Noname manuscript No.
(will be inserted by the editor)

Reactive Mission and Motion Planning with Deadlock Resolution Avoiding Dynamic Obstacles

Javier Alonso-Mora · Jonathan A. DeCastro · Vasumathi Raman · Daniela Rus · Hadas Kress-Gazit

Received: date / Accepted: date

Abstract In the near future mobile robots, such as personal robots or mobile manipulators, will share the workspace with other robots and humans. We present a method for mission and motion planning that applies to small teams of robots performing a task in an environment with moving obstacles, such as humans. Given a mission specification written in Linear Temporal Logic, such as patrolling a set of rooms, we synthesize an automaton from which the robots can extract valid strategies. This centralized automaton is executed by the robots in the team at runtime, and in conjunction with a distributed motion planner that guarantees avoidance of moving obstacles. Our contribution is a correct-by-construction synthesis approach to multi-robot mission planning that guarantees collision avoidance with respect to moving obstacles, guarantees satisfaction of the mission specification and resolves encountered deadlocks, where a moving obstacle blocks the robot temporally.

Our method provides conditions under which deadlock will be avoided by identifying environment behaviors that, when encountered at runtime, may prevent the robot team from achieving its goals. In particular, i) it identifies deadlock conditions; ii) it is able to check whether they can be resolved; and iii) the robots implement the deadlock resolution policy locally in a distributed manner. The approach is capable of synthesizing and executing plans even with a high density of dynamic obstacles. In contrast to many existing approaches to mission and motion planning, it is scalable with the number of moving obstacles. We demonstrate the approach in physical experiments with walking humanoids moving in 2D environments and in simulation with aerial vehicles (quadrotors) navigating in 2D and 3D environments.

Keywords Multi-robot systems · Formal Methods · Mission Specification · Motion Planning · Deadlock Resolution · Dynamic Environments

Jonathan DeCastro and Javier Alonso-Mora contributed equally to this work. This work was supported in part by NSF Expeditions in Computer Augmented Program Engineering (ExCAPE), pDOT ONR N00014-12-1-1000, SMARTS N00014-09-1051, the Boeing Company and TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. We thank their support.

J. Alonso-Mora
Massachusetts Institute of Technology &
Delft University of Technology
Mekelweg 2, 2628 CD Delft, The Netherlands
Tel.: +31 (0)15 27 85489
Fax: +31-15-27 86679
E-mail: j.alonsomora@tudelft.nl

J. DeCastro
Cornell University
Ithaca, NY, USA
E-mail: jad455@cornell.edu

V. Raman
Zoox, Inc.
Menlo Park, CA, USA
E-mail: vasumathi.raman@gmail.com

D. Rus
Massachusetts Institute of Technology
Cambridge, MA, USA
E-mail: rus@csail.mit.edu

H. Kress-Gazit
Cornell University
Ithaca, NY, USA
E-mail: hadaskg@cornell.edu

1 Introduction

Mobile robots, such as package delivery robots, personal assistants, surveillance robots, cleaning robots, mobile manipulators or autonomous cars, execute possibly complex tasks and must share their workspace with other robots and humans. For example, consider the case shown in Fig. 1 in which two mobile robots are tasked with patrolling and cleaning the rooms of a museum. What makes this task challenging is that the environment in which the robots operate could be filled with static obstacles, as well as dynamic obstacles, such as people or doors, that could lead to collisions or block the robot. To guarantee the task of continuously monitoring all the rooms, each robot must react to the environment at runtime in a way that does not prevent making progress toward fulfilling the overall mission. In particular, we describe an approach for navigation in dynamic environments that is able to satisfy a mission by resolving *deadlocks*, i.e. situations where a robot is temporally blocked by a dynamic obstacle and can not make progress towards achieving its mission, at runtime.

Planning for multi-agent systems has been explored extensively in the past. Many have focused on approaches for local motion planning [1, 6] that offer collision avoidance

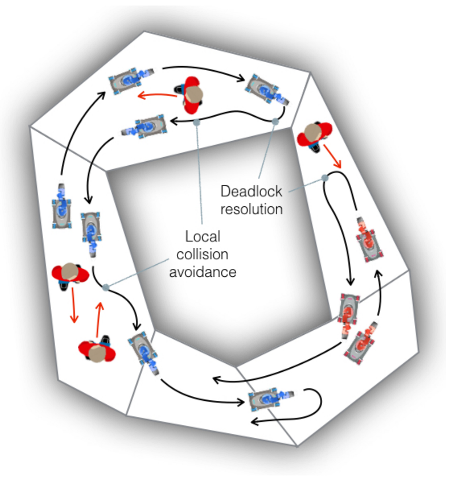


Fig. 1: Surveillance/cleaning scenario. Two robots are tasked with actively monitoring the rooms of a museum. The robots must avoid collisions with static and moving obstacles and resolve deadlocks in order to achieve their goals.

in cluttered, dynamic environments. While these approaches are effective for point-to-point navigation, the planning is myopic and could fail when applied to complex tasks in complex workspaces. On the other hand, it has been demonstrated that correct-by-construction synthesis from linear temporal logic (LTL) specifications has utility for composing basic (atomic) actions to guarantee the task in response to sensor events [18, 26, 29, 44]. Such approaches are naturally conducive to mission specifications written in structured English [25], which are translatable into LTL formulas over variables representing the atomic actions and sensor events associated with the task.

In the surveillance-cleaning scenario of Fig. 1, the motion (moving between rooms), atomic actions (e.g., "remove garbage", "identify a subject"), and binary sensors (e.g. "intruder sensing", "garbage sensing") are assumed to be perfect: they are treated as black boxes that always return the correct result and hence admit a *discrete abstraction* that is appropriate for the task and workspace. A major challenge underpinning this approach is in creating atomic elements holding guarantees for correct execution of the discrete abstraction. To guarantee motion fulfillment, researchers have explored combining LTL-based planners with grid planners [7], sampling-based planners [22], or planners for multiple robots predicated on motion primitives [38]. Such approaches are able to guarantee motion in cluttered environments but do not readily extend these guarantees to cases where the environment is dynamic in nature. Solutions have been sought that, in a computationally expensive manner, partition the workspace finely [30, 44] or re-compute the motion plan [7], or else apply conservative constraints forbidding the robot to occupy the same region as an adversarial agent [24].

1.1 Approach

In the approach introduced in this paper, we alleviate such difficulties by considering an integration of a high-level mission planner with a local planner that guarantees collision-free motion in three dimensional workspaces when faced with both static and dynamic obstacles, under the assumption that the dynamic obstacles are not intentionally adversarial. In this context, "intentionally adversarial" means that the dynamic obstacles may behave in a way that may temporarily prevent the robot from achieving a goal, but cannot move in a way that actively always prevents the robot from achieving its goals, for instance by blocking the robot forever. Our integration involves two components: an offline algorithm for plan synthesis adopting the benefit of an LTL formalism, and an online local planning algorithm for executing the plan. Our approach is centralized for the robots in the team, which execute the high-level specification, and decentralized with respect to moving obstacles, i.e. we do not control the moving obstacles yet perform decentralized avoidance and deadlock resolution strategies. While the robots are able to measure the position and velocity of moving obstacles, they only need to do so within a local range of the robot – the key assumption in this paper is that the robots are not required to have global knowledge of their environment.

The basis of the offline synthesis is a novel discrete abstraction of the problem that applies simple rules to resolve *physical deadlocks*, between two or more robots in a team or between a robot and a dynamic obstacle. This abstraction is composed with a specification of a multi-agent task to synthesize a strategy automaton encoding the mission plan. In contrast to approaches that would require on-the-fly replanning upon encountering a physical deadlock [7, 22, 32], the approach we propose automatically generates alternative plans within the synthesized automaton. As with any reactive task, there may exist no mission plan that guarantees the task, due to the conservative requirement that a mission plan must execute under *all possible* environment behaviors. To address this conservatism, our approach automatically identifies for which environment behaviors the mission is guaranteed to hold. These additional assumptions are transformed succinctly into a certificate of task infeasibility that is explained to the user.

The online execution component is based on a local planner that can optimally avoid dynamic obstacles in two- or three-dimensions, executed as a service called during execution of the strategy automaton. Given a dynamic model of the robots and a coarse description of the moving agents (e.g. their maximum velocities) our local planner computes a plan that guarantees collision-free motion between the robot and static and dynamic obstacles. The collision-avoidance feature obviates the need for collision avoidance to be taken care of by the discrete abstraction. It furthermore allows our local planner to preserve the behaviors of the strategy automaton, by preventing a robot from entering unintended regions as it carries out its task. To the authors' knowledge, this is the first end-to-end system that has been devised to guarantee multi-agent mission-level tasks in dynamic environments using optimization-based local planners.

The proposed deadlock resolution approach is motivated by works in event-driven planning (e.g. [14]), but yields a strategy that scales well with the number of dynamic obstacles without incurring conservatism that would prevent mission plans from being synthesized. In particular,

- Our approach establishes proof for task success without requiring a costly re-planning step or fine workspace discretization, as long as the environment that causes deadlocks behaves according to the generated assumptions.
- Our approach comes with proof that admissible deadlocks are always resolved and livelocks (the situation where a robot is free to move but unable to reach a goal) never occur.
- The fully automated nature of our approach has practical utility, since the user does not need to intervene to debug specifications. In fact, our approach explains, in an intelligible way, any additional environment assumptions it has added.
- Another practical feature of our approach is that, unlike related planners [24, 30], we do not require global knowledge of the obstacles. As we show, this allows our approach to scale to an arbitrary number of dynamic obstacles, as long as the aggregate behavior of the obstacles adhere to all specified assumptions.

Our approach is well-suited for any dynamic environment, but we emphasize its particular value to human environments. Specifically, our automatically-generated environment assumptions are transformed into human-readable certificates such as:

The synthesized controller is certified for this task, if any encountered deadlock between the robot and a dynamic obstacle in the hallway resolves eventually.

The certificates provide, at synthesis time, a set of rules defining situations which could make it impossible for the robot to achieve its goals, with the purpose of creating a layer of cooperation between the user (i.e. the human that performs the controller synthesis and deploys the system) and the robots. This frees a user from having to come up with assumptions that characterize the environment's behavior, a difficult proposition in practice. If these assumptions are broken at runtime, then this signifies that the task is no longer strictly guaranteed. Our approach also aims to reduce situations where members of the robot team become deadlocked with one another, by adopting a coordination strategy in the specification preventing actions that may induce deadlocks.

A more detailed overview of the approach is given in Sec. 4, right after formalizing the problem in Sec. 3.

1.2 Contribution

This paper presents two main contributions toward *reactive mission and motion planing with deadlock resolution among dynamic obstacles.*

- A holistic synthesis approach to provably achieve collision-free behaviors in dynamic environments with an arbitrary number of moving obstacles that does not require mutual exclusion. The approach leverages (a) reactive mission planning to globally resolve deadlocks and achieve the specified task, and (b) online local motion planning to guarantee collision free motion and respect the robot kinodynamics.
- An automatic means for encoding tasks that resolve deadlock based on automatically-generated revisions to a specification. Our approach automatically generates human-comprehensible assumptions in LTL that, if satisfied by the controlled robots and the dynamic obstacles, would ensure correct behavior. We show that our revision approach is sufficient in making the original specification realizable.

We also contribute an optimization-based method for local motion planning that guarantees real-time collision avoidance with *static* and dynamic obstacles in 3D environments while remaining faithful to the robot's dynamics. The method extends [3] by efficiently computing the robot's local free-space in cluttered environments. Yet, the reader may opt for a different local planner and maintain the synthesis approach, as long as the local planner provides avoidance guarantees. The method is evaluated in experiments with ground robots and in simulations with aerial vehicles.

In a preliminary version of this work, [12], a strategy was developed for synthesizing controllers for guaranteed collision-free motion of a robot team. In this paper, we extend those results by presenting a complete description of the proposed abstraction method and offline controller synthesis procedure, solidify details on the mathematical derivation for the constraints of the local motion planner, and provide in-depth evaluation of our proposed synthesis techniques aided by both simulation and physical experiments. Additionally, we enhance the approach in two ways. First, our approach reasons about the geometry of workspace regions in order to avoid preventable deadlock. For instance, if a corridor is only wide enough for one robot, we offer an approach that coordinates the actions of two robots so that they do not head in opposite directions in the corridor. Second, we present a general approach that allows a richer set of deadlock resolution rules to be chosen at synthesis time.

1.3 Related Work

1.3.1 Reactive Synthesis for Mission Planning

A number of approaches are suited to automatic synthesis of correct-by-construction controllers from mission specifications written as temporal logic formulas [7, 22, 31]. Reactive synthesis [26, 44] extends these capabilities to tasks in which the desired outcome depends on uncontrolled events in the environment and changing sensor inputs, and is especially compelling given the complex nature of multi-agent scenarios. For instance, [41] synthesized control and communication for producing optimal multi-robot trajectories, [9] distributed a specification among a robot team, and [35, 36] synthesized centralized reactive controllers based on analytically constructed multi-robot motion controllers. Distributed and decomposition-based planning approaches tackle the complexity problem when scaling to a large number of robots. For instance, [40] construct distributed controllers from a specifications already separated into coordinating and non-coordinating tasks, while [39] automatically decompose a specification into independent, distributed task specifications. In most approaches, moving obstacles are modeled in a discrete manner as part of the abstraction, leading to over-conservative restrictions like requiring robots to be at least one region apart. In contrast, our method only requires local awareness of the robot's surroundings, and guarantees collision-avoidance via a local planner.

Reactive synthesis in dynamically-changing environments presents a crucial dilemma: explicitly modeling the state of all other agents can be computationally prohibitive, but incomplete models of the environment destroy task satisfaction guarantees. To address the state-explosion problem while tracking the state of uncontrollable agents, [45] formulated an incremental synthesis procedure that started with a set number of agents assumed observable, and added more agents to this set depending on available computational resources; however, unlike our approach, they still required global knowledge of the external agents. The authors in [30], on the other hand, made local modifications to the synthesized strategy when new elements of the environment were discovered that violated the original assumptions. While we also update our specification, we differ from [30] in that no re-synthesis step is needed at runtime, thereby preserving guarantees before runtime.

Our goal is different in that we assume a centralized high-level controller that guarantees the specification through deadlock resolution by choosing environment assumptions to avoid both deadlock and livelock.

1.3.2 Specification Revisions

Recent efforts in reactive synthesis have focused on automatically identifying certain environment assumptions that may prevent the existence of a controller that satisfies the task. Approaches to assumption-mining have provided techniques that enable automatic specification debugging for specifications of any structure [4, 28]. While providing the ability to automate the debugging process, they still requires input from the user, for instance the variables the user desires and a final selection of candidate assumptions generated by the algorithm, which has drawbacks for realizing a fully-automated robotic mission planner. An assumption-mining approach to certify the necessary environment assumptions for a given task and robot dynamics was introduced in [11], however, the dynamics-based abstraction do not extend naturally to multi-agent scenarios. This proposed approach obviates the need for the user to intervene during the planning process.

We propose a novel approach in which assumptions on the environment are generated to identify likely deadlock situations. These added assumptions may be interpreted as restricting the mobility of the uncontrolled agents and are relaxed, when possible, by identifying when they may be violated, if only on a temporary basis. In this regard, our approach is inspired by works on error resilience [17] and recovery [43] in reactive synthesis.

1.3.3 Motion Planning in Dynamic Environments

Collision-free (and deadlock-free) motion planning for multi-robot teams has been successfully demonstrated via non-convex optimization, as proposed in [5, 33], but these approaches did not account for dynamic obstacles, nor could be computed in real-time. On the other hand, convex optimization approaches for collision avoidance, such as [6] and [1], are online and account for dynamic obstacles, but cannot reason globally to resolve deadlocks. In this work, we extend these works to enforce collision avoidance and motion constraints over a short time horizon, where the global execution is given by a discrete controller synthesized from a mission specification.

Also relevant to our efforts are the works in *deadlock resolution*. The authors of [23] applied pedestrian-avoidance principles to deadlock resolution in narrow passageways. While our approach is similarly reactive to the environment, we additionally reason about situations that cannot be locally resolved (e.g. a blocked corridor). Along similar lines, [10] described a centralized graph search technique for motion planning, but did not consider dynamic obstacles, and required a rich underlying graph to represent multi-robot motions with kinematic constraints. In contrast, our proposed local planning approach presents a more concise discrete abstraction and also applies to 3D environments. Traditional motion planning approaches such as RRT [27], PRM [20] and lattice based planners [34] can also be applied to compute collision - and deadlock - free motions for a single robot. But, in contrast to our synthesis approach, they do not typically reason about the mission strategy of multiple robots, nor encode logical constraints representing mission specifications.

1.4 Organization

The remainder of this paper is structured as follows. The required concepts for offline synthesis and online motion planning are described in Sec. 2. We formalize the problem in Sec. 3 and give an overview of the method in Sec. 4. In Sec. 5, we introduce a strategy for mission planning for resolving deadlock at runtime, while, in Sec. 6, we introduce an automated approach for generating runtime certificates and a coordination scheme for mission planning. In Sec. 7, we describe the online motion planner. We provide theoretical guarantees of the integrated approach in Sec. 8. In Sec. 9, we present extensive simulation and experimental results. Conclusions and future work are provided in Sec. 10.

2 Preliminaries

Throughout this paper scalars are denoted in italics, x, and vectors in bold, $\mathbf{x} \in \mathbb{R}^n$, with n denoting the dimension of the workspace. The robot's current position is denoted by $\mathbf{p} \in \mathbb{R}^n$ and its current velocity by $\mathbf{v} = \dot{\mathbf{p}}$. A map of the workspace $W \subset \mathbb{R}^n$ is considered, and formed by a set of static obstacles, given by a list of polytopes, $\mathcal{O} \subset \mathbb{R}^n$. For mission synthesis the map is abstracted by a set of discrete regions $\mathcal{R} = \{R_1, \ldots, R_p\}$, and their topological connections, covering the obstacle-free workspace $F = \mathbb{R}^n \setminus \mathcal{O}$, where the open sets $R_\alpha \subseteq W$.

We consider robots moving in $\mathbb{R}^3$ and approximate them by their smallest enclosing cylinder of radius r and height $2h$, denoted by V. Its ε-additive dilation of radius $\bar{r} = r + \varepsilon$ and height $\bar{h} = h + \varepsilon$ is denoted by V_ε. For a set $X \subset \mathbb{R}^n$ we denote the collision set by $X + V = \{\mathbf{p} \in \mathbb{R}^n \mid X \cap V(\mathbf{p}) \neq \emptyset\}$, with $V(\mathbf{p})$ a volume V at position $\mathbf{p}$. Throughout, the notation $\|\cdot\|$ is used to denote the Euclidean norm.

We consider a set of dynamic obstacles DO and denote the volume occupied by a dynamic obstacle $i \in DO$, at position $\mathbf{p}_i$, by $V_i(\mathbf{p}_i)$. To be able to prove safety in dynamic environments, we assume that all moving obstacles either maintain a constant velocity during the planning horizon (a couple of seconds), or that they employ an identical algorithm for collision avoidance as our robots, as introduced in the Reciprocal Velocity Obstacles literature [3]. In this work we do not treat the case where moving obstacles seek collisions and are capable of overtaking the robots. Instead, we assume a fair environment - one where it is always possible for the robots to avoid collisions - such as the case when operating with humans or other risk-adverse agents.

2.1 Linear Temporal Logic

LTL formulas are defined over the set AP of atomic (Boolean) propositions by the recursive grammar $\varphi ::= \pi \in AP \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$. From the Boolean operators $\wedge$ "conjunction" and $\neg$ "negation", and the temporal operators $\bigcirc$ "next" and $\mathcal{U}$ "until", the following operators are derived: "disjunction" $\vee$, "implication" $\Rightarrow$, "equivalence" $\Leftrightarrow$, "always" $\Box$, and "eventually" $\Diamond$. We refer the reader to [42] for a description of the semantics of LTL. Let AP represent the set of atomic propositions, consisting of *environment* propositions ($\mathcal{X}$) corresponding to thresholded sensor values, and *system* propositions ($\mathcal{Y}$) corresponding to the robot's actions and location with respect to a partitioning of the workspace. The value of each $\pi \in \mathcal{X} \cup \mathcal{Y}$ is the abstracted binary state of a low-level component. These might correspond to, for instance, thresholded sensor values, discrete actions that a robot can take, or a discrete region (e.g. room in a house).

Definition 1 ***(Reactive Mission Specification)*** A *Reactive Mission Specification* is a LTL formula of the form $\varphi = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e \implies \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$, with s and e standing for 'system' and 'environment', such that

- φ_i^e, φ_i^s are formulas for the *initial conditions* free of temporal operators.
- φ_t^e, φ_t^s are the *safety conditions* (transitions) to be satisfied always, and are of the form $\Box\psi$, where ψ is a Boolean formula constructed from subformulas in $AP \cup \bigcirc AP$.
- φ_g^e, φ_g^s are the *liveness conditions* (goals) to be satisfied infinitely often, with each taking the form $\Box\Diamond\psi$, with ψ a Boolean formula constructed from subformulas in $AP \cup \bigcirc AP$.

A strategy automaton that *realizes* a reactive mission specification φ is a deterministic strategy that, given a finite sequence of truth assignments to the variables in $\mathcal{X}$ and $\mathcal{Y}$, and the next truth assignment to variables in $\mathcal{X}$, provides a truth assignment to variables in $\mathcal{Y}$ such that the resulting infinite sequence satisfies φ. If such a strategy can be found, φ is *realizable*. Otherwise, it is *unrealizable*. Using a fragment of LTL known as *generalized reactivity(1)*, a strategy automata for φ of the form above can be *efficiently* synthesized [8], and converted into hybrid controllers for robotic systems by invoking atomic controllers [26]. These controllers are *reactive*: they respond to sensor events at runtime.

2.2 LTL Encoding for Multi-Robot Tasks

We adopt a LTL encoding of a centralized multi-robot mission that is robust to the inherent variability in the duration of inter-region robot motion in continuous environments [37]. Let $AP_{\mathcal{R}} = \{\pi_\alpha^i \mid R_\alpha \in \mathcal{R}\}$ be the set of Boolean propositions representing the workspace regions, such that $\pi_\alpha^i \in AP_{\mathcal{R}}$ is `True` when robot i is physically in R_α for $\alpha \in [1, \dots, p]$. We call π_α^i in $AP_{\mathcal{R}} \subseteq \mathcal{X}$ a *completion* proposition, signaling when robot i is physically inside R_α. We also define the set $AP_{\mathcal{R}}^{act} \subseteq \mathcal{Y}$ that captures robot commands that *initiate* movement between regions. We call $\pi_{act,\alpha}^i$ in $AP_{\mathcal{R}}^{act}$ an *activation* variable for moving to R_α (but has not necessarily completed motion to R_α). Non-motion actions are handled similarly. Observe that π_α^i and $\pi_{act,\alpha'}^i$ may be true at the same time if robot i is in R_α and is moving toward $R_{\alpha'}$, where R_α and $R_{\alpha'}$ are adjacent regions. Also note that this is sufficient for the special case π_α^i and $\pi_{act,\alpha}^i$ (the robot stays put). We assume reasonably that non-motion actions are independent of motion, so that actions themselves do not involve moving within any particular region and, if it is possible to execute a particular action within a region, it can be performed anywhere within that region.

We now solidify the semantics of the LTL formulas in the context of robot mission and motion planning. Let T denote a particular fixed time step at which the strategy automaton is updated with sensory information and supplies a new input to the local planner (as described in Sec. 4.3). A proposition $\pi \in AP$ is `True` at time $t > 0$ iff $\bigcirc \pi \in \bigcirc AP$ is `True` at $t + T$.

Definition 2 ***(LTL Encoding of Motion*** **[37])** A task encoding that admits arbitrary controller execution durations is

$$\varphi_t^s : \bigwedge_{\substack{\pi_\alpha^i \in AP_{\mathcal{R}}, \\ i \in [1, n_{robots}]}} \square(\bigcirc \pi_\alpha^i \Rightarrow \bigvee_{R_\beta \in Adj(R_\alpha)} \bigcirc \pi_{act,\beta}^i),$$

$$\varphi_t^e : \bigwedge_{\substack{\pi_\alpha^i \in AP_{\mathcal{R}}, \\ R_\beta \in Adj(R_\alpha), \\ i \in [1, n_{robots}]}} \square(\pi_\alpha^i \wedge \pi_{act,\beta}^i \Rightarrow \bigcirc \pi_\alpha^i \vee \bigcirc \pi_\beta^i),$$

$$\varphi_g^e : \square \diamondsuit \bigwedge_{\substack{\pi_{act,\alpha}^i \in AP_{\mathcal{R}}^{act}, \\ i \in [1, n_{robots}]}} \left((\pi_{act,\alpha}^i \wedge \bigcirc(\pi_\alpha^i \vee \neg \pi_{act,\alpha}^i)) \vee \right.$$
$$\left. (\neg \pi_{act,\alpha}^i \wedge \bigcirc(\neg \pi_\alpha^i \vee \pi_{act,\alpha}^i)) \right),$$

where $Adj : \mathcal{R} \to 2^{\mathcal{R}}$ is an adjacency relation on regions in $\mathcal{R}$ and n_{robots} is the number of robots. The φ_t^s-formula is a system safety condition describing which actions can occur ($\bigcirc \pi_{act,\beta}^i$) given the observed completion variables ($\bigcirc \pi_\alpha^i$). Formula φ_t^e captures the allowed transitions ($\bigcirc \pi_\beta^i$) given past completion (π_α^i) and activation ($\pi_{act,\beta}^i$) variables. Formula φ_g^e enforces that every motion and every action eventually completes (first disjunct) as long as the activation variable is held fixed (second disjunct). Specifically, the second disjunct in this formula allows the system to change its mind for a given action, absolving the environment from having to complete motion for that action. Both φ_t^e and φ_g^e are included as conjuncts to the antecedent of φ.

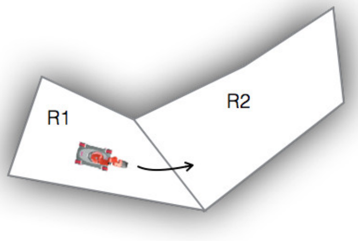


Fig. 2: Example of two connected regions.

Take, for example, two regions $R1$ and $R2$, arranged as shown in Fig. 2, with a robot positioned in $R1$ and heading toward $R2$. The system can only take a subset of actions; in this case, it is free to stay in $R1$ or move to $R2$:

$$\square(\bigcirc \pi_{R1} \implies \bigcirc \pi_{act,R1} \vee \bigcirc \pi_{act,R2}).$$

Upon taking an action, for instance move to $R2$ (activate $\pi_{act,R2}$), the system is allowed to be in either of the two regions

$$\square(\pi_{R1} \wedge \pi_{act,R2} \implies \bigcirc \pi_{R1} \vee \bigcirc \pi_{R2}),$$

and the environment must eventually allow the system to either arrive at this region or change course

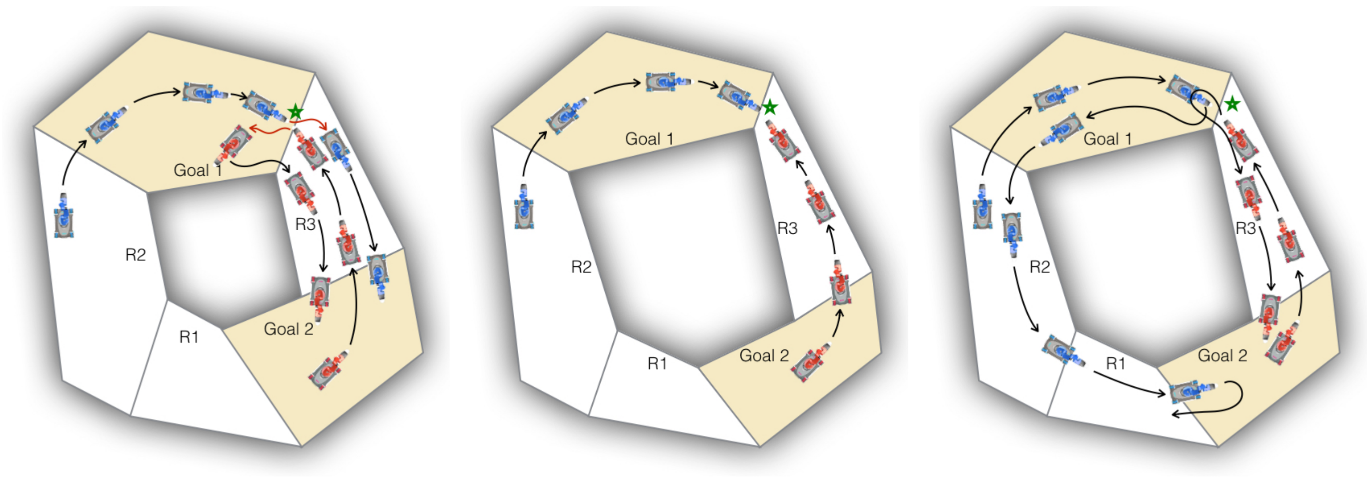
$$\square \diamondsuit((\pi_{act,R2} \wedge \bigcirc(\pi_{R2} \vee \neg \pi_{act,R2})) \vee$$
$$(\neg \pi_{act,R2} \wedge \bigcirc(\neg \pi_{R2} \vee \pi_{act,R2}))).$$

To complete the motion encoding, mutual exclusion is also enforced to express the fact that the robot can only be in one region at a time and must decide on one motion at a time. That is, $\square(\pi_{R1} \vee \pi_{R2})$ and $\square(\pi_{act,R1} \vee \pi_{act,R2})$.

We note that it is shown in [15] that complexity of synthesis under the generalized reactivity(1) fragment is polynomial in the size of the state space of the game structure that is, in turn, at most exponential in the total number of propositions. Considering motion alone, the formulas effectively impose restrictions to the allowed state transitions to only consider those that are physically adjacent, effectively reducing the size of the synthesis problem.

3 Problem Formulation

This work combines global planning with local motion planning to produce a correct-by-construction synthesis method



(a) Specification φ with local planner (b) Specification φ with local planner (c) Proposed approach with deadlock resolution

Fig. 3: Examples of integrated mission and motion planning. The blue robot starts in the region Goal 1 (top) and is tasked to visit Goal 2 (bottom right) and return to Goal 1. The red robot is placed in the region Goal 2 and is tasked to visit Goal 1 and return. The shortest path for both robots, given by solving a specification φ is to go through the corridor on the right. In (a), an execution of a specification φ using a local planner that locally avoids the collision between both robots and succeeds in executing the mission. (b) employs the same specification as (a), but the workspace is shrunk, resulting in a deadlock at location $\star$. (c) shows an execution of a controller synthesized from the modified specification φ'' using the deadlock resolution strategy and local planner developed in this work. With our approach, dynamic obstacles can be avoided locally, as in (a), and deadlocks can also be resolved.

that avoids collisions locally yet is able to resolve deadlocks. Synthesis is carried out in a fully-automated way; when modifications to the original specification are necessary, these are explained to the user in an intelligible manner. We provide an example to motivate our correct-by-construction synthesis method.

Example 1 Consider the workspace in Fig. 3b, where two robots are tasked with visiting regions Goal1 and Goal2 infinitely often; that is,

$$\varphi_s^g = \bigwedge_{i\in\{1,2\}} \square\lozenge(\pi^i_{Goal1}) \wedge \square\lozenge(\pi^i_{Goal2}).$$

Figure 3 illustrates two approaches for solving this task. Figures 3a and 3b show the result of applying a local motion planning scheme to locally avoid collisions with other robots or dynamic obstacles. In certain instances, such as the case shown in Fig. 3b, deadlocks can lead to the execution failing to satisfy the task.

Our approach, shown in Fig. 3c, relies on a local motion planner to allow several agents per region and avoid dynamic obstacles, as in Fig. 3a. Furthermore, it is able to resolve encountered deadlocks that may arise. In this example, when one of the robots encounters deadlock, it reverses its motion to allow the other one to pass into Goal 1, ultimately taking another route to Goal 2.

Definition 3 ***(Collision)*** A robot at position $\mathbf{p}$ is in collision with a static obstacle if $V(\mathbf{p}) \cap \mathcal{O} \neq \emptyset$. The robot is in collision with a dynamic obstacle i at position $\mathbf{p}_i$ and of volume $V_i(\mathbf{p}_i)$ if $V(\mathbf{p}) \cap V_i(\mathbf{p}_i) \neq \emptyset$.

Denote by $\mathbf{p}(t)$ the position of a robot at time t and by $\mathbf{p}_i(t)$ the position of a dynamic obstacle i at time t. The trajectory of the dynamic obstacles is estimated between the current time t_k and a time horizon τ. In our model we consider constant velocity.

Definition 4 ***(Collision Free Local Motion)*** A trajectory is said to be collision free if for all times between t_k and the time horizon there is no collision between the robot and any static or dynamic obstacle,

$$V(\mathbf{p}(t)) \cap \left(\mathcal{O} \bigcup_{i\in DO} V_i(\mathbf{p}_i(t))\right) = \emptyset \qquad \forall t \in [t_k, t_k+\tau]. \tag{1}$$

Which is equivalent to

$$V(\mathbf{p}(t)) \subset F \qquad \text{and}$$
$$V(\mathbf{p}(t)) \cap V_i(\mathbf{p}_i(t)) = \emptyset \quad \forall t \in [t_k, t_k+\tau],\ \forall i \in DO. \tag{2}$$

Definition 5 ***(Deadlock)*** In this work we consider motion related deadlocks. A robot at position $\mathbf{p}$ is said to be in a deadlock if it is not in a collision, it has not achieved the target given by the automaton and it can not make progress towards the goal, i.e. it is not moving, for a prespecified amount of time.

The goal of this work is to solve a set of problems as follows.

Problem 1 ***(Local Collision Avoidance)*** Given the dynamics for each robot in the team, construct an online local planner that guarantees collision avoidance with static and dynamic (moving) obstacles.

Problem 2 ***(Synthesis of Strategy Automaton with Deadlock Resolution)*** Given a topological map, a local motion planner that solves Problem 1 and a realizable mission specification φ that ignores collisions, automatically construct a specification φ' that includes both φ and a model of deadlock between robots and unmodeled dynamic obstacles. Use φ' to synthesize a controller that satisfies φ'.

This synthesized controller will re-route the robots to resolve deadlocks (should they occur), while satisfying the reactive mission specification and remaining livelock free. For mission specifications that consider the presence of possible deadlocks, there may be no satisfying controller. We therefore synthesize environment assumption revisions as additional LTL formulas to identify cases where dynamic obstacles may trigger deadlock and trap the system from achieving its goals. These formulas are significant because they offer *certificates* explaining the required behaviors of the environment that, if followed, guarantee that the robot team will carry out the task. Such certificates must be conveyed to the user in a clear, understandable manner. An example of such a condition is: "the environment will never cause deadlock if robot 1 is in the kitchen and moving to the door". This leads to the following Problem.

Problem 3 ***(Revising Environment Assumptions)*** Given an *unrealizable* reactive mission specification φ', synthesize environment assumption revisions $[\varphi_t^e]^{rev}$ such that the specification φ'' formed by replacing φ_t^e with $[\varphi_t^e]^{rev}$ is realizable, and provide the user with a human-readable description of these revisions as certificates for guaranteeing the task.

4 Approach

This work solves Problems 1, 2 and 3 via a combined offline and online approach, which (a) synthesizes a strategy automaton that realizes the mission and (b) computes a local motion planner that executes the automaton in a collision-free manner. Figure 4 highlights the offline and online components and their interconnections, which we now introduce.

4.1 Offline

The inputs for the offline part of the method are: (a) a user given mission specification, (b) a discrete topological map of the workspace (which ignores dynamic obstacles) and (c) the dynamic model and controller of the robots in the team. The offline part of the method consists of two independent parts.

4.1.1 Mission Planning

In this step we synthesize a centralized controller, or finite state machine, that will guide the robots in the team through the topological map. This controller considers possible *physical deadlocks* between robots in the team as well as with moving obstacles. Since the position of the moving obstacles is not known at synthesis time, environment assumptions are iteratively revised as necessary. The resulting strategy automaton with the revisions included accommodate deadlocks wherever they may occur at runtime, and fulfillment of the specification is guaranteed as long as the environment behaves according to the assumptions explained to the user in the revisions generation step. We also adopt a recovery scheme [43] that synthesizes a strategy that allows violations of environment *safety* assumptions to be tolerated, retaining satisfaction guarantees as long as the violation is transient.

The *mission planning* part of the offline synthesis approach is described in detail in Sec. 5 and Sec. 6.

4.1.2 Motion Planning

The automaton is agnostic to the robot's dynamics, which are instead accounted for by the local planner. For a given robot model and controller a set of motion constraints, or tracking errors, are precomputed at synthesis time. This part is described in Sec. 7.2.

During execution, the local planner is fed, at runtime, a set of constraints that are then solved for in an efficient manner. These constraints include region boundaries, static and dynamic obstacles and kinodynamic model of the robot.

4.2 Online

At each time step of the execution, the synthesized strategy automaton provides a desired goal for each controlled robot in the team. Then, each robot independently computes a local trajectory that achieves its goal while avoiding other agents.

If a physical deadlock is sensed, an alternative goal is extracted for the robot from the synthesized strategy automaton. The existence of such an alternative in the automaton is guaranteed by construction if the environment assumptions

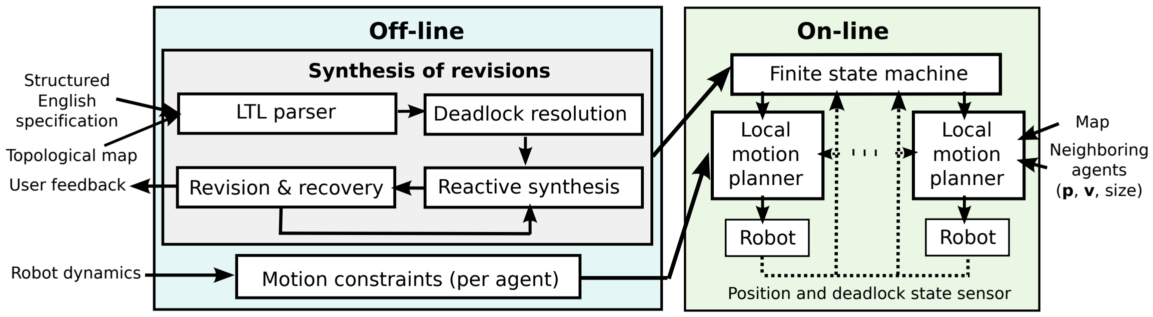


Fig. 4: Structure of the proposed mission and motion planner, with offline and online parts. The mission planning is offline and is described in Sec. 5 and in Sec. 6. The motion planner, Sec. 7, is computed at runtime and utilizes the strategy automaton (finite-state machine) synthesized offline by the mission planner.

are satisfied. The local planner builds on [3] by adopting a convex optimization approach as described in Sec. 7.

4.3 Integration of mission and motion planning

The proposed method consist of two interconnected parts, the mission planner and the motion planner. Figure 4 highlights the components and their interconnections.

The mission planner is computed offline, prior to execution. It requires a topological map of the environment given by a description of the regions, such as rooms, and their connections. It creates a finite state machine or automaton that achieve the high-level specification and from which the robots in the team can extract a strategy at runtime. Note that we do not optimize the mission planner in this work, but our framework allows us to readily adopt techniques for optimal execution such as [21] to extract an optimal strategy automaton.

At each time instance in the execution, a target motion is extracted from the automaton. The motion planner computes a collision-free motion to make progress towards the target. If a physical deadlock is sensed, an alternative strategy is extracted from the automaton.

The motion planner requires a local map of the environment W, containing all the static and moving obstacles. The regions in the free space F of the local map - used at runtime - must be labeled to match the regions $\mathcal{R}$ of the topological map - used for offline synthesis.

If the automaton commands a robot to transition between two connected regions, a path is computed from the current position of the robot to the border of the destination region and then is followed by the local planner. If the automaton commands a robot to remain in a region, the local planner moves the robot towards the middle point of the region.

5 Offline Synthesis: Resolving Deadlock

In this section, we discuss how to synthesize a strategy automaton given a mission specification and a topological map of the environment, provided that, at runtime, a low-level control strategy is applied that guarantees collision-free motion. We assume that the task specification φ ignores collisions, but we allow the possibility that deadlocks can occur at any time during the robot's execution. Deadlocks can trap the robot from achieving its goals, rendering the specification unrealizable. The crux of this work is an approach that systematically modifies the specification with additional behaviors that redirect the robot team in order to resolve deadlocks, whenever possible. If a satisfying mission plan does not exist, the approach iteratively adds assumptions on the deadlock behavior to the specification until a satisfying strategy can be found for the robot team. By focusing on deadlock rather than the positioning of dynamic obstacles, it allows our approach to be valid for any number of dynamic obstacles, as long as they fulfill the stated assumptions returned by our synthesis approach. It also removes the need to globally track the positions of every obstacle at runtime.

An outline of the general approach is shown in Fig. 5. Such a strategy was chosen to disable any blocked routes to the goal and thereby enable the strategy automaton to seek alternate routes once deadlock has been encountered. In this section, we detail the steps involved to implement the overall approach.

5.1 Deadlock Resolution

We declare a robot to be physically in deadlock with another agent if it has not reached its goal but cannot move. This can happen when an agent becomes blocked either by another agent or by a dynamic obstacle. To keep track of which robot is in deadlock, we introduce Boolean input signals $x^{ij} \in \mathcal{X}$, where $i = 1, \ldots, n_{robots}$ and $j = 0, \ldots, n_{robots}$ (the index $j = 0$ representing a dynamic obstacle). Without loss of generality, we consider only deadlock between pairs of agents at a time. For the case where a robot is in deadlock while in proximity to a dynamic obstacle, we let $j = 0$ and refer to this case as *singleton deadlock*. Otherwise, the robot is in deadlock with another robot on its team, $j \neq 0$, and is considered to be in a state of *pairwise deadlock*. The proposition x^{i0} is `True` iff robot i is in singleton deadlock and x^{ij} is `True` iff robots i and j are in pairwise deadlock. We defer detailing our approach for detecting deadlock at runtime to Sec. 7.

To simplify the notation in what follows, we introduce the following shorthand:

$\theta^{ij}_P = \neg x^{ij} \wedge \bigcirc x^{ij}$	rising edge–pairwise deadlock between robots i and j
$\theta^{i}_S = \neg x^{i0} \wedge \bigcirc x^{i0}$	rising edge–singleton deadlock for robot i
$\psi^{i}_{\alpha\beta} = \pi^i_\alpha \wedge \bigcirc \pi^i_\alpha \wedge \pi^i_{act,\beta}$	incomplete transition ($\alpha \neq \beta$); remain in region ($\alpha = \beta$)

The definition for singleton deadlock is abstract enough to capture the case where deadlock occurs between the robot and any number of dynamic obstacles - singleton deadlock will be set if the robot stops moving when encountering one or more dynamic obstacles blocking its path. On the other hand, since the members of the team are controlled by the same mission planner, pairwise deadlock can be resolved separately. For instance, if three robots on a team converge on the same point, then three pairwise deadlock propositions will be set.

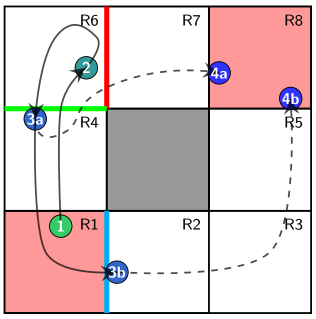


Fig. 5: Diagram illustrating the deadlock resolution strategy for a single robot tasked with visiting $R1$ and $R8$. Starting in region $R1$ (marked '1'), the robot encounters deadlock (2) in region $R6$, while heading to $R7$. The $R6$-to-$R7$ transition is prevented (red line), and the robot must move a discrete radius m away from the deadlock event to resolve deadlock. If $m = 1$, then deadlock is resolved once the robot crosses the green line, leaving $R6$ (3a). From there, it may reach $R8$ (4a) if no other deadlocks are encountered. On the other hand, when $m = 3$, deadlock is resolved only when crossing the cyan line (3b); an alternate path to the goal may result (4b).

Resolving deadlock by redirecting the robot's motion based on the instantaneous value of x^{ij} alone may result in livelock, where the robot may be trapped from achieving its goals as a result of repeated deadlock status changes. For this reason, our scheme automatically introduces additional memory propositions that are set when deadlock is sensed, and reset once the robot moves a predefined *discrete radius*, denoted m, defining the a deadlock resolution horizon (i.e. it traverses m regions away from the region where deadlock occurred in order for the deadlock to be considered "resolved").

Definition 6 ***(Discrete Radius)*** Let $\pi^i_{curr(k_i)} \in AP_{\mathcal{R}}$ and $\pi^i_{act,curr(k_i-1)} \in AP^{act}_{\mathcal{R}}$ be, respectively, the configuration and action taken by robot i, where $k_i = 1, 2, \ldots$ represents an event that is incremented when robot i enters a new region, i.e. k_i is incremented at the time instant when $curr(k_i-1) \leftarrow curr(k_i)$. The current region index $curr(\cdot) \in [1, p]$ is defined recursively, initialized such that $\pi^i_{curr(1)}$ is the robot's completion when deadlock was recorded and $\pi^i_{act,curr(0)}$ is the robot's action when deadlock was recorded. Then, the *discrete radius* m is the number of successive steps $k_i \in [1, m]$ for which we impose the restriction $\pi^i_{act,curr(k_i)} \in AP^{act}_{\mathcal{R}} \backslash \{\pi^i_{act,curr(k_i-1)}\}$ on the robot's actions. This ensures that the robot makes a move that does not re-enter the region just visited.

The concept behind the proposed deadlock resolution approach is to force the robot to actively alter its strategy to overcome a deadlock by imposing a small number of constraints without directly prescribing the path the robot is required to take. The path is derived once a strategy automaton is synthesized from the specification augmented with these revisions. For instance, as illustrated in Fig. 5, for the case $m = 1$ (resp. $m = 3$), if a deadlock is sensed at point (2), the revisions forbid the robot from crossing the red line until it reaches the green line (resp. cyan line). As a result, different choices of m will lead to the synthesis of strategies that give rise to different subsequent paths to goal region $R8$ and decisions whether or not to revisit the location where deadlock had occurred.

We first introduce an approach where resolution occurs when the robot leaves its current region, then generalize this approach to allow the user to choose any number of discrete steps, $m \geq 0$, to be taken by the robot before deadlock is declared as resolved. In this work, we assume m to be chosen ahead of time.

5.2 Resolving Deadlock when $m = 0$

Our deadlock resolution approach for the case $m = 0$ amounts to the situation where robot i is forced to move in another direction whenever x^{ij} becomes `True` for $j = 0, \ldots, n_{robots}$.

As long as x^{ij} remains `True` when robot i is in region R_α, we disallow motion to R_β as follows:

$$\square \bigwedge_{\substack{\pi^i_\alpha \in AP_\mathcal{R},\\ R_\beta \in Adj(R_\alpha)}} \left(\bigcirc x^{ij} \wedge \pi^i_\alpha \implies \bigcirc(\neg\pi^i_{act,\alpha} \wedge \neg\pi^i_{act,\beta})\right). \quad (3)$$

It is easily observed that, as soon as the robot's motion is nonzero when it begins to move in a direction opposite to its previous motion, x^{ij} becomes `False` again and the robot is free to resume its motion to R_β. This can lead to unwanted behaviors, such as chattering. To avoid chattering behaviors, we enrich the deadlock resolution approach to allow for any choice of $m > 0$.

5.3 Resolving Deadlock when $m = 1$

For each robot, we introduce into $\mathcal{Y}$ the system propositions $\{y^i_\beta \mid R_\beta \in \mathcal{R}\} \subset \mathcal{Y}$ representing the *deadlock flag* occurring when activating a transition from a given region R_α to region R_β. When the flag is set, the following formula restricts the robot's motion:

$$\square \bigwedge_{\substack{\pi^i_\alpha \in AP_\mathcal{R},\\ R_\beta \in Adj(R_\alpha)}} \left(y^i_\beta \wedge \pi^i_\alpha \implies \bigcirc(\neg\pi^i_{act,\alpha} \wedge \neg\pi^i_{act,\beta})\right). \quad (4)$$

The role of y^i_β is to disallow the current transition (from R_α to R_β), as well as the self-transition from R_α to R_α. The self-transition is disallowed to force the robot to leave the region where the deadlock occurred (R_α), instead of waiting for it to resolve; R_β is disallowed since the robot cannot make that transition.

Next, we encode conditions for detecting *singleton deadlock* at runtime, and storing these as propositions y^i_β that memorize that singleton deadlock had occurred:

$$\square \bigwedge_{\substack{\pi^i_\alpha \in AP_\mathcal{R},\\ R_\beta \in Adj(R_\alpha)}} \left(\neg y^i_\beta \Rightarrow \left((\theta^i_S \wedge \psi^i_{\alpha\beta}) \Rightarrow \bigcirc y^i_\beta\right)\right), \quad (5)$$

$$\square \bigwedge_{\substack{\pi^i_\alpha \in AP_\mathcal{R},\\ R_\beta \in Adj(R_\alpha)}} \left(y^i_\beta \Rightarrow \left((\pi^i_\alpha \wedge \bigcirc \pi^i_\alpha) \Leftrightarrow \bigcirc y^i_\beta\right)\right). \quad (6)$$

The first formula sets the deadlock flag y^i_β if the robot is activating transition from R_α to R_β. The second formula keeps the flag set until a transition has been made out of R_α (to a region different from R_β). Notice that, in our construction, singleton deadlock considers deadlock between one robot and *any number of* dynamic obstacles, alleviating the need to globally track or identify obstacles at runtime. While this construction could introduce cycling, we prefer it over an approach that stores the entire path because we can limit the number of propositions added to $\mathcal{Y}$ in order to manage complexity. For instance, if we are aware that deadlock does not occur when the robot is trying to reach a given region $R_\cdot$, we can eliminate the variable $y^i_\cdot$.

For *pairwise deadlock*, we add the following formulas encoding the conditions for declaring that pairwise deadlock has been detected. Note that the disjunction in the formula allows the synthesis tool to decide which one of the two robots should react to the deadlock:

$$\square \left(\theta^{ij}_P \implies \Big(\bigvee_{\ell \in \{ij\}} \bigwedge_{\substack{\pi^\ell_\alpha \in AP_\mathcal{R},\\ R_\beta \in Adj(R_\alpha)}} (\neg y^\ell_\beta \wedge \psi^\ell_{\alpha\beta}) \implies \bigcirc y^\ell_\beta\Big)\right). \quad (7)$$

We also add the following to ensure that the memory propositions are only set when the rising edge of deadlock (singleton or pairwise) is sensed.

$$\square \left(\bigwedge_{\substack{i \in [1,n_{robots}]\\ R_\beta \in \mathcal{R}}} \Big(\neg y^i_\beta \wedge \neg\theta^i_S \wedge \bigwedge_{\substack{j \in [1,n_{robots}]\\ j \neq i}} \neg\theta^{ij}_P\Big) \implies \bigcirc \neg y^i_\beta\right). (8)$$

In practice, we do not need a proposition y^i_β for every $R_\beta \in \mathcal{R}$, but only $d = \max_{R_\alpha \in \mathcal{R}} (|Adj(R_\alpha)|)$ such propositions for each robot in order to remember all of the deadlocks around each region of the workspace. Here $|\cdot|$ denotes the set cardinality. The number of conjuncts required for condition (7) is $\binom{n_{robots}}{2}$, but, since the number of formulas contributes at worst linear complexity (due to parsing of each formula), the conjuncts contribute only a small amount to the overall complexity. Note that the complexity of the synthesis algorithm is a function of the number of propositions and not the size of the specification.

Conjuncting the conditions (4)–(8) with φ^s_t yields a modified formula $[\varphi^s_t]'$ over the set AP, and the new *abstracted* specification $\varphi^{abstr} = \varphi^e_i \wedge \varphi^e_t \wedge \varphi^e_g \implies [\varphi^s_i]' \wedge [\varphi^s_t]' \wedge \varphi^s_g$. The initial conditions are modified by setting the additional propositions x^{ij}, y^i_α to `False`.

5.4 Resolving Deadlock when $m > 1$

In some cases, having a deadlock resolution strategy in which multiple discrete steps must be made away from any encountered deadlock may result in different behavior than a strategy in which deadlock is resolved when moving away just one step. Considering Fig. 5, the case $m = 3$ results in greater exploration of the workspace, whereas the case $m = 1$ results in confinement to a smaller portion of the workspace.

We generalize the strategy presented in Sec. 5.3 by considering the case where deadlock is resolved once $m > 1$ discrete moves have been taken away from the last encountered deadlock. In what follows, the same formulas as in Sec. 5.3 apply; here, we only describe modifications to this setup. To ensure each robot moves away from deadlock a discrete radius, we require $m - 1$ propositions (for robot i, $y^i_{out,1}, \ldots, y^i_{out,m-1}$) that are set and reset in a chain in order to memorize the robot's position from the encountered deadlock. $y^i_{out,k}$ are initially `False` for all i, k.

In order to set the first such memory proposition in the chain, the terms $\bigcirc y^i_\beta$ in (5) and $\bigcirc y^\ell_\beta$ in (7) are replaced with $\bigcirc y^i_\beta \wedge \bigcirc y^i_{out,1}$ and $\bigcirc y^\ell_\beta \wedge \bigcirc y^\ell_{out,1}$, respectively, and the abstracted specification φ^{abstr} is constructed based on these formulas. For each subsequent discrete step away from deadlock, we require the remaining propositions to be set when the one with next lowest index has been reset. This behavior occurs through the formula:

$$\square \bigwedge_{\substack{k=2,\ldots,\\ m-1}} \left(\neg y^i_{out,k} \Rightarrow \left((y^i_{out,k-1} \wedge \bigcirc \neg y^i_{out,k-1}) \Rightarrow \bigcirc y^i_{out,k}\right)\right) \quad (9)$$

Additionally, for each $k = 1, \ldots, m-1$, we require that each $y^i_{out,k}$ be reset only when the robot has left the current region; specifically,

$$\square \bigwedge_{\substack{\pi^i_\alpha \in AP_{\mathcal{R}},\\ k=1,\ldots,m-1}} \left(y^i_{out,k} \Rightarrow \left((\pi^i_\alpha \wedge \bigcirc \pi^i_\alpha) \Leftrightarrow \bigcirc y^i_{out,k}\right)\right). \quad (10)$$

Finally, as long as some $y^i_{out,k}$ is set, we also set the deadlock flag memory proposition y^i_α corresponding to the region R_α that the robot had immediately departed. That is,

$$\square \bigwedge_{\pi^i_\alpha \in AP_{\mathcal{R}}} \left((\pi^i_\alpha \wedge \bigvee_{k=1,\ldots,m-1} y^i_{out,k}) \Rightarrow \bigcirc y^i_\alpha\right). \quad (11)$$

This prevents the robot from re-entering the region from which it just departed.

The safety revisions restrict the system's moves in the execution sequence be ones that actively take it m away from the location where the deadlock flag was raised. Since waiting in a region is disabled in (4), and reentering a region is disabled in (11), these safety revisions will cause the system to move m steps away from deadlock in finite time.

In general, setting m large, could lead to behavior that "explores" more of the workspace, but also could result in unrealizability. Consider again the scenario in Fig. 5, but with R2 always blocked. In this case, $m = 3$ would result in an unrealizable specification because the robot cannot make three discrete steps away from R6 without entering R2. Such design tradeoffs therefore depend on the workspace and its partitioning. Automatic selection of m for a given specification and collection of regions is the subject of future work, as is the use of $\square \diamondsuit$ liveness formulas to resolve livelock in a more direct manner similarly to [4, 11] while remaining scalable to the number of robots on the team.

6 Offline Synthesis: Environment Assumptions and Coordination

If the specification φ^{abstr} is synthesizable, then Problem 2 has been solved and no further modifications to the abstracted specification are necessary. But, the possible presence of humans or other uncontrollable agents in some parts of the environment may cause the abstracted specification to be unrealizable. Then, it becomes necessary to solve Problem 3 to find a minimal set of environment assumptions that restores the guarantees.

We automatically generate assumptions on the environment's behavior in cases where the modified specification is unrealizable. To prevent any unreasonable assumptions (assumptions that the robot can overcome deadlock when it is impossible to do so), we provide a means for coordinating robot actions to prevent such assumptions from being given to the user. Combining the encoding and revisions approach, we formally show that the synthesized automaton is guaranteed to fulfill the task under these assumptions, showing that our approach also removes the possibility of deadlock and livelock from occurring.

6.1 Runtime Certificates for the Environment

We note that the dynamic obstacles are uncontrollable agents, and lacking behavioral information, so altering environment assumptions does nothing to characterize their behavior. Rather, we may still provide the user with a certificate under which the environment's behavior will guarantee that the team can achieve all its goals without being trapped permanently in a state of deadlock or livelock. Such assumptions can be given to the user to allow him/her to be mindful of any condemning situations when co-inhabiting the robots' environment. As such, we call these added assumptions *runtime certificates*.

When a specification is unrealizable, there exist environment behaviors (called *environment counterstrategies*) that prevent the system from achieving its goals safely. Here we build upon the work of [4, 11, 28], processing synthesized counterstrategies to mine the necessary assumptions. Rather than synthesize assumptions from the counterstrategy as in [4], which requires specification revision templates to be specified by hand, we automate the counterstrategy search by searching for all deadlock occurrences, then store the corresponding conditions as assumptions.

We denote $\mathcal{C}_{\varphi^{abstr}}$ as an automaton representing the counterstrategy for φ^{abstr}. Specifically, a counterstrategy is the tuple $\mathcal{C}_{\varphi^{abstr}} = (\mathcal{Q}, \mathcal{Q}_0, \mathcal{X}, \mathcal{Y}, \delta, \gamma_{\mathcal{X}}, \gamma_{\mathcal{Y}})$, where $\mathcal{Q}$ is the set of counterstrategy states; $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is the set of initial counterstrategy states; $\mathcal{X}, \mathcal{Y}$ are sets of propositions in AP; $\delta : \mathcal{Q} \times 2^{\mathcal{Y}} \rightarrow 2^{\mathcal{Q}}$ is a transition relation returning the set of possible successor states given the current state and valuations of robot commands in $\mathcal{Y}$; $\gamma_{\mathcal{X}} : \mathcal{Q} \rightarrow 2^{\mathcal{X}}$ is a labelling function mapping states to the set of environment propositions that are `True` for incoming transitions to that state; and $\gamma_{\mathcal{Y}} : \mathcal{Q} \rightarrow 2^{\mathcal{Y}}$ is a labelling function mapping states to the set of system propositions that are `True` in that state. We compute $\mathcal{C}_{\varphi^{abstr}}$ using the `slugs` synthesis tool [16].

To find the graph cuts in the counterstrategy graph that prevent the environment from impeding the system, we first define the following propositional representation of state $q \in \mathcal{Q}$ as $\psi(q) = \psi_{\mathcal{X}}(q) \wedge \psi_{\mathcal{Y}}(q)$, where

$$\psi_{\mathcal{Y}}(q) = \bigwedge_{\pi \in \gamma_{\mathcal{Y}}(q)} \pi \wedge \bigwedge_{\pi \in \mathcal{Y} \setminus \gamma_{\mathcal{Y}}(q)} \neg\pi,$$
$$\psi_{\mathcal{X}}(q) = \bigwedge_{\pi \in \gamma_{\mathcal{X}}(q)} \pi \wedge \bigwedge_{\pi \in \mathcal{X} \setminus \gamma_{\mathcal{X}}(q)} \neg\pi.$$

Next, letting $\delta_{\mathcal{Y}}(p) = \{q \in \mathcal{Q} | \exists \pi \in \mathcal{Y} : q \in \delta(p, \pi)\}$, the set of *cut transitions* S_{cuts} is computed as $S_{cuts} = \{(p,q) \in \mathcal{Q}^2 \mid q \in \delta_{\mathcal{Y}}(p), \psi(p) \wedge \psi(q) \models \bigvee_{i \in [1, n_{robots}]} \bigcirc \theta_S^i\}$. S_{cuts} collects those transitions on which the environment has intervened (by setting deadlock) to prevent the system from reaching its goals.

Finally, the following safety assumptions are found:

$$\varphi_{rev}^e = \square \bigwedge_{(p,q) \in S_{cuts}} (\psi_{\mathcal{Y}}(p) \wedge \psi_{\mathcal{X}}(p) \implies \neg \bigcirc \psi_{\mathcal{X}}(q)) \quad (12)$$

If any of the conjuncts in (12) falsify the antecedent of φ (the environment assumptions), they are discarded. Then, set $[\varphi_t^e]^{rev} = \varphi_t^e \wedge \varphi_{rev}^e$ and construct the final *revised* specification $\varphi^{rev} = \varphi_i^e \wedge [\varphi_t^e]^{rev} \wedge \varphi_g^e \implies [\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$.

Algorithm 1 expresses our proposed approach for resolving deadlock. The automatically generated assumptions act to restrict the behavior of the dynamic obstacles. Each revision of the high-level specification excludes at least one environment move in a given state. Letting $|\cdot|$ denote set cardinality, with $2^{|\mathcal{X}|}$ environment actions and $2^{|\mathcal{Y}|}$ states, at most $2^{(|\mathcal{Y}|+|\mathcal{X}|)}$ iterations occur, though in our experience far fewer are needed. The generated assumptions are minimally restrictive – omitting even one allows the environment to cause deadlock, resulting in unrealizability. Note that the parsing step in line 8 creates statements that are displayed to the user. The user display step is explained in detail in the implementation in Sec. 9.

Algorithm 1 Find realizable φ^{rev} fulfilling task φ and resolving deadlock.

```
1: $\varphi^{abstr} \leftarrow \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e \implies [\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$
2: $[\varphi_t^e]^{rev} \leftarrow \varphi_t^e$
3: $\varphi^{rev} \leftarrow \varphi_i^e \wedge [\varphi_t^e]^{rev} \wedge \varphi_g^e \Rightarrow [\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$
4: while $\varphi^{rev}$ is unrealizable do
5:     Extract $\mathcal{C}_{\varphi^{rev}}$ from $\varphi^{rev}$
6:     $\varphi_{rev}^e \leftarrow$ Eq. (12)
7:     for each $k$th conjunct of $\varphi_{rev}^e$ s.t. $\varphi_{rev}^e[k] \wedge [\varphi_t^e]^{rev} \neq$ False do
8:         Parse $\varphi_{rev}^e[k]$ into human-readable statements and display to user.
9:         $[\varphi_t^e]^{rev} \leftarrow [\varphi_t^e]^{rev} \wedge \varphi_{rev}^e[k]$
10:    end for
11:    $\varphi^{rev} \leftarrow \varphi_i^e \wedge [\varphi_t^e]^{rev} \wedge \varphi_g^e \Rightarrow [\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$
12: end while
```

In practice, many of the added environment safety statements can be violated by dynamic obstacles at runtime without consequence, if these violations can be assumed to be temporary. For this reason, we introduce a recovery scheme that synthesizes a strategy that allows environment *safety* assumption violations to be tolerated. We refer the reader to [43] for these technical details of the details of this strategy. Note that we modify the approach to attempt a recovery only for violations of the newly added assumption φ_{rev}^e, rather than for the entire formula $[\varphi_t^e]'$, since our goal is to only make assertions on the environment's behavior with respect to deadlock and not all behaviors in general. The requirement for temporary deadlock is less restrictive than the requirement that deadlocks should *never* occur, but it nonetheless places additional requirements on the environment's behaviors, i.e. that the dynamic obstacles cannot infinitely often cause deadlock. Hence such conditions are displayed to the user in an easily-interpretable form.

Runtime certificates are displayed to the user in a format such as: `The task is guaranteed as long as for robot 1 any singleton deadlock in the kitchen while heading to the door is eventually resolved on its own.` In this specific case, dynamic obstacles may enter deadlock with robot 1, but the obstacles are obligated to eventually resolve deadlock. If the dynamic obstacle is a person, the certificate may have no impact on the true behavior of the environment, as social norms deem it natural for people to resolve deadlocks on their own. If the dynamic obstacle is a door, then the certificate could alert that the door should eventually be opened to allow the robot to pass through. On the other hand, if the door never opens, then the certificate could help to explain that the door being closed as the reason the task remains unfulfilled.

It is possible that many such certificates are required, which may overwhelm the user. We address this in two ways. First, we project the found certificates onto the set of propositions relating to motion only, eliminating any propositions that do not relate to motion. Second, we use a graphical visualization of the certificates overlaid on a map of the physical workspace. In addition to the above provisions, the work in [11] offers an approach that can be adopted to further reduce the number of revisions fed to the user. There, a method is introduced for grouping regions that share the same properties for the revisions, and convey to the user metric information that is necessary for fulfilling the added revisions. Such an integration is left for future work. We refer the reader to Sec. 9 for implementation details.

We point out that (12) gives revisions that are possibly conservative. The formula is created from a counterstrategy that is extracted from a game structure capturing the environment's behaviors for every possible behavior of the system [11]. In the current implementation, the counterstrategy is computed without regard to the number of revisions that could be generated. Future iterations of the approach could make use of an optimality criterion to extract a counterstrategy with a minimal number of revisions. Another cause for conservatism is due to the fact that the approach abstracts away the actual behavior of the dynamic obstacles, neglecting the physical behavior of the dynamic obstacles. This can be improved by enhancing the existing approach with LTL formulas that impose physical constraints on the environment, for instance mutual exclusion conditions on deadlocks.

6.2 Coordination Between Robots

Since the strategy for the robots' motion is completely determined at synthesis time, the controllers we synthesize should not lead to deadlocks if they can be safely avoided. For instance, two robots on the team should not enter a narrow doorway from opposite ends, only to become deadlocked there. This motivates the creation of a method for automatically inserting dimension-related information into the specification based on the workspace geometry and the volume of the robot so that the robots can pre-coordinate, at synthesis time, to avoid unneeded deadlock. This pre-coordination serves two purposes: 1) it allows to eliminate any environment assumptions between two robots in a region where there is high likelihood of deadlock if both are occupying that region, and 2) it changes the behavior of the agents to actively avoid potential deadlock in such high-risk regions, such as one-way corridors.

The modification considers the restrictions on what robots are allowed to do in certain regions, based on the dimension of the region and the size of the robot. We introduce an encoding of LTL formulas that eliminate the actions of robots that would result in deadlock. Specifically, we consider two cases: 1) a robot will not enter a region if the move will exceed the region's capacity and, 2) it will be prevented that two or more robots enter through opposite sides a one-way narrow region. We then create a new specification $\varphi^{abstr,coord}$ with pre-coordination of robots, and apply Algorithm 1 on $\varphi^{abstr,coord}$ by swapping out φ^{abstr} in line 1.

To create the LTL encoding, we introduce Algorithm 2 to enforce pairwise coordination amongst robots in the controlled team. If the region is too small to contain a pair of robots, any robot outside of the region is prevented from entering (line 6). If the boundary between two regions R_α and R_β is too small for two robots to pass through at once, and one robot is approaching the boundary from R_α (resp. R_β), then no other robot may approach that boundary if in R_β (resp. R_α). This requirement is encoded in lines 11–12. Note that Algorithm 2 is general to any workspace with convex regions.

Algorithm 2 Augmenting a specification with agent coordination with respect to region geometry.

```
 1: $D \leftarrow$ max dimension of the enclosing hull of the robots on the team
 2: for each $R_\alpha \in \mathcal{R}$ do
 3:     $A \leftarrow$ area of region $R_\alpha$
 4:     if $\frac{A}{D} < 1$ then
 5:         // Region capacity is too small
 6:     $\varphi_t^s \leftarrow \varphi_t^s \wedge (\bigcirc \pi_\alpha^i \implies \neg \bigcirc \pi_{act,\alpha}^j) \wedge (\bigcirc \pi_\alpha^j \implies \neg \bigcirc \pi_{act,\alpha}^i)$
 7:     end if
 8:     for each $R_\beta \in Adj(R_\alpha)$ do
 9:         if $\|R_\alpha \cap R_\beta\| < 2D$ then
10:             // Boundary between $R_\alpha$ and $R_\beta$ is too narrow
11:             $\varphi_t^s \leftarrow \varphi_t^s \wedge \bigwedge_{i,j=1}^{n_{robots}} \left( (\psi_{\alpha\beta}^i \wedge \bigcirc \pi_\beta^j) \implies \bigcirc \neg \pi_{act,\alpha}^j \right)$
12:             $\varphi_t^s \leftarrow \varphi_t^s \wedge \bigwedge_{i,j=1}^{n_{robots}} \left( (\psi_{\beta\alpha}^i \wedge \bigcirc \pi_\alpha^j) \implies \bigcirc \neg \pi_{act,\beta}^j \right)$
13:         end if
14:     end for
15: end for
```

7 Online Local Motion Planning

In this section we describe the local planner that links the mission plan with the physical robot (recall Fig. 4). The offline synthesis and generated state machine are agnostic to the local planner, which can be substituted as long as avoidance of unmodeled moving obstacles is guaranteed. Our online local planner does account for the robot dynamics, which were abstracted for high-level synthesis.

At each step of the online execution, the synthesized strategy automaton provides a desired goal position for each robot and a preferred velocity $\bar{\mathbf{u}} \in \mathbb{R}^n$ towards it. An overview of the algorithm is given in Algorithm 3 and each step is described in detail in the following sections. We note that the reader may choose any other method for online planning as long as it preserves the avoidance guarantees with the kinematic model of the robots.

7.1 Overview

We build on the work on distributed Reciprocal Velocity Obstacles with motion constraints [2], and its recent extension to aerial vehicles [3].

As described by [2], the method follows two ideas. (a) The radius of the robot is enlarged by a pre-defined and typically fixed value $\varepsilon > 0$ for collision avoidance. This value depends on the kinodynamic model of the robot and can be reduced in real time without having to recompute the stored maximum tracking errors. And, (b) in run time, the local trajectories are limited to those with a tracking error below ε with respect to their reference trajectory. Recall that the tracking errors were precomputed in the offline process.

At each time-step an optimal reference velocity $\mathbf{u}^* \in \mathbb{R}^n$ is obtained by solving a convex optimization in reference velocity space. The associated local trajectory is guaranteed to be collision-free, satisfies the motion constraints and minimizes a cost function. The cost function minimizes the deviation to a preferred velocity $\bar{\mathbf{u}}$, corrected by a small repulsive velocity $\mathring{\mathbf{u}}$ inversely proportional to the distance to the neighboring obstacles when in close proximity. As described by [3] this additional term introduces a desired separation between robots and obstacles. Note that the avoidance guarantees arise from the constrained optimization and not from the repulsive velocity.

7.2 Robot Dynamics

Letting $t \in \mathbb{R}_+$ denote time and t_k the current time instant, we define the relative time $\tilde{t} = t - t_k \in [0, \dots, \infty)$ and the time horizon of the local planner $\tau > 0$, greater than the required time to stop if moving at maximum speed. Note that different robots may present different dynamic models. We denote the state of a robot by $\mathbf{z} = [\mathbf{p}, \dot{\mathbf{p}}, \ddot{\mathbf{p}}, \dots]$, which includes its position and velocity and may include additional terms such as acceleration and orientation. Given a control input $\nu(t)$ the dynamical model is $\dot{\mathbf{z}} = g(\mathbf{z}, \nu)$.

In our local planner, we consider a set of candidate local trajectories, each defined by a straight-line reference $\mathbf{p}_{\text{ref}}(\tilde{t}) = \mathbf{p} + \mathbf{u}\tilde{t}$ of constant velocity $\mathbf{u} \in \mathbb{R}^n$ and starting at the current position $\mathbf{p}$ of the robot. Each motion primitive is then given by an appropriate trajectory tracking controller $\mathbf{p}^{robot}(\tilde{t}) = f(\mathbf{z}, \mathbf{u}, \tilde{t})$ that is continuous in the initial state $\mathbf{z}$ of the robot, respects its dynamical model and converges to the straight-line reference trajectory. Local trajectories are now parametrized by $\mathbf{u}$, see Fig. 6 for an example. Suitable controllers defining the function $f(\mathbf{z}, \mathbf{u}, \tilde{t})$ include LQR control and second order exponential curves, for ground robots [2] and quadrotors [3].

Algorithm 3 Execution of the local planner using the synthesized strategy automaton.

```
 1: Input: Current state of the robot, a local map, position and velocity
    of neighbors and a synthesized strategy automaton (FSM).
 2: At each time instance ($\sim$ 10 Hz) do the following:
 3: if the robot is in deadlock with any other agent then
 4:     Send a deadlock flag to the FSM.
 5: end if
 6: Obtain command from the FSM (e.g. "stay in the current room"
    or "move to the next room"), based on current state and deadlock
    flag.
 7: Convert command into a goal position and preferred velocity $\bar{\mathbf{u}}$.
 8: Compute constraints to satisfy the dynamic model of the robot.
 9: for each neighboring agent do
10:     Compute pairwise collision avoidance constraint.
11: end for
12: Compute largest obstacle-free convex region wrt static obstacles.
13: Solve constrained optimization to obtain collision-free motion
14: Output: A collision-free motion for the robot and the time horizon
```

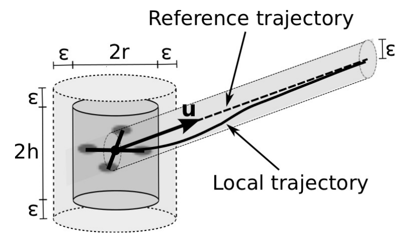


Fig. 6: Schema local and reference trajectories for an aerial vehicle, generated from the reference velocity $\mathbf{u}$. The tracking error is limited by ε and the robot volume dilated by ε.

For fixed robotic platform, controller, initial state $\mathbf{z}$ and reference velocity $\mathbf{u}$, the maximum deviation (initial position independent) between the reference and the simulated trajectory is given by

$$\gamma(\mathbf{z}, \mathbf{u}) = \max_{\tilde{t} > 0} ||(\mathbf{p} + \tilde{t}\mathbf{u}) - f(\mathbf{z}, \mathbf{u}, \tilde{t})||_2. \tag{13}$$

In an offline procedure, we precompute the maximal tracking errors $\gamma(\mathbf{z}, \mathbf{u})$ via forward simulation of the robot dynamics and controller $f(\mathbf{z}, \mathbf{u}_i, \tilde{t})$ for a discretization of reference velocities $\mathbf{u}$ and initial states $\mathbf{z}$ - we only discretize in initial velocity since the error is independent of the initial position of the robot. They are stored for online use in a look-up table.

7.3 Constraints

To define the motion and inter-agent avoidance constraints we build on the approach in [3]. We additionally introduce constraints for avoiding static obstacles. For completeness, we give an overview of each of the constraints.

7.3.1 Robot dynamics

Recalling Eq. (13) the motion constraint is given by the reference velocities for which the tracking error is below ε,

$$R(\mathbf{z}, \varepsilon) = \{\mathbf{u} \,|\, \gamma(\mathbf{z}, \mathbf{u}) \leq \varepsilon\}. \tag{14}$$

approximated by the largest inscribed convex polytope/ellipsoid $\hat{R}(\mathbf{z}, \varepsilon) \subset R(\mathbf{z}, \varepsilon)$.

7.3.2 Avoidance of other agents

Denote by $\mathbf{p}_j$, $\mathbf{v}_j$, $\bar{r}_j$ and $\bar{h}_j$ the position, velocity, dilated radius and height of a neighboring agent j. Assume that it keeps its velocity constant for $\tilde{t} \leq \tau$. Reciprocity (i.e. the other agent follows the same algorithm) can as well be assumed and is discussed in [3]). For every neighboring agent

j, the constraint is given by the reference velocities $\mathbf{u}$ for which the agents' enveloping shape do not intersect within the time horizon. For cylindrically-shaped agents moving in 3D the velocity obstacle of colliding velocities is a truncated cone

$$VO_j^\tau = \{\mathbf{u} \mid \exists \tilde{t} \in [0,\tau] : \|\mathbf{p}^H - \mathbf{p}_j^H + (\mathbf{u}^H - \mathbf{v}_j^H)\tilde{t}\| \le \bar{r} + \bar{r}_j \\ \text{and } |p^V - p_j^V + (u^V - u_j^V)\tilde{t}| \le \bar{h} + \bar{h}_j\},$$

where $\mathbf{p} = [\mathbf{p}^H, p^V]$, with $\mathbf{p}^H \in \mathbb{R}^2$ its projection onto the horizontal plane and $p^V \in \mathbb{R}$ its vertical component. The constraint is linearized to $A_j(\mathbf{p},\varepsilon) = \{\mathbf{u} \mid \mathbf{n}_j^T\mathbf{u} \le b_j\}$, where $\mathbf{n}_j \in \mathbb{R}^3$ and $b_j \in \mathbb{R}$ maximize $\mathbf{n}_j^T\mathbf{v} - b_j$ subject to $A_j(\mathbf{p},\varepsilon) \cap VO_j^\tau = \emptyset$.

7.3.3 Avoidance of static obstacles

We extend a recent fast iterative method to compute the largest convex polytope in free space [13], by directing the growth of the region in the preferred direction of motion and enforcing that both the current position of the robot and a look ahead point in the preferred direction of motion are within the region. The convex polytope is computed in position space ($\mathbb{R}^3$ for aerial vehicles) and then converted to an equivalent region in reference velocity space. See Algorithm 4, where $directedEllipsoid(\mathbf{p},\mathbf{q})$ is the ellipsoid with one axis given by the segment $\mathbf{p} - \mathbf{q}$ and the remaining axis infinitesimally small, and K the number of steps in the linear search, typically between 2 and 4.

7.3.4 Avoiding incorrect region transitions

The local planner prevents incorrect region transitions (for instance, avoiding entering another region if the robot's local goal is within the current one) by introducing "virtual" doors at borders between workspace regions. These virtual doors may be closed or opened depending on the desired transition. A closed door is introduced as an obstacle in $\mathcal{O}$.

Algorithm 4 Largest collision-free directed convex polytope.

```
1:  $L \leftarrow \mathbf{p} + \bar{\mathbf{u}}\{\tau, \frac{K-1}{K}\tau, \frac{K-2}{K}\tau, \ldots, 0\}$;   $P := \emptyset$;
2:  $\mathbf{q} \leftarrow L[0]$;   $L := L \setminus \mathbf{q}$;
3:  while $L \neq \emptyset$ and $\mathbf{p}, \mathbf{q} \notin P$ do
4:      $E \leftarrow directedEllipsoid(\mathbf{p},\mathbf{q})$
5:      // Largest polytope seeded in $E$ computed as in [13]
6:      while not converged do
7:          $P \leftarrow$ separating planes of $E$ and dilated $\mathcal{O}$ (QP)
8:              such that $P \subset \mathbb{R}^n \setminus (\mathcal{O} + V_\varepsilon)$
9:          If $\mathbf{p}, \mathbf{q} \notin P$ then { $\mathbf{q} \leftarrow L[0]$; $L := L \setminus \mathbf{q}$; break; }
10:         $E \leftarrow$ ellipsoid $E \subset P$ of maximal volume (SDP)
11:     end while
12: end while
13: $F(\mathbf{p},\varepsilon) := (P - \mathbf{p})/\tau$ // Converts to ref. velocity, $\mathbf{u}$, space
```

7.4 Optimization

The optimization cost is given by two parts. As described in Sec. 7.2, the first one is a regularizing term, weighted by a design constant $\bar{\alpha}$, and the second one is a minimizer with respect to a preferred velocity.

A convex optimization with quadratic cost and linear and quadratic constraints is solved

$$\begin{aligned} \mathbf{u}^* := \underset{\mathbf{u}\in\mathbb{R}^n}{\arg\min}\ & (\alpha\|\mathbf{u}-\mathbf{v}\|^2 + \|\mathbf{u} - (\bar{\mathbf{u}} + \mathring{\mathbf{u}})\|^2), \\ \text{s.t. } & \mathbf{u} \in \hat{R}(\mathbf{z},\varepsilon) \cap F(\mathbf{p},\varepsilon) \\ & \mathbf{u} \in A_j(\mathbf{p},\varepsilon) \quad \forall j \text{ neighbor agent} \end{aligned} \tag{15}$$

The solution of this optimization is a collision-free reference velocity $\mathbf{u}^*$ which minimizes the deviation towards the goal specified by the strategy automaton. The associated trajectory (see Sec. 7.2) is followed by the robot and is collision-free.

7.5 Deadlock Detection

To allow the strategy automaton to resolve deadlock at runtime, we set the deadlock proposition x^{ij} ($i = 1, \ldots, n_{robots}$, $j = 0, \ldots, n_{robots}$; $j = 0$ implying a dynamic obstacle), according to the following rule:

$$x^{ij} \Leftarrow (\|\mathbf{u}_i^*\| < k_1) \wedge (\|\bar{\mathbf{u}}_i\| > k_2) \wedge (\|\mathbf{p}_i - \mathbf{p}_j\| < k_3), \tag{16}$$

with $k_1, k_2, k_3 > 0$ being tunable parameters. This states that a necessary condition for x^{ij} to be set is when the agent velocity magnitude $\|\mathbf{u}_i^*\|$ is low, the preferred velocity magnitude $\|\bar{\mathbf{u}}_i\|$ is high, and the unsigned distance between agents $\|\mathbf{p}_i - \mathbf{p}_j\|$ is within a prescribed tolerance. In our experiments these values are chosen experimentally to detect all deadlocks while minimizing false positives. We introduce a small hysteresis in the flag activation. In particular, we activate the deadlock flag when the right-hand condition of (16) has been `True` for a minimum period of time $T_{dk-true}$. When the flag becames active, x^{ij} is kept in `True` for a minimum period of time $T_{dk-false}$. In our experiments we employ $8s$ and $5s$ respectively. This hysteresis prevents false alarms and chattering when the velocity is small (e.g. while the robot is accelerating).

8 Theoretical Guarantees

We provide proofs for the guarantees inherent to our synthesized controller. The following three subsections are sufficient to show that, under the collision-free guarantees provided by the local planner, the synthesized strategy realizes the reactive task specification and resolves deadlocks.

8.1 Correctness With Respect to Robot Dynamics

By construction of the local planner, the controller is guaranteed correct with respect to the low-level controller $f(\mathbf{z}, \mathbf{u}, \tilde{t})$, which is continuous on the initial state of the robot and respects its dynamics. We do assume that the model of the robot is accurate and that there are no external disturbances.

8.2 Collision-Free Motion

Theorem 1 *The local planner of Sec. 7 yields collision-free motion in dynamic environments, under the constant velocity assumption.*

If (15) is *feasible*, collision-free motion is guaranteed for the local trajectory up to time τ with the assumption that all interacting agents maintain a constant velocity.

Proof. Avoidance of dynamic obstacles was shown in our previous work [3]. Here we reproduce it for the case of a dynamic obstacle maintaining a constant velocity, and it extends to the case where all agents do reciprocal collision avoidance.

Recall that t_k represents the current time instant and $\tilde{t} = t - t_k \in [0, \infty)$ the relative time. Let $\mathbf{p}(t)$ denote the position at time $t \geq t^k$, and if not specified, variables are evaluated at t^k. The idea is that the optimal reference trajectory is collision-free for an agent whose volume is enlarged by ε and the robot stays within ε of it. Formally,

$$
\begin{aligned}
(\mathbf{p} + \mathbf{u}\tilde{t}) - (\mathbf{p}_j + \mathbf{v}_j\tilde{t}) \underset{\text{Avoidance constraint, } \mathbf{u}\in A_j(\mathbf{p},\varepsilon)}{\notin} V_\varepsilon + V_j \Rightarrow \\
\underset{\mathbf{u}\in\hat{R}(\mathbf{z},\varepsilon)}{\Rightarrow} \mathbf{p}(t) - \mathbf{p}_j(t) = f(\mathbf{z}, \mathbf{u}, \tilde{t}) - (\mathbf{p}_j + \mathbf{v}_j\tilde{t}) \notin V + V_j
\end{aligned}
$$

For the case of planar disk robots, this is equivalent to showing the relative distance is greater than the sum of radii,

$$
\begin{aligned}
||\mathbf{p}(t) - \mathbf{p}_j(t)|| = ||f(\mathbf{z}, \mathbf{u}, \tilde{t}) - (\mathbf{p}_j + \mathbf{v}_j\tilde{t})|| &\underset{\mathbf{u}\in\hat{R}(\mathbf{z},\varepsilon)}{\geq} \\
||(\mathbf{p} + \mathbf{u}\tilde{t}) - (\mathbf{p}_j + \mathbf{v}_j\tilde{t})|| - \varepsilon &\underset{\mathbf{u}\in A_j(\mathbf{p},\varepsilon)}{\geq} \\
r + \varepsilon + r_j - \varepsilon = r + r_j&,
\end{aligned}
$$

For avoidance of static obstacles, $\mathbf{u} \in F(\mathbf{p}, \varepsilon)$ implies

$$
\begin{aligned}
\mathbf{u} \in F(\mathbf{p}, \varepsilon) \underset{\text{Alg. 4, } P \text{ convex}}{\Rightarrow} & (\mathbf{p} + \mathbf{u}\tilde{t}) \notin \mathcal{O} + V_\varepsilon \quad \forall \tilde{t} \in [0, \tau] \\
\underset{\mathbf{u}\in\hat{R}(\mathbf{z},\varepsilon)}{\Rightarrow} & f(\mathbf{z}, \mathbf{u}, \tilde{t}) \notin \mathcal{O} + V \quad \forall \tilde{t} \in [0, \tau].
\end{aligned}
$$

□

If the assumptions are violated, e.g. the moving obstacles quickly change their velocity, the constrained optimization of Eq. (15) can be *infeasible*. In that case, no collision-free solution exists that respects all of the constraints and a collision may arise. In this case the robot decelerates at its maximum deceleration rate until full stop or a feasible collision-free trajectory is found. In practice, since this computation is performed at a high frequency, each individual robot is able to adapt to changing situations, and the resulting motion is collision-free if the moving obstacles behave fairly (i.e. never cause collisions).

8.3 Correctness with Respect to the Task Specification

Since the local planner is myopic, it provides guarantees up to a time horizon τ and consequently may result in deadlock and livelock. However, as we have shown, the planner's local guarantees allow a discrete abstraction that the strategy automaton can use to resolve deadlocks and avoid livelocks. Here we formally prove the guarantees on the execution provided by our synergistic online and offline synthesis.

Proposition 1 *Given a task specification φ that ignores collisions, if the resulting specification φ^{abstr} defined in Sec. 5 is realizable, then the corresponding strategy automaton also realizes φ.*

Proof. Assume given $\varphi = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e \implies \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$. Recall that $\varphi^{abstr} = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e \implies [\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$, where $[\varphi_i^s]'$ and $[\varphi_t^s]'$ contain φ_i^s and φ_t^s as subformulas, respectively. Suppose that strategy automaton $\mathcal{A}_{\varphi^{abstr}}$ realizes φ^{abstr}. This means that the resulting controller is guaranteed to fulfill the requirement $[\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$ as long as the environment fulfills the assumption $\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e$. This implies that $A_{\varphi^{abstr}}$ fulfills $\varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$ as long as the environment fulfills the assumption $\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e$. □

Proposition 2 *Given a task specification φ that ignores collisions, if φ is realizable but the resulting specification φ^{abstr} is not realizable, then the revision procedure in Sec. 6.1 will find an assumption φ_{rev}^e to add to φ^{abstr} that renders the resulting specification φ^{rev} realizable and the resulting strategy $\mathcal{A}_{\varphi^{rev}}$ free of deadlock and livelock.*

Proof. Suppose φ is realizable by strategy $\mathcal{A}_\varphi$, but φ^{abstr} is not realizable, admitting counterstrategy $\mathcal{C}_{\varphi^{abstr}} = (\mathcal{Q}, \ldots)$. It suffices to show that the set S_{cuts} is nonempty. Assume by way of contradiction that S_{cuts} is empty. Then the rising edge of deadlock θ_s^i never occurs for any i, so no robot transitions are ever disabled. Since we assume that deadlock does not occur in the initial state, this means that x^{ij} is always `False` for every i,j. Therefore $[\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$ defined in Sec. 5 reduces to $\varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$. The lack of deadlock means that any region transition contained in $\mathcal{A}_\varphi$ is still admissible, and therefore $\mathcal{A}_\varphi$ can be used as a strategy to realize φ^{abstr}, a contradiction. Therefore, there must be deadlock and S_{cuts} is not empty. Now, upon addition of the assumptions φ^{abstr}, existence of $\mathcal{A}_{\varphi^{rev}}$ that satisfies φ^{abstr} implies, by construction, that $\mathcal{A}_{\varphi^{rev}}$ is livelock-free. □

Note that it may be the case that S_{cut} is nonempty, but for every $(p,q) \in S_{cuts}$, the resulting revision

$$(\psi_{\mathcal{Y}}(p) \wedge \psi_{\mathcal{X}}(p) \implies \neg \bigcirc \psi_{\mathcal{X}}(q))$$

contradicts φ_e^t. This indicates that φ is only realizable because it makes unreasonable assumptions on the environment. Our approach identifies this fact as a by-product of the revision process.

8.4 Computational Complexity

For a given choice of m, the offline reactive synthesis algorithm used in this work is exponential in the number of propositions [8, 16]. Using our encoding, the problem scales linearly with n_{robots} – no worse than existing approaches (e.g. [41]). When one or more dynamic obstacles are considered, the number of propositions does not change. As stated in Sec. 6, $2^{(|\mathcal{Y}|+|\mathcal{X}|)}$ iterations of the main loop in Algorithm 1 are needed in the worst case, yielding a theoretical complexity that is doubly exponential in the number of propositions.

For the online component, a convex program is solved independently for each robot, with the number of constraints linear in the number of neighboring robots. The runtime of the iterative computation of the convex volume in free space barely changes with the number of obstacles, up to tens of thousands [13], and a timeout can be set, with the algorithm returning the best solution found.

9 Experiments and Simulations

We present results of our end-to-end approach both in simulation and on hardware. Our evaluation is meant to illustrate the various parts of the synthesis and execution process, and provide a statistically-grounded evaluation of the approach when placed in a difficult environment that does not necessarily behave according to the automatically-generated environment assumptions. In this context, our results reveal that our approach has merit in dealing with such environments to execute the task successfully. We furthermore show that our approach is scalable to any number of dynamic obstacles, and that the local planner applies to 3-D workspaces. Lastly, we show that our approach may be executed in real time on actual hardware.

The synthesis procedure described in Sec. 5 was implemented with the `slugs` synthesis tool [16], and executed with the LTLMoP toolkit [19]. The local motion planner, Sec. 7, was implemented with the IRIS toolbox [13] and an off-the-shelf convex optimizer. We assume the dynamic obstacles are cooperative in avoiding collisions, therefore, each one is controlled by a local planner. Many of the experiments presented in this section are available in the accompanying video.

In what follows, we consider a "garbage collection" scenario, upon which we synthesize a strategy automaton.

Example 2 ***(Garbage Collection)*** A robot team is required to patrol the *Living Room* (R_{LR}) and *Bedroom* (R_{BR}) of the workspace in Fig. 7. For two robots, the specification is:

$$\square\diamondsuit(\pi^1_{LR}) \wedge \square\diamondsuit(\pi^1_{BR}) \wedge \square\diamondsuit(\pi^2_{LR}) \wedge \square\diamondsuit(\pi^2_{BR})$$

and if *garbage* is observed, pick it up

$$\square(\pi^1_{garb} \implies \pi^1_{act,pickup}) \wedge \square(\pi^2_{garb} \implies \pi^2_{act,pickup}).$$

Additionally, the robots must always avoid other moving agents.

The system propositions are actions to move between regions ($\pi^i_{act,LR}, \ldots, \pi^i_{act,BR}$) and to pick up ($\pi^i_{act,pickup}$). The environment propositions are sensed garbage (π^i_{garb}), region completions ($\pi^i_{LR}, \ldots, \pi^i_{BR}$), and pick up completion (π^i_{pickup}).

We omit the complete encoding of Def. 2, however, for illustration we supply the transition formulas for the case where robot 1 is in Hall:

$$\varphi_t^s : \begin{cases} \square(\pi^1_{Hall} \vee \pi^1_{LR} \vee \pi^1_{BR} \vee \pi^1_{Kitchen} \vee \pi^1_{Door}) \\ \square(\bigcirc\pi^1_{Hall} \Longrightarrow \bigcirc\pi^1_{act,Hall} \vee \bigcirc\pi^1_{act,BR} \vee \bigcirc\pi^1_{act,LR}) \end{cases}$$

$$\varphi_t^e : \begin{cases} \square(\pi^1_{act,Hall} \vee \pi^1_{act,LR} \vee \pi^1_{act,BR} \vee \pi^1_{act,Kitchen} \vee \\ \quad \pi^1_{act,Door}) \\ \square(\pi^1_{Hall} \wedge \pi^1_{act,Hall} \Longrightarrow \bigcirc\pi^1_{Hall}) \\ \square(\pi^1_{Hall} \wedge \pi^1_{act,LR} \Longrightarrow \bigcirc\pi^1_{Hall} \vee \bigcirc\pi^1_{LR}) \\ \square(\pi^1_{Hall} \wedge \pi^1_{act,BR} \Longrightarrow \bigcirc\pi^1_{Hall} \vee \bigcirc\pi^1_{BR}) \end{cases}$$

The initial conditions φ_i^s and φ_i^e are `True`.

We implement the above example using humanoid robots (able to rotate in place, move forward and along a curve) and simulated quadrotor UAVs.

9.1 Synthesis and Revisions

Upon synthesizing a controller for single robot, we obtain 16 revisions to the environment assumptions. These are displayed to the user as runtime certificates. One example is: `Deadlock should not occur when the robot is in the Hall moving toward the Living Room and had already been blocked from entering the Bedroom.` Note that, with each robot added to the team, the number of revisions grows combinatorially. In contrast to the single-robot case, there are a total of 1306 statements given to the user in the case of two robots. In these cases, we display the

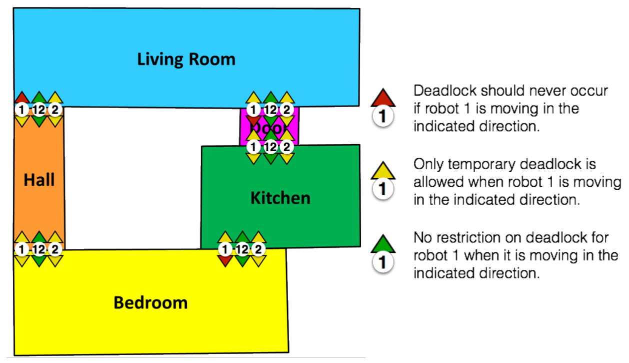

Living Room
Hall
Kitchen
Bedroom
Deadlock should never occur if robot 1 is moving in the indicated direction.
Only temporary deadlock is allowed when robot 1 is moving in the indicated direction.
No restriction on deadlock for robot 1 when it is moving in the indicated direction.

Fig. 7: Workspace showing specification revisions for each region completion/activation pairs where singleton or pairwise deadlock may occur. An arrow's color indicates the type of assumption that has been made. The number (or pair of numbers) indicates the robot (or robot pairs) concerned with the assumption. The placement of the arrow indicates the region that the robot is headed (i.e. its action commands $AP_{\mathcal{R}}^{act}$) when the given assumption holds true.

revisions graphically, by projecting over the variables of interest: the current region and action for each robot. Rather than displaying all 1306 statements, we show the projection consisting of 45 statements projected onto the set of each robot's motion and activation propositions. Satisfying these 45 statements implies that we also satisfy the 1306 statements. To further aid the user, we display them graphically on the workspace as shown in Fig. 7.

For instance, a red arrow on the boundary of the Hall indicates that the automaton cannot guarantee the task if the robot experiences deadlock when it is in the Hall and while activating a motion to the Living Room. The certificates displayed in Fig. 7 are projections onto a subset of the complete set of propositions (i.e. deadlocks, memory propositions, robot positions, and robot actions for *each* of the robots in the team), by abstracting those variables away. That is, if there exist restrictions on deadlock for *any* of the propositions that have been abstracted away, then the revision displayed will be a conservative overapproximation to the true revision and the dot will be labeled red.

9.2 Scalability with Respect to Dynamic Obstacles

Considering Example 2, the specification for the single-robot case consists of 14 propositions, while that for the two-robot case consists of 29 propositions. The specification is invariant to the number of dynamic obstacles in either case.

One could also consider a two-robot team controlled by a baseline strategy that relies on mutual exclusion (one robot per region) to be kept with other robots and dynamic obstacles (DO). That strategy required 20 propositions for the case without DOs. One additional proposition is added for each region for each DO (producing 25 for one DO, 35 for three DO, 60 propositions for eight DO, etc.). Because the obstacles are assumed to behave in an adversarial manner, they can violate mutual exclusion if they enter a neighboring region of the robot. Hence, the baseline synthesis procedure is *not realizable* for one or more dynamic obstacles.

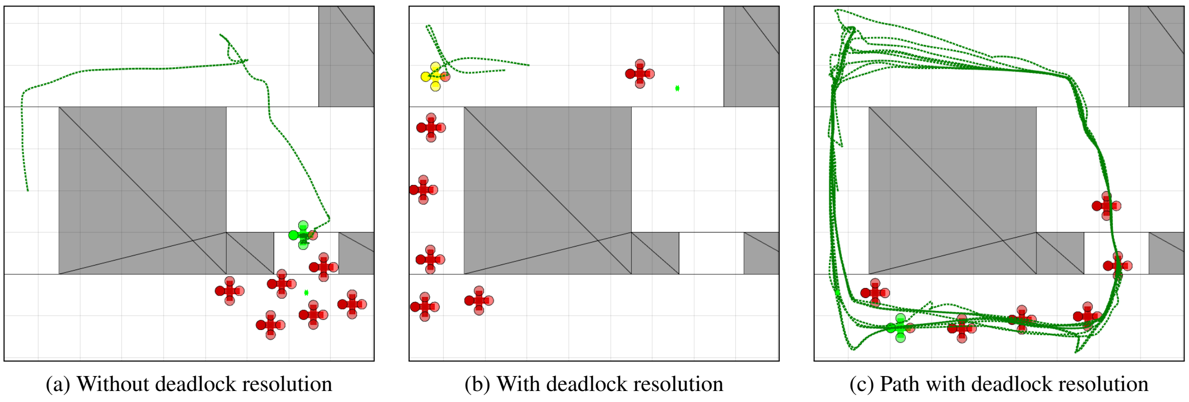
In contrast, our approach is realizable independently of the number of dynamic obstacles and requires fewer propositions than the case with two or more DO.

9.3 Performance Evaluation

We directly compare the proposed approach with a baseline approach where the robots execute a local planner, but there is no deadlock resolution in the strategy. Recall that there is no guarantee of mission satisfaction in that case. Figs. 8, 9 and 10 display results for various problem scenarios. In each experiment, we use the model described in [3] to model the robots and dynamic obstacles as quadrotors. The "counter-flow" cases follow a pre-defined set of waypoints that allow DOs to circulate within the workspace in one direction (counter to the flow of the robots), while, in the "random waypoints" cases, DOs randomly select a neighboring waypoint once a waypoint has been achieved. To detect deadlock, we use the criteria in (16) with the choice of parameters $k_1 = \frac{1}{3}$, $k_2 = \frac{1}{4}$, and $k_3 = 1.5$.

Each test case consisted of 133 minutes of data obtained over multiple simulation runs lasting 200 seconds each. The simulation was terminated before 200 seconds if none of the controlled robots reached their goal, but none had been moving (their velocity falls below a threshold) for 100 seconds or longer. Any such runs were flagged as *unresolved deadlock*, at which point the robots are deemed unable to continue their task. The robots in the team were initialized randomly at different regions in the workspace.

In the "counter-flow" example of Fig. 10, 100% of the simulation runs without the proposed deadlock resolution approach eventually enter unresolved deadlock at some point during the run. In contrast, when the proposed approach is used, deadlock is able to be resolved, resulting in more goals being visited. In the single-robot case, only 5% of the runs lead to unresolved deadlock. In all such runs, the DOs had violated a runtime certificate (note that the DOs were not programmed to satisfy any such certificates); in some cases the DOs surrounded the robot. In the two-robot case, nearly 20% of the runs lead to unresolved deadlock. This number is higher than in the one-robot case because there is more than one robot whose motion could be blocked by the DOs, leading to more encountered deadlocks. Additionally, when one robot has already become deadlocked, the objects in the environment effectively act as static obstacles to the remaining robot, increasing the chance it will become deadlocked as



(a) Without deadlock resolution (b) With deadlock resolution (c) Path with deadlock resolution

Fig. 8: Example of the approach in a scenario with six dynamic obstacles (dark red) and one controlled quadrotor (light green/yellow). The path of the controlled quadrotor is shown with a dashed green line. (a) The original approach without deadlock resolution avoids collisions but can get into unresolved deadlocks. The path leading to the deadlock is shown. (b) The proposed approach successfully resolves deadlocks, like the one shown here. The path leading to and resolving the deadlock are shown. (c) Path of the controlled quadrotor using the proposed approach during a ten minutes simulation. The quadrotor successfully avoids collisions and reverts the motion when it encounters a deadlock.

compared with moving, dynamic obstacles. The combined effect of these two factors is the reason why there is a fourfold increase in the number of encountered deadlocks.

The "random waypoints" cases are included to evaluate the performance of the proposed approach where the DOs do not all move in the same direction, but instead move randomly in the workspace. In the case of a single robot, deadlock resolution allows the robots to find alternate routes around deadlocks, and thus the robot is able to visit 40% more goals than the case without deadlock resolution. In the case of two robots, the team is able to achieve 136% more goals than without resolution. As may be observed in the supplementary videos, deadlock resolution gives the robots an ability to exploit areas of the workspace containing a lower density of dynamic obstacles to achieve their goals. The cases where deadlock resolution is included results in greater likelihood of task achievement over a 200-second interval. As compared with the counter-flow cases, there are fewer cases of unresolved deadlock because the random nature of the DOs allows the robots to move more freely in some cases than in others.

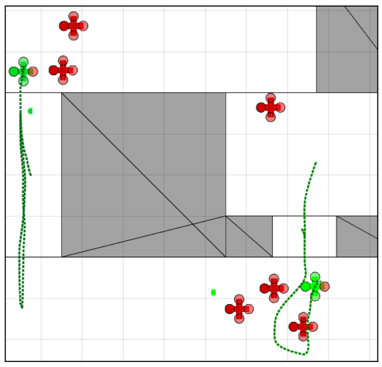
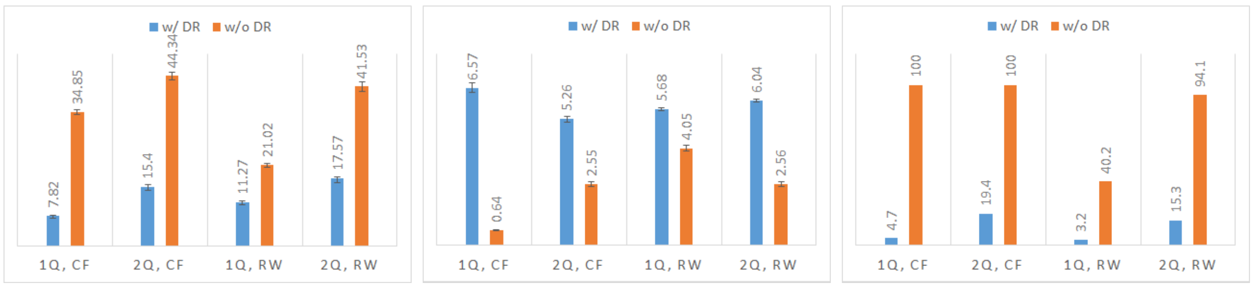


Fig. 9: Example of the approach with six dynamic obstacles (dark red) and two controlled quadrotors (light green). The dynamic obstacles navigate to randomized locations and the controlled robots execute the proposed framework. The path of the controlled quadrotors is shown with a dashed green line for one minute of the simulation. The quadrotors successfully avoid collisions, reverse motion when they encounter a deadlock and explore the top and bottom rooms.

9.4 3D Problem Domain

We next demonstrate the effectiveness of the approach in a 3D scenario where, in the $5 \times 5 \times 5$ m^3 two-floor workspace of Fig. 11, robots move between floors through a vertical opening at the left corner or the stairs at the right side. The two robots on the team as well as the dynamic obstacle are simulated quadrotors. The task is to infinitely often visit the top and bottom floors while avoiding collisions and resolving deadlock. The strategy automaton is synthesized as described in Sec. 5. A local planner for the 3D environment is constructed following Sec. 7. A representative experiment is shown in the snapshots in Fig. 11. The green robot enters deadlock when moving towards the upwards corridor; however, deadlock is resolved by taking the alternative route up the stairs.



(a) # encountered deadlocks per 200-sec run (b) # goals visited per 200-sec run (c) # unresolvable deadlocks per 200-sec run

Fig. 10: Comparison of the results of the “garbage collection” scenario with DOs exhibiting counter-flow (CF) or random waypoints (RW) behaviors, with either one quadrotor (1Q) or two quadrotors (2Q). For each scenario, we evaluate the results for data collected over 40 200-sec runs. Six quadrotors were used for the DOs. Over each run, (a), (b), and (c) show, respectively, data for the number of encountered deadlocks, the number of goals visited, and the number of unresolvable deadlocks. Standard deviations are indicated as error bars in (a) and (b).

9.5 Physical Experiments

To demonstrate effectiveness in a physical setting, we employ two Aldebaran Nao robots to carry out the planar garbage collection scenario, with a teleoperated KUKA youBot serving as the dynamic obstacle. The model for the Nao robots is one where the robots are are able to rotate in place, move forward, and move along a curve at a constant velocity. The size of the field is 5m by 3m, and the sensing range for the local planner is 1m. The size is such that only one Nao robot may fit through the Hall and Door at a time. The positions of each robot are measured through a motion capture system. The local planner is implemented on a laptop computer communicating via a WiFi connection to the robots. In the local planner, the Nao robots are taken to have a circular footprint with effective radius of 0.2 m.

We carried out experiments using two robots on the team, using the workspace shown in Fig. 7. The revisions for these two robots are pictured in the figure for the synthesized mission plan. As demonstrated in the snapshots in Fig. 12, the Naos can execute the task, by avoiding collisions and resolving deadlocks with one another and with the dynamic obstacle (the KUKA youBot). At the particular deadlock event shown in Fig 12b, the youBot must eventually move away from the Door region, as the assumption pictured in Fig. 7 states that ‘only temporary deadlock is allowed’ when either of the robots are trying to enter it from the Kitchen. The experiments demonstrate that the approach is effective at deadlock resolution and at achieving collision free motion, thereby satisfying the mission specification.

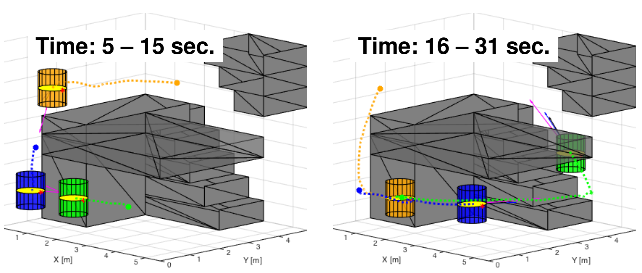
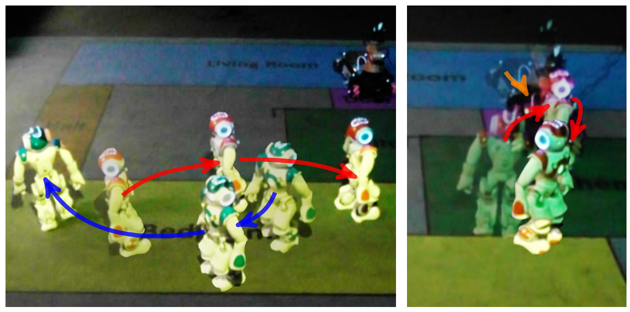


Fig. 11: Deadlock resolution (green robot) and safe navigation in a 3D environment. Quadrotors are displayed at the final time and their paths for the time interval. Each yellow disk represents a quadrotor and the cylinder its safety volume. The orange robot represents the dynamic obstacle.

10 Conclusion

We present a framework for synthesizing a strategy automaton and collision-free local planner that guarantees comple-



(a) Avoidance maneuver (b) Deadlock resolution

Fig. 12: Planar scenario with two centrally-controlled Nao robots and a dynamic obstacle (youBot). In each image, three consecutive frames of the robot’s motion are superimposed. In (a), the local planner enables the two Naos to avoid collisions with each other. In (b), one of the Naos reverses direction to resolve the deadlock with the youBot.

tion of a task specified in linear temporal logic, where we consider reactive mission specifications abstracted with respect to basic locomotion, sensing and actuation capabilities. Our approach is less conservative than current approaches that impose a separation between agents, and is computationally cheaper than explicitly modeling all possible obstacles in the environment. If no controller is found that satisfies the specification, the approach automatically generates the needed assumptions on deadlock to render the specification realizable and communicates these to the user. The approach generates controllers that accommodate deadlock between robots or with dynamic obstacles *independently of* the precise number of obstacles present, and we have shown that the generated controllers are correct with respect to the original specification. Experiments with ground and aerial robots demonstrate collision avoidance with other agents and obstacles, satisfaction of a task, deadlock resolution and livelock-free motion. Future work includes optimizing the set of revisions and decentralizing the synthesis.

References

1. Alonso-Mora J, Breitenmoser A, Rufli M, Beardsley P, Siegwart R (2010) Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots. Proc Int Symp on Distributed Autonomous Robotics Systems
2. Alonso-Mora J, Gohl P, Watson S, Siegwart R, Beardsley P (2014) Shared control of autonomous vehicles based on velocity space optimization. In: IEEE Int. Conf. on Robotics and Automation (ICRA)
3. Alonso-Mora J, Naegeli T, Siegwart R, Beardsley P (2015) Collision avoidance for aerial vehicles in multi-agent scenarios. Autonomous Robots
4. Alur R, Moarref S, Topcu U (2013) Counter-strategy guided refinement of GR(1) temporal logic specifications. In: Formal Methods in Computer-Aided Design (FMCAD), pp 26–33
5. Bento J, Derbinsky N, Alonso-Mora J, Yedidia JS (2013) A message-passing algorithm for multi-agent trajectory planning. Annual Conference on Neural Information Processing Systems NIPS
6. van den Berg J, Guy SJ, Lin M, Manocha D (2009) Reciprocal n-body Collision Avoidance. In: Int. Symp. on Robotics Research (ISRR)
7. Bhatia A, Kavraki L, Vardi M (2010) Sampling-based motion planning with temporal goals. In: IEEE Int. Conf. on Robotics and Automation (ICRA)
8. Bloem R, Jobstmann B, Piterman N, Pnueli A, Sa'ar Y (2012) Synthesis of reactive (1) designs. Journal of Computer and System Sciences 78(3):911–938
9. Chen Y, Ding XC, Stefanescu A, Belta C (2012) Formal approach to the deployment of distributed robotic teams. IEEE Transactions on Robotics 28(1):158–171
10. Cirillo M, Uras T, Koenig S (2014) A lattice-based approach to multi-robot motion planning for non-holonomic vehicles. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, USA
11. DeCastro J, Ehlers R, Rungger M, Balkan A, Kress-Gazit H (2016) Automated generation of dynamics-based runtime certificates for high-level control. Discrete Event Dynamic Systems pp 1–35, DOI 10.1007/s10626-016-0232-7, URL `http://dx.doi.org/10.1007/s10626-016-0232-7`
12. DeCastro JA, Alonso-Mora J, Raman V, Rus D, Kress-Gazit H (2015) Collision-free reactive mission and motion planning for multi-robot systems. In: Proceedings of the International Symposium on Robotics Research (ISRR)
13. Deits R, Tedrake R (2014) Computing large convex regions of obstacle-free space through semidefinite programming. Workshop on the Algorithmic Fundamentals of Robotics
14. Dimarogonas DV, Frazzoli E, Johansson K (2012) Distributed event-triggered control for multi-agent systems. IEEE Trans Automatic Control 57(5):1291–1297
15. Ehlers R (2013) Symmetric and efficient synthesis. PhD thesis, Saarland University
16. Ehlers R, Raman V (2016) Slugs: Extensible GR(1) synthesis. In: Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II, pp 333–339, DOI 10.1007/978-3-319-41540-6_18, URL `http://dx.doi.org/10.1007/978-3-319-41540-6_18`
17. Ehlers R, Topcu U (2014) Resilience to intermittent assumption violations in reactive synthesis. In: Proc. of the Int. Conf. on Hybrid Systems: Computation and Control
18. Ehlers R, Könighofer R, Bloem R (2015) Synthesizing cooperative reactive mission plans. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on
19. Finucane C, Jing G, Kress-Gazit H (2010) Ltlmop: Experimenting with language, temporal logic and robot control. In: Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems
20. Hsu D, Latombe JC, Kurniawati H (2007) On the probabilistic foundations of probabilistic roadmap planning. The International Journal of Robotics Research 25(7):627–643
21. Jing G, Ehlers R, Kress-Gazit H (2013) Shortcut through an evil door: Optimality of correct-by-construction controllers in adversarial environments. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 4796–4802, DOI 10.1109/

IROS.2013.6697048

22. Karaman S, Frazzoli E (2009) Sampling-based motion planning with deterministic μ-calculus specifications. In: IEEE Conf. on Decision and Control (CDC)
23. Knepper RA, Rus D (2012) Pedestrian-inspired sampling-based multi-robot collision avoidance. In: RO-MAN, IEEE, pp 94–100
24. Kress-Gazit H, Conner DC, Choset H, Rizzi AA, Pappas GJ (2008) Courteous cars. IEEE Robot Automat Mag 15(1):30–38
25. Kress-Gazit H, Fainekos GE, Pappas GJ (2008) Translating structured english to robot controllers. Advanced Robotics 22(12):1343–1359
26. Kress-Gazit H, Fainekos GE, Pappas GJ (2009) Temporal logic based reactive mission and motion planning. IEEE Transactions on Robotics 25(6):1370–1381
27. LaValle SM, Kuffner JJ (2001) Randomized kinodynamic planning. The International Journal of Robotics Research 20(5):378–400
28. Li W, Dworkin L, Seshia SA (2011) Mining assumptions for synthesis. In: IEEE/ACM Int. Conf. on Formal Methods and Models for Codesign
29. Liu J, Ozay N, Topcu U, Murray RM (2013) Synthesis of reactive switching protocols from temporal logic specifications. IEEE Trans Automat Contr 58(7):1771–1785
30. Livingston SC, Prabhakar P, Jose AB, Murray RM (2013) Patching task-level robot controllers based on a local μ-calculus formula. In: Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), Karlsruhe, Germany
31. Loizou SG, Kyriakopoulos KJ (2004) Automatic synthesis of multiagent motion tasks based on ltl specifications. In: Proc. of IEEE Conf. on Decision and Control (CDC)
32. Maly M, Lahijanian M, Kavraki LE, Kress-Gazit H, Vardi MY (2013) Iterative temporal motion planning for hybrid systems in partially unknown environments. In: ACM International Conference on Hybrid Systems: Computation and Control (HSCC), ACM, ACM, Philadelphia, PA, USA, pp 353–362
33. Mellinger D, Kushleyev A, Kumar V (2012) Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. Robotics and Automation, IEEE Int Conf on
34. Pivtoraiko M, Knepper RA, Kelly A (2009) Differentially constrained mobile robot motion planning in state lattices. Journal of Field Robotics 26(3):308–333
35. Raman V (2014) Reactive switching protocols for multi-robot high-level tasks. In: IEEE/RSJ International Conference on Intelligent Robots and Systems
36. Raman V, Kress-Gazit H (2014) Synthesis for multi-robot controllers with interleaved motion. In: IEEE Int. Conf. on Robotics and Automation
37. Raman V, Piterman N, Kress-Gazit H (2013) Provably correct continuous control for high-level robot behaviors with actions of arbitrary execution durations. In: IEEE Int. Conf. on Robotics and Automation
38. Saha I, Ramaithitima R, Kumar V, Pappas GJ, Seshia SA (2016) Implan: Scalable incremental motion planning for multi-robot systems. In: Proceedings of the 7th International Conference on Cyber-Physical Systems (ICCPS)
39. Schillinger P, Bürger M, Dimarogonas DV (2016) Decomposition of finite LTL specifications for efficient multi-agent planning. In: 13th International Symposium on Distributed Autonomous Robotic Systems, Springer Tracts in Advanced Robotics
40. Tumova J, Dimarogonas DV (2015) Decomposition of multi-agent planning under distributed motion and task LTL specifications. In: 54th IEEE Conference on Decision and Control, CDC 2015, Osaka, Japan, December 15-18, 2015, IEEE, pp 7448–7453, DOI 10.1109/CDC.2015.7403396, URL `http://dx.doi.org/10.1109/CDC.2015.7403396`
41. Ulusoy A, Smith SL, Ding XC, Belta C, Rus D (2013) Optimality and robustness in multi-robot path planning with temporal logic constraints. I J Robotic Res 32(8):889–911
42. Vardi MY (1996) An automata-theoretic approach to linear temporal logic. In: Logics for concurrency, Springer, pp 238–266
43. Wong KW, Ehlers R, Kress-Gazit H (2014) Correct high-level robot behavior in environments with unexpected events. In: Proc. of Robotics: Science and Systems
44. Wongpiromsarn T, Topcu U, Murray R (2012) Receding horizon temporal logic planning. Automatic Control, IEEE Transactions on 57(11):2817–2830
45. Wongpiromsarn T, Ulusoy A, Belta C, Frazzoli E, Rus D (2013) Incremental synthesis of control policies for heterogeneous multi-agent systems with linear temporal logic specifications. In: Robotics and Automation (ICRA), IEEE Int. Conf. on