shown in Algorithm 2. In lines 1-2, the liveness requirements are classified as complete or incomplete goals. In line 3, the user's predefined group for the

Algorithm 2 Rewriting the LTL formula for liveness requirements

1: CompGoals $\Leftarrow \varphi \in \varphi_g^s$, $gNum(\varphi) <$ CurrGoalNum
2: IncompGoals $\Leftarrow \varphi \in \varphi_g^s$, $gNum(\varphi) \geq$ CurrGoalNum
3: *groupName* $\leftarrow$ *regionSensor*
4: Translate specs with *groupName* into LTL
5: newLiveness $\Leftarrow$ liveness with newRegion in new LTL
6: **if** Depth First Order **then**
7: $\quad \varphi_g^s \Leftarrow$ newLiveness $\wedge$ IncompGoals $\wedge$ CompGoals
8: **end if**
9: **if** Breadth First Order **then**
10: $\quad \varphi_g^s \Leftarrow$ IncompGoals $\wedge$ newLiveness $\wedge$ CompGoals
11: **end if**

new region is loaded. In lines 4-5, the liveness with the relevant *groupName* is translated into LTL, as described in Section IV-A. In lines 6-8, the goals are re-ordered, if following the Depth First strategy, the new goal is put before incomplete goals. In lines 9-11, if following the Breadth First strategy, the new goal is added after the set of incomplete goals.

Revisiting Example 1, assume that when the new region *danger1* is detected, the robot has already visited r_1, r_2, so the priority of the original goals have been changed. The robot is also capable of choosing between either a depth-first order or breadth-first order, which is achieved as follows:

Depth First(Alg 2, ln 6-8):

$$\begin{aligned}&\varphi_g^s = \\ &\square \diamond ((\neg a^{guide}) \Rightarrow danger_1) \\ &\wedge \square \diamond ((\neg a^{guide}) \Rightarrow r_3) \\ &\wedge \square \diamond ((\neg a^{guide}) \Rightarrow r_2) \\ &\wedge \square \diamond ((a^{guide}) \Rightarrow (r_0 \vee r_1))\end{aligned}$$

Breadth First(Alg 2, ln 9-11):

$$\begin{aligned}&\varphi_g^s = \\ &\square \diamond ((\neg a^{guide}) \Rightarrow r_3) \\ &\wedge \square \diamond ((\neg a^{guide}) \Rightarrow danger_1) \\ &\wedge \square \diamond ((\neg a^{guide}) \Rightarrow r_2) \\ &\wedge \square \diamond ((a^{guide}) \Rightarrow (r_0 \vee r_1))\end{aligned}$$

VI. Experiments

The new algorithms and procedures for detecting and adding unknown regions to the synthesized controller were validated in a real environment with a physical robot. The experiments illustrates how high-level tasks are automatically adjusted while the environments are being expanded.

Example 2: Classroom assistant

The robot is looking for students who need to submit assignments to their professors in the workspace depicted in Fig. 3. The robot is wandering through the classrooms until it finds such a student. Once it is handed an assignment, it searches for a professor in the offices until it finds one. If a door opens (new classroom or new office), the robot explores that region, classifies it, and continues executing its task accordingly. If the robot detects a hazardous item in front of *classroom2*, it avoids entering that classroom and it raises its flag as warning. The specifications, given in structured English, are presented in Listing 1.

A. Experimental Setup

The experimental test bed includes a mobile robot which can move autonomously in the environment and detect new regions. Additionally, the robot is able to perform other actions

Listing 1 Specification for the experiments.

```
Environment starts with false
Robot starts with false
Always not (newClassroom and newOffice)
Always not ((newClassroom or newOffice) and hazardous)
Group Classrooms is classroom1
Group Offices is office1
Do flag if and only if you are sensing hazardous
If you are sensing newClassroom then do re-synthesize and
     add to Classrooms
If you are sensing newOffice then do re-synthesize and add
     to Offices
If you are sensing newClassroom or you are sensing
     newOffice then stay
If you are not sensing newClassroom and you are not sensing
      newOffice then do not re-synthesize
If you are not sensing assignment and you are not
     activating re-synthesize then visit all Classrooms
If you are sensing assignment and you are not activating re
     -synthesize then visit all Offices
If you are not sensing hazardous and you are not sensing
     assignment and you are not activating re-synthesize
     then visit classroom2
If you are sensing hazardous then always not classroom2
```

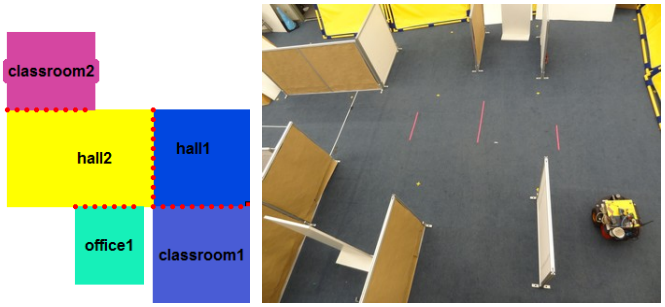


Fig. 3: Left: The initial map is composed of *hall1, hall2, classroom1, classroom2, office1*. Right: field top view

such as sensing objects (e.g., cones) and performing actions (e.g., raising a flag).

The mobile robot used is based on the Pioneer 3-DX mobile robot platform, upgraded with a compact PC on top, and sensors and actuators as follows. For sensing new regions, a Hokuyo URG04-LX Scanning Laser Range Finder was used. The range finder scans at 10 Hz with a field of view of 240 degrees, and angular step size of 0.36 degree. The URG range is approximately 4 meters. The scanner data was overlaid on an occupancy grid with resolution of 10 [cm]. LTLMoP communicated with the robot via wireless communication.

The experiments were conducted in an indoor environment. The Vicon Motion Capture system was used for obtaining accurate pose information for the robot. The Vicon system consists of 24 infrared cameras mounted on a truss attached to the ceiling, allowing for accurate tracking of rigid bodies, marked with reflective markers.

The new-region sensor detects new regions as explained in Section V-B. The new region must be at least sD wide and sD deep relative to the current region edge it is emanating from. The new region sensor scans the occupancy grid for unoccupied cells, starting from the edges of the current region. It only scans edges which are contained in boundary edges. If the sensor finds an opening with minimal width, it continues

to scan parallel lines on the grid, advancing outward from the current region. The scan continues until a line is not wide enough. During the scan, the start and end points of each line are recorded, along with its depth, such that at the end of the scan, the new region can be defined according to the maximal width, depth or area, as long as it fulfills the minimal size requirements. The new-region sensor distinguishes between a small new region and a large new region by checking whether either the width or the depth of the new region exceeds a certain threshold. In the reported experiment, the threshold was set to $2sD$, $D = 0.46$ [m] and $s = 1.5$. In the experiments scenario, the small regions are referred to as offices and the large regions as classrooms. In order to maintain the guarantees, the new-region sensor is implemented conservatively. The **sensors** and robot propositions used are:

Sensors: newClassroom, newOffice, hazardous, assignment

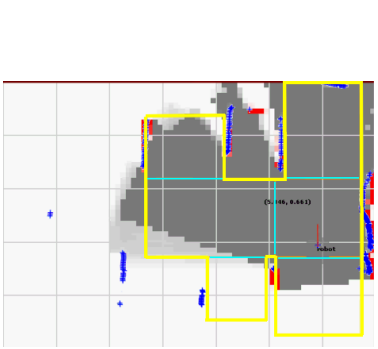
- **newClassroom/newOffice**: The two propositions are attached to the same new-region sensor. If the detected new region is larger than the threshold, the sensor returns true in **newClassroom**. Otherwise **newOffice** returns true.
- **hazardous**: A red cone served as a hazardous item. The cone detector is composed of a software blob detector, which receives the image frames from a video camera mounted on the mobile robot.
- **assignment**: The object sensor, which is a cup that can sense the presence or absence of objects within, was used to sense the presence of an assignment.

Actuators: flag, re-synthesize:

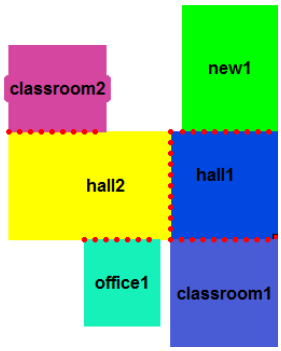
- **flag**: The robot action is raising a flag if the proposition value is true, or lowering a flag if false.
- **re-synthesize**: When the proposition is true, it activates the local controller that terminates the execution and starts the re-synthesis process.

B. Description of the experiment

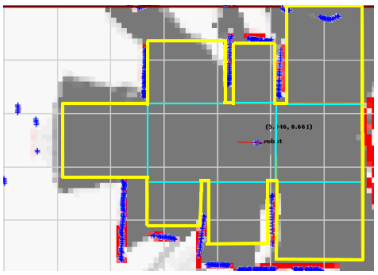
The robot starts in *classroom1*, and heads toward *hall1*. As soon as it enters *hall1*, it detects *new1* (Fig. 4a,4b) and classifies it as a large room, thus adding it to the group *classrooms* and then re-synthesizing. It visits region *new1* and *classroom1* according to the specifications, then it heads toward *classroom2*. On its way, the robot passes through *hall2*, where it detects *new2* (large) (Fig. 4c,4d), adds it to group *classrooms*, and re-synthesizes. Immediately afterwards, it detects *new3* (small) (Fig. 4c,4d), adds it to group *offices* and re-synthesizes. Afterward, it continue to classroom *new2*, there, a student hands the robot an assignment (Fig. 4e), and the robots start searching for the professor to submit the assignment to in the offices *new3* and *office1*. Since the robot does not find the professor in those offices, it continues to search the offices until a door to a new office is opened, the robot detects *new4* (Fig. 4c,4d), adds it to group *offices* and re-synthesizes. The robot moves to the *new4* office, there it finds the professor and hands him the assignment (Fig. 4f). Since now the robot does not have assignments to deliver, it continues to search for students in the classrooms, starting with *classroom2* which it did not visit, yet.



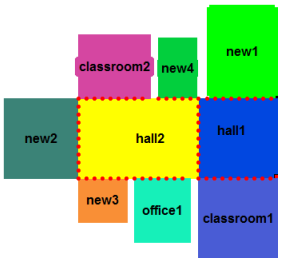
(a) After detecting the classroom *new1*. The boundary is in yellow, the regions are separated by turquoise lines.



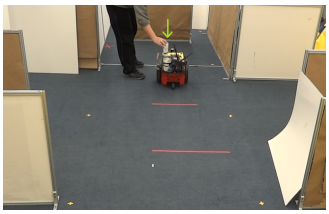
(b) LTLMoP map after adding classroom *new1*.



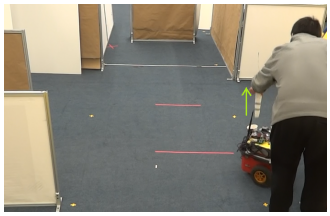
(c) After adding classroom *new2* and offices, *new3,new4*.



(d) LTLMoP map after the addition of the office *new4*.



(e) Student submit assignment to robot in *classroom2*.



(f) Robot submit assignment to professor in office *new4*.

(g) Avoiding *classroom2* and raising the warning flag as a response to the hazardous cone.

(h) Visiting the office *new3* with assignment in the cup

Fig. 4: Assignment collecting and handing experiment

The robot continues its correct execution according to the specifications, continuously searching for students with assignments and handing them to professors. After some time, a hazardous item appears in front of *classroom2* (Fig. 4g). When the robot is in *hall2* and heading to *classroom2*, it detects the hazardous item, consequently, it raises the warning flag (Fig. 4g), skips *classroom2* and continues to *new2* instead. After the hazardous item exits the view of the sensor, the robot lowers the flag, and continues execution with correct behavior.

From the experiment execution [19], it is evident that under the assumptions of Section III, correct behavior is guaranteed even when adding regions that are unknown a priori. Moreover, the robot correctly classified the new regions to *offices* and to *classrooms* according to their sizes, and added them to the specifications accordingly.

VII. Conclusions and Future Work

This paper describes an approach to guaranteeing high-level robot behaviors in partially known and continuously updated workspaces through re-synthesis. This includes modifying the discrete abstraction of the workspace on-the-fly, preserving the robot's state and history of task progress and sequencing the liveness requirements such that the robot exhibits the desired behavior. The proposed methods and algorithms were successfully tested in experiments. Moreover, it was shown that specific features of the new detected regions can be used to classify them into different groups, enabling the automation of redefinition of the high level task accordingly, and maintaining the correct execution.

The process of changing the controller is automated and a new controller is guaranteed to be correct with respect to the specifications. Furthermore, if a controller cannot be generated, it means that the task can no longer be achieved in the modified workspace and LTLMoP can provide explanations for the failure.

Ongoing research is focused on improving the new region detection, such that the under approximation will be closer to the exact free area. Another research direction aims to deal with the changes in known regions, such as the unpredictable appearance of obstacles which may result in re-decomposition of the regions, or even in unrealizable specification.

Acknowledgments

The authors gratefully acknowledge the contribution of Cameron Finucane, Eric Sample, and Nyk Lotocky to the development process and to the implementation.

References

[1] C. Belta and L.C.G.J.M. Habets. Constructing decidable hybrid systems with velocity bounds. In *IEEE Conference on Decision and Control*, Bahamas, 2004.

[2] A. Bhatia, L.E. Kavraki, and M.Y. Vardi. Sampling-based motion planning with temporal goals. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2689–2696, 2010.

[3] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, Cambridge, Massachusetts, 1999.

[4] D.C. Conner, A.A. Rizzi, and H. Choset. Composition of Local Potential Functions for Global Robot Control and Navigation. In *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, pages 3546 – 3551, Las Vegas, NV, October 2003.

[5] H.S.M. Coxeter. *Regular polytopes*. Dover Publications, 1973.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000.

[7] A. Elfes. Sonar-based real-world mapping and navigation. *Robotics and Automation, IEEE Journal of*, 3(3): 249 –265, June 1987.

[8] G.E. Fainekos, A. Girard, H. Kress-Gazit, and G.J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343 – 352, 2009.

[9] C. Finucane, G. Jing, and H. Kress-Gazit. LTLMoP : Experimenting with Language , Temporal Logic and Robot Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems.*, pages 1988–1993, Taipei, Taiwan, 2010.

[10] B. Johnson and H. Kress-Gazit. Probabilistic Analysis of Correctness of High-Level Robot Behavior with Sensor Error. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[11] S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *IEEE Conference on Decision and Control*, pages 2222–2229, 2009.

[12] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from LTL specifications. In *Hybrid Systems: Computation and Control*, volume 3927, pages 333–347, 2006.

[13] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Translating Structured English to Robot Controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.

[14] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Temporal Logic based Reactive Mission and Motion Planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[15] M. Lahijanian, J. Wasniewski, S.B. Andersson, and C. Belta. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3227–3232, May 2010.

[16] S. R. Lindemann and S. M. LaValle. Smooth Feedback for Car-Like Vehicles in Polygonal Environments. In *IEEE Conference on Robotics and Automation*, 2007.

[17] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of Reactive(1) Designs. In *VMCAI*, pages 364–380, Charleston, SC, Jenuary 2006.

[18] V. Raman and H. Kress-Gazit. Analyzing unsynthesizable specifications for high-level robot behavior using ltlmop. In *CAV*, pages 663–668, July 2011.

[19] S. Sarid, B. Xu, and H. Kress-Gazit. Guaranteed High-level robot behavior while exploring unknown workspace, 2012. URL http://www.youtube.com/watch?v=oiF4Wt8xgio. Video (YouTube).

[20] T. Wongpiromsarn, U. Topcu, and R.M. Murray. Receding Horizon Temporal Logic Planning. In *IEEE Conference on Decision and Control (CDC)*, pages 5997–6004, December 2009.