LTL TASK DECOMPOSITION FOR 3D HIGH-LEVEL PATH PLANNING IN KNOWN AND STATIC ENVIRONMENTS ⋆

Sofia Hustiu [*,**] **Ioana Hustiu** [*] **Marius Kloetzer** [*] **Cristian Mahulea** [**]

[*] *Dept. of Automatic Control and Applied Informatics, "Gheorghe Asachi" Technical University of Iasi, Romania; {hustiu.sofia, hustiu.ioana, kmarius}@ac.tuiasi.ro*

[**] *Engineering Research Institute of Aragon (I3A), University of Zaragoza, Spain; cmahulea@unizar.es*

Abstract: The paper addresses the problem of planning the motion of a team of drones such that a co-safe Linear Temporal Logic (LTL) formula is accomplished. The high-level formula can include visits or avoidance of some known and static regions of interest from the 3D environment. The team mission is decomposed into independent tasks that can be accomplished by each drone, while the workspace is abstracted by a cell decomposition algorithm based on rectangular cuboid partitions. An optimization problem for the task assignment yields the tasks to be fulfilled by each drone, and independent trajectories are automatically obtained. The contributions include an algorithm for decomposing the co-safe LTL specification into independent tasks, a recursive cell decomposition method for 3D environments with regions of interest rather than obstacles, and an overall planning procedure that returns a trajectory for each drone. The independent tasks imply that the drones fulfilling them do not need to constantly communicate, the flight of each agent relying on positioning sensors for following the predefined path. Simulations and comparative studies based on different partitions and scenarios are included.

Keywords: mobile robots, path planning, discrete event systems

1. INTRODUCTION

The interest of high-level path planning for multirobot systems has increased in the last years, especially since the technological needs evolved and become more present in people's live. The high-level specifications are important when a certain execution of tasks is desired (Belta et al., 2007), and not only the achievement of one target as in the classical navigation problem in robotics (LaValle, 2006). One of the most used formalisms to specify high-level specifications is through Linear Temporal Logic (LTL) formulas (Ding et al., 2014; Guo and Dimarogonas, 2015; Kloetzer and Mahulea, 2015; Lacerda and Lima, 2019; Lindemann et al., 2019; Moarref and Kress-Gazit, 2020), which has been used by Computer Science community from decades ago for the verification of properties of discrete systems. Other high-level formulas can be used to specify the scope of a team of robots, such as: probabilistic Computation Tree Logic (Li and Fu, 2019), parametric LTL (Alur et al., 2001). Another importance is represented by the high-level specification, where time constraints are concerned, for instance: Time Window Temporal Logic (TWTL) (Mosca et al., 2019), Metric Interval Temporal Logic (MITL) (Nikou et al., 2018) or Signal Temporal Logic (STL) (Yang et al., 2020). The present paper focuses on the benefits of using a co-safe LTL specification as a general framework for specifications, while the resulted plans can constitute inputs for available simulators (Gânsari and Buiu, 2017; Dosoftei et al., 2020; Mahulea et al., 2020) or for real-time experiments.

The motivation of using high-level specifications is twofold: (a) specifications that require infinite length trajectories, e.g., surveillance tasks, and (b) complex tasks in which conditionals and/or a given order of performing tasks are needed, e.g., "first visit region A or B and then region C, while always avoiding region D". There are many possible applications of using high-level specifications. For example, in the case of transportation and delivery of goods when different possible destinations are available for a good; or in the case of factory of the future where the Automated Guided Vehicles (AGVs) should coordinate with the human workers to transport intermediate products in the factory; or in the agriculture where a team of drone should perform a task depending on some measurements acquired by other drones; or in applications based on Search and Rescue (SAR) missions, where drones play a crucial part in visiting essential regions of interest.

In the case of high-level specifications, almost all proposed solutions are based on a *discretization of the state space*, i.e., the environment where the robots are moving. This allows the high-level planners to compute trajectories that include some necessary intermediate points that should be reached, while possibly avoiding other positions. For solving a high-level path planning problem where the specification is an LTL formula, the formula is usually translated to a Büchi or Rabin automaton that *accepts runs* of infinite lengths, each of these runs containing a sequence of tasks that would satisfy the

⋆ Corresponding author: Ioana Hustiu.

The work was partially supported by the CCDI-UEFISCDI grant of the Romanian Ministry of Research and Innovation, project number PN-III-P2-2.1-PTE-2019-0731, at the Technical University of Iasi.

formula. Having the automaton of the specification, the next step is to combine it with the discrete model of the team of robots. This model is obtained by the discretization of the continuous state space in which the robots are moving (e.g., 2D in the case of ground robots or 3D in the case of drones) and it abstracts the possible movements of each agent between adjacent regions (cells) of the workspace.

The result of the discretization process is an automaton describing the movement capabilities of one agent, and in order to obtain the model of the team, the product of a number of identical automatons may be used. This process could induce a very large number of states in the team model, resulting in the so-called state explosion problem that would yield intractable computation complexity. Finally, having both discrete systems (one of the specification and one of the team), they are combined by using a particular product (using the property that the outputs of the team model are inputs in the specification model) and a search algorithm is applied on the global model to compute a path from the initial to final state.

In the last years, many efforts have been put in obtaining computational attractive solution for the high-level path planning by overcoming the mentioned state explosion problem. The first one is based on using Petri nets for modeling the team (Mahulea et al., 2020; Kloetzer and Mahulea, 2020; Mahulea and Kloetzer, 2018; Lacerda and Lima, 2019; Mahulea et al., 2021), since this type of models are scalable with respect to the number of robots. In particular, the size of the team model is comparable with the size of the corresponding automaton model of one robot. Adding one robot to the team does not change the model structure, but only the initial (numerical) state. Another solution is based on decomposing the LTL specification into sub-formulas (or tasks) that can be achieved in parallel (independently) by the robots (Schillinger et al., 2018; Tumova and Dimarogonas, 2016). By using this last solution, it is not necessary to compute the full team model, and furthermore, no additional synchronizations among the robots during their movements are required. This independence has beneficial aspects in terms of requiring less sensing and communication devices on the drones.

Recently, the developments moved in 3D domains, where mobile robots are represented by Unmanned Aerial Vehicles (UAVs). The appearing challenge is due to the 3D discretization approaches, which are more complex than the ones from 2D. The most utilized techniques for path planning in 3D are: rapidly exploring random tree (RRT) (Sahil, 2019), cell decomposition (Vespa et al., 2018; Lupascu et al., 2019), probabilistic graph (Sanchez-Lopez et al., 2019), simultaneous localization and mapping (SLAM) (Azim, 2013; Hafez, 2015).

In this paper we consider an LTL formula given for a whole team of drones evolving in a static 3D environment. The formula expresses the required mission over visiting or avoiding some 3D regions of interest. For overcoming the state explosion problem, our method is related to the decomposition of the LTL formula in independent tasks. The first contribution is to propose an algorithm returning the decomposition of the LTL specification for the team in individual tasks, each task being achievable by one agent. The algorithm is based on theoretical results presented in (Tumova and Dimarogonas, 2016), but, as far as we know, there is no formal method to automatically obtain such a decomposition. Our algorithm is based on the conference version we proposed in (Hustiu et al., 2020), and it works on a particular sub-class of LTL formulas, namely co-safe LTL (which induce finite lengths or accepting runs). The work embodied in this paper extends the outcome from (Hustiu et al., 2020) in two directions: first, it considers a 3D environment suitable for drone applications, and therefore it develops a partitioning and modelling technique suitable for this scenario. Second, the individual agent models enable the computation of individual trajectories via graph searches rather than using Petri net models and optimization problems as in (Hustiu et al., 2020), and hence the computational complexity is reduced.

The second contribution refers to obtaining a computationally tractable method for 3D planning in the given scenario. For this, we use discretization approaches for 3D environments into rectangular cuboid cells. Two distinct techniques are considered: (a) grid (all the cells have the same size and volume) and (b) octtree (the cells have different size and volume). The latter technique extends the one from (Lupascu et al., 2019) (which works only for obstacles to be avoided), and it also represents the regions of interest, rather than removing from the model such parts of the environment. Formal details are given for LTL decomposition and for environment abstraction, and based on the two mapping techniques of the 3D domain, we provide a numerical evaluation of the proposed LTL decomposition and planning.

The proposed approach implies a contribution in the sense of fulfilling a complex mission for the whole team, by having drones with a reduced number of communication and sensing devices. This is because once each drone receives its computed path from a central unit, it can follow it independently of other robots, by using on-board positioning sensors.

The paper is structured as follows: Sec. 2 describes the problem formulation for our study. Our main contributions regarding this research are highlighted in the next three sections: Sec. 3 contains the algorithmic solution for the decomposition of the co-safe LTL specification into independent tasks. The automaton representing the considered environment is defined and constructed in Sec. 4. Sec. 5 assigns the independent tasks to each UAV in an optimal way and uses the drone automaton to produce the trajectories to be followed. For all the mentioned sections, formal details are accompanied by small examples. Sec. 6 contains more complex examples and a detailed comparative study of the influence of environment representation on the resulted trajectories. In the end, conclusions are provided together with future work proposal.

2. PROBLEM DEFINITION

Let us assume a team of identical robots denoted $\mathcal{R} = \{r_1, r_2, \ldots, r_{|\mathcal{R}|}\}$ that evolve in an 3D environment. The agents are assumed omni-directional and of negligible size. A typical example is that the robots are represented by small drones. The environment is a rectangular cuboid $E = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}] \subset \mathbb{R}^3$ and in E there exist some convex polyhedral regions of interest labeled with elements of set $Y = \{y_1, y_2, \ldots, y_{|Y|}\}$. The environment is assumed known and static, in terms that the vertices of each region from set Y are fixed and given, and the initial deployment of robots is also known. The assumption of omni-directional robots is natural in case of drones, and the assumption of their negligible size can be achieved by reducing each robot to a point and

correspondingly enlarging regions from Y (Choset et al., 2005; LaValle, 2006).

The work is devoted to develop a planning strategy for our multi-robot system based on a high-level specification. Due to this top-down approach, we are not interested in the actual low-level control routines for each robot, which would depend on the specific dynamic model of each agent. Instead, we provide algorithm for the path-planning part, which constitutes the base for the upcoming path-following controllers. This motivates our above assumptions for simplifying the agents to omni-directional point robots. Referring to the applicability of the results to be presented, the planning part is centralized, being computed by a unit that returns individual trajectories. It is assumed that the central unit can communicate with the drones only when they are in their initial position (e.g., in charging stations that are also equipped with communication devices). Once each drone receives its movement plan from this central unit, the movement of the team is performed in a decentralized manner, since each robot independently follows the designated trajectory. The decomposition of the global task into independent ones guarantees that no intermediate coordinations or synchronizations are required, with possible exception of some local collision-avoidance rules in case that two drones fly dangerously close to each other. During movement, each drone would use localization sensors (as an indoor/outdoor GPS system) in order to execute some predefined trajectory-tracking control laws - e.g., see (Rubí et al., 2019) for a selection of such routines. Proximity and communication sensors would also be used in real-time experiments only for avoiding collisions among agents.

Our goal is to derive *independent* trajectories for the robots such that their overall movement fulfills a specification given for the whole team. The "independent" attribute means that the each robot can move without having to communicate or to synchronize with other team members, as for example in centralized approaches from (Kloetzer and Mahulea, 2020). The global specification will be given as a so-called co-safe $\mathrm{LTL_X}$ formula over set Y, this class being able to express rich behaviors including visits or avoidance of specific regions of interest.

In short, an LTL formula is recursively defined over the set of atomic propositions Y, by using (i) the standard Boolean operators ($\neg$ - negation, $\vee$ - disjunction, $\wedge$ - conjunction, $\Rightarrow$ - implication, and $\Leftrightarrow$ - equivalence) and (ii) some temporal operators with straightforward meaning ($\mathcal{U}$ - until, $\Diamond$ - eventually, $\Box$ - always, $\bigcirc$ - next). Extensive details on this language can be found in studies as (Clarke et al., 1999; Baier and Katoen, 2008). The $\mathrm{LTL_X}$ subclass lacks the "next" symbol, this operator being meaningless for continuous trajectories (as the ones generated by drones). In this paper we use *syntactically co-safe* $\mathrm{LTL_X}$ formulas (Kupferman and Vardi, 2001). While the satisfaction of LTL and $\mathrm{LTL_X}$ formulas has to be interpreted on infinite strings with elements from 2^Y, a co-safe LTL formula is satisfied by a finite sequence with elements from the power-set 2^Y (called prefix) continued by any infinite string over 2^Y. Semantically, an $\mathrm{LTL_X}$ formula including only the temporal symbols $\Diamond$ and $\mathcal{U}$ when written in positive normal form (i.e., $\neg$ can precede only elements of Y) is syntactically co-safe (Kupferman and Vardi, 2001). In the remainder of this work we simply denote as LTL the co-safe $\mathrm{LTL_X}$ specifications that are used. As short examples of such formulas, $\varphi = \Diamond(y_1 \vee y_2)$ requires that at least one of regions y_1 and y_2 is eventually visited by a robot, while $\varphi = \neg(y_1)\mathcal{U}y_2$ imposes that y_1 should be avoided until y_2 is visited.

Any LTL formula over set Y can be transformed into a Büchi automaton that accepts all and only the input strings from 2^Y that satisfy the formula (Wolper et al., 1983). The Büchi automaton can be obtained by feeding existing software tools as (Gastin and Oddoux, 2001) with an LTL formula φ.

Definition 1. The Büchi automaton corresponding to an LTL formula over the set Y is denoted by $B = (S, S_0, \Sigma_B, \rightarrow_B, F)$, where:

- S is a finite set of states;
- $S_0 \subseteq S$ is the set of initial states;
- $\Sigma_B = 2^Y$ is the set of inputs;
- $\rightarrow_B \subseteq S \times \Sigma_B \times S$ is the transition relation;
- $F \subseteq S$ is the set of final states. ■

For $s_i, s_j \in S$, we denote by $\varrho_{\mathcal{B}}(s_i, s_j)$ the set of all inputs of B that enable transition from s_i to s_j. An infinite input word (sequence with elements from Σ_B) is *accepted* by B if the word produces at least one sequence of states of B that visits set F infinitely often. However, for syntactically co-safe LTL formulas, an input word is accepted by B if it starts with a finite sequence (prefix) that drives B from S_0 to the set of final states F (while any continuation of the prefix cannot violate the formula). Thus, for satisfying a co-safe formula φ we have to find a finite sequence with elements from 2^Y that drives the formula's Büchi automaton to one of its final states.

Summing up, the *problem* we here solve receives as inputs the environment E, regions Y, robots $\mathcal{R}$ (each with a different initial position) and formula φ. The solution we seek is a trajectory for each drone, such that when the robots individually follow their trajectories the formula φ is accomplished.

Example 1. For easier following the problem and its solution, we provide a simple example that will be revisited throughout the next sections. Figure 1 depicts an environment $E = [0, 80] \times [0, 50] \times [0, 100]$ with four disjoint regions of interest. Two robots evolve in this environment, their initial deployment being marked with the red (for r_1) and blue (for r_2) circles. The imposed specification is

$$\varphi = \Diamond y_1 \wedge \neg y_2 \mathcal{U} (y_3 \vee y_4), \tag{1}$$

which requires that (1) region y_1 is eventually reached, and (2) region y_2 is avoided until either of the y_3 and y_4 regions is eventually visited. Intuitively, if a robot solves part (1) while it avoids y_2 and the other solves part (2) of φ, they can do this in a distributed manner, i.e. without any synchronization or communication. Contrary, if we change the last parenthesis to $(y_3 \wedge y_4)$, both robots would have to cooperate in fulfilling it, e.g. if a robot reaches y_3 it should wait there until the other robot visits the disjoint region y_4. The next three sections provide an automated method to decide and solve the situations of distributing LTL specifications. As mentioned, this planning method is running on a central unit that sends the independent paths to robots before their actual movement. All the algorithms to be discussed were implemented in MATLAB, and the included computation times were obtained on a system with i7 8^{th} gen. CPU and 8 GB RAM.

As the ideas leading to problem's solution, instead of finding a single finite sequence over 2^Y that satisfies the formula, we look for individual robot sequences that can be independently followed. The trajectory of each robot will show the sequence

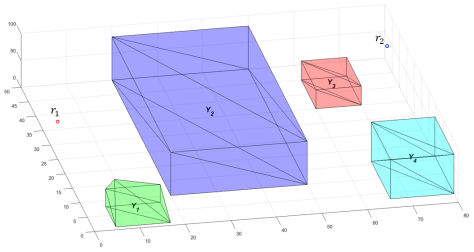


Fig. 1. Environment with four regions of interest and initial positions of two robots (r_1 - red, r_2 - blue).

of regions visited by the agent. Due to independent movements of robots and since we do not account the exact time when each robot enters (visits) a region, the team can actually produce a multitude of possible strings over 2^Y. We have to make sure that any such possible string satisfies the imposed formula, and for this we first decompose the specification as in Sec. 3. Then, we abstract the environment and the possible motion of each robot into an automaton, for this adapting a cell decomposition method in Sec. 4. Finally, we assign parts of the decomposed formula to robots and we search trajectories in their abstractions as in Sec. 5.

3. TASK DECOMPOSITION

For distributing a given LTL mission φ among team members, the goal of this section is to automatically obtain decompositions of φ. Each decomposition is a set containing subformulas, denoted as *tasks*, that have the independence property and can be executed by a single robot. Collectively, the tasks must assure two properties: *independence* and *fullness*. As stated previously, the first property is referring to the constraint that we assume no synchronizations between the drones, therefore the independence implies that each task can be accomplished by only one robot. This also implies that a task φ_i should not violate another task φ_j , where $\{\varphi_1, \ldots, \varphi_n\}$ is the decomposition set, $j \neq i$, $j = 1, \ldots, n$. The second property implies that the global mission φ becomes accomplished once all tasks are executed.

Algorithm 1 describes a way of obtaining decomposition sets for a mission φ. The LTL formula φ is first converted into a Büchi automaton $\mathcal{B}$ using the existing software from (Gastin and Oddoux, 2001). Besides input $\mathcal{B}$, Algorithm 1 needs two additional inputs, a partition P of the environment and an observation map h showing the partition regions that intersect with regions of interest from Y. These two inputs will be algorithmically constructed in Sec. 4, and for easier understanding we mention that set O from line 1 contains all observations (regions of interest from Y) that can be generated by a single robot.

The beginning of Algorithm 1 trims the Büchi automaton such that all the observations that define transitions in $\mathcal{B}$ can be enabled by a single robot (lines 1-3). The purpose for this is to be able to decompose the mission φ into independent tasks, which are basically elements from Σ_B. Thus, we will have the guarantee that a robot can perform a specific task (since it can generate observations from trimmed $\mathcal{B}$), whereas without this trimming more robots might have to cooperate for enabling a single transition from set $\rightarrow_B$. The trimming procedure is here adapted from (Kloetzer and Mahulea, 2020), with the mention that in (Kloetzer and Mahulea, 2020) it was designed for cooperating robots rather than individual ones.

Further, in lines 4-11, all loopless accepted runs from $\mathcal{B}$ are found by using k-shortest path algorithm (Yen, 1971) and stored in a set $Runs$. For this, we consider each pair (s_0, s_f) of initial and final states, we initially set k to be the number of states of $\mathcal{B}$, i.e. $|S|$ (first call of line 8) and we use the k-shortest path algorithm with these inputs (line 9). If there are exactly k loopless paths returned, we increase k (on subsequent iterations of line 8) until we are sure that set $|Runs_{s_0 \rightarrow s_f}|$ includes all possible paths from s_0 to s_f. Thus, when finishing the loop from lines 5-11, set $Runs$ includes all loopless accepted runs of $\mathcal{B}$.

Next, a run ρ is selected from $Runs$ (line 13) and all the transitions from ρ are stored. Between lines 17-22 we verify if all possible permutations of these transitions are feasible runs in $\mathcal{B}$, in the sense that they would conduct from the same initial state to the same final state as the selected run ρ. If the previous statement is true, then we can affirm that a decomposition set $Decomp_\rho$ was found and the observation set corresponding to each transition becomes a task. In other words, each task from the identified decomposition is a subset of 2^Y that contains observations realizable by a single robot. These tasks guarantee the fullness property since ρ is an accepted run of $\mathcal{B}$.

To ensure the independence of tasks we must also take into account the self-loops of states of $\mathcal{B}$. Therefore, a constraint is imposed such that the self-loops of all states that have the same output transition have to be replaced with the intersection of all self-loops of these states. Through lines 23-25 we satisfy this necessary constraint, so that a task cannot violate another part of the main goal φ. Additionally, the selected feasible run ρ and its permutations are removed from the set $Runs$ (lines 14 and 21) to avoid unnecessary iterations pertaining the same tasks.

Algorithm 1 has a high number of iterations, as implied by the given pseudo-code. The trimming of the Büchi automaton has linear complexity on the number of transitions in $\rightarrow_B$, while the size of $\mathcal{B}$ depends on the imposed LTL specification. The k-shortest algorithm is reiterated on lines 5-11 for an initially unknown number of times that is imposed by the structure of $\mathcal{B}$, but each of these iterations has a pseudo-polynomial complexity (Yen, 1971). From the tests we performed, there are normally less than 5 iterations of lines 7-10, while S_0 and S_F usually include up to two states. Although there are an order of $|Runs|^2$ iterations imposed by lines 12 and 17, each of these iterations includes simple set operations, and the cardinality of $Runs$ is greatly decreased due to trimming $\mathcal{B}$, rather than considering its initial structure. Consequently, the trimming - which is necessary for obtaining independent tasks - also alleviates the necessary computations. Complexity of Algorithm 1 does not depend on the number of robots $|\mathcal{R}|$.

Algorithm 1 is not complete in terms of obtaining all possible decomposition sets, because it does not account possible sequences of more tasks in a specific order that could be latter assigned to a single agent, while guaranteeing the independence and fullness properties.

***Example** 2.* We advance the Example 1 on the basis presented in this section. The Büchi automaton corresponding to formula φ from (1) is given in Figure 2(a). As noted, $\mathcal{B}$ should be trimmed before applying the task decomposition algorithm. In this context, observations as $y_1 \wedge y_3$ or $y_1 \wedge y_4$ are redundant since these regions are disjoint and a drone cannot be at the same time in y_1 and in y_3 or y_4. Similarly, observation that enable the transition from s_0 to s_2 becomes y_1, because y_1 and

Algorithm 1 Mission decomposition

Input : Büchi automaton $\mathcal{B}$, partition P and observation map h
Output: Feasible decompositions $TaskSet$

1 Compute $O = \bigcup_{p \in P} h(p)$
2 **for** $(s_i, \rho(s_i, s_j), s_j) \in \rightarrow_B$ **do**
3 $\quad \rho(s_i, s_j) = \rho(s_i, s_j) \cap O$
4 $\quad$ Initialize $Runs := \emptyset$
5 $\quad$ **for** $s_0 \in S_0$ *and* $s_f \in F$ **do**
6 $\quad\quad k := 0$
7 $\quad\quad$ **repeat**
8 $\quad\quad\quad k := k + |S|$
9 $\quad\quad\quad$ Let $Runs_{s_0 \rightarrow s_f} := k\_shortest\_path(k, s_0, s_f)$
10 $\quad\quad$ **until** $|Runs_{s_0 \rightarrow s_f}| < k$;
11 $\quad\quad Runs := Runs \cup Runs_{s_0 \rightarrow s_f}$
12 **while** $Runs \neq \emptyset$ **do**
13 $\quad$ Choose a run $\rho \in Runs$
14 $\quad Runs := Runs \setminus \{\rho\}$
15 $\quad$ Compute $Decomp_\rho := \bigcup_{i=1}^{|\rho|-1} \{\varrho_{\mathcal{B}}(\rho(i), \rho(i+1))\}$
16 $\quad$ Set $counter := 1$
17 $\quad$ **for** $\gamma \in Runs$ **do**
18 $\quad\quad$ **if** $(|\gamma| = |\rho|)$ **then**
19 $\quad\quad\quad$ Compute $Decomp_\gamma := \bigcup_{i=1}^{|\gamma|-1} \{\varrho_{\mathcal{B}}(\gamma(i), \gamma(i+1))\}$
20 $\quad\quad\quad$ **if** $(\,(Decomp_\rho \setminus Decomp_\gamma) = \emptyset\,)$ **then**
21 $\quad\quad\quad\quad Runs := Runs \setminus \{\gamma\}$
22 $\quad\quad\quad\quad counter := counter + 1$
23 $\quad\quad\quad\quad$ **for** $\rho(i) \in \rho$ *and* $\gamma(j) \in \gamma$ *such that sets* $\varrho_{\mathcal{B}}(\rho(i), \rho(i+1))$ *and* $\varrho_{\mathcal{B}}(\gamma(j), \gamma(j+1))$ *are identical* **do**
24 $\quad\quad\quad\quad\quad$ Set $\varrho_{\mathcal{B}}(\rho(i), \rho(i)) := \varrho_{\mathcal{B}}(\rho(i), \rho(i)) \cap \varrho_{\mathcal{B}}(\gamma(j), \gamma(j))$
25 $\quad\quad\quad\quad\quad$ Set $\varrho_{\mathcal{B}}(\gamma(j), \gamma(j)) := \varrho_{\mathcal{B}}(\rho(i), \rho(i)) \cap \varrho_{\mathcal{B}}(\gamma(j), \gamma(j))$
26 $\quad\quad\quad\quad$ **if** $(\,counter = (|\rho|!)\,)$ **then**
27 $\quad\quad\quad\quad\quad Decomp_\rho$ is a feasible decomposition; append it to $TaskSet$
28 **if** $TaskSet$ *is empty* **then**
29 $\quad$ Mission cannot be decomposed

y_2 are disjoint. Figure 2(b) shows the trimmed Büchi automaton obtained after lines 1-3 of Algorithm 1. In this case, the set of accepted runs obtained after line 11 is $Runs = \{\rho_1 = s_0, s_2, s_3; \rho_2 = s_0, s_1, s_3\}$. The construction of $\mathcal{B}$, its trimming and the computation of set $Runs$ took in our implementation less than 0.22 seconds.

At the first iteration in Algorithm 1, ρ_1 is chosen on line 13 and the decomposition $Decomp_{\rho_1} = \{\{y_1\}, \{y_3 \vee y_4\}\}$ is obtained on line 15. Since ρ_1 was eliminated after being selected, only ρ_2 can be now considered on iteration starting on line 17, and in this case the decomposition set $Decomp_{\rho_2} = \{\{y_3 \vee y_4\}, \{y_1\}\}$ is computed on line 19. The decomposition sets are identical (line 20), and condition from line 26 becomes accomplished, thus one decomposition is found, namely $\{\{y_1\}, \{y_3 \vee y_4\}\}$. To guarantee that tasks are not violating the mission during their independent executions, the self-loops of the states must be taken into account as previously explained, this being handled on lines 23-25. Therefore, the self-loop of s_1 becomes $\neg y_2$ (as the self-loop of s_2), and this implies that the region y_2 will not be visited along any trajectory. For this simple example, Algorithm 1 returned the formula decomposition in about 0.04 seconds. ■

4. ABSTRACTION FOR 3D ENVIRONMENT

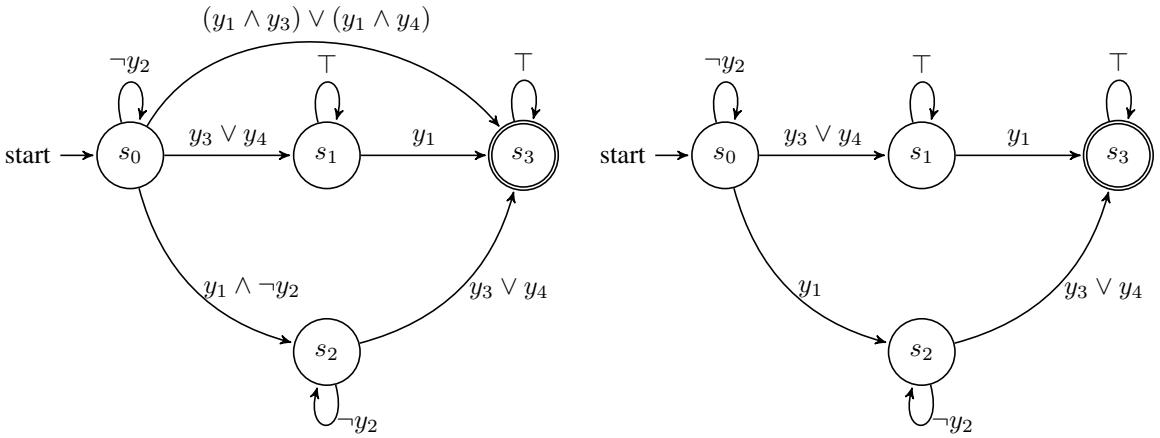
The goal of this section is to construct for each robot r an automaton denoted by T_r. This automaton abstracts the drone's possible motions in environment E into a form suitable for connections with the Büchi automaton corresponding to an LTL formula over set of regions Y.

***Definition** 2.* For each $r \in \mathcal{R}$, the automaton T_r has the structure $T_r = (P, p_{0r}, Adj, 2^Y, h, mixed, Waypoint)$, where:

- P is the set of states, being a partition of E whose elements are also denoted by places or cells. Each element $p \in P$ will be a rectangular cuboid, the intersection of any two different places has volume zero ($Volume(p_i \cap p_j) = 0$, $\forall p_i \neq p_j$), and $\cup_{p \in P} p = E$;
- $p_{0r} \in P$ is the initial state, i.e. the place in which drone r is initially deployed;
- $Adj : P \times P \rightarrow \{0, 1\}$ is the adjacency relation, with $Adj(p_i, p_j) = 1$ if $p_i \cap p_j$ is a set with area different than zero. I.e., two cells are adjacent if they share an entire part of their facets, such that a robot can pass through this part for going from p_i to p_j without intersecting any other cell. Conversely, if two cells intersect only in one point or have only a common line segment, they are not adjacent since enforcing that a drone passes exactly through that intersection would be impossible in real scenarios;
- 2^Y is the set of observations;
- $h : P \rightarrow 2^Y$ is the observation map, showing the regions of interest intersected by partition element p, i.e., $h(p) = \{y \in Y \,|\, Volume(p \cap y) \neq 0\}$. Thus, $h(p) = \emptyset$ if p does not intersect with any region of interest. Note that we consider the volume of intersection based on the same ideas as before, of generating an observation when the drone's position is strictly inside a region from Y;
- $mixed : P \rightarrow \{0, 1\}$ is a flag showing if p lies entirely or only partially inside regions from $h(p)$, i.e. $mixed(p) = 0$ if $h(p) = \emptyset$ or $p \cap y = p$, $\forall y \in h(p)$, and $mixed(p) = 1$ otherwise;
- $Waypoint : P \times P \rightarrow \mathbb{R}^3 \cup \emptyset$ is a map giving the waypoint through which the robot is allowed to pass from place p_i to an adjacent place p_j. Thus, for $p_i, p_j \in P$ with $Adj(p_i, p_j) = 1$, we choose $Waypoint(p_i, p_j)$ as being the centroid of the intersection $p_i \cap p_j$, and since this intersection has a non-empty area we allow small deviations of the drone from its trajectory, without affecting the sequence of cells to be visited. For $p_i, p_j \in P$ with $Adj(p_i, p_j) = 0$, we let $Waypoint(p_i, p_j) = \emptyset$. ■

Since the robots are identical, the only difference between the automatons T_r is given only by their initial states. While there are many partition techniques for 2D environments (Berg et al., 2008; Belta and Habets, 2006; Mahulea et al., 2020), for 3D ones with regions of interest we here focus on two methods:

(i) *Grid cell decomposition* - the workspace E is divided into equal rectangular cuboids. To this end, a precision ϵ is imposed, showing how many equal cuboids (cells) are along any axis of E. E.g., for $\epsilon = 8$, the whole



(a) Original Büchi automaton (b) Trimmed Büchi automaton

Fig. 2. Büchi automaton corresponding to the LTL formula (1), $\varphi = \Diamond y_1 \wedge \neg y_2 \mathcal{U} (y_3 \vee y_4)$. After running Algorithm 1, the self-loop of s_1 from the trimmed automaton becomes $\neg y_2$.

environment is divided in 8^3 equal rectangular cuboids, each of them forming a place in T_r.

(ii) *Octtree based decomposition* - the imposed precision ϵ is a power of 2, and now the environment is recursively divided in rectangles of different size. The idea is to divide parts of E where an increased resolution (smaller cells) is needed, thus obtaining an abstraction with fewer states than in the previous method (Vespa et al., 2018; Lupascu et al., 2019).

Both methods will be used for simulations and compared in Sec. 6. The only conceptual difference between these methods is given by the partition, since the other elements of T_r (adjacency, observation etc.) are constructed identically once P is obtained. The partition for method (i) is easy to understand and therefore it is not detailed.

The remainder of this section details method (ii), which extends a cell decomposition technique that we previously used in 3D environments cluttered with obstacles (Lupascu et al., 2019). The extension refers to the fact that the technique in (Lupascu et al., 2019) partitions only a part of the free space, i.e. space that does not intersect any obstacle, and thus it could only be used for avoiding obstacles, but not for capturing possible movements to regions of interest. Here, we tailor this method for abstracting the entire 3D workspace in a finite discrete representation based on cells. The idea is inspired by so-called octtrees, and we employ a procedure that labels any given rectangular cuboid as *free* (no intersection with any region of interest), *occupied* (included in one of more regions of interest) or *mixed* (partially intersecting at least one region). The concept is to start by labeling the environment E (which is clearly mixed) and then recursively split every mixed rectangular cuboid in eight equal cuboids, by cutting in half each of its edges. Due to recursive procedure, each of these eight cuboids is labeled and the mixed ones are split as long as the precision ϵ is not reached, i.e. the smallest obtained cuboids have edges ϵ-times smaller than E. At the end, the partition elements (cells from P) will be the free, the occupied and the small mixed rectangular cuboids.

Due to splitting in equal cuboids, all rectangular cuboids from P will have the same ratio of edges as environment E, and the number of obtained cells is usually much smaller than ϵ^3, as it results in case of method (i). For each free cell p we will have in T_r the output $h(p) = \emptyset$ and $mixed(p) = 0$. For each occupied cuboid $h(p)$ shows the regions from Y that include the current cuboid and $mixed(p) = 1$. For each mixed cuboid (too small to further split) $h(p)$ contains the regions from Y that are intersecting p and we will have $mixed(p) = 1$, inducing that when the drone is somewhere inside p it is possible - but not certain - that the robot visits one or more regions from $h(p)$. This information of $mixed$ flag will be exploited in Sec. 5.

The pseudo-code expressed in Algorithm 2 includes the previously-discussed ideas for constructing automatons T_r. The lines 1-31 contain the recursive procedure for labeling and partitioning a generic rectangular cuboid RC. Along Algorithm 2, the intersection of any two polyhedra (e.g. lines 4, 37) is computed based on half-space (H-) representations of these shapes (Grünbaum, 2003). The test from line 5 is fulfilled only for *free* cells, which are added to P. If RC partially intersects at least one region from Y (test on line 10), then RC is *mixed*. RC is split in eight equal parts if these smaller cells do not go beyond precision ϵ, and for each of these smaller cuboids the procedure $recursive\_partitioning$ follows (lines 12-21). Too small mixed rectangular cuboids become cells in P with the $mixed$ flag set to 1 (lines 22-26). If the current RC is not free nor mixed, then it is completely included in some region(s) of interest (*occupied*), and it becomes a place in P (lines 27-31).

Based on the above, by applying the $recursive\_partitioning$ procedure for the full 3D environment E, the partition P results along with its observations and mixed map (line 34). The adjacency map Adj and the map $Waypoint$ are built according to Def. 2 on lines 35-39. As mentioned, 2 cells RC_i, RC_j are adjacent if their intersection has a positive area. Because all partition elements are cuboids with parallel edges, if test on line 37 is true, $p_i \cap p_j$ is a rectangle, and the corresponding waypoint is its centroid.

From a complexity point of view, there is an upper-bound ϵ^3 for the number of cells returned by Algorithm 2. Each iteration for the recursive partitioning of the environment has a polynomial complexity, while it mainly includes operations based on intersection of convex polyhedra. The scalability of Algorithm 2 is provided by its independence on the formula

Algorithm 2 Robot abstractions

Input : Environment E, regions Y, initial positions of robots $\mathcal{R}$, precision ϵ
Output: Automaton T_r, $r = 1, \ldots, |\mathcal{R}|$

/* —— Structure of the recursive partitioning procedure —— */
$(P, h, mixed) = recursive\_partitioning(RC, P, h, mixed, \epsilon, Y)$
Denote RC as $RC = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_2]$
Compute $Volume(RC \cap y_i), \forall i = 1, \ldots, |Y|$
if $\sum_{i=1}^{|Y|} Volume(RC \cap y_i) = 0$ **then**
    /* the current rectangular cuboid RC is *free* */
    $P := P \cup RC$, $h(RC) := \emptyset$, $mixed(RC) := 0$
    return $P, h, mixed$

else
    if $\exists i = 1, \ldots, |Y|$ *such that* $Volume(RC \cap y_i) \in (0, Volume(RC))$ **then**
        /* the current rectangular cuboid RC is *mixed* */
        if $2 \cdot \epsilon \cdot (x_2 - x_1) \geq (x_{max} - x_{min})$ **then**
            /* the mixed RC is further split in eight equal cuboids */
            $(P, h, mixed) = recursive\_partitioning([x_{min}, \frac{x_{min}+x_{max}}{2}] \times [y_{min}, \frac{y_{min}+y_{max}}{2}] \times [z_{min}, \frac{z_{min}+z_{max}}{2}], P, h, mixed, \epsilon, Y)$
            $(P, h, mixed) = recursive\_partitioning([x_{min}, \frac{x_{min}+x_{max}}{2}] \times [\frac{y_{min}+y_{max}}{2}, y_{max}] \times [z_{min}, \frac{z_{min}+z_{max}}{2}], P, h, mixed, \epsilon, Y)$
            $(P, h, mixed) = recursive\_partitioning([x_{min}, \frac{x_{min}+x_{max}}{2}] \times [y_{min}, \frac{y_{min}+y_{max}}{2}] \times [\frac{z_{min}+z_{max}}{2}, z_{max}], P, h, mixed, \epsilon, Y)$
            $(P, h, mixed) = recursive\_partitioning([x_{min}, \frac{x_{min}+x_{max}}{2}] \times [\frac{y_{min}+y_{max}}{2}, y_{max}] \times [\frac{z_{min}+z_{max}}{2}, z_{max}], P, h, mixed, \epsilon, Y)$
            $(P, h, mixed) = recursive\_partitioning([\frac{x_{min}+x_{max}}{2}, x_{max}] \times [y_{min}, \frac{y_{min}+y_{max}}{2}] \times [z_{min}, \frac{z_{min}+z_{max}}{2}], P, h, mixed, \epsilon, Y)$
            $(P, h, mixed) = recursive\_partitioning([\frac{x_{min}+x_{max}}{2}, x_{max}] \times [\frac{y_{min}+y_{max}}{2}, y_{max}] \times [z_{min}, \frac{z_{min}+z_{max}}{2}], P, h, mixed, \epsilon, Y)$
            $(P, h, mixed) = recursive\_partitioning([\frac{x_{min}+x_{max}}{2}, x_{max}] \times [y_{min}, \frac{y_{min}+y_{max}}{2}] \times [\frac{z_{min}+z_{max}}{2}, z_{max}], P, h, mixed, \epsilon, Y)$
            $(P, h, mixed) = recursive\_partitioning([\frac{x_{min}+x_{max}}{2}, x_{max}] \times [\frac{y_{min}+y_{max}}{2}, y_{max}] \times [\frac{z_{min}+z_{max}}{2}, z_{max}], P, h, mixed, \epsilon, Y)$

        else
            /* the mixed RC reached the imposed precision */
            $P := P \cup RC$, $mixed(RC) := 1$
            $h(RC) := \{y_i \in Y \mid Volume(RC \cap y_i) > 0\}$
            return $P, h, mixed$

    else
        /* RC is included by the regions of interest it intersects */
        $P := P \cup RC$, $mixed(RC) := 0$
        $h(RC) := \{y_i \in Y \mid Volume(RC \cap y_i) > 0\}$
        return $P, h, mixed$

/* —— Main algorithm —— */
Initialize $P = \emptyset$, $h(RC) = \emptyset$, $mixed(RC) = \emptyset$
$(P, h, mixed) = recursive\_partitioning(E, P, h, mixed, \epsilon, Y)$
Initialize $Adj = 0_{|P| \times |P|}$
for $p_i, p_j \in P$, $i \neq j$ **do**
    if $Area(p_i \cap p_j) > 0$ **then**
        $Adj(p_i, p_j) := 1$
        $Waypoint(p_i, p_j) := centroid(p_i \cap p_j)$
For each $r \in \mathcal{R}$, set p_{0r} as the cell where r is initially deployed
Return $T_r = (P, p_{0r}, Adj, 2^Y, h, mixed, Waypoint), \forall r \in \mathcal{R}$

φ and on the number of agents - $|\mathcal{R}|$ being necessary only for returning the initial cell for each robot.

Example 3. Let us revisit Example 1 for an easier understating of the abstraction approach from Algorithm 2. Thus, Figure 3(a) reveals a visual representation of the partitioning of environment from Figure 1, with precision $\epsilon = 16$. Figure 3(b) represents an enlarged part of this partition, by emphasizing with black a free cuboid, with yellow a mixed cell and with red an occupied one (the red cuboid being completely included in region y_1). For this environment, the resulted abstraction has 1002 cells in P, 7254 adjacency relations (transitions between states), and it was computed in 2.7 seconds. ■

5. TASK ASSIGNMENT

If Algorithm 1 returns a nonempty $TaskSet$, it means that the formula can be decomposed in independent tasks, and we now have to assign these tasks to the $|\mathcal{R}|$ robots. For this, we choose any decomposition from $TaskSet$, generically denoted as $\bigcup_{i=1}^{|\rho|-1}\{\varrho_{\mathcal{B}}(\rho(i),\rho(i+1))\}$. As shown in Sec. 3, this decomposition is in fact a set of elements of $\Sigma_{\mathcal{B}}$ that enable transitions along run ρ of $\mathcal{B}$.

For simplifying the notations, denote the task $\varrho_{\mathcal{B}}(\rho(i),\rho(i+1))$, $i = 1,\ldots,|\rho|-1$ by φ_i. According to notations from Sec. 2, φ_i contains the inputs of $\mathcal{B}$ that induce transition from state $\rho(i)$ to $\rho(i+1)$. Thus, a drone from the team should generate any observation from φ_i (i.e., an element of 2^Y), such that the task is accomplished. Since $\mathcal{B}$ was trimmed before computing its accepted runs, we have the guarantee that any element of φ_i can be produced by a proper position of a single robot, e.g., φ_i cannot be a conjunction of two disjoint regions. Also, the independence and fullness properties guarantees that the imposed LTL formula is satisfied if all tasks φ_i are independently accomplished by the robots, i.e., without necessary synchronizations or order of visiting regions.

Therefore, the $|\rho|-1$ tasks φ_i should be allocated to the $|\mathcal{R}|$ robots, and to this end we solve the following steps:

(i) Construct a cost matrix $W \in \mathbb{R}^{(|\rho|-1)\times|R|}$, where $W(i,r)$ is the cost incurred if robot r satisfies task φ_i.
(ii) Having matrix W, assign all tasks to drones such that a desired cost function for the whole team is minimized. Note that we may end up with more tasks being sequentially solved by a robot.

For fulfilling step (i), we consider each robot $r \in \mathcal{R}$ and each task φ_i, $i = 1,\ldots,|\rho|-1$, and we iterate a procedure that drives r from its initial deployment to a position where it satisfies φ_i. This procedure is basically a graph search, but instead of considering the underlying graph of the model T_r, we have to consider a reduced graph, as follows. Robot r should reach a place where where an element of $\varrho_{\mathcal{B}}(\rho(i),\rho(i+1))$ is true (for accomplishing φ_i), hence the set of possible destination nodes for r is given by $D = \{p \in P \mid h(p) \subseteq \varrho_{\mathcal{B}}(\rho(i),\rho(i+1)) \text{ and } mixed(p) = 0\}$. Note that we avoid going in a mixed state, because we need to have desired observations generated in the reached state, rather than a possible generation. While drone r moves towards any state from set D, it should produce in any intermediate position observations from set $\varrho_{\mathcal{B}}(\rho(i),\rho(i))$, such that the state $\rho(i)$ of $\mathcal{B}$ is not left until transition to $\rho(i+1)$ becomes enabled. Therefore, r is allowed to move only through intermediate nodes from set $I = \{p \in P \mid h(p) \subseteq \varrho_{\mathcal{B}}(\rho(i),\rho(i))\}$. Thus, from automaton T_r we keep the graph having nodes $D \cup I$ and transitions inherited from Adj, and on this graph we run a shortest path search from initial node p_{0r} to any node from set D. For this, we use the Dijkstra algorithm (Cormen et al., 2001), and the cost of the returned path is the value to be saved in $W(i,r)$.

The procedure for solving step (i) is captured by lines 7-15 from the overall algorithmic solution provided in Algorithm 3.

Remark 1. (Unfeasible distribution). It is possible that for a pair i,r the graph search does not give a solution - in cases when p_{0r} does not belong to or is disconnected from set $D \cup I$. If this happens, it means that robot r cannot move to a position where φ_i becomes true while yielding along the path observations in $\varrho_{\mathcal{B}}(\rho(i),\rho(i))$, and we store in $W(i,r)$ a big number N (with the significance of infinite cost). If the obtained matrix W has at least one row containing only values N, we conclude that the problem is infeasible for the current distribution. In this case, one should choose a different distribution returned by Algorithm 1, or should use the centralized approach from (Kloetzer and Mahulea, 2020). The condition from line 16 of Algorithm 3 handles such cases, by switching to the centralized planning method (Kloetzer and Mahulea, 2020).

Three observations are in order regarding step (i):

- The graph search approach is totally different than the methods from (Hustiu et al., 2020; Kloetzer and Mahulea, 2020), where two versions of a Mixed Integer Linear Programming (MILP) problem were used for computing $W(i,r)$. In (Kloetzer and Mahulea, 2020) the MILP was necessary for planning the whole team of robots in a centralized manner by using a Petri net model, while in (Hustiu et al., 2020) the MILP was inherited and tailored for a single robot. Thus, the current approach benefits from the computational point of view, since the complexity of the Dijkstra graph search algorithm is $|P|^2$, while a MILP optimization belongs to the NP-hard class.
- Based on the above, the costs from W are the sum of weights on transitions of T_r, and thus they can be cast to reflect the expected time or energy for moving between adjacent cells from the environment partition. Here, we consider unitary costs on transitions on T_r, thus resulting in costs in W that minimize the number of drone movements, in terms of direction changes in visited cells.
- Any cost $W(i,r)$ is computed by considering that r starts from its initial position p_{0r}. However, if step (ii) assigns more tasks to a robot, the total moving cost incurred by the robot will be sightly different that the sum of corresponding costs from W.

Step (ii) is in essence a form of optimal task allocation method, since the $|\rho|-1$ tasks have to be independently solved by the $|\mathcal{R}|$ drones, while incurring a minimum cost based on elements from W. This step is solved by MILP problem (2) (Hustiu et al., 2020), with the following accompanying explanations:

- The variables are: a binary matrix $Z \in \{0,1\}^{|\mathcal{R}|\times(|\rho|-1)}$ and a real variable λ.
- The solution Z returned by the optimization indicates the task(s) that should be satisfied by each robot, in the sense that robot $r \in \mathcal{R}$ is allocated to task(s) φ_i for which $Z(r,\varphi_i) = 1$.
- Variable λ is used for transforming a type of mimimax optimization into a MILP formulation. Basically we are interested in minimizing the maximum cost of any indi-

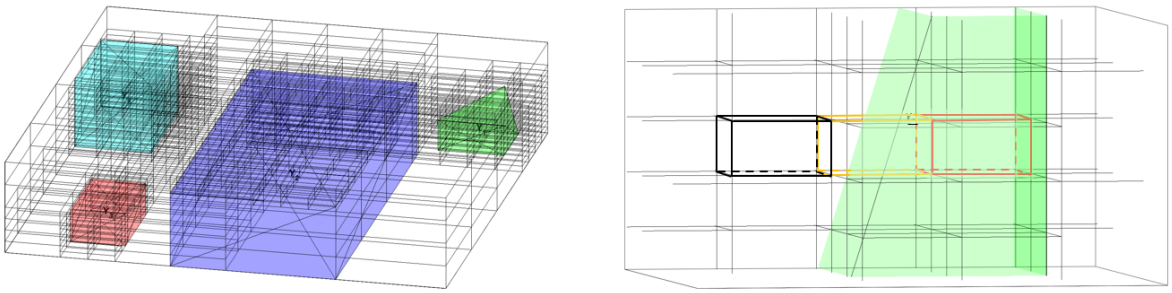


Fig. 3. (a) Decomposition in rectangular cuboids of the environment from Figure 1. (b) Enlarged part of the partition, showing a free cell (black), a mixed cell yellow and an occupied one (red).

vidual robot, and this is done by upper-bounding with λ the cost incurred by any drone.

- The cost function, where N has a big value, reduces the number of cells visited by any robot (through term in λ), while avoiding unnecessary movements of "faster" robots (through second term).
- The first set of constraints imposes that all tasks from the chosen decomposition will be accomplished.
- The second set of constraints upper-bounds by λ the cost incurred by any robot r. Note that if r will be allocated to multiple tasks, e.g., φ_i and φ_j, its actual moving cost is approximated with $W(r,\varphi_i)+W(r,\varphi_j)$ - recall that these costs were computed in W by assuming that the drone always starts from its initial position.

$$\begin{aligned} &\min\ N\cdot\lambda+\textstyle\sum_{r=1}^{|\mathcal{R}|} Z(r,:)\cdot W(:,r)\\ &\text{s.t. } \textstyle\sum_{r=1}^{|R|} Z(i,r)=1,\ \forall i=1,\ldots,|\rho|-1\\ &\quad Z(r,:)\cdot W(:,r)\le\lambda,\ \forall r=1,\ldots,|R|\\ &\quad Z\in\{0,1\}^{|\mathcal{R}|\times|\rho|-1},\ \lambda\in\mathbb{R}_{\ge 0}\end{aligned} \tag{2}$$

MILP (2) can be solved with existing software tools (IBM, 2016). Its feasibility (existence of a solution) is guaranteed because the MILP is not called when some tasks cannot be accomplished, as mentioned in Rem. 1. The solution Z returned by (2) shows the tasks that should be satisfied by each robot, but without a specific order. In our approach, for avoiding an additional computation overhead, we simply order the tasks to be solved by each agent based on their cost from W, i.e., drone r first heads to accomplishing its lower-cost task, and so on until it finishes all assignments.

The pseudo-code description of our entire method is given in Algorithm 3, which finishes with constructing the trajectory (sequence of cells) to be individually followed by each drone. We consider that an individual robot movement cannot solve the problem in two cases: if the mission cannot be decomposed, or the chosen decomposition is unfeasible as in Rem. 1. In either of these two situations, the approach from (Kloetzer and Mahulea, 2020) can be used for returning a centralized solution where the robots will have to communicate and synchronize along paths (lines 5, 17 from Algorithm 3). If a feasible distribution is found, MILP (2) is used for showing the individual tasks to be solved by robots. For each robot r, the sequence of cells to be traversed, Seq_r, is build by imposing that it sequentially accomplishes its allocated tasks, in an ascending expected cost from matrix W (order chosen on line 22). For accomplishing each of these tasks φ_i, we use the graph search that was iterated for finding the element $W(i,r)$, by simply updating the start node based on the current position of robot r (lines 24, 30). Finally, the drone follows the imposed sequence of cells by connecting the waypoints from T_r (centroids of common facets shared by successive cells from Seq_r), thus obtaining a piece-wise linear trajectory that can be followed by an omni-directional agent. Additionally, we enforce that the robot enters in each cell p_{nr} (instead of reaching only one of its facets) by inserting in its trajectory the centroid of p_{nr} (line 28). This is done since a task φ_i is satisfied in each such cell p_{nr}. All drones individually follow their trajectories $Trajectory_r$, and thus the global mission φ is accomplished in a distributed manner. As anticipated at the beginning of Sec. 2, Algorithm 3 yields a centralized path-planning part that constitutes the base for trajectory-following routines, with the advantage that the robot movements are executed in a decentralized manner.

***Remark** 2.* **Inter-robot collisions.** Note that the individual robot trajectories may intersect. Thus, for implementations in real scenarios, the robots should have local collision-avoidance rules, e.g., based on priorities and waiting modes, or on techniques inspired from resource allocation systems (Reveliotis and Roszkowska, 2011). These local rules fall beyond the scope of the current work, but they would require only a small communication radius of agents, without affecting the distribution of the imposed formula.

The complexity of Algorithm 3 depends on those mentioned regarding Algorithms 1 and 2, each being called one time. Also, there is a single call to MILP (2). The number of iterations is imposed by $\mathcal{R}$ and $|\rho|-1$ (lines 7, 19, 22), in each iteration the Dijkstra graph search being the most demanding part, and hence its small complexity is promising in terms of scalability on the number of robots.

Algorithm 3 is not complete in terms of independence between robots. Thus, it may sometimes switch to the centralized solution from (Kloetzer and Mahulea, 2020), even if the current formula may have independent tasks. This is because we consider only one decomposition from $TaskSet$ (rather than testing all of them), Algorithm 1 does not account specific sequences of tasks to be imposed to the same robot. Besides fully independent tasks, future work may also be targeted towards identifying some possible intermediate synchronizations between which it may be possible to have independent tasks.

***Example** 4.* We here conclude the example introduced in Sec. 2 and revisited at the end of each Sec. 3 and Sec. 4. Step (i) returns in 0.3 seconds the cost matrix $W=\begin{bmatrix}11 & 3\\ 4 & 12\end{bmatrix}$. MILP (2) from step (ii) is solved in 0.45 seconds and it allocates the task

Algorithm 3 Overall solution

Input : Environment (initial position of robots and regions of interest), LTL specification φ
Output: Individual robot movement plans $Plan_r$

1 Run Algorithm 2 to obtain robot model T_r, $r = 1, \ldots, |\mathcal{R}|$
2 Convert φ to Büchi automaton $\mathcal{B}$
3 Run Algorithm 1 to obtain $TaskSet$ - the possible decompositions of φ
4 **if** $TaskSet = \emptyset$ **then**
5 Use approach from (Kloetzer and Mahulea, 2020) and **return** (synchronized) movement plans
6 Choose a decomposition from $TaskSet$, denoted by $\bigcup_{i=1}^{|\rho|-1}\{\varrho_{\mathcal{B}}(\rho(i), \rho(i+1))\}$
7 **for** $i = 1, \ldots, |\rho| - 1$ and $r = 1, \ldots, |\mathcal{R}|$ **do**
8 $D_i := \{p \in P \mid h(p) \subseteq \varrho_{\mathcal{B}}(\rho(i), \rho(i+1)) \text{and} mixed(p) = 0\}$
9 $I_i := \{p \in P \mid h(p) \subseteq \varrho_{\mathcal{B}}(\rho(i), \rho(i))\}$
10 Construct graph with set of nodes $D_i \cup I_i$ and corresponding arcs inherited from transitions in T_r
11 Run Dijkstra's graph search, with source node p_{0r} and destination set D_i
12 **if** Solution is obtained **then**
13 $W(i, r)$:= minimum cost of the returned paths to nodes from D_i
14 **else**
15 $W(i, r) := N$
16 **if** $\exists i \in \{1, \ldots, |\rho| - 1\}$ such that $W(i, :) \cdot \mathbf{1} = N \cdot |\mathcal{R}|$ **then**
17 Use approach from (Kloetzer and Mahulea, 2020) and **return** (synchronized) movement plans
18 Solve MILP (2) and obtain Z - robot-to-task(s) allocations
19 **for** $r = 1, \ldots, |\mathcal{R}|$ **do**
20 Set $Seq_r := p_{0r}$
21 Set $Trajectory_r := x_{0r}$
22 **for** $i \in \{1, \ldots, |\rho| - 1\}$ such that $Z(r, i) = 1$ *and* $W(i, r) \leq W(j, r), \forall j \in \{1, \ldots, |\rho| - 1\}, j \neq i$ **do**
23 Consider the graph from line 10, with set of nodes $D_i \cup I_i$ and arcs inherited from T_r
24 Run Dijkstra's graph search, with source node p_{0r} and destination set D_i
25 Denote the returned minimum cost path with $p_{0r}, p_{1r}, p_{2r}, ..., p_{nr}$
26 Append sequence of cells $p1, p2, ..., p_n$ to Seq_r
27 Append to $Trajectory_r$ the centroids of common facets, $Waypoint(p_{(k)r}, p_{(k+1)r})$, $k = 1, \ldots, n$
28 Append to $Trajectory_r$ the centroid of cell p_{nr}
29 Set $Z(r, i) := 0$ - task φ_i was solved
30 Update $p_{0r} := p_{nr}$
31 Return individual piece-wise linear trajectories $Trajectory_r$, $\forall r = 1, \ldots, |\mathcal{R}|$

$\varphi_1 = y_1$ to the first robot (red) and $\varphi_2 = y_3 \vee y_4$ to r_2 (blue). Thus, r_1 should move to a cell with observation y_1 (set D for φ_1 containing all places with observation y_1) and should cross only through cells not intersecting y_2 (according to set I that includes all places with observations different than y_2). Drone r_2 independently moves to a cell with observation belonging to set $\{y_3, y_4\}$, while also avoiding y_2. There result 4 cells in the sequence Seq_{r_1} from Algorithm 3 and 3 cells in Seq_{r_2}. The trajectories obtained by linking the corresponding waypoints are shown in Figure 4. ■

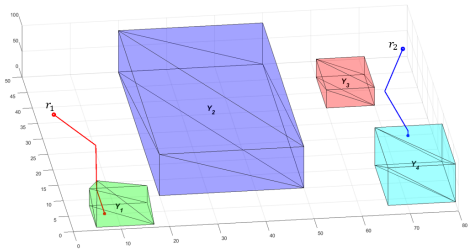


Fig. 4. Independent trajectories of the two drones, giving a solution to Example 1.

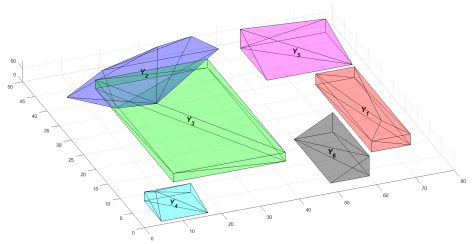


Fig. 5. Environment E with 6 regions of interest Y.

6. NUMERICAL EVALUATION

This section numerically evaluates our solution, by considering a team of 3 drones and the configuration of the workspace depicted in Figure 5: a 3D environment $E = [0, 80] \times [0, 50] \times [0, 100]$ (in any length units lu) with 6 regions of interest $Y = \{y_1, y_2, y_3, y_4, y_5, y_6\}$. All regions are convex bounded polyhedra with flat base ($z = 0$), having distinct shapes and heights. The regions are disjoint, except y_2 and y_3 that intersect. The LTL specification is given in (3), requiring that the team of drones must eventually visit regions y_1, y_4, y_5, y_6 and the intersection of y_2 and y_3.

$$\varphi = \Diamond y_1 \wedge \Diamond (y_2 \wedge y_3) \wedge \Diamond y_4 \wedge \Diamond y_5 \wedge \Diamond y_6. \tag{3}$$

The two types of partitions described in Sec. 4 are shortly referred here as $Grid$ and $OctTree$, respectively. We decomposed the environment with a precision $\epsilon = 16$. As a result, we obtained a graph with 1849 nodes and 13081 transitions for the $OctTree$ approach, with a run time of 6.7 seconds. On the other hand, for the $Grid$ procedure we achieved a graph with 4096 nodes with 27136 transitions in 19 seconds. As mentioned in Sec. 2, the algorithms were implemented and run in MATLAB, on a laptop with i7 - 8^{th} gen. CPU @ 2.20Ghz and 8GB RAM.

The initial drone deployments are not displayed in the Figure 5, since we will assume 100 random deployments. For each experiment, the initial positions yield different sequences of cells and trajectory lengths for the robots, with small variations regarding the computation times. For the numerical interpretation, we will report average results over these 100 experiments.

First we discuss execution times of different phases of our proposed method. The computing time for trimming the Büchi automaton and for decomposing the mission into independent tasks as in Algorithm 1 took 17.58 seconds, of course this time being independent of the used partitioning method. The computation for obtaining the cost matrix W (average over 100 different initial deployments of drones) took 6.84 seconds for the $Grid$ case and 1.51 seconds for the $OctTree$ one. As

expected, $Grid$ method induces more time, since computing each costs requires a graph search on a subset of states from P. Solving the MILP allocation took around 0.03 seconds, this formulation being independent of size of automatons T_r.

The second part of the comparison contains performances for the resulted trajectories of drones. Thus, Table 1 gives the average length of trajectories for each drone. Recall that the length is influenced by the size of each traversed cell from Seq_r (Algorithm 3), and we optimized the number of traversed places (due to unitary costs in Adj). The $OctTree$ partition includes larger cells, and this coarser abstraction may attract longer trajectories than $Grid$, but with smaller computation times.

Table 1. Average lengths of the trajectory for each drone, $OctTree$ and $Grid$ decompositions.

Average trajectory length	$OctTree$	$Grid$
r_1 $[lu]$	76.84	68.55
r_2 $[lu]$	86.02	69.56
r_3 $[lu]$	79.44	71.54

Table 2 includes information about the maximum cost and the total cost, as per elements from W. Thus, the meaning of the cost is the number of cells traversed by the drone, i.e., the number of direction changes. The maximum cost from the first line means the maximum number of traversed cells over the 3 robots, while each robot satisfies all its assigned tasks. The total cost include the number of places visited by all robots until specification φ is accomplished. The fewer states yielded by $OctTree$ imply less direction changes for drone flight, even if the actual trajectory length is longer.

Table 2. Averages for maximum cost and total cost, in terms of number of traversed places.

Average cost	$OctTree$	$Grid$
Maximum cost	11.31	19.13
Total cost	27.52	47.10

For a better visualization of the trajectories and for a simulation of drone's flight, the reader is directed to the video available at `https://youtu.be/awzeIZbexng`. We mention that the motivation behind the considered simulation with 3 drones and 5 independent tasks is to highlight a scenario in which at least one drone must execute multiple tasks. More complicated formulas with a greater number of robots would yield to hard-to-visualize trajectories.

One important contribution of this paper is the algorithm for decomposing the LTL mission. To demonstrate this, we computed the cost for fulfilling the mission from (3) with only one drone, instead of distributing it among the team members. Thus, for the $OctTree$ partition a single drone would have to cross in average through 34.6 cells, and for the $Grid$ case it would traverse 68.74 cells. By comparing these data with Table 2, we can affirm that the decomposition is truly beneficial both from the point of view of the total number of traversed cells, as well as from the parallel execution of independent tasks, whose completion time is reflected by the maximum cost from Table 2.

7. CONCLUSIONS

This paper presents an approach for path planning in a 3D environment, by considering a team of drones that should complete a co-safe $\mathrm{LTL_X}$ mission over some regions of interest. Before drones start their movement, a central unit decomposes the overall requirement into independent tasks and find trajectories that are independently followed by the drones, hence the motion itself occurs in a distributed manner. The team mission is modeled as a Büchi automaton and based on this model a decomposition into independent tasks is computed. The workspace with regions of interest is abstracted into an automaton for each robot, by using either a grid decomposition with equal rectangular cuboid cells, or a recursive octtree-based partitioning method.

The contributions include an automated way for decomposing the LTL formula into independent tasks, an extension of the recursive partitioning method for 3D environments with regions of interest. Moreover, all these parts are integrated in an overall algorithmic solution whose performance is numerically evaluated based on several criteria.

For future work we intend to include collision avoidance methods based on safe distances between drones and to support the simulation results through real-time experiments in a laboratory setup. Here, we aim to use the advantage of having independent trajectories that can be followed by drones based on some feedback from on-board positioning sensors, while a reduced usage of communication or proximity sensors may be necessary for enforcing local collision-avoidance rules.

REFERENCES

Alur, R., Etessami, K., La Torre, S., and Peled, D. (2001). Parametric temporal logic for "model measuring". *ACM Transactions on Computational Logic (TOCL)*, 2(3), 388–407.

Azim, A. (2013). *3D perception of outdoor and dynamic environment using laser scanner*. Ph.D. thesis, Grenoble.

Baier, C. and Katoen, J.P. (2008). *Principles of model checking*. MIT Press.

Belta, C., Bicchi, A., Egerstedt, M., Frazzoli, E., Klavins, E., and Pappas, G. (2007). Symbolic Planning and Control of Robot Motion. *IEEE Robotics and Automation Magazine*, 14(1), 61 – 71.

Belta, C. and Habets, L. (2006). Controlling a class of nonlinear systems on rectangles. *IEEE Transactions on Automatic Control*, 51(11), 1749–1759.

Berg, M.D., Cheong, O., and van Kreveld, M. (2008). *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition.

Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L.E., and Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Boston.

Clarke, E.M.M., Peled, D., and Grumberg, O. (1999). *Model checking*. MIT Press.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. (2001). *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, Cambridge, Massachusetts and New York, 2nd edition.

Ding, X., Smith, S.L., Belta, C., and Rus, D. (2014). Optimal control of Markov decision processes with linear temporal

logic constraints. *IEEE Transactions on Automatic Control*, 59(5), 1244–1257.

Dosoftei, C., Horga, V., Doroftei, I., Popovici, T., and Custura, S. (2020). Simplified mecanum wheel modelling using a reduced omni wheel model for dynamic simulation of an omnidirectional mobile robot. In *Int. Conf. and Exposition on Electrical And Power Engineering (EPE)*, 721–726.

Gânsari, M. and Buiu, C. (2017). Software system integration of heterogeneous swarms of robots. *Journal of Control Engineering and Applied Informatics*, 19, 49–57.

Gastin, P. and Oddoux, D. (2001). Fast ltl to büchi automata translation. In *Proc. of the* 13^{th} *Conference on Computer Aided Verification (CAV)*, 53–65.

Grünbaum, B. (2003). *Convex Polytopes*. Springer-Verlag.

Guo, M. and Dimarogonas, D.V. (2015). Multi-agent plan reconfiguration under local LTL specifications. *Int. Journal of Robotics Research*, 34(2), 218–235.

Hafez, A. (2015). Sequential covariance-weighted quasiconvex solution to mapping in visual slam. *Journal of Control Engineering and Applied Informatics*, 13, 61–69.

Hustiu, I., Kloetzer, M., and Mahulea, C. (2020). Distributed path planning of mobile robots with ltl specifications. In *2020 24th International Conference on System Theory, Control and Computing (ICSTCC)*, 60–65. IEEE.

IBM (2016). IBM ILOG CPLEX Optimization Studio. Software.

Kloetzer, M. and Mahulea, C. (2015). LTL-based planning in environments with probabilistic observations. *IEEE Transactions on Automation Science and Engineering*, 12(4), 1407–1420.

Kloetzer, M. and Mahulea, C. (2020). Path planning for robotic teams based on LTL specifications and Petri net models. *Discrete Event Dynamic Systems: Theory and Applications*, 30, 55—-79.

Kupferman, O. and Vardi, M.Y. (2001). Model checking of safety properties. *Formal Methods in System Design*, 19(3), 291–314.

Lacerda, B. and Lima, P.U. (2019). Petri net based multi-robot task coordination from temporal logic specifications. *Robotics and Autonomous Systems*, 122, 343–352.

LaValle, S.M. (2006). *Planning Algorithms*. Cambridge. Available at `http://planning.cs.uiuc.edu`.

Li, L. and Fu, J. (2019). Approximate dynamic programming with probabilistic temporal logic constraints. In *2019 American Control Conference (ACC)*, 1696–1703. IEEE.

Lindemann, L., Nowak, J., Schönbächler, L., Guo, M., Tumova, J., and Dimarogonas, D.V. (2019). Coupled multi-robot systems under linear temporal logic and signal temporal logic tasks. *IEEE Transactions on Control Systems Technology*, 1–8.

Lupascu, M., Hustiu, S., Burlacu, A., and Kloetzer, M. (2019). Path planning for autonomous drones using 3d rectangular cuboid decomposition. In *2019 23rd International Conference on System Theory, Control and Computing (ICSTCC)*, 119–124. IEEE.

Mahulea, C., González, R., Montijano, E., and Silva, M. (2021). Path planning of multirobot systems using Petri net models. Results and open problems. *Revista Iberoamericana de Automática e Informática industrial*, 18(1), 19–31.

Mahulea, C. and Kloetzer, M. (2018). Robot Planning based on Boolean Specifications using Petri Net Models. *IEEE Trans. on Automatic Control*, 63(7), 2218–2225.

Mahulea, C., Kloetzer, M., and Gonzalez, R. (2020). *Path Planning of Cooperative Mobile Robots Using Discrete Event Models*. Wiley-IEEE Press.

Moarref, S. and Kress-Gazit, H. (2020). Automated synthesis of decentralized controllers for robot swarms from high-level temporal logic specifications. *Autonomous Robots*, 44, 585–600.

Mosca, A., Vasile, C.I., Belta, C., and Raimondo, D.M. (2019). Multi-robot routing and scheduling with temporal logic and synchronization constraints. In *Proceedings of the 2019 2nd International Conference on Control and Robot Technology*, 40–45.

Nikou, A., Boskos, D., Tumova, J., and Dimarogonas, D.V. (2018). On the timed temporal logic planning of coupled multi-agent systems. *Automatica*, 97, 339–345.

Reveliotis, S. and Roszkowska, E. (2011). Conflict Resolution in Free-Ranging Multivehicle Systems: A Resource Allocation Paradigm. *IEEE Trans. on Robotics*, 27(2), 283–296.

Rubí, B., Pérez, R., and Morcego, B. (2019). A survey of path following control strategies for UAVs focused on quadrotors. *Journal of Intelligent & Robotic Systems*, 1–25.

Sahil, I. (2019). Battery and wifi aware 3d exploration based on temporal logics.

Sanchez-Lopez, J.L., Wang, M., Olivares-Mendez, M.A., Molina, M., and Voos, H. (2019). A real-time 3d path planning solution for collision-free navigation of multirotor aerial robots in dynamic environments. *Journal of Intelligent & Robotic Systems*, 93(1-2), 33–53.

Schillinger, P., Bürger, M., and Dimarogonas, D. (2018). Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. *The International Journal of Robotics Research*, 37(7), 818–838.

Tumova, J. and Dimarogonas, D.V. (2016). Multi-agent planning under local ltl specifications and event-based synchronization. *Automatica*, 70, 239–248.

Vespa, E., Nikolov, N., Grimm, M., Nardi, L., Kelly, P.H.J., and Leutenegger, S. (2018). Efficient octree-based volumetric slam supporting signed-distance and occupancy mapping. *IEEE Robotics and Automation Letters*, 3(2), 1144–1151.

Wolper, P., Vardi, M., and Sistla, A. (1983). Reasoning about infinite computation paths. In *Proc. of the 24th IEEE Symposium on Foundations of Computer Science*, 185–194.

Yang, G., Belta, C., and Tron, R. (2020). Continuous-time signal temporal logic planning with control barrier functions. In *2020 American Control Conference (ACC)*, 4612–4618. IEEE.

Yen, J.Y. (1971). Finding the K Shortest Loopless Paths in a Network. *Management Science*, 17(11), 712–716.