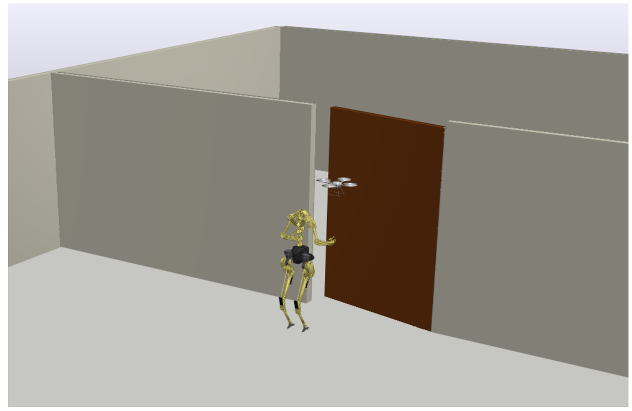
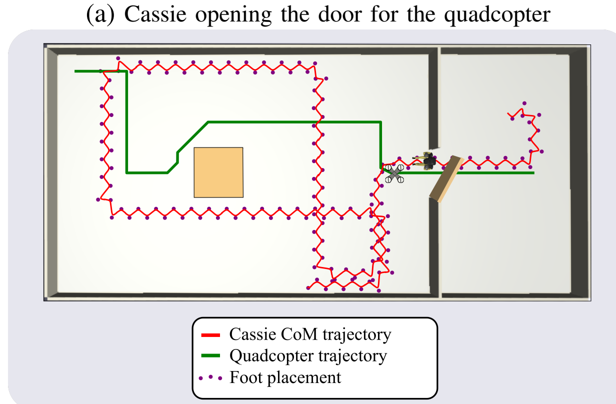
Leveraging Heterogeneous Capabilities in Multi-Agent Systems for Environmental Conflict Resolution

Michael E. Cao*, Jonas Warnke*, Ye Zhao, and Samuel Coogan

***Abstract*— In this paper, we introduce a high-level controller synthesis framework that enables teams of heterogeneous agents to assist each other in resolving environmental conflicts that appear at runtime. This conflict resolution method is built upon temporal-logic-based reactive synthesis to guarantee safety and task completion under specific environment assumptions. In heterogeneous multi-agent systems, every agent is expected to complete its own tasks in service of a global team objective. However, at runtime, an agent may encounter un-modeled obstacles that prevent it from achieving its own task. To address this problem, we take advantage of the capability of other heterogeneous agents to resolve the obstacle. A controller framework is proposed to detect such a situation and redirect agents with the capability of resolving the appropriate obstacles to the required target, either by resynthesis of the controllers with new objectives, or by initially including a runtime-assignable goal in the controller, resulting in a non-resynthesis solution. A set of case studies involving the bipedal robot Cassie and a quadcopter are used to evaluate the controller performance in action.**



(a) Cassie opening the door for the quadcopter



(b) A bird's eye view of Cassie's and the quadcopter's trajectories

Fig. 1: 3D rendering of case study 1 in Drake simulation [4]. The quadcopter approaches an unknown door obstacle preventing it from achieving its goal. The quadcopter requests assistance from Cassie. Accordingly, Cassie opens the door and both agents resume their original task.

I. Introduction

Heterogeneous multi-agent systems with distinct mobility capabilities are generally capable of accommodating a larger variety of tasks than those consisting of a homogeneous team of agents [1], [2]. To achieve autonomous team behaviors such as the multi-room patrolling shown in Figure 1b, a common approach is to automatically synthesize a controller for each agent, as this is often more efficient than crafting controllers by hand. However, creating controllers in this way has its own set of challenges, among which ensuring that the generated controllers do not cause any agents to perform tasks that would induce breakage or otherwise risk the agent's safety is a top priority [3].

In this paper, we study controller synthesis to resolve environmental conflicts such that multi-agent team specifications are fulfilled using the generalized reactivity (1) (GR(1)) fragment [5] of Linear Temporal Logic (LTL). The GR(1) formula, in particular, allows for reactive synthesis algorithms that have favorable polynomial complexity while retaining the ability to encode a large variety of specifications [6]–[8]. These reactive synthesis methods are powerful because they provide formal guarantees on correct-by-construction controller behavior under any modeled action from the environment [9].

A major challenge of reactive synthesis methods is that they require an explicit model of the environment's capabilities and may not be robust to unexpected changes in these capabilities encountered at runtime [10]. For example, an unmodeled obstacle that interferes with the operation of the system may unexpectedly appear. Ensuring that the system's operation is robust to environmental changes is therefore an important area of research. To this end, multiple lines of research have been proposed in the literature, such

M. E. Cao and S. Coogan are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, 30332, USA {mcao34, sam.coogan}@gatech.edu. S. Coogan is also with the School of Civil and Environmental Engineering, Georgia Institute of Technology, Atlanta, 30332, USA. J. Warnke and Y. Zhao are with the School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, 30332, USA jwarnke@gatech.edu & ye.zhao@me.gatech.edu.

This research was supported in part by the National Science Foundation under the National Robotics Initiative, award #1924978. The authors would also like to thank Yuki Yoshinaga and Aziz Shamsah for their assistance in rendering 3D representations of the simulation environment.

*The first two authors equally contributed to this paper.

as online synthesis of local strategies which are further patched to the original controller [11], offline analysis of counter-strategies to resolve unrealizable specifications [12], controller synthesis that can tolerate a finite sequence (up to N steps) of environmental assumption violations [13], or robust metric automata design such that the system state is maintained within a bounded ϵ-distance from the nominal state under unmodeled disturbances [14]. There also exist robustness methods that allow the system to identify specific broken environment assumptions [15]. However, none of these works studied the strategy of employing other agents to resolve the environmental conflict, which will be the focus of this paper, as little has been explored in this direction. For example, the authors in [16] have studied the correction of broken assumptions, but focus on the cases where the broken assumption is due to unexpected behaviors by another agent operating within the workspace, which is resolved by changing that agent's behavior.

As heterogeneous agents are generally expected to interact with their surrounding environment, certain agents may have the capability to manipulate and correct a broken environment assumption for another agent. This paper focuses on leveraging each agent's individual capabilities to resolve broken environment assumptions that prevent another agent from achieving its objective. Formally, we consider scenarios in which the broken environment assumption causes an agent's specification to become unrealizable, yet another agent has the ability to fix the violation.

The contribution of this paper is to propose a navigation planning framework with four main components that enable agents to assist each other in resolving obstacles using reactive synthesis. These components are summarized as:

- **Environment Characterization:** Observe the environment at runtime and verify whether the next state in the controller automaton would satisfy or violate any new specifications generated from runtime observations.
- **Safe Action Replanning:** Backtrack states in the automaton and replace an action that will lead to a safety violation with a known safe maneuver.
- **Violation Resolution:** Identify other agents with the capability of resolving the violation and assign one to be responsible for violation resolution.
- **Task Replanning:** Add the resolution of the violation to the assigned agent's objectives and trigger a change in behavior of that agent to execute its new objective.

We refer to these components collectively as the "coordination layer" that interacts with the other elements of the controller (see Figure 2).

The rest of the paper is outlined as follows: In Section II, we introduce the basics of LTL specifications and GR(1) formula. We then formally define our problem statement in Section III. In Section IV, we provide an overview of the previous work that this study is built upon before detailing the main approach that enables heterogeneous cooperation in Section V. Section VI outlines several case studies showcasing our approach, and Section VII concludes the study and discusses potential future work.

II. Preliminaries

In this study, we use the General Reactivity of Rank 1 (GR(1)) fragment [5] of Linear Temporal Logic (LTL) to specify desired tasks for each agent in a given environment. GR(1) synthesis is used to automatically generate correct-by-construction finite state machines (FSM). The generated strategy is implemented as a two-player game between the agent and the environment, where the FSM guarantees the agent satisfies the goal and safety specifications for any modeled environment action [6], [8].

GR(1) allows for efficient synthesis while maintaining much of the expressiveness of LTL. In particular, GR(1) allows us to design temporal logic formulas (φ) with atomic propositions (AP) that can either be True ($\varphi \vee \neg\varphi$) or False ($\neg\mathsf{True}$). With negation ($\neg$) and disjunction ($\vee$) one can also define the following operators: conjunction ($\wedge$), implication ($\Rightarrow$), and equivalence ($\Leftrightarrow$). There also exist temporal operators "next" ($\bigcirc$), "eventually" ($\Diamond$), and "always" ($\Box$). Further details of GR(1) can be found in [5].

Our implementation uses the SLUGS reactive synthesis tool [17], which allows rules to be specified in a more human-intelligible structured slugs format using infix notation, non-negative integers, comparisons, and addition. These rules are automatically converted to GR(1) formulas which are used to synthesize a reactive controller.

III. Problem Formulation

This paper studies a particular control synthesis problem where an agent in a heterogenous multi-agent team cannot complete its tasks because the environment has violated its assumptions at runtime. Formally, this occurs because an unmodeled environment behavior causes the specification to become unrealizable.

Let $\mathcal{P}$ denote the set of heterogeneous agents in a multi-agent team. When synthesizing a multi-agent controller, each agent $p \in \mathcal{P}$ within the team is given its own set of goal and safety specifications, denoted as φ_o^p and φ_s^p, respectively.

At the high-level, the environment is modeled using a coarse abstraction that divides the workspace into a set of N discrete regions $\mathcal{S} = \{s_0, s_1, ..., s_{N-1}\}$. As low-level controllers are responsible for planning agent actions within each coarse region, they can be swapped in and out to accommodate different agent types without largely affecting the high-level actions.

The set of known, irresolvable obstacles $\mathcal{O} \subset \mathcal{S}$ are accounted for as the set of safety specifications

$$\varphi_s^p := \bigwedge_{s \in \mathcal{O}} \Box \neg s. \tag{1}$$

To account for the heterogeneity of the system, each agent $p \in \mathcal{P}$ is also modeled with an a priori known finite set of capabilities $C_p = \{c_{p0}, c_{p1}, c_{p2}, ...\}$. Examples of capabilities include "open doors", "inspect regions for hazards", or "climb stairs".

Obstacles that are resolvable but not known a priori are modeled as another subset $\mathcal{R} \subset \mathcal{S}$. Each resolvable obstacle $r \in \mathcal{R}$ has an associated action c_r and set of states S_r within which that action may be performed in order to resolve the obstacle and remove it from the environment. These properties are such that

$$c_r \in \bigcup_{p \in \mathcal{P}} C_p, \quad S_r \subset (\mathcal{S} \setminus \mathcal{O}). \tag{2}$$

Thus, an agent p is considered to be capable of resolving an obstacle r if $c_r \in C_p$. It follows that any instance of an agent encountering an obstacle that it does not have the capability to resolve is considered a safety violation. We introduce an "augmented" set of safety specifications φ_a^p, which contains the same specifications as φ_s^p but in addition contains all of the additional safety specifications originating from unknown, resolvable obstacles:

$$\varphi_a^p := \varphi_s^p \bigwedge_{r \in R \ | \ c_r \notin C_p} \Box \neg r \tag{3}$$

Given the necessary preliminaries above, we can now formally define the problem statement.

Problem Statement: Assume a set of given controllers synthesized using φ_s^p, and a set of "actual" environment specifications φ_a^p such that one or more φ_o^p are unrealizable under φ_a^p. Once the system is detected to violate φ_a^p at runtime, we aim to create a generalizable formulation that can assign an agent p to resolve and remove the conflicting specification in φ_a^p such that the original synthesized controller satisfies φ_a^p.

IV. Controller Synthesis

To leverage the formal guarantees afforded by LTL, we synthesize navigation planners for each agent based on the planning framework detailed in [18]. In this section, we provide an overview of the task and motion planners, which serve as the foundation that the proposed coordination layer will be built on. In subsequent sections, we augment the high-level navigation planning structure to further encode collaborative behaviors that are able to resolve environment assumption violations at runtime.

The approach from [6]–[8] adopted here is summarized as follows. To synthesize task planners, we construct two-player games between each agent and an abstracted environment. We automatically encode a variety of propositions about how the game may evolve within LTL specifications: we encode initialization assumptions, environment safety assumptions, system safety guarantees, and system liveness properties. We synthesize an automaton that guarantees the agent will always win the game as long as all the environment assumptions hold true at runtime. The automaton is represented as a finite state machine (FSM). At runtime the current environment state is an input to the FSM, which outputs an action for the agent. Each action provided by the FSM is guaranteed to meet the safety specifications while bringing the agent closer to completing its task.

In this work we synthesize planners for a bipedal robot Cassie [19] and a quadcopter to study heterogeneous autonomous multi-agent navigation. When constructing a two-player game between each agent and its environment, we treat other agents as part of a partially observable environment and encode the possible moves each agent can expect the others to make in environment safety specifications. Deadlock resolution becomes a challenge when guaranteeing collision avoidance and task completion in the presence of other agents. In this study, we circumvent this challenge by assuming that the quadcopter is going to fly above Cassie at all times, so we do not encounter deadlock, which allows us to focus on mission replanning for assistive behaviors at runtime. Deadlock resolution techniques such as the one presented in [20] are implementable in our planning framework.

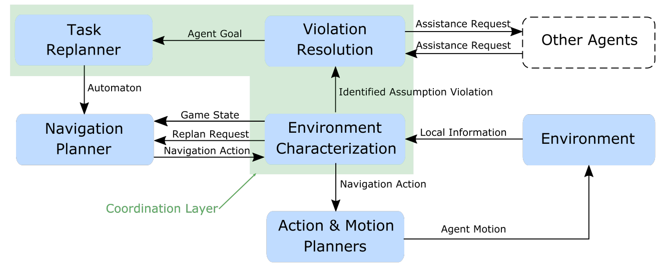


Fig. 2: Block diagram of collaborative task and motion planning framework. A coordination layer verifies that the desired actions generated by an offline-synthesized navigation planner are still safe based on environment information observed at runtime. If an environment assumption is violated, the navigation planner replans its current action to ensure the system enters a safe state while the coordination layer determines if the violation can be resolved by any other agent. The resolution is encoded and the plan progresses.

V. Coordination Layer Design

In this section, we introduce the coordination layer on top of the synthesized controllers, containing the elements which enable agents to identify new safety specifications at runtime, replan actions when necessary, identify and assign other agents to resolve obstacles, and adjust the behavior of each agent to execute the conflict resolution. An overview of the proposed planning framework is depicted in Figure 2.

A. Environment Characterization

At runtime, the Environment Characterization (EC) block passes the game state abstracted from the environment to the Navigation Planner (NP) block and requests a navigation action. The EC block determines new safety specifications based on the observed environment and verifies if the navigation action would violate these specifications. This block is the main component responsible for observing the specifications in φ_a^p that were not known during synthesis. We assume that the agent has adequate sensing capabilities to determine if any states reachable by the possible current navigation actions are safe. If a safety violation occurs, the EC block signals the NP block to replan its current action.

B. Safe Action Replanning

The safe action replanning occurs within the NP block. At each step, the current environment state is fed to the FSM to generate a correct system action. When the replanning flag is raised, the navigation planner backtracks to the previous state in the FSM and extracts a new system action to avoid the safety violation. The new action is passed to the EC block, which passes it on to the lower-level planners if deemed safe.

C. Violation Resolution

When the EC block determines that the safety specifications based on the observed environment do not match the safety specifications used during offline synthesis, it also passes the details of the violation to the Violation Resolution (VR) block, including the action(s) c_r required, as well as the state(s) S_r at which those actions must be performed in to resolve the obstacle. The VR block then identifies which agent $p \in \mathcal{P}$ has the capability to resolve the obstacle, i.e., any agent p such that $c_r \in C_p$. This component then broadcasts a request for an appropriate agent to come to assist.

D. Task Replanning

The Task Replanner (TR) block receives incoming requests for assistance in the form of updated system goals and is responsible for adjusting the agent's strategy to assist the agent in need. This can be accomplished either by resynthesizing the automaton based on the new system goals or by augmenting the initially synthesized controller with an additional runtime-assignable objective, which can then be assigned to the appropriate obstacle as needed. A more detailed elaboration of these two strategies is provided in the following subsections.

E. Resynthesis Method

Once a violation has been detected and assisting agents have been assigned, the TR block can trigger a resynthesis of each of the affected agent's controllers with their new objectives. After the controller detects that the obstacle has been resolved, another resynthesis is triggered, returning the agents to their original objectives.

This method is achieved recursively by storing previous objectives in a stack; if a new resolvable obstacle is encountered while resolving a known obstacle, the resynthesis targets the new obstacle and pushes the previous set of objectives onto the stack. Once an obstacle is resolved, the top set of objectives in the stack is popped off, and controllers are resynthesized to these objectives. Beyond changing the target locations in the original objective, any number of new goal tasks or locations can be encoded in the new specifications to assist another agent that has encountered an assumption violation. Additionally, the agent that has encountered an obstacle that it cannot resolve on its own can be assigned a new task to complete while it waits for the obstacle to be resolved. A detailed pipeline of this process is shown in Algorithm 1.

F. Non-resynthesis Method

For the targeted collaborative behavior, resynthesis is not strictly necessary if a resolution does not require a completely new task to be specified for the assisting agent. Instead, the agent is just required to visit one additional location. This allows the system to remain in continuous operation rather than halting for a period of time to allow for controller resynthesis. Assistance is achieved by including an additional runtime-assignable goal location in the assisting agent's liveness specifications. This behavior is described in more detail in Algorithm 2.

Algorithm 1: Resynthesis for Conflict Resolution

```
for p ∈ P do
    φ_o^p ← initial objectives;
end
φ_stack ← ∅;
synthesize controllers;
while system active do
    execute controllers;
    if resolvable obstacle r encountered by agent p_r at s_o then
        if c_r ∉ C_pr then
            push all of current φ_o^p onto φ_stack;
            change φ_o^{p_r} to safe behavior;
            for p ∈ P do
                if c_r ∈ C_p then
                    add state in S_r to φ_o^p;
                    synthesize controllers;
                    break;
            end
            continue;
        else if r ∈ φ_o^{p_r} then
            resolve obstacle r;
            pop φ_o^p off of φ_stack;
            synthesize controllers;
            continue;
        end
    end
end
```

Algorithm 2: Non-resynthesis for Conflict Resolution

```
for p ∈ P do
    φ_o^p ← initial objectives + runtime assignable objective;
end
synthesize controllers;
while system active do
    execute controllers;
    if resolvable obstacle r encountered by agent p_r at s_o then
        if c_r ∉ C_pr then
            push all of current φ_o^p onto φ_stack;
            change φ_o^{p_r} to safe behavior;
            for p ∈ P do
                if c_r ∈ C_p then
                    assign runtime objective to state in S_r;
                    break;
            end
            continue;
        else
            resolve obstacle r;
            continue;
        end
    end
end
```

For the full implementation of either of the methods outlined in this paper, we point the reader to the GitHub repository located at `https://github.com/GTLIDAR/safe-nav-locomotion`.

VI. Results

In this paper, we implement and evaluate the framework described in Section IV, which was primarily built to synthesize controllers for the bipedal walking robot platform Cassie, designed by Agility Robotics [19]. We consider a second quadcopter agent in the environment that has dramatically different capabilities in both mobility and manipulation. A quadcopter, which lacks the ability to manipulate objects, is instead able to perform maneuvers that are unavailable to Cassie, such as backward movement or $180°$ turns in a single region. In this study, the quadcopter is assumed to fly above Cassie at all times, so that collision is not a concern. Additionally, we preserve the belief space planning framework proposed in [18], [21], allowing Cassie to infer the quadcopter location if it is not within Cassie's visible range. Cassie's locomotion planner is designed based on the phase-space planning framework in [22].

Resolvable obstacles are also implemented within the simulated environment. We recall from Section III that each resolvable obstacle r has an associated state S_r where an action c_r must be performed by an agent p for which $c_r \in C_p$ in order to resolve the obstacle and remove it from the environment. These properties are directly inserted into the simulation environment.

The set of resolvable obstacles $\mathcal{R}$ and the set of agents and their capabilities C_p are implemented as separate dictionary data structures. As such, this framework is generalizable as one would simply need to add the appropriate agent capabilities and obstacle resolutions to each dictionary.

To represent the potential for obstacles to appear at runtime, two separate simulated environments are initialized, one of which does not contain the resolvable obstacles that will need to be resolved. This instance of the environment, representing φ_s^p, is used for the initial synthesis, and then the resulting controller is applied to the environment instance containing the resolvable obstacles, which corresponds to φ_a^p. At runtime, if an agent enters a state containing a resolvable obstacle r that it is unable to resolve (i.e., violates φ_a^p), the controller is able to check that a violation has happened in the simulated environment, and sends this information to the simulation to assign new objectives to each agent accordingly, such that the agent p tasked with resolving the obstacle fulfills $c_r \in C_p$. Thus, the simulation running each of the controllers is responsible for the VR and TR blocks of the coordination layer, while the separate simulated environment instances simulate the EC component. The Safe Action Replanner is built directly into the NP block.

Three case studies utilizing the synthesized controllers are presented to evaluate the proposed approach. For each case study, an environment is created where a quadcopter and Cassie are each running on their own controller and have their own task objectives to complete. The environment is abstracted into a 7×13 coarse set of regions $\mathcal{S} = \{s_0, s_1, ..., s_{90}\}$ such that s_0 is the northwestern-most region and increments following English reading orientation (i.e. incrementing left to right, then starting at the leftmost region on the next row). This setup can be seen in Figures 3-6.

For each case study, we consider a team of agents consisting of $P = \{\mathtt{quadcopter}, \mathtt{Cassie}\}$ with unique capabilities $C_{\text{quad}} := \{\mathtt{sense}\}$, $C_{\text{Cassie}} := \{\mathtt{push}\}$. These do not represent the full capabilities of each agent, but only represent the ones that are relevant for solving the conflict resolution.

Resolvable obstacles that may appear within a region s_i in the environment consist of two types: $r = \mathtt{door}$ and $r = \mathtt{uncertainty}$. A resolvable obstacle of type $r = \mathtt{door}$, if found in s_i, has properties

$$S_r = \{s_i\},\ c_r = \mathtt{push}, \tag{4}$$

and represents physical doors that the quadcopter cannot fly through, but are able to be opened by Cassie. Resolvable obstacles of type $r = \mathtt{uncertainty}$ have the properties

$$S_r = \{s_{\text{north}}, s_{\text{east}}, s_{\text{south}}, s_{\text{west}}\},\ c_r = \mathtt{sense}, \tag{5}$$

and represent regions in which Cassie is uncertain about its capabilities to safely traverse through the environment, but the quadcopter is able to scout them by visiting any adjacent region.

We design a set of objective specifications for each agent $p \in \mathcal{P}$ such that the agent alternates between visiting two regions in the environment $s_{\text{A}}, s_{\text{B}} \in \mathcal{S}$, with an AP $\mathtt{scout}$ unique to each agent, which initializes to False, to track which region the agent should head towards. The set of objective specifications is thus given as

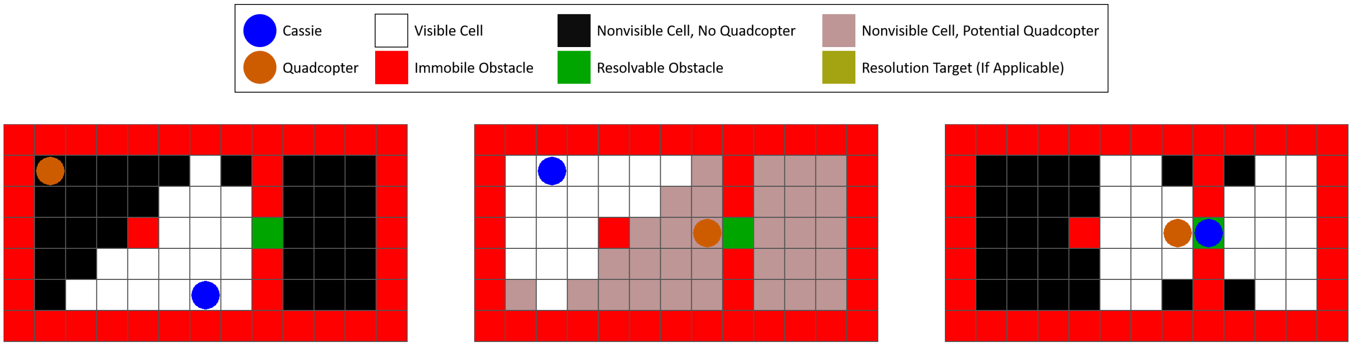
$$\begin{aligned} \text{Patrol}_p(s_{\text{A}}, s_{\text{B}}) =& \square\lozenge(s_{\text{A}} \land \neg\mathtt{scout}_p) \\ &\land \square((s_{\text{A}} \land \neg\mathtt{scout}_p) \Rightarrow \bigcirc\mathtt{scout}_p) \\ &\land \square\lozenge(s_{\text{B}} \land \mathtt{scout}_p) \\ &\land \square((s_{\text{B}} \land \mathtt{scout}_p) \Rightarrow \bigcirc\neg\mathtt{scout}_p) \end{aligned} \tag{6}$$

Each implementation of these case studies utilizes the resynthesis method detailed in Section V-E due to its overall lower synthesis time, but it should be noted that case study 1 is fully implementable using the non-resynthesis method outlined in Section V-F. As case studies 2 and 3 require multiple locations to be visited and resolved, additional work is required to enable the non-resynthesis method to work in these cases.

Additionally, while the figures in this section mainly feature abstractions of the environment in order to easily illustrate the behaviors of each agent, the computed control actions are applicable to a real 3D simulation environment, as shown in Figure 1. Those two subfigures show the real-world interactions resulting from the behavior in case study 1.

A. Case Study 1: Opening A Door

The first case study leverages Cassie's manipulation capability in the environment with higher dexterity and power than the quadcopter. The quadcopter is tasked with patrolling between s_{homeQ} and s_{awayQ}, where s_{homeQ} is a region in the left room and s_{awayQ} is in the right room. Cassie is



(a) Initial Configuration

(b) The quadcopter's controller encounters a door, so instead it elects to have the quadcopter wait in the preceding region until it is opened.

(c) Cassie opens the door and the original objectives are reinstated. Both agents are now able to complete their objectives.

Fig. 3: Execution of case study 1 leveraging Cassie's higher strength and manipulation abilities to open up the path for the quadcopter. Cassie's objective is to patrol the left room, while the quadcopter's objective is to deliver something to the right room. However, the quadcopter discovers a closed door separating the two rooms at runtime, prompting Cassie to come over and open it so that both agents are able to complete their objectives.

tasked with patrolling between s_{homeC} and s_{awayC}, where both s_{homeC} and s_{awayC} are in the left room:

$$\begin{aligned} \varphi_o^{\text{quad}} &:= \text{Patrol}_{\text{quad}}(s_{\text{homeQ}}, s_{\text{awayQ}}), \\ \varphi_o^{\text{Cassie}} &:= \text{Patrol}_{\text{Cassie}}(s_{\text{homeC}}, s_{\text{awayC}}). \end{aligned} \tag{7}$$

At runtime, the quadcopter discovers an obstruction at $s_{\text{door}} = s_{47}$ while in $s_{\text{safe}} = s_{46}$ that prevents it from accomplishing its objective in the form of a closed door that cannot be flown through but can be opened by Cassie. A resynthesis of objectives is triggered, where the quadcopter is now tasked with hovering outside the door, and Cassie is tasked with visiting one of its initial patrol points and the closed door:

$$\begin{aligned} \varphi_o^{\text{quad}} &:= \text{Patrol}_{\text{quad}}(s_{\text{safe}}, s_{\text{safe}}), \\ \varphi_o^{\text{Cassie}} &:= \text{Patrol}_{\text{Cassie}}(s_{\text{homeC}}, s_{\text{door}}). \end{aligned} \tag{8}$$

Once Cassie visits the door, it is considered open and the obstacle is removed from the environment, triggering another resynthesis which returns both agents to their original target objectives. A walkthrough of the execution of this case study is shown in Figure 3. For this case study, we also used low-level planners to generate safe motions for the quadcopter and Cassie, including center of mass (CoM) trajectories and foot placements. The 3D visualization can be seen in Figure 1 and the attached video.

B. Case Study 2: Scouting Ahead

The second case study involves Cassie encountering several states and not knowing whether each state is safe to traverse on foot, requiring the help of the quadcopter's heightened sensing capabilities. To this end, the quadcopter is set to patrol between s_{homeQ} and s_{awayQ}, where s_{homeQ} and s_{awayQ} are in the left room, while Cassie must patrol between s_{homeC} and s_{awayC}, where s_{homeC} is a region in the left room and s_{awayC} is in the right room:

$$\begin{aligned} \varphi_o^{\text{quad}} &:= \text{Patrol}_{\text{quad}}(s_{\text{homeQ}}, s_{\text{awayQ}}), \\ \varphi_o^{\text{Cassie}} &:= \text{Patrol}_{\text{Cassie}}(s_{\text{homeC}}, s_{\text{awayC}}). \end{aligned} \tag{9}$$

At runtime, Cassie encounters region $s_{\text{uncertain1}} = s_{34}$ while at $s_{\text{safe1}} = s_{33}$, and it is unsure about its ability to traverse this region. A resynthesis is triggered, where the quadcopter is tasked with observing the uncertain region by visiting any of the adjacent regions (in this case, we select the region $s_{\text{uncertain1W}}$ directly west of $s_{\text{uncertain1}}$):

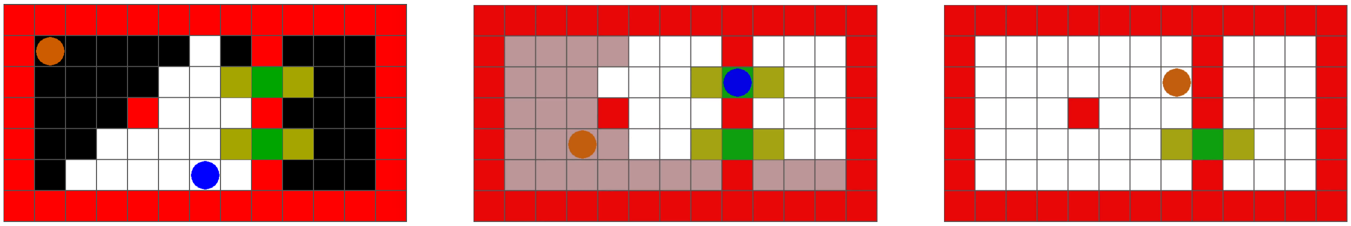
$$\begin{aligned} \varphi_o^{\text{quad}} &:= \text{Patrol}_{\text{quad}}(s_{\text{homeQ}}, s_{\text{uncertain1W}}), \\ \varphi_o^{\text{Cassie}} &:= \text{Patrol}_{\text{Cassie}}(s_{\text{safe1}}, s_{\text{safe1}}). \end{aligned} \tag{10}$$

Once the quadcopter observes the unknown region, the resolvable obstacle is removed from the environment. If the quadcopter senses that the region is not traversable by Cassie, then the region is added to O and will be considered as an immovable obstacle during future synthesis. In this specific case study, $s_{\text{uncertain1W}}$ is found to be untraversable.

The two agents are returned to their original objectives, outlined in (9), before Cassie encounters another uncertain state at $s_{\text{uncertain2}} = s_{60}$ while in $s_{\text{safe2}} = s_{59}$, where the process repeats. The quadcopter is tasked with visiting $s_{\text{uncertain2W}}$, directly west of $s_{\text{uncertain2}}$, while Cassie is instructed to stay at s_{safe2}. The uncertain region is found to be traversable by Cassie, and both agents are returned to their original objectives, now able to fulfill them. A walkthrough of the execution of this case study is shown in Figures 4 and 5.

C. Case Study 3: Chain of Conflicts

The third case study merges the previous two and requires both agents to resolve an obstacle. For this case study, the quadcopter encounters a door while on its way to resolving an uncertain region encountered by Cassie, requiring Cassie to first open the door, thus demonstrating the coordination

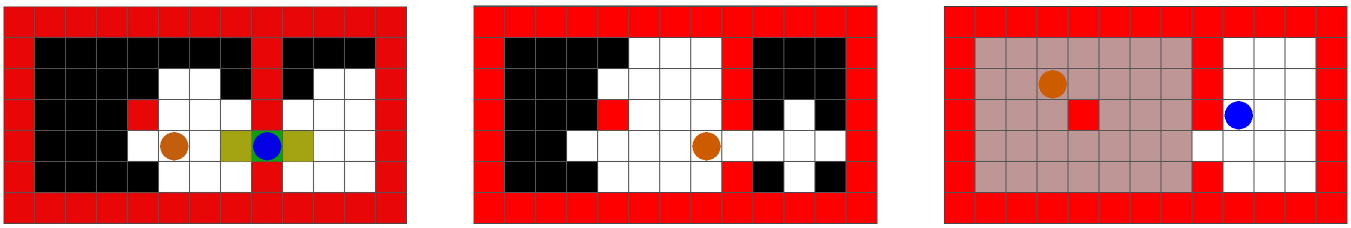


(a) Initial Configuration

(b) Cassie's controller encounters an uncertain region, so instead Cassie's controller elects to have Cassie wait in the preceding region until it is resolved.

(c) The quadcopter senses that the region is actually an immovable obstacle, resolving it, and original objectives are reinstated.

Fig. 4: Partial execution of case study 2 leveraging the quadcopter's more powerful sensory capabilities to find a traversable path for Cassie. The quadcopter's objective is to patrol the left room, while Cassie's objective is to deliver something to the right room. However, Cassie encounters an uncertain region, prompting the quadcopter to observe the region and determine that it is nontraversable.



(a) Cassie's controller encounters another uncertain region, so again Cassie's controller elects to have Cassie wait in the preceding region until it is resolved instead.

(b) The quadcopter senses that this region is traversable by Cassie.

(c) Original objectives are again reinstated, with both agents now able to meet their objectives.

Fig. 5: Latter half of the execution of case study 2. The quadcopter and Cassie are executing their original objectives when Cassie encounters another uncertain region. The quadcopter observes the new uncertainty, this time determining that the region is traversable, thus allowing both agents to fully complete their original objectives.

layer's ability to handle multiple resolvable obstacles in a chain when required.

Initially, Cassie is tasked with patrolling between s_{homeC} in the leftmost room and s_{awayC} in the center room, while the quadcopter is tasked with patrolling between s_{homeQ} and s_{awayQ}, both in the rightmost room:

$$\begin{aligned} \varphi_o^{\mathrm{quad}} &:= \mathrm{Patrol}_{\mathrm{quad}}(s_{\mathrm{homeQ}}, s_{\mathrm{awayQ}}), \\ \varphi_o^{\mathrm{Cassie}} &:= \mathrm{Patrol}_{\mathrm{Cassie}}(s_{\mathrm{homeC}}, s_{\mathrm{awayC}}) \end{aligned} \tag{11}$$

Cassie encounters an uncertain state at $s_{\mathrm{uncertain}} = s_{43}$ while in $s_{\mathrm{safeC}} = s_{43}$, triggering a resynthesis requiring the quadcopter to sense the true traversibility of that state by visiting $s_{\mathrm{uncertainE}} = s_{44}$:

$$\begin{aligned} \varphi_o^{\mathrm{quad}} &:= \mathrm{Patrol}_{\mathrm{quad}}(s_{\mathrm{homeQ}}, s_{\mathrm{uncertainE}}), \\ \varphi_o^{\mathrm{Cassie}} &:= \mathrm{Patrol}_{\mathrm{Cassie}}(s_{\mathrm{safeC}}, s_{\mathrm{safeC}}) \end{aligned} \tag{12}$$

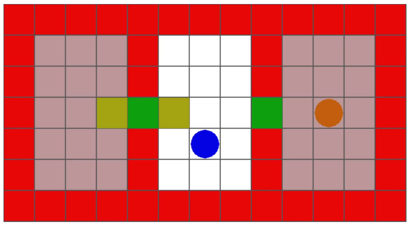
However, the quadcopter encounters a closed door at $s_{\mathrm{door}} = s_{47}$ while in $s_{\mathrm{safeQ}} = s_{48}$ on its way to resolve Cassie's uncertainty, triggering another resynthesis where Cassie is tasked with opening the door:

$$\begin{aligned} \varphi_o^{\mathrm{quad}} &:= \mathrm{Patrol}_{\mathrm{quad}}(s_{\mathrm{safeQ}}, s_{\mathrm{safeQ}}), \\ \varphi_o^{\mathrm{Cassie}} &:= \mathrm{Patrol}_{\mathrm{Cassie}}(s_{\mathrm{safeC}}, s_{\mathrm{door}}) \end{aligned} \tag{13}$$

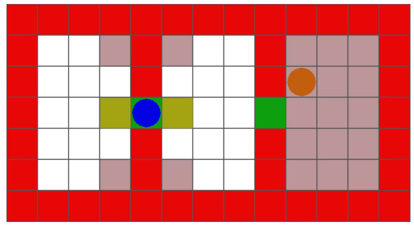
Once the door is opened and resolved, the quadcopter travels to the uncertain state and resolves that obstacle as well, resulting finally in both agents being able to accomplish their objectives. A walkthrough of the execution of this case study is shown in Figure 6.

VII. Conclusion and Future Work

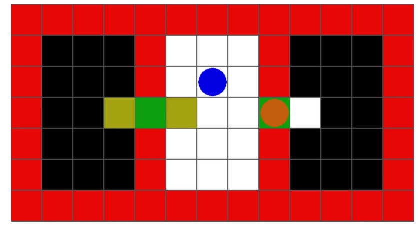
We presented a generalizable approach to identifying and resolving environment assumption violations discovered at runtime by automatically leveraging the capability of heterogeneous agents. This allows the team of agents to recover from cases where their objectives become unrealizable due to runtime-observed violations. We implemented this approach in a grid world simulation and generated safe 3D CoM motion plans for a Cassie bipedal robot and a quadcopter. Future directions of research include implementing deadlock resolution strategies, further developing the non-resynthesis solution to enable more complex runtime-assignable behaviors, and cataloguing a comprehensive library of robots and capabilities, such as multi-quadcopter teaming to deliver a battery for Cassie charging and Cassie long-duration navigation for package delivery.



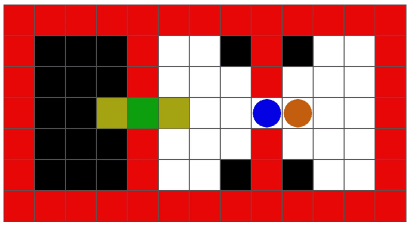
(a) Initial Configuration



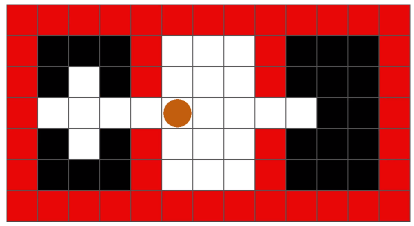
(b) Cassie's controller encounters an uncertain region, so it elects to have Cassie wait in the preceding region until it is resolved.



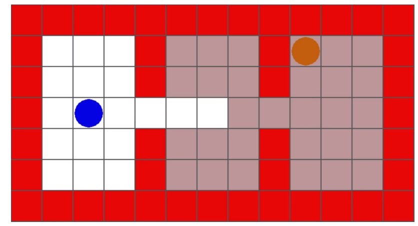
(c) On the way to sense the region that Cassie is uncertain about, the quadcopter encounters a closed door and waits on the other side.



(d) Cassie opens the door for the quadcopter, allowing it to come sense the uncertain region.



(e) The quadcopter senses that the region is passable by Cassie.



(f) Original objectives are now reinstated, with both agents now able to meet their objectives.

Fig. 6: Execution of case study 3 showcasing the capabilities of both agents and how they must each contribute in order to ensure a successful mission. Cassie's objective is to patrol the left and center rooms while the quadcopter is tasked with patrolling the right room. However, Cassie is unsure whether it is able to pass into the left room, prompting the quadcopter to fly towards the region in question. On the way, the quadcopter encounters a closed door, which Cassie must open before the quadcopter can continue.

References

[1] Y. Emam, S. Mayya, G. Notomista, A. Bohannon, and M. Egerstedt, "Adaptive task allocation for heterogeneous multi-robot teams with evolving and unknown robot capabilities," in *IEEE International Conference on Robotics and Automation*, pp. 7719–7725, 2020.

[2] J. L. Kit, A. G. Dharmawan, D. Mateo, S. Foong, G. S. Soh, R. Bouffanais, and K. L. Wood, "Decentralized multi-floor exploration by a swarm of miniature robots teaming with wall-climbing units," in *International Symposium on Multi-Robot and Multi-Agent Systems*, pp. 195–201, 2019.

[3] T. Wongpiromsarn, U. Topcu, and R. Murray, "Synthesis of control protocols for autonomous systems," *Unmanned Systems*, vol. 01, pp. 21–39, 07 2013.

[4] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019.

[5] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *Verification, Model Checking, and Abstract Interpretation*, pp. 364–380, Springer, 2006.

[6] J. Liu, N. Ozay, U. Topcu, and R. M. Murray, "Synthesis of reactive switching protocols from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 58, no. 7, pp. 1771–1785, 2013.

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

[8] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu, "Correct, reactive, high-level robot control," *IEEE Robotics & Automation Magazine*, vol. 18, no. 3, pp. 65–74, 2011.

[9] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, (New York, NY, USA), p. 179–190, Association for Computing Machinery, 1989.

[10] K. W. Wong, R. Ehlers, and H. Kress-Gazit, "Correct high-level robot behavior in environments with unexpected events," in *Robotics: Science and Systems*, 2014.

[11] S. C. Livingston, R. M. Murray, and J. W. Burdick, "Backtracking temporal logic synthesis for uncertain environments," in *2012 IEEE International Conference on Robotics and Automation*, pp. 5163–5170, IEEE, 2012.

[12] W. Li, L. Dworkin, and S. A. Seshia, "Mining assumptions for synthesis," in *ACM/IEEE International Conference on Formal Methods and Models for Codesign*, pp. 43–50, IEEE, 2011.

[13] R. Ehlers and U. Topcu, "Resilience to intermittent assumption violations in reactive synthesis," in *International Conference on Hybrid Systems: Computation and Control*, pp. 203–212, 2014.

[14] R. Majumdar, E. Render, and P. Tabuada, "Robust discrete synthesis against unspecified disturbances," in *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pp. 211–220, 2011.

[15] V. Raman and H. Kress-Gazit, "Automated feedback for unachievable high-level robot behaviors," in *2012 IEEE International Conference on Robotics and Automation*, pp. 5156–5162, 2012.

[16] K. W. Wong, R. Ehlers, and H. Kress-Gazit, "Resilient, provably-correct, and high-level robot behaviors," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 936–952, 2018.

[17] R. Ehlers and V. Raman, "Slugs: Extensible gr(1) synthesis," 2016.

[18] J. Warnke, A. Shamsah, Y. Li, and Y. Zhao, "Towards safe locomotion navigation in partially observable environments with uneven terrain," in *IEEE Conference on Decision and Control*, pp. 958–965, 2020.

[19] Agility Robotics in *http://www.agilityrobotics.com/*.

[20] J. Alonso-Mora, J. A. DeCastro, V. Raman, D. Rus, and H. Kress-Gazit, "Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles," *Autonomous Robots*, vol. 42, pp. 801–824, 2018.

[21] S. Bharadwaj, R. Dimitrova, and U. Topcu, "Synthesis of surveillance strategies via belief abstraction," in *IEEE Conference on Decision and Control*, pp. 4159–4166, IEEE, 2018.

[22] Y. Zhao, B. R. Fernandez, and L. Sentis, "Robust optimal planning and control of non-periodic bipedal locomotion with a centroidal momentum model," *The International Journal of Robotics Research*, vol. 36, no. 11, pp. 1211–1242, 2017.