# SAFETY-GUARANTEED TASK PLANNING FOR BIPEDAL NAVIGATION IN PARTIALLY OBSERVABLE ENVIRONMENTS

A Thesis
Presented to
The Academic Faculty

By

Jonas Warnke

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
George W. Woodruff School of Mechanical Engineering

Georgia Institute of Technology

December 2021

**SAFETY-GUARANTEED TASK PLANNING FOR BIPEDAL NAVIGATION IN PARTIALLY OBSERVABLE ENVIRONMENTS**

Thesis committee:

Dr. Ye Zhao
Department of Mechanical Engineering
*Georgia Institute of Technology*

Dr. Sam Coogan
Department of Electrical Engineering
*Georgia Institute of Technology*

Dr. Jonathan Rogers
Department of Aerospace Engineering
*Georgia Institute of Technology*

Date approved: 11/24/21

Dedicated to my loving parents who have cultivated my intellectual curiosity and whose unwavering support made this work possible.

## ACKNOWLEDGMENTS

I would like to thank the members of my thesis committee for their help in preparation of this work. I would like to extend my sincere thanks to Prof. Ye Zhao for his guidance and expertise throughout my graduate studies, his enthusiasm for the potential of our work kept me motivated. I am also grateful to Prof. Sam Coogan for many discussions on collaborative robotics. I would like to thank my collaborators Aziz Shamsah, Michael Cao, and Yingke li, who helped shape my research and allowed me to extend my work to have more impact. I would also like to acknowledge Suda Bharadwaj and Ufuk Topcu for their discussions on belief space planning.

I cannot begin to express my gratitude to my friends and family who supported me during my time at Georgia Tech. I am thankful to my parents Gilla and Peter who encouraged me to pursue my interests without letting anything stand in my way, and who, along with my sisters Lena and Louisa, offered their unwavering support. I am extremely grateful to my girlfriend Shweta, who's sympathetic ear and backing helped me push through and finish my thesis. Lastly, special thanks to all my friends who offered me company and encouragement during my studies, you made enduring graduate school during a global pandemic bearable: Ben, Eric, Dilip, and Luke to name a few.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

## LIST OF ACRONYMS

$\boldsymbol{a}_{\mathrm{HL}}$ high-level action

$\boldsymbol{a}_{\mathrm{LL}}$ high-level action

$\boldsymbol{e}_{\mathrm{HL}}$ robot state in action planning environment

**AP** atomic propositions

**BDD** binary decision diagram

**CEGAR** counter example guided abstraction refinement

**CoM** center-of-mass

**DoF** Degrees of Freedom

**EC** Environment Characterization

**FSM** Finite State Machine

**GR(1)** General Reactivity of Rank 1

**HL** high-leve

**LL** low-level

**LTL** Linear Temporal Logic

**MPC** model predictive control

**NP** Navigation Planner

**OWS** One Walking Step

**PIPM** Prismatic Inverted Pendulum Model

**PSP** phase-space planning

**ROM** Reduced Order Model

**TR** Task Replanner

**VR** Violation Resolution

## SUMMARY

Bipedal robots are becoming more capable as basic hardware and control challenges are being overcome, however reasoning about safety at the task and motion planning levels has been largely underexplored in the field. My work makes key steps towards guaranteeing safe locomotion in cluttered environments in the presence of humans or other dynamic obstacles by designing a hierarchical task planning framework that incorporates multi-level safety guarantees. This layered planning framework is composed of a coarse high-level symbolic navigation planner and a lower-level local action planner. A belief abstraction at the global navigation planning level enables belief estimation of non-visible dynamic obstacle states and guarantees navigation safety with collision avoidance. Both planning layers employ linear temporal logic (LTL) for a reactive game synthesis between the robot and its environment while incorporating lower-level safe locomotion keyframe policies into formal task specification design. The synthesized task planner commands a series of locomotion actions, including walking step length, step height, and heading angle changes, to a motion planner which generates a center of mass trajectory and foot placements.

The modularity of the planning framework allows it to be applied to a diverse selection of robot agents such as bipedal robots, mobile robots, or quadcopters. Each type of robot simply requires a modified local action planner to generate actions and navigation waypoints that take the robot's dynamic limitations into account. The high-level symbolic navigation planner has been extended to leverage the capabilities of a team of heterogeneous agents to resolve unmodeled environmental conflicts that appear at runtime. When an environment assumption that was used to synthesize a given navigation planner is violated at runtime, the navigation strategy loses its safety and task completion guarantees and becomes invalid. Modifications in the navigation planner in conjunction with a coordination layer allow each agent to guarantee immediate safety and eventual task completion in the presence of an assumption violation if another agent exists that can resolve the said vio-

lation, e.g., a door is closed that another dexterous agent can open. This is achieved in four steps: 1) The environment is characterized at runtime, and it is verified whether the next state in the controller automaton would satisfy or violate any safety specifications based on runtime observations, 2) the immediate control action is replanned by backtracking states in the automaton and replacing unsafe actions with known safe actions, 3) a resolution is identified and assigned to another agent, 4) involved agents replan to eliminate the violation.

The planning framework leverages the expressive nature and formal guarantees of LTL to generate provably correct controllers for complex robotic systems. The use of belief space planning for dynamic obstacle belief tracking and heterogeneous robot capabilities to assist one another when environment assumptions are violated allows the planning framework to reduce the conservativeness traditionally associated with using formal methods for robot planning.

# CHAPTER 1
# INTRODUCTION AND BACKGROUND

Robots are increasingly being integrated in real-life scenarios as they can provide a plethora of physical, economic, and societal benefits when deployed correctly. Robots are able to complete physical tasks that humans can't or don't want to perform, they provide businesses with novel solutions to automate tasks from logistics to maintenance, and have the capability of operating in environments that are unsafe for humans, such as performing search and rescue operations. Legged robots specifically present the most utility potential in complex workspaces as they have the capability of operating in environments designed for humans, outperforming wheeled robots at negotiating uneven terrain such as debris or stairs. However, many problems still need to be solved before legged robots can be fully integrated into our society.

Navigation in real-life workspaces presents the challenge of task and motion planning as the environments may contain dynamic obstacles such as humans or other autonomous robots. Dynamic obstacles may be adversarial or, more likely, may be oblivious to an autonomous legged robot and will therefore not necessarily attempt to avoid collision with such a robot. The burden of collision avoidance must lie on the robot planner to enable such a robot to be safely inserted into existing environments. Collision avoidance with dynamic obstacles is further complicated by static obstacles as they can occlude a robot sensor's view of the environment, making it challenging to safely navigate around them, this can bee seen in Figure 1.1. In the context of dynamic legged locomotion, maintaining dynamic balancing, i.e., avoiding a fall [1, 2, 3], becomes an additional essential safety criterion beyond avoiding collisions. Reasoning about safety from both levels has been largely under-explored in the field. As one closely-related line of research, Wieber's recent studies [4, 5] proposed a model predictive control (MPC) method to address safe naviga-

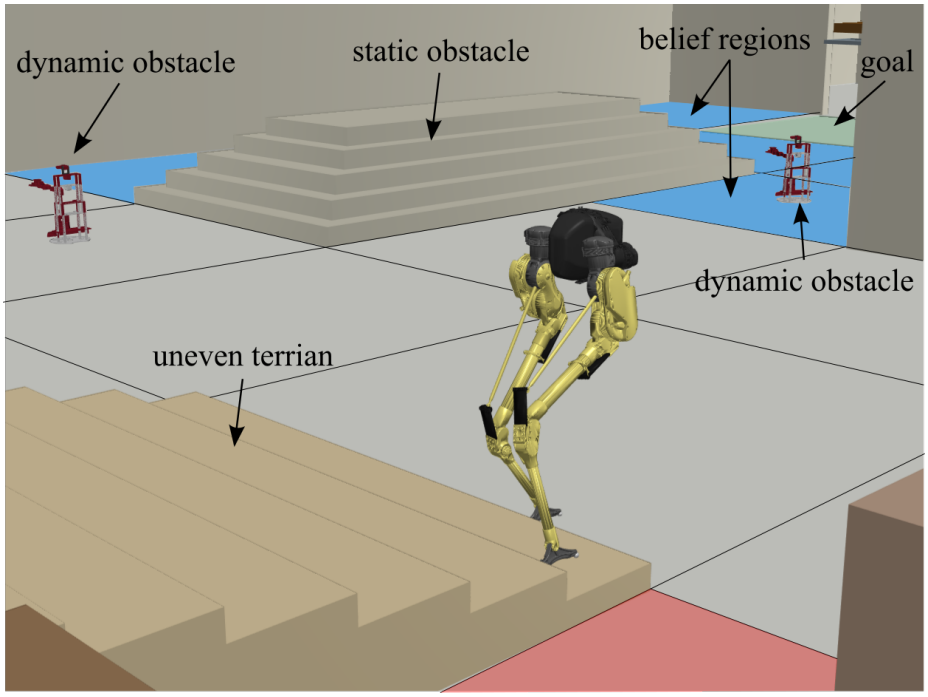



Figure 1.1: A snapshot of the simulation environment for the proposed TAMP framework. The walking robot is deployed to accomplish safe navigation tasks. The environment contains static and dynamic obstacles, and uneven terrains.

tion problems for bipedal walking robots in a crowded environment. Nevertheless, their work mainly focused on *passive* safety, i.e., the robot comes to a stop for collision avoidance. Their MPC optimization weighs the safety criteria and lacks formal guarantees on navigation safety.

Formal guarantees on safe task completion in complex environment has been gaining interest in recent years [6, 7, 8], however the literature for multi-level guarantees for underactuated legged robots in a dynamic environments remains lacking. An intrinsic challenge of multi-level formal guarantees for bipedal systems, is guaranteeing viable execution of discretized high-level commands and generating continuous motion plans for the inherently complex bipedal dynamics. This study explicitly addresses this challenge by encoding low-level physics-consistent safety criteria into the high-level task specification design. This strategy ensures that, on top of collision avoidance, the task planner commands actions that can be safely executed by a low-level planner. The high level planning method presented here takes one step towards using a symbolic planning method to design *active*

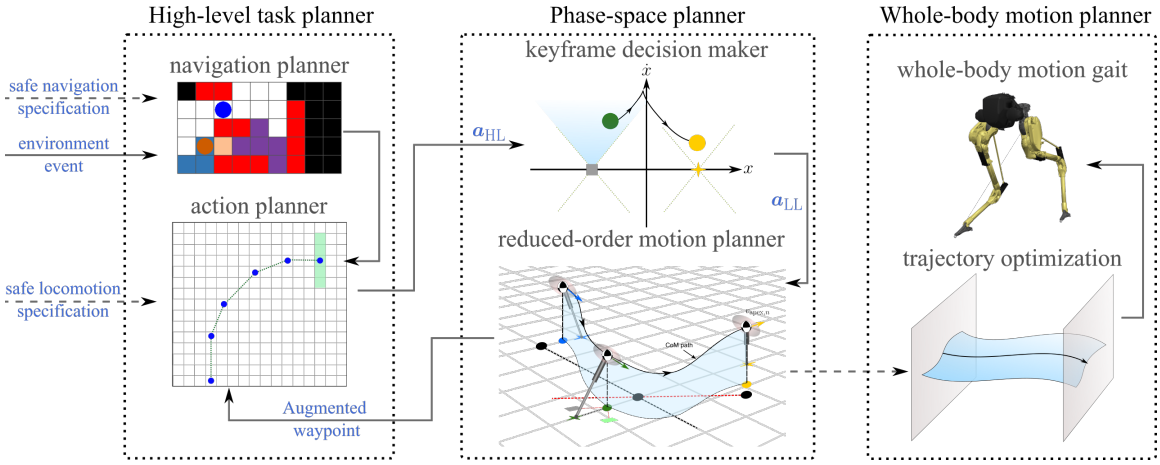



Figure 1.2: Block diagram of the proposed locomotion planning framework. The task planner employs a linear temporal logic approach to synthesize actions. At the low-level, the keyframe decision-maker generates the keyframe states sent to the motion planner. Locomotion specifications from the low-level will be incorporated into the task planner.

navigation decisions for safety guarantees, i.e., the robot steers its walking direction to avoid collisions besides the coming-to-a-stop strategy. A proposed integrated hierarchical framework seen in Figure 1.2 provides multi-layer formal safety and navigation guarantees for underactuated bipedal system operating in dynamic, cluttered, and partially observable environments. The work presented here is a continuation of prior work [9], with extensions toward formal guarantees on low-level motion plans, non-determistic high-level transitions, and joint-belief abstractions for tracking the belief state of multiple dynamic obstacles to generate safe reactive plans for the 20 Degrees of Freedom (DoF) Cassie bipedal system [10].

A major challenge of reactive synthesis methods for formal guarantees on a system is that they require an explicit model of the environment's capabilities and are not inherently robust to unexpected changes in these capabilities encountered at runtime [11]. For example, an unmodeled obstacle that interferes with the operation of the system may unexpectedly appear. General Reactivity of Rank 1 (GR(1)) is a fragment [12] of Linear Temporal Logic (LTL), a common and suitable specification language for designing correct-by-construction robot controllers due to its expressiveness and relative computational efficiency. GR(1) specifically relies on an assumption guarantee structure where the

Figure 1.3: A conceptual illustration of a heterogeneous multi-agent team of robots completing diverse tasks in an indoor logistics environment (Special Credit to Yuki Yoshinaga for sharing this simulation).

system behavior is only guaranteed when the environment assumptions used during controller synthesis hold true at runtime. Ensuring that the system's operation is robust to environmental changes is therefore a challenging yet important consideration when designing a reactive task and motion planner [13].

It is particularly important for task and motion planning to be robust to common unpredictable obstructions that are unreasonable and unnecessary to explicitly and completely model within the environment assumptions, such as physically obstructed hallways and doorways, or changes in terrain estimation confidence due to new debris that may affect traversability. We envision that teams of heterogeneous multi-agent systems with distinct mobility capabilities are deployed to cooperatively solve a larger variety of tasks than those consisting of heterogeneous agents [14, 15]. An illustration of such collaboration can be seen in Figure 1.3. To achieve autonomous team behaviors such as the multi-room patrolling, a common approach is to automatically synthesize a controller for each agent that contains a decomposed component of a global task. The task and motion planning frame-

Table 1.1: Key performance classifications of this thesis.

| Performance index | Navigation planner (chapter 3) | Action planner (chapter 3) | Multi agent coordination layer (chapter 4) |
|---|---|---|---|
| Navigation safety | Static and dynamic obstacle collision avoidance guarantees via LTL synthesis | Guarantee desired navigation transitions via LTL synthesis | Static & dynamic obstacle collision avoidance guarantees via LTL synthesis |
| Locomotion safety | N/A | Commanded actions guaranteed to meet safety criteria encoded in LTL specifications | (To be explored) |
| Computational tractability | Joint belief space planning | (To be explored) | Online resynthesis at coarse abstraction only |
| Robustness | Belief tracking over approximates possible obstacle movement | Online action replanning for disturbances & imperfect tracking | Multi agent collaborative conflict resolution |

work created for bipedal planning is amenable to synthesizing controllers for a diverse set of robots as specific robot dynamics considerations are contained within the action planning layer, a single interchangeable component within the high level task planner. Utilizing a cohesive framework for a set of heterogeneous agents allows for a unified solution for inter-agent collaboration. The proposed solution 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 planning framework proposed in this work leverages the expressive nature and formal guarantees of LTL to generate provably correct controllers for complex robotic systems. Low level continuous dynamics are captured in abstract discrete high level transition specifications to create a reactive and provably safe task planner for bipedal locomotion navigation in partially observable environments. The use of belief space planning for dynamic obstacle belief tracking and heterogeneous robot capabilities to assist one another when environment assumptions are violated allows the planning framework to reduce the conservativeness traditionally associated with using formal methods for robot planning. Table 1.1 shows a summary of the performance benefits of each component proposed in this framework.

## 1.1 Related Work

Formal synthesis methods have been well established to guarantee high-level robot behaviors in dynamic environments [16, 17, 18, 19, 20]. Collision-free navigation in the presence of dynamic obstacles has been achieved via multiple approaches such as local collision avoidance controllers in [21], incrementally expanding a motion tree in sampling-based approaches [22], and Velocity Obstacle Sets generated by obstacle reachability analysis in [23]. Collision avoidance and task completion become more challenging to guarantee when the environment is only partially observable as such an environment has a strategic advantage in being adversarial. Navigating through partially known maps with performance guarantees has been achieved through exploring [24], updating the discrete abstraction, and re-synthesizing a controller at runtime in [25]. To avoid the computational costs of online re-synthesis, others have proposed patching a modified local controller into an existing global controller when unmodeled non-reachable cells, i.e. static obstacles, are discovered at runtime [26, 27]. The authors in [25] have proposed a satisfaction metric of specification to meet the specification as closely as possible when run-time discovered environment constraints render the specification unsatisfiable. Lastly, the work of [28] proposes a receding horizon planning method for efficient synthesis of short-horizon plans. As unmodeled obstacles appear in the planning horizon, a goal generator re-computes a path to a satisfying state. These approaches above are better suited for guaranteeing successful navigation and collision avoidance in environments that are uncertain only with respect to static obstacles as they can not reason about when and where a dynamic obstacle may appear.

Collision avoidance with dynamic obstacles in partially observable environments has been achieved through approaches such as POMDPs [29], Probabalistic velocity obstacle modeling [30], and object occlusion cost metrics [31]. The authors in [32] guarantee passive motion safety by avoiding braking Inevitable Collision States (ICS) at all times via a braking ICS-checking algorithm. While these solutions provide collision avoidance guar-

antees, they assume dynamic obstacles could appear at any time and result in an overly conservative strategies. Our method investigates belief-space planning to provide the controller additional information on when and where dynamic obstacles may appear in the robot's visible range to inform the synthesized strategy if navigation actions are guaranteed to be safe, even when static obstacles occlude the robot's view adjacent environment locations. We have devised a variant of the approach in [33] to explicitly track a belief of which non-visible environment locations are obstacle free, reducing the conservativeness of a guaranteed collision-free strategy. This belief tracking method is then integrated into our hierarchical TAMP framework.

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 [11]. 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 as online synthesis of local strategies which are further patched to the original controller [26], offline analysis of counter-strategies to resolve unrealizable specifications [34], controller synthesis that can tolerate a finite sequence (up to $N$ steps) of environmental assumption violations [35], or robust metric automata design such that the system state is maintained within a bounded $\epsilon$-distance from the nominal state under unmodeled disturbances [36]. There also exist robustness methods that allow the system to identify specific broken environment assumptions [37]. However, none of these works studied the strategy of employing other agents to resolve the environmental conflict, which will be the focus of the planning framework extension presented in this paper, as little has been explored in this direction. For example, the authors in [38] 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.

## 1.2 Thesis Structure

This work proposes a novel planning framework that utilizes reactive synthesis methods for safe collision free locomotion navigation with guarantees on task completion. The Framework has been extended to handle collaborative agent solvable assumption violations that occur at runtime. The structure of this thesis is organized as follows.

First, an introduction to the GR(1) fragment of LTL used for reactive controller synthesis, and the fundamentals of motion planning for bipedal locomotion using a Reduced Order Model (ROM) will be provided in chapter 2.

Next the hierarchical task planning framework will be outlined in chapter 3. The task planner is split into two layers with different environment abstractions, a global navigation planner and a local action planner. The details of the navigation planner's obstacle belief tracking are expounded for single and multiple obstacle scenarios, and the planners efficacy is demonstrated.

Extensions to the planning framework to allow for multi-agent collaboration in the presence of runtime environment assumption violations are described in chapter 4. This is achieved through four steps: environment characterization, safe action replanning, violation resolution, and task replanning. We refer to these components collectively as the "coordination layer" that interacts with the other elements of the planner. Three case studies highlighting the opperation of this solution in 2D gridworld simulations are presented.

Finally, an overview of the presented work is given in chapter 5 and opportunities for future extensions are discussed.

# CHAPTER 2
# PRELIMINARIES

This chapter will explain the preliminaries required for constructing a high level task planner for bipedal locomotion navigation with formal correctness guarantees, namely the dynamic constraints by which the high level planner needs to adhere and the details of the GR(1) fragment of LTL used for reactive synthesis are introduced.

## 2.1 Phase-space planning

First the dynamics of a reduced order Prismatic Inverted Pendulum Model (PIPM) [39] are derived. Then, the phase-space planning approach to generate the center-of-mass (CoM) trajectories of the reduced order model [40] are summarized. In addition, the locomotion keyframe state, a discretized state of our phase-space planning (PSP) approach, used as a connection between the high-level planner layer and phase-space planner layer is defined. Finally the locomotion safety criteria used to inform the high level task planning strategy are detailed.

### 2.1.1 Reduced-order Locomotion Planning

This subsection first introduces mathematical notations of our reduced-order model. As shown in Figure 2.1, the CoM position $\boldsymbol{p}_{\text{com}} = (x, y, z)^T$ is composed of the sagittal, lateral, and vertical positions. We denote the apex position as $\boldsymbol{p}_{\text{apex}} = (x_{\text{apex}}, y_{\text{apex}}, z_{\text{apex}})^T$, the foot placement as $\boldsymbol{p}_{\text{foot}} = (x_{\text{foot}}, y_{\text{foot}}, z_{\text{foot}})^T$, and $h_{\text{apex}}$ is the relative apex CoM height with respect to the stance foot height. $v_{\text{apex}}$ denotes the CoM velocity at $\boldsymbol{p}_{\text{apex}}$. $\Delta y_1$ is the lateral distance between CoM and the high-level waypoint $\boldsymbol{w}$[1] at apex. $\Delta y_2 := y_{\text{apex}} - y_{\text{foot}}$ is defined to be the lateral CoM-to-foot distance at apex. This parameter will be used to

[1]The high-level discrete representation of the robot location.

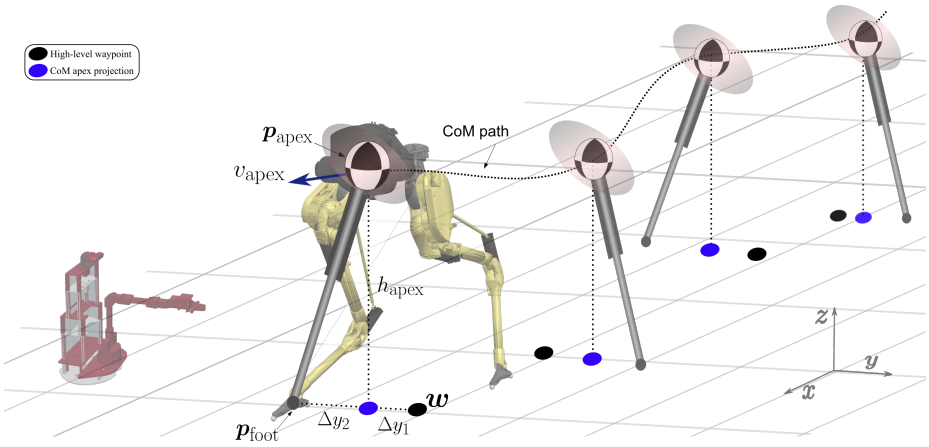



Figure 2.1: Reduced-order modeling of Cassie robot as a 3D prismatic inverted pendulum model with all of its mass concentrated on its CoM and a telescopic leg to comply to the varying CoM height. $\Delta y_1$ is the relative lateral distance between lateral CoM apex position and the high-level waypoint $\boldsymbol{w}$, and $\Delta y_2$ is the lateral distance between the CoM lateral apex position and the lateral foot placement.

determine the allowable steering angle in subsection 2.1.3.

PIPM has been widely explored in the literature [39]. Here we reiterate for completeness the derivation of the centroidal dynamics of such model. The centroidal momentum dynamics are formalized for our single contact case using moment balance along with linear force equilibrium as such

$$(\boldsymbol{p}_{\text{com}} - \boldsymbol{p}_{\text{foot}}) \times (\boldsymbol{f}_{\text{com}} + m\boldsymbol{g}) = -\boldsymbol{\tau}_{\text{com}} \tag{2.1}$$

where $\boldsymbol{\tau}_{\text{com}}$ is the angular moment of the modeled flywheel on the CoM, and $\boldsymbol{g}$ is the gravitational vector. For nominal planning we set $\boldsymbol{\tau}_{\text{com}} = 0$. Formulating the dynamics (Equation 2.1) for $q^{\text{th}}$ walking step as hybrid control system

$$\ddot{\boldsymbol{p}}_{\text{com},q} = \Phi(\boldsymbol{p}_{\text{com},q}, \boldsymbol{u}_q) = \begin{pmatrix} \omega_q^2(x - x_{\text{foot},q}) \\ \omega_q^2(y - y_{\text{foot},q}) \\ a\omega_q^2(x - x_{\text{foot},q}) \end{pmatrix} \tag{2.2}$$

where the asymptote slope $\omega = \sqrt{g/h_{\text{apex}}}$. The hybrid control input is $\boldsymbol{u}_q = (\omega_q, \boldsymbol{p}_{\text{foot},q})$,

with $\boldsymbol{p}_{\text{foot},q}$ being the hybrid input[2].

When the CoM motion is constrained within a piece-wise linear surface parameterized by $h = a(x - x_{\text{foot}}) + h_{\text{apex}}$, where $h$ denotes the CoM height from the stance foot, the reduced-order model becomes linear and an analytical solution exists. Detailed derivations are elaborated in Appendix A.

**Summary of Phase-space Planning:** In PSP, the sagittal planning takes precedence over the lateral planning. The decisions for the planning algorithm are formulated in the sagittal phase-space, like step length and CoM velocity. On the other hand, the lateral phase-space parameters are searched for to adhere to the sagittal phase-space plan. In this paper we build on our previous work on PSP [39, 9], by formulating safety guarantees for sagittal planning in order to achieve successful transition between keyframe states in case of perturbation in subsection 2.1.3. Moreover, we employ an optimization algorithm based on the lateral apex states that selects the next sagittal apex velocity that would allow the lateral dynamics to comply to high-level waypoint tracking in subsection 2.1.4.

### 2.1.2 Locomotion Keyframe

PSP uses keyframe states for non-periodic dynamic locomotion planning[39]. Our study generalizes the keyframe definition in our previous work by introducing diverse navigation actions in 3D environments.

**Definition 2.1.1** (Locomotion keyframe state)**.** *A keyframe state of our reduced-order model is defined as* $\boldsymbol{k} = (d, \Delta\theta, \Delta z_{\text{foot}}, v_{\text{apex}}, z_{\text{apex}}) \in \mathcal{K}$*, where*

- $d := x_{\text{apex},n} - x_{\text{apex},c}$ *is the walking step length*[3]*;*
- $\Delta\theta := \theta_{\text{apex},n} - \theta_{\text{apex},c}$ *is the heading angle change at two consecutive CoM apex states;*

[2]Hereafter, we will ignore the subscript q for notation simplicity. We will instead use $\cdot_c$ and $\cdot_n$ denoting the current and next walking steps, respectively

[3]while in straight walking $d$ represents the step length, this step length during steering walking is adjusted to reach the next waypoint on the new local coordinate.

- $\Delta z_{\text{foot}} := z_{\text{foot},n} - z_{\text{foot},c}$ *is the height change for successive foot placements;*
- $v_{\text{apex}}$ *is the CoM sagittal apex velocity;*
- $z_{\text{apex}}$ *is the global CoM height at apex.*

The keyframe state is composed of a high-leve (HL) action ($\boldsymbol{a}_{\text{HL}}$) and a low-level (LL) action ($\boldsymbol{a}_{\text{LL}}$). The HL action is defined as $\boldsymbol{a}_{\text{HL}} = (d, \Delta\theta, \Delta z_{\text{foot}},) \in \mathcal{A}_{\text{HL}}$, which is determined by the navigation policy to be designed in the task planner. The parameters $d$, $\Delta\theta$, and $\Delta z_{\text{foot}}$ are expressed in the Cartesian space as the high-level way-points $\boldsymbol{w}$. On the other hand, the LL action is $\boldsymbol{a}_{\text{LL}} = (v_{\text{apex}}, z_{\text{apex}}) \in \mathcal{A}_{\text{LL}}$, which is determined in the low-level phase-space planner. The keyframe parameters are sent from the high-level task a planner to the phase-space planner *online* as shown in Figure 1.2.

### 2.1.3 Locomotion Safety Criteria

As bipedal robots become increasingly integrated in dynamic workspaces, safe operation in presence of unexpected perturbation is critical. In this section we propose safe locomotion criteria and navigation guarantees based on the locomotion keyframe state, which includes both high-level action ($\boldsymbol{a}_{\text{HL}}$) and high-level action ($\boldsymbol{a}_{\text{LL}}$), thus providing safety guarantees for our integrated framework. In this subsection we present locomotion safety theorems based on PIPM introduced in subsection 2.1.1.

As a general principle of balancing safety, the sagittal CoM position should be able to cross the sagittal apex with a positive CoM velocity while the lateral CoM velocity should be able to reach the zero lateral velocity threshold at the next apex state. Ruling out the fall situations provides us upper and lower bounds of the balancing safety region.

First, we study the constraints between apex velocities of two consecutive walking steps and propose the following theorems and corollaries.

**Theorem 2.1.1.** *For safety-guaranteed straight walking, given $d$ and $\omega$, the apex velocity*

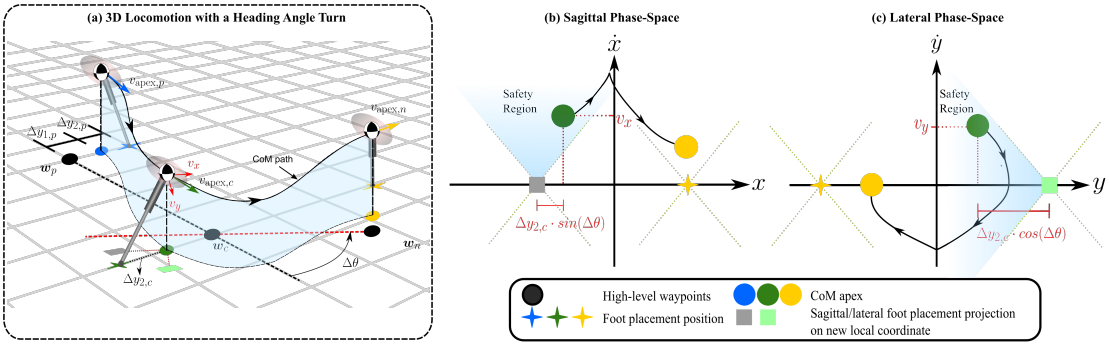



Figure 2.2: Phase-space safety region for steering walking: (a) shows three consecutive keyframes with a heading angle change ($\Delta\theta$) between the current keyframe and the next keyframe. The CoM trajectory and its projection on the sagittal-lateral space is represented by the blue surface. The direction change introduces a new local coordinate, where the dashed black line is the sagittal coordinate before the turn, and the red dashed line is the sagittal coordinate after the turn. Subfigures (b) and (c) show the sagittal and lateral phase-space plots respectively, both satisfying the safety criteria proposed in Theorem 2.1.2. The subscripts $p$, $c$ and $n$ denote the previous, current, and next walking steps, respectively.

*for two consecutive walking steps ought to satisfy the following velocity constraint:*

$$-\omega^2 d^2 \leq \underbrace{v_{\text{apex},n}^2 - v_{\text{apex,c}}^2}_{\substack{\textit{apex velocity square difference} \\ \textit{for two consecutive steps}}} \leq \omega^2 d^2 \tag{2.3}$$

*where* $d^2 = (x_{\text{apex},n} - x_{\text{apex},c})(x_{\text{apex},c} + x_{\text{apex},n} - 2x_{\text{foot},c})$. *Notably,* $d$ *is equal to the step length in Def. 2.1.1, i.e.,* $d = x_{\text{apex},n} - x_{\text{apex},c}$*, during a straight walking where* $x_{\text{apex},c} = x_{\text{foot},c}$. *The proof of this criterion can be seen in Appendix B.*

Another consideration for safety is to limit the maximum allowable velocity of the CoM. Since the maximum velocity occurs at the foot switching juncture, we explicitly enforce an upper velocity bound to this switching velocity $v_{\text{switch}}$ to avoid jerky motions magnified by the ground impact dynamics from the real system. Through the analytical solution of the reduced-order model in Appendix A, we solve for the sagittal CoM velocity at the switching juncture $v_{\text{switch}} = \Psi(v_{\text{apex},c}, v_{\text{apex},n}, d)$. Therefore, we set an upper bound on $v_{\text{switch}}$ to limit the maximum allowed CoM velocity, i.e., $v_{\text{switch}} \leq v_{\max}$.

Similar to Theorem 2.1.1, $v_{\text{switch}}$ provides a nonlinear relationship between sagittal

apex velocities for two consecutive apex states. Combining the boundary conditions in Theorem 2.1.1 and the limit of $v_{\text{switch}}$ allows us to quantify the viable region of $v_{\text{apex},n}$ given $v_{\text{apex},c}$, $d$, and $\omega$ to achieve locomotion safety.

The steering case however, requires a more restrictive criterion. A fall will occur when $v_{\text{apex},c}$ is out of a safety range such that either the lateral CoM velocity cannot reach zero at the next apex state or the sagittal CoM can not climb over the next sagittal CoM apex.

**Theorem 2.1.2.** *For safety-guaranteed steering walking, the current sagittal CoM apex velocity $v_{\text{apex},c}$ in the original local coordinate should be bounded by*

$$\Delta y_{2,c} \cdot \omega \cdot \tan \Delta\theta \leq v_{\text{apex},c} \leq \frac{\Delta y_{2,c} \cdot \omega}{\tan \Delta\theta} \tag{2.4}$$

Figure 2.2 shows a steering walking trajectory and phase-space plot that satisfy Theorem 2.1.2. Namely, the CoM in the sagittal and lateral phase-space should not cross the asymptote line of the shaded safety region as seen in Figure 2.2. This criterion is specific to steering walking, as the heading change ($\Delta\theta$) introduces a new local frame, which yields the current state $\boldsymbol{\xi}_c$ to no longer be an apex state in the new coordinate. As such, it has non-apex sagittal and lateral components, i.e., $v_{y,c} \neq 0$, and $x_{\text{apex,c}} \neq x_{\text{foot,c}}$.

**Corollary 1.** *For steering walking in Theorem 2.1.2, given $d$, $\Delta\theta$, $\Delta y_{2,c}$ and $\omega$, two consecutive apex velocities ought to satisfy the following velocity constraint:*

$$-\omega^2 d^2 \leq v_{\text{apex},n}^2 - (v_{\text{apex},c} \cos \Delta\theta)^2 \leq \omega^2 d_+^2 \tag{2.5}$$

*where $d_+^2 = d^2 + 2\Delta y_{2,c} d \sin \Delta\theta$.*

**Corollary 2.** *For steering walking in Theorem 2.1.2, similarly, given $d$, $\Delta\theta$, $\Delta y_{2,c}$, and $\omega$, two consecutive apex velocities ought to satisfy the following velocity constraints,*

$$-\omega^2 d^2 \leq v_{\text{apex},n}^2 - (v_{\text{apex},c} \cos \Delta\theta)^2 \leq \omega^2 d_-^2 \tag{2.6}$$

*where* $d_-^2 = d^2 - 2\Delta y_{2,c} d \sin \Delta\theta$. *Note that, parameters* $v_{\text{apex},n}$, $d$, *and* $\Delta\theta$ *in Equation 2.3-Equation 2.6 are the keyframe states.*

The aforementioned safety theorems provide quantification bounds on next keyframe selection that leads to viable transitions under nominal conditions. These theorems are used to generate locommotion keyframe transitions that ensure locomotion safety based on controllable regions and sequential composition to provide guarantees on the progression of the system states $\boldsymbol{\xi}$ adhering to Theorems 2.1.1- 2.1.2 under bounded disturbance.

**Definition 2.1.2** (Locomotion safety)**.** *Safety for a locomotion process is defined as a formally-guaranteed successful transition between consecutive locomotion keyframe states* $\boldsymbol{k} \in \mathcal{K}$ *while the robot maintains its balance, i.e., avoids a fall.*

### 2.1.4 Keyframe Decision Maker for Waypoint Tracking

In the previous section, we defined safety theorems that guarantees locomotion safety as defined in Def. 2.1.2. Now we turn our focus on another consideration for safe task completion by ensuring tracking of the high-level waypoints. The lateral phase-space plan is determined based on the sagittal phase-space plan, as the lateral dynamics transition between two apex states needs to obey a consistent timing as that of the sagittal dynamics transition. Therefore, the lateral dynamics depend on sagittal apex velocities and sagittal step length. In previous work [39], the lateral foot placement is solved through a Newton-Raphson search method, such that the lateral CoM velocity is bounded and equal to zero at the next lateral CoM apex. While our previous method achieved stable walking and turning, it lacks the consideration of high-level navigation task accomplished through tracking of the high-level waypoints. Therefore, it provides no guarantee that the lateral CoM motion will be able to track the high-level waypoint. In my work [9] we proposed a heuristic based policy that restricted the allowable keyframe transitions to achieve waypoint tracking for specific locomotion plans. In [41], we extend our previous work by adding an algorithm

that manipulates the sagittal phase-space plan to allow high-level waypoint tracking. First let's define the viable ranges for $\Delta y_2$ and $\Delta y_1$.

**Definition 2.1.3** (Viable range for lateral apex CoM-to-waypoint distance)**.** $\mathcal{R}_{\Delta y_1} := \{\Delta y_1 | \Delta y_1 + \Delta y_2 \leq b_{\text{safety}}\}$*, where* $b_{\text{safety}}$ *denotes the safety boundary around the waypoint.*

**Definition 2.1.4** (Viable range for lateral apex CoM-to-foot distance)**.** *Given the safety criterion for steering walking defined in Theorem 2.1.2, the viable range for lateral CoM-to-foot distance at apex is defined as* $\mathcal{R}_{\Delta y_2} := \{\Delta y_2 | v_{\text{apex,max}} \cdot \tan \Delta\theta/\omega \leq \Delta y_2 \leq (v_{\text{apex,min}})/(\omega \cdot \tan \Delta\theta)\}$.

$\mathcal{R}_{\Delta y_1}$ is defined in such a manner to limit the lateral deviation of the robot's CoM and foot location from the high-level waypoint, in order to avoid collisions with obstacles. Given Defs. 2.1.3-2.1.4, we can track the high-level waypoint as follows.

**Proposition 2.1.3.** *Viable lateral tracking of the high-level waypoint is achieved only if (i)* $\Delta y_2$ *and* $\Delta y_1$ *are bounded within their respective viable ranges, i.e.,* $\Delta y_1 \in \mathcal{R}_{\Delta y_1}$ *and* $\Delta y_2 \in \mathcal{R}_{\Delta y_2}$*, and (ii)* $\textit{sign}(\Delta y_{1,k+1} + \Delta y_{2,k+1}) = -\textit{sign}(\Delta y_{1,k} + \Delta y_{2,k})$*, where* $k$ *indices the* $k^{\text{th}}$ *walking step.*

Proposition 2.1.3 requires that the sign of the sum of $\Delta y_1$ and $\Delta y_2$ alternates between consecutive keyframes in order to guarantee that the high-level waypoint and lateral CoM are contained between the lateral foot placement for One Walking Step (OWS).

The analytical solutions of $\Delta y_{1,n}$ and $\Delta y_{2,n}$ are highly nonlinear and depend on multiple parameters including the step length $d$, heading angle change $\Delta\theta$, current and next apex velocities $v_{\text{apex},c}$, $v_{\text{apex},n}$ and the current lateral state of the system $\Delta y_{1,c}$ and $\Delta y_{2,c}$. However, $(d, \Delta\theta) \in \boldsymbol{a}_{\text{HL}}$ are determined by the navigation policy designed in the high-level task planner, and $v_{\text{apex},c}$, $\Delta y_{1,c}$ and $\Delta y_{2,c}$ are fixed from the previous step. Therefore, we manipulate $v_{\text{apex},n}$ to adjust the sagittal phase-space plan and subsequently the lateral phase-space plan. To this end, we sample a set of equidistant values $v_{\text{apex},n} \in [v_{\text{apex,min}}, v_{\text{apex,max}}]$ and

calculate a cost $\lambda$, which penalizes deviation of $\Delta y_{1,n}$ and $\Delta y_{2,n}$ from their respective desired values $\Delta y_{1,d}$ and $\Delta y_{2,d}$ meeting Proposition 2.1.3 and selected by the designer[4]. After the sampling, we set $v_{\text{apex},n}$ to the optimal next apex velocity $v_{\text{apex,opt}}$ that results in the minimum cost. This procedure is presented in algorithm 1.

**Algorithm 1:** Optimal Next Apex Velocity for Waypoint Tracking

**Input:** $d$, $v_{\text{apex},c}$, $\Delta y_{1,c}$, $\Delta y_{2,c}$, and a velocity sampling increment $v_{\text{inc}}$;
**Set:** $v_{\text{apex},n} = v_{\text{apex,min}}$, cost $\lambda \leftarrow \infty$, $\Delta y_{1,d} \in \mathcal{R}_{\Delta y_1}$ and $\Delta y_{2,d} \in \mathcal{R}_{\Delta y_2}$, and cost weights $c_1$ and $c_2$;
**while** $v_{\text{apex},n} \leq v_{\text{apex,max}}$ **do**
  $t_{\text{FHWS}}$, $t_{\text{SHWS}} \leftarrow$ sagittal PSP with ($d$, $v_{\text{apex},c}$, $v_{\text{apex},n}$);
  $\Delta y_{1,n}$, $\Delta y_{2,n} \leftarrow$ Newton-Raphson Search [39];
  $\lambda_{\text{new}} = c_1(\Delta y_{1,d} - \Delta y_{1,n}) + c_2(\Delta y_{2,d} - \Delta y_{2,n})$;
  **if** $\lambda_{\text{new}} < \lambda$ **then**
    $\lambda \leftarrow \lambda_{\text{new}}$;
    $v_{\text{apex,opt}} \leftarrow v_{\text{apex},n}$;
  **end**
  $v_{\text{apex},n} \leftarrow v_{\text{apex},n} + v_{\text{inc}}$;
**end**
**Output:** $v_{\text{apex},n} = v_{\text{apex,opt}}$;

## 2.2 Reactive Synthesis

Straightforward robotic tasks in simple environments may be addressed by a manually designed state machine that models a set of robot-environment state combinations and encodes the correct high-level actions. However, as the task complexity, the number of inputs, or the number of decision variables increases, manually modeling all the combinations is no longer feasible to guarantee correct execution of the system. Reactive synthesis methods have been studied for automatic high-dimensional state machine generation from high-level symbolic specifications [42]. Specifically LTL allows system constraints and desired properties to be expressed in a mathematically precise yet intuitive manner. Once system characteristics and desired behaviors have been captured in LTL specifications, off-the-shelf

[4]The middle value of the viable range as the desired value in the implementation.

solvers can be used to synthesize a state machine that is able to output a correct high-level action for any modeled environment behavior guaranteed to meet safety and task specifications. LTL is a systematic approach to design a robot task planner and makes the resulting controller modular, allowing the planner to be adapted to changing task requirements with minimal modification to the controller specification design. The use of symbolic language further facilitates the implementation of a tailored LTL planning structure as it is human-interpretable [43], allowing the task planner do be understood by non-technical specialists. This customizability lowers the cost of deploying a robotic sorting solution in intricate new scenarios, lowering the barriers of entry to autonomous robotic solutions for complex tasks.

To formally guarantee goal tasks are achieved *infinitely often* while the safety specifications are met, we use GR(1) [44], a fragment of 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 [45, 46, 47]. GR(1) allows us to design temporal logic formulas ($\varphi$) with atomic propositions (AP) that can either be $\mathsf{True}$ ($\varphi \vee \neg\varphi$) or $\mathsf{False}$ ($\neg\mathsf{True}$). 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" ($\Box$).

GR(1) mission specifications are of the form:

$$(\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e) \Rightarrow (\varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s) \tag{2.7}$$

The specification is an implication between a set of assumptions and a set of guarantees. On the left hand side of Equation 2.7 we have the environment initialization assumption ($\varphi_i^e$), the environment transition (safety) assumption ($\varphi_t^e$), and the environment liveness assumption ($\varphi_g^e$). On the right hand side we have the system initialization guarantees ($\varphi_i^s$), the system transition (safety) guarantees ($\varphi_t^s$), and the system liveness guarantees ($\varphi_g^s$). Initial conditions are free from temporal operators and simply dictate the initial state of the system and environment. Safety conditions are to always be satisfied and dictate how the

APs of the system may evolve between the current and next time step of the execution. Finally the liveness conditions are to be satisfied infinitely often and are of the form $\square\lozenge\psi$. Known properties of the environment in which a system is intended to operate are encoded in the assumptions, while desired behavior the system guaranteed to satisfy are captured in the guarantees. If a mission specification is realizable the system ensures the guarantees are satisfied *only* when the environmnet assumptions hold true. Further details of GR(1) can be found in [12]. Our implementation uses the SLUGS reactive synthesis tool [48] to design specifications with APs and natural numbers, which are automatically converted to ones using only APs.

# CHAPTER 3
# TASK PLANNING VIA BELIEF ABSTRACTION

This section will employ the locomotion keyframe properties from chapter 2 for the high-level task specification design. The goal of our temporal-logic-based task planner is to achieve safe locomotion navigation in a partially observable environment with dynamic obstacles as defined below

**Definition 3.0.1** (Navigation Safety)**.** *Navigation safety is defined as dynamic maneuvering over uneven terrain without falling while avoiding collisions with static and dynamic obstacles.*

To achieve safe navigation, the task planner evaluates observed environmental events at each walking step and commands a safe action set to the middle-level phase-space planner as shown in Figure 1.2 while guaranteeing goal positions to be visited *in order* and *infinitely often*. In particular, we study a pick-up and drop-off task while guaranteeing static and dynamic obstacle collision avoidance.

We design our task planner using formal synthesis methods to ensure locomotion actions guarantee navigation safety and task completion. The discrete abstraction granularity required to plan walking actions for each keyframe is too fine to synthesize plans for large environment navigation. Therefore, we have split the task planner into two layers: A high-level navigation planner that plays a navigation and collision avoidance game against the environment on a global coarse discrete abstraction, and an action planner that plays a local game on a fine abstraction of the local environment (i.e., one coarse cell). The action planner generates action sets at each keyframe to progress through the local environment to achieve the desired coarse-cell transition in the navigation game after multiple walking steps.

### 3.1 Navigation Planner Design

A top-down projection of the navigation environment is discretized into a coarse two-dimensional grid as shown in Figure 3.8. Each time the robot enters a new cell, the navigation planner evaluates the robot's discrete location ($l_{r,c} \in \mathcal{L}_{r,c}$) and heading ($h_{r,c} \in \mathcal{H}_{r,c}$) on the coarse grid, as well as the dynamic obstacle's location ($l_o \in \mathcal{L}_o$), and determines a desired navigation action ($n_a \in \mathcal{N}_a$). The planner can choose for the robot to stop, or to transition to any reachable safe adjacent cell. $\mathcal{L}_{r,c}$ and $\mathcal{L}_o$ denote sets of all coarse cells the robot and dynamic obstacle can occupy, while $\mathcal{H}_{r,c}$ represents the four cardinal directions in which the robot can travel on the coarse abstraction. The dynamic obstacle moves under the following assumptions: (a) it will not attempt to collide with the robot when the robot is standing still, (b) it's maximum speed only allows it to transition to an adjacent coarse cell during one turn of the navigation game, and (c) it will eventually move out of the way to allow the robot to pass. Assumption (c) prevents a deadlock [49]. Static obstacle locations are encoded as safety specifications. Given these assumptions, the task planner in section 3.4 will guarantee that the walking robot can achieve a specific navigation goal while preventing collisions with static and dynamic obstacles.

### 3.2 Action Planner Design

The local environment, i.e., one coarse cell, is further abstracted into a fine discretization. At each walking step, the action planner evaluates the robot state in action planning environment ($\boldsymbol{e}_{\mathrm{HL}}$)[1] consisting of the discrete waypoint location ($l_{r,f} \in \mathcal{L}_{r,f}$) and heading ($h_{r,f} \in \mathcal{H}_{r,f}$) on the fine grid, as well as the robots current stance foot index ($i_{\mathrm{st}}$), and determines an appropriate action set ($\boldsymbol{a}_{\mathrm{HL}}$) defined in Def. 2.1.1. The action planner generates a sequence of locomotion actions guaranteeing that the robot eventually transitions to the next desired coarse cell while ensuring all action sets are safe and achievable based on $\boldsymbol{e}_{\mathrm{HL}}$

[1] We use the symbol $\boldsymbol{e}_{\mathrm{HL}}$ to represent the robot state, since this symbol represents the second player in the game, i.e., the environment player.

and $\boldsymbol{a}_{\mathrm{HL}}$. Note that, the fine abstraction also models the terrain height for each fine-level cell, allowing the action planner to choose the correct step height $\Delta z_{\mathrm{foot}}$ for each keyframe transition.

During locomotion, the nominal robot state transitions are deterministically modeled within the action planner based on the current game state and system action, however, the nominal transition is not guaranteed. To account for this, we model additional necessary nondeterministic transitions to handle the following cases:

- The robot location is far enough from the centroid of a cell that the same geometric cell transition puts the robot in a different cell at the next step than expected. This occurs because infinite number of continuous locations are captured in one discrete cell.

- Not all the robot states can be captured in the discrete abstraction, such as the robot CoM velocity, which, however, may still affect transitions.

- The robot may be perturbed externally while walking, altering the foot location at the next walking step.

We have encoded nondeterministic transitions, and associated transition flags ($t_{nd}$), to capture these cases into action planner's environment assumptions. This flag variable $t_{nd}$ is encoded as a special automaton state that will be used to replan the foot location of the next walking step. An example of addressing a sagittal perturbation will be shown in Figure 3.7 (c).

An example of modeled nondeterministic transitions can be seen in Figure 3.1. The CoM trajectory sometimes imperfectly tracks the waypoints due to accumulated differences in the continuous keyframe state represented by the same discrete state $\boldsymbol{e}_{\mathrm{HL}}$. The reduced-order motion planner identifies when the waypoint needs to be shifted from the lateral case and informs the action planner, which verifies the updated waypoint is allowed by the non-deterministic transition model and continues planning from the new waypoint.

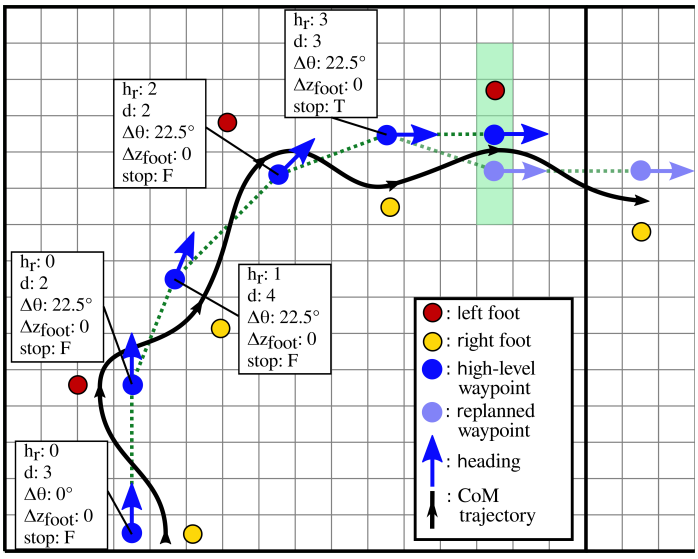



Figure 3.1: Illustration of fine-level steering walking within one coarse cell. Discrete actions are planned at each keyframe allowing the robot to traverse the fine grid toward the next coarse cell. The waypoint transitions nondeterministically following the turn. A set of locomotion keyframe decisions are also annotated.

### 3.3 Capturing Low-level Constrains in the High-level Planner

To ensure the action planner only commands safe and feasible actions, we must take into account the underlying *Locomotion Safety*. This is achieved by capturing low-level constraints in the high-level planner specifications. Action planner state transition limitations based on straight walking step length constraints in Theorem 2.1.1, and kinematic constraints from the Cassie leg, are directly encoded in the action planner specifications. *Locomotion safety* is guaranteed when the combination of apex velocity, heading angle change, and foot placement meets Theorems 2.1.1- 2.1.2. These constraints are not able to be directly captured as the action planner does not reason about CoM velocity and the dynamic equations of motion can not be encoded in symbolic specifications. Instead, they are captured by generating a library of permissible turning sequences based on discrete robot states that are known to meet the above constraints (see Table 3.2). For example, given $\omega = 3.15$ rad/s, $\Delta y_{1,c} = 0.14$ m (equals to $\Delta y_{1,d}$ in algorithm 1), and an allowable $v_{\text{apex}}$ range $[0.2, 0.7]$ m/s, Theorem 2.1.2 results in $\Delta\theta \leq 24.40°$. Any turning angle larger than this value will results in a high-level action that is not executable by the middle-level

motion planner. Thus we choose $\Delta\theta = 22.5^\circ$ such that we can complete a $90^\circ$ turn in $4$ consecutive walking steps. A safe turning sequence can be seen in Figure 3.1.

To ensure that collision avoidance in the abstract game translates to collision-free locomotion in the continuous domain, we guarantee the location $l_{r,f}$ stays far enough away from any obstacles. algorithm 1 ensures that the distance between $l_{r,f}$ and the robot's desired foot placement does not exceed $b_{\text{safety}}$ as detailed in subsection 2.1.4. The action planner guarantees $l_{r,f}$ is never in a cell that is less than a distance $b_{\text{safety}}$ away from the neighboring coarse cell that may contain static or dynamic obstacles via safety specifications. The planner guarantees this distance even after non-deterministic sagittal and lateral transitions, ensuring collision avoidance.

### 3.4 Task Planner Synthesis

To formally guarantee that the goal locations are reached *infinitely often* while the safety specifications are met, we use GR(1), a fragment of LTL.

A navigation game structure is proposed by including robot actions in the tuple $\mathcal{G} := (\mathcal{S}, s^{\text{init}}, \mathcal{T}_N)$ with

- $\mathcal{S} = \mathcal{L}_{r,c} \times \mathcal{L}_o \times \mathcal{H}_{r,c} \times \mathcal{N}_a$ is the augmented state;
- $s^{\text{init}} = (l_{r,c}^{\text{init}}, l_o^{\text{init}}, h_{r,c}^{\text{init}}, n_a^{\text{init}})$ is the initial state;
- $\mathcal{T}_N \subseteq \mathcal{S} \times \mathcal{S}$ is a transition relation describing the possible moves of the robot and the obstacle.

To synthesize the transition system $\mathcal{T}_N$, we define the rules for the possible successor state locations which will be further expressed in the form of LTL specifications $\psi$. The successor location of the robot is based on its current state and action $succ_r(l_{r,c}, h_{r,c}, n_a) = \{l'_{r,c} \in \mathcal{L}_{r,c} | \exists l'_o, h'_{r,c}((l_{r,c}, l_o, h_{r,c}, n_a), (l'_{r,c}, l'_o, h'_{r,c}, n'_a)) \in \mathcal{T}_N\}$. We define the set of possible successor robot actions at the next step as $succ_{n_a}(n_a, l_{r,c}, l'_{r,c}, l_o, l'_o, h_{r,c}, h'_{r,c}) = \{n'_a \in \mathcal{N}_a | ((l_{r,c}, l_o, h_{r,c}, n_a), (l'_{r,c}, l'_o, h'_{r,c}, n'_a)) \in \mathcal{T}_N\}$. We define the set of successor locations of

the obstacle. $succ_o(l_{r,c}, l_o, n_a) = \{l'_o \in \mathcal{L}_o | \exists l'_{r,c}, h'_{r,c}.((l_{r,c}, l_o, h_{r,c}, n_a), (l'_{r,c}, l'_o, h'_{r,c}, n'_a)) \in \mathcal{T}_N\}$. Later we will use a belief abstraction inspired by [33] to solve our synthesis in a partially observable environment.

The task planner models the robot and environment interplay as a two-player game. The robot action is Player 1 while the possibly adversarial obstacle is Player 2. The synthesized strategy guarantees that the robot will always win the game by solving the following reactive problem.

**Reactive synthesis problem:** Given a transition system $\mathcal{T}_N$ and linear temporal logic specifications $\psi$, synthesize a winning strategy for the robot such that only correct decisions are generated in the sense that the executions satisfy $\psi$.

The action planner is synthesized using the same game structure as the navigation planner, with possible states and actions corresponding to section 3.2. Nondeterministic robot location transitions are captured in the robot successor function $succ_{r,f}(l_{r,f}, h_{r,f}, \boldsymbol{a}_{\mathrm{HL}}) = \{l'_{r,f} \in \mathcal{L}_{r,f}, h'_{r,f} \in \mathcal{H}_{r,f} | ((l_{r,f}, h_{r,f}, \boldsymbol{a}_{\mathrm{HL}}), (l'_{r,f}, h'_{r,f}, \boldsymbol{a}'_{\mathrm{HL}})) \in \mathcal{T}_A\}$, where $\mathcal{T}_A$ is the transition relation in the action planner. Compared to the transition relation $\mathcal{T}_N$, $\mathcal{T}_A$ does not have the obstacle location $l_o$ but includes locomotion actions $\boldsymbol{a}_{\mathrm{HL}}$. Given the current robot state and action, $succ_{r,f}$ provides a set of possible locations at the next turn in the game. Obstacle avoidance is taken care of in the navigation game the obstacle location $\mathcal{L}_o$ and successor function $succ_o$ are not needed for action planner synthesis. Since reactive synthesis is used for both navigation and action planners, and the action planner guarantees the robot transition in the navigation game, the correctness of this hierarchical task planner is guaranteed.

### 3.4.1 Belief Space Planning in Partial Observable Environment

The navigation planner above synthesizes a safe game strategy that is always winning but only in a fully observable environment. We relax this assumption by assigning the robot a visible range only within which the robot can accurately identify a dynamic obstacle's

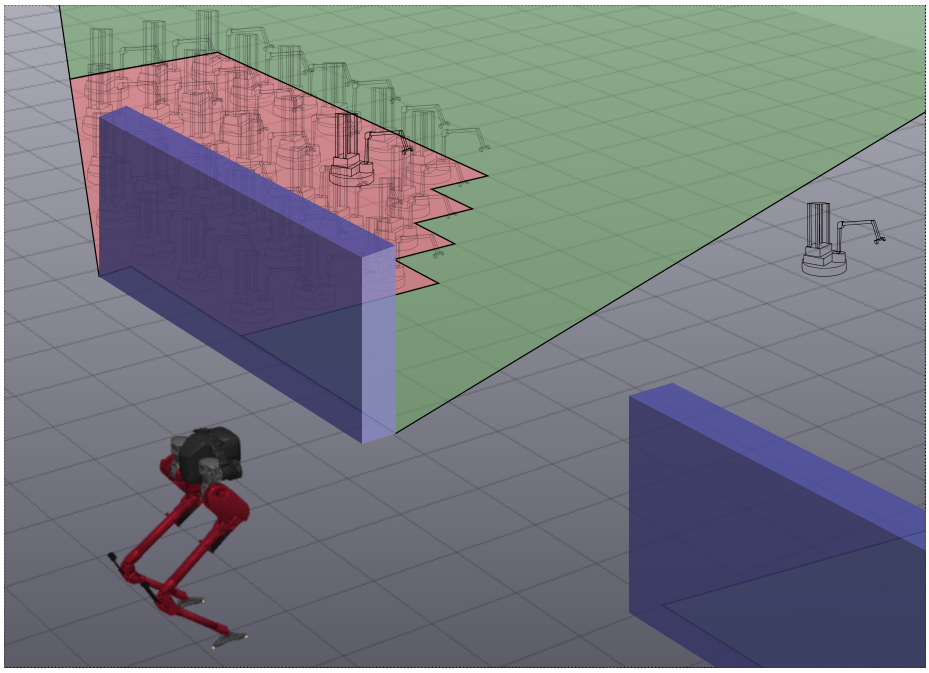
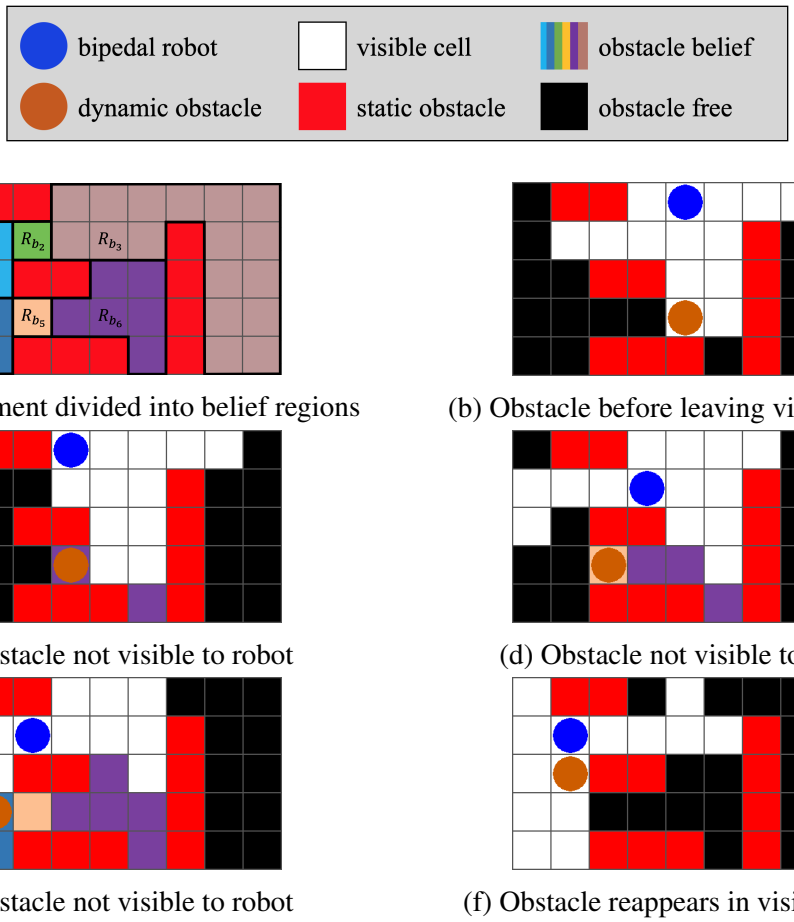


Figure 3.2: Conceptual Visualization of biped tracking the believed location of a nonv-visible mobile robots.

location. To reason about where an out-of-sight obstacle could be, we devise an abstract belief set construction method based on the work in [33]. Using this belief abstraction, we explicitly track the possible discrete locations of a dynamic obstacle, rather than assuming it could be in any non-visible cell. The abstraction is designed by partitioning regions of the environment into sets of discrete belief regions ($R_b$) and constructing a powerset of these regions ($\mathcal{P}(R_b)$). We choose smaller partitions around static obstacles that may block the robot's view as this allows the planner to guarantee collision-free navigation for a longer horizon like the scenario depicted in Figure 3.3. It is also possible to start with a coarse partition and automatically refine the abstraction using counter example guided abstraction refinement (CEGAR) [50] until a guaranteed winning strategy can be synthesized. We index each set in $\mathcal{P}(R_b)$ to represent a belief state $b_o \in \mathcal{B}_o$ that captures non-visible regions potentially with a dynamic obstacle.

The fully observable navigation game structure is modified to generate a partially observable belief-based navigation game with an updated state $\mathcal{S}_{\text{belief}}$ and transition system $\mathcal{T}_{\text{belief}}$ In addition to the obstacle location $l_o \in \mathcal{L}_o$, $\mathcal{S}_{\text{belief}}$ captures the robot's be-




(a) Environment divided into belief regions

(b) Obstacle before leaving visible range

(c) Obstacle not visible to robot

(d) Obstacle not visible to robot

(e) Obstacle not visible to robot

(f) Obstacle reappears in visible range

Figure 3.3: Simulation showing how the navigation planner's belief evolves when the dynamic obstacle leaves the visible range for several turns. 6 colored belief regions are shown, as well as the robot (blue circle), the dynamic obstacle (orange circle) and static obstacles (red cells). Black cells represent non-visible cells believed to be obstacle free while white cells are visible. The planner believes the obstacle could be in any colored cell depicted, and can therefore reason where the obstacle could and could not reappear, allowing the planner to determine which navigation actions are safe.

lief of the obstacle $b_o \in \mathcal{B}_o$. A visibility function $vis : \mathcal{S}_{\text{belief}} \to \mathbb{B}$ is added such that it maps the state $(l_{r,c}, l_o)$ to the Boolean as True if and only if $l_o$ is a location in the visible range of $l_{r,c}$. We do not need to modify $succ_{n_a}$ since the dynamic obstacle only affects the possible one-step robot action if it is in the visible range. $succ_r$ also remains the same as the relationship between the robot's actions and its state is not changed by the belief. The set of possible successor beliefs of the obstacle location, $b'_o$, is defined as $succ_{b_o} = \{b'_o \in \mathcal{B}_o | ((l_{r,c}, b_o, l_o), (l'_{r,c}, b'_o, l_o)) \in \mathcal{T}_{\text{belief}}\}$ where $b'_o$ indexes $\emptyset$ when $vis(l_{r,c}, l'_o) =$ True and $b'_o$ indexes a nonempty set in $\mathcal{P}(R_b)$ when $vis(l_{r,c}, l'_o) =$ False.

There are four classes of belief transitions, shown in Figure 3.3, that need to be defined for accurate and meaningful belief tracking:

- Visible to visible: as in the fully observable case, the obstacle may transition to any adjacent visible cell.
- Visible to belief: if the obstacle is not visible after its turn, the belief state represents the set of regions containing non-visible cells adjacent to the obstacle's previous location.
- Belief to belief: during the current turn the obstacle could be in any non-visible (or newly visible[2]) cell represented by the current belief, at the next turn the obstacle can move to any adjacent cell, therefore the next belief state represents the regions encompassing all adjacent cells captured by the current belief state
- Belief to visible: similar to the previous case the obstacle may be in any non-visible or newly visible cell represented by the planner's belief, and may move to any adjacent cell, which defines the visible cells it could appear in at the next time step.

This method of belief tracking guarantees that all real transitions the obstacle can make during its turn are captured in the planner's belief. When the obstacle can enter cells in a new belief region, the planner believes it could be anywhere in that region at the next step, therefore the belief is an over-approximation of possible obstacle locations. We guarantee that the obstacle is within the regions captured by the belief state, therefore we can guarantee that the obstacle can only appear in a visible cell when there is a modeled transition from the current belief state to that cell. This allows us to guarantee that with some beliefs, the obstacle can not enter the visible range in front of the robot. Since both the action planner and the allowable navigation actions remain the same for the partially observable game,

[2]Due to the turn based nature of the game, an obstacle may be in the robot's visible range after the robot makes a move, but the obstacle may move from this newly visible cell before the robot reevaluates its new visible range.

the game still incorporates the low-level safety constraints using the same specifications, but allows for a larger set of navigation options than would be possible without explicitly tracking the belief of the dynamic obstacle's location.

### 3.5 Belief Tracking of Multiple Obstacles

Our task planning framework is extensible to environments with multiple dynamic obstacles. It is possible to directly add any number of additional obstacles and their associated beliefs to the navigation planning game, however, the synthesis has polynomial time complexity. To improve computational tractability, we merge all non-visible obstacles' believed states into one combined belief region. Reasoning about a combined belief region still allows the planner to guarantee collision-free navigation without the complexity of tracking each obstacle individually.

To model a combined belief state we must separate the obstacles' state from it's belief. Each obstacle's state is either a visible cell on the grid, or and index representing the obstacle is not visible ($l_{o,i,c} \in L_{o,i,c} | L_{o,i,c} = \mathcal{L}_o + \mathcal{I}_{nv}$). The joint belief state consists of the powerset of belief regions, including the empty set when all obstacles are visible. ($b_{oj} \in \mathcal{B} | \mathcal{B} = \mathcal{P}(R_b)$).

We generate a new multi-obstacle game structure $\mathcal{G}_{\text{combined}-\text{belief}} :=$ $(\mathcal{S}_{\text{belief}}, s^{\text{init}}_{\text{belief}}, \mathcal{T}_{\text{belief}}, vis)$ with

- $\mathcal{S}_{\text{belief}} = \mathcal{L}_{r,c} \times \mathcal{L}_{o,i,c} \times \mathcal{B}_o \times \mathcal{H}_{r,c} \times \mathcal{N}_a$;
- $s^{\text{init}}_{\text{belief}} = (l^{\text{init}}_{r,c}, l^{\text{init}}_{o,i,c}, \{b^{\text{init}}_o\}, h^{\text{init}}_{r,c}, n^{\text{init}}_a)$ is the initial location of the obstacle known a priori;
- $\mathcal{T}_{\text{belief}} \subseteq \mathcal{S}_{\text{belief}} \times \mathcal{S}_{\text{belief}}$ are possible transitions where $((l_{r,c}, l_{o,i,c}, b_o, h_{r,c}, n_a),$ $(l'_{r,c}, l'_{o,i,c}, b'_o, h'_{r,c}, n'_a)) \in \mathcal{T}_{\text{belief}}$;
- $vis : \mathcal{S}_{\text{belief}} \to \mathbb{B}$ is a visibility function that maps the state $(l_{r,c}, l_{o,i})$ to the boolean as $\mathsf{True}$ iff $l_{o,i}$ is a real location in the visible range of $l_{r,c}$.

This game requires new specifications that govern $succ_{o,i}(l_{r,c}, l_{o,i,c}, b)$ and $succ_b(l_{r,c}, l_{o,i,c}, b)$, the allowable successor obstacle state and joint belief state, all other successor functions remain the same. Even though the belief can represent multiple obstacles, the possible belief-to-belief transitions are the same as when the belief state represents a single obstacle. The key specifications to be changed are those governing $succ_{b_o}$ when an obstacle enters or exits the visible range. These changes can be made in the specifications defining the successor belief state $succ_{b_o}$.

## 3.6 Results

This result section evaluates the performance of (i) the high-level task planner by assessing its task completion, collision avoidance, and safe action execution; (ii) the middle-level motion planner by employing our designed keyframe decision maker to choose proper keyframe states and generating safe locomotion trajectories; and (iii) recoverability against perturbations using the reachability-based synthesized controller; The results are simulated using the Drake toolbox [51], and the open-source code can be found here https://github.com/GTLIDAR/safe-nav-locomotion.git. A video of the simulations is viewable here https://youtu.be/w-SrjuUbO78.

### 3.6.1 LTL Task Planning Implementation

The task planner is evaluated in an environment with multiple static and dynamic obstacles, and two rooms with different ground heights connected by a set of stairs as seen in Figure 3.4. To generate the navigation planning abstraction, the environment is discretized into a $10 \times 5$ coarse grid, with a $2.7 \times 2.7$ m$^2$ cell size. $\mathcal{L}_{r,c}$ is the set of all accessible discrete cells, $\mathcal{H}_{r,c}$ is the set of cardinal directions, and $\mathcal{N}_a$ is a set of navigation actions in those cardinal directions (N, E, S, W). Each coarse cell is further discretized into a finer $26 \times 26$ grid for local action planning. We model the possible actions as step length $d \in \{\mathsf{small1}, \mathsf{small2}, \mathsf{medium1}, \mathsf{medium2}, \mathsf{large1}, \mathsf{large2}\}$, heading change

$\Delta\theta \in \{\mathsf{left}, \mathsf{none}, \mathsf{right}\}$), and step height $\Delta z_{\text{foot}} \in \{z_{\text{down2}}, z_{\text{down1}}, z_{\text{flat}}, z_{\text{up1}}, z_{\text{up2}}\}$. The possible heading changes $\Delta\theta \in \{-22.5^\circ, 0^\circ, 22.5^\circ\}$, are constrained by the minimum number of steps needed to make a $90^\circ$ turn, and the maximum allowable heading angle change that results in viable keyframe transitions as defined in Theorem 2.1.2. We choose $\Delta\theta = \pm 22.5^\circ$ so that a $90^\circ$ turn can be completed in four steps as shown in Figure 3.1. Completing the turn in fewer steps is not feasible as it would overly constrain $v_{\text{apex}}$, as can be seen in Figure 2.2(b). Due to the allowable heading change of $\pm 22.5^\circ$, $\mathcal{H}_{r,f}$ contains a discrete representation of the 16 possible headings the robot could have.

A set of specifications is designed to describe the allowable successor locations and actions in the transition system. Here, we only show a few specifications as examples:

$$\square\big((h_{r,f} = \mathcal{H}_{r,c} \wedge ((i_{\text{st}} = \mathsf{left} \wedge \Delta\theta = \mathsf{right}) \vee (i_{\text{st}} = \mathsf{right} \wedge \Delta\theta = \mathsf{left})) \Rightarrow \bigcirc(d = \mathsf{medium2})\big), \tag{3.1}$$

$$\square\big((h_{r,f} = \mathcal{H}_{r,c} \wedge ((i_{\text{st}} = \mathsf{left} \wedge \Delta\theta = \mathsf{left}) \vee (i_{\text{st}} = \mathsf{right} \wedge \Delta\theta = \mathsf{right})) \Rightarrow \bigcirc(d = \mathsf{small2})\big), \tag{3.2}$$

which govern the allowable step length during the first step of a turning process.

Both navigation and action planners are constructed by combining environment assumptions and system specifications generated by the successor functions described in section 3.4 and section 3.5 into a transition system and using the LTL synthesis tool SLUGS to generate a winning strategy. Synthesis occurs offline, and the winning strategy is efficiently encoded in a binary decision diagram (BDD) [52] which can be accessed online by interfacing the controller directly with SLUGS. At each turn of the game, the controller computes the new abstracted environment state and passes it to SLUGS which returns the corresponding system action.

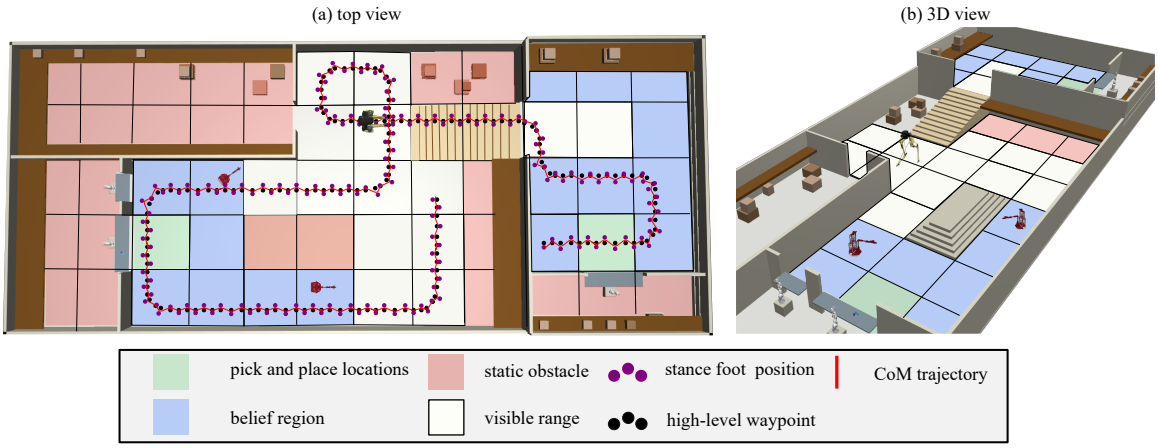



Figure 3.4: 3D simulation of the Cassie robot dynamically navigating in the partially observable environment while avoiding collisions with two mobile robot. Trajectories of Cassie CoM and the moving obstacle, Cassie foot placements as well as high-level abstraction are supeimposed in subfigure (a). Subfigure (b) shows the tested enviroment in 3D.

### 3.6.2 Nominal Motion Plan for Pick and Place Task

The middle-level motion planner is able to generate CoM trajectories of the ROM for a pick and place task infinitely often that include traversing stairs, steering, stopping, and avoiding dynamic obstacles. The keyframe decision maker, detailed in subsection 2.1.4, selects the optimal next keyframe for waypoint tracking. The action planner interfaces with the middle-level motion planner *online* to pass the action set for the next keyframe. In the case when the keyframe decision maker cannot satisfy the lateral tracking of high-level waypoints in Proposition 2.1.3, a new non-deterministic transition from the action planner is selected based on the modified lateral phase-space plan *online*. The action planner receives the updated waypoint which allows the planner to chose the correct transition to the next game state. Our simulation shows that the robot successfully traverses uneven terrain to complete its navigation goals while steering away from dynamic obstacles when they appear in the robot's visible range. The robot's navigation trajectory is shown in Figure 3.4. The tracking results for multiple plans with different obstacle paths are detailed in Table 3.1 using PSP parameters given in Table 3.2. Waypoint correction only occurs in the last step of a turning sequence due to the complexity of lateral tracking during steering scenarios.

Table 3.1: Successful motion plan results for the pick and place task

| **Steps** | **turns** | **waypoint correction** |
|---|---|---|
| 200 | 9 | 4 |
| 260 | 17 | 12 |
| 500 | 29 | 22 |

Table 3.2: Nominal PSP parameters values

| **parameter** | **value** | **parameter** | **value** |
|---|---|---|---|
| $v_{\text{apex,min}}$ | 0.20 m/s | $v_{\text{apex,max}}$ | 0.70 m/s |
| $h_{\text{apex}}$ | 0.985 m | $\Delta z_{\text{foot}}$ | $\{0, \pm 0.1, \pm 0.2\}$ m |
| $\Delta y_{1,d}$ | 0.10 m (0.0 m for steering) | $\Delta y_{2,d}$ | 0.14 m |
| $c_1$ | 1.0 (7 for steering) | $c_2$ | 1.0 (4 for steering) |
| $\Delta\theta$ | $\{0°, \pm 22.5°\}$ | $b_{\text{safety}}$ | 0.52 m |
| $d$ | $\{0.21, 0.28, 0.31, 0.38, 0.42, 0.43, 0.47, 0.52\}$ m | $v_{\text{inc}}$ | 0.01 m/s |

12 out of 260 steps[3] result in alternative discrete state transitions in the lateral direction, all of which were seamlessly handled by the action planner as shown in Figure 3.5. This result show that the integration of the high-level planner and the middle-level motion planner in an *online* fashion allows for successful and safe TAMP.

### 3.6.3 Safe Recoverability and Replanning

The proposed sequential composition of controllable regions and reachability analysis in [41] allows our middle-level motion planner to be robust against perturbations exerted on the CoM in the sagittal space. Given a keyframe transition for OWS, the synthesized controller is able to guarantee that the CoM state reaches the targeted state within OWS, thus successfully completing a OWS safely. In Figure 3.6(b) we show the composition of controllable regions for multiple walking step and demonstrate that the CoM trajectory is recoverable when employing the synthesized controller. Table 3.3 shows the success

[3]the step count refers to the number of high-level actions received by the middle-level motion planner, which includes stopping actions

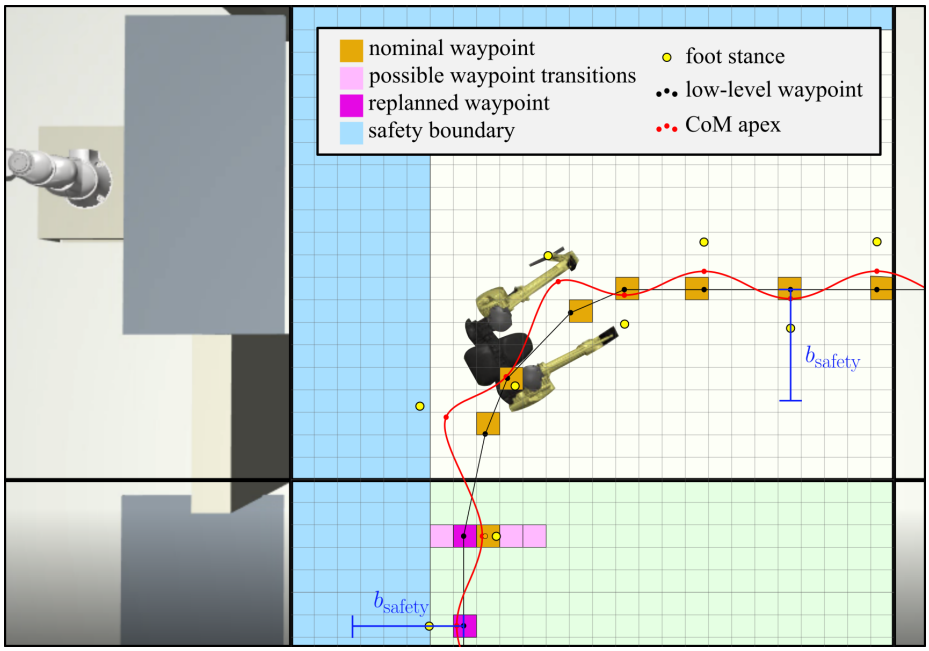



Figure 3.5: Illustration of *online* updating the high-level waypoint to maintain lateral tracking at the middle-level motion planner. The high-level waypoint is also required to keep a safe distance away from the adjacent coarse cell to avoid collisions with static or dynamic obstacles. In this run, we set the safety boundary to be 6 fine cells as shown in light blue.

rate for randomly generated keyframe transitions, where the step length is $d_1 = 0.312$ m, $d_3 = 0.416$ m and $d_3 = 0.52$ m. The data is generated using ROCS [53] with 1000 runs for each desired keyframe transition, a randomly selected $\boldsymbol{\xi}_c \in \boldsymbol{\Xi}_c$ and the applied disturbance bound $\tilde{\boldsymbol{\Xi}}_{\text{applied}}$ is uniformly distributed within $[-2, 2]$ m for CoM position and $[-5, 5]$ m/s for CoM velocity. The controllable regions are synthesized with state space granularity of $(0.002$ m, $0.004$ m/s), a control input $\omega \in [2.8, 3.5]$ rad/s with a granularity of $0.02$ rad/s, and the added noise bound at synthesis $\tilde{\boldsymbol{\Xi}}_{\text{synthesis}}$ is uniformly distributed within $[-0.01, 0.01]$ m for CoM position and $[-0.02, 0.02]$ m/s for CoM velocity. In Figure 3.6(a), we show 15 successful random keyframe transitions where $v_{\text{apex},n} = [0.45, 0.7]$ m/s and $d = 0.415$ m.

Large perturbations can push the system state outside of the controllable regions and the synthesized controller cannot recover to $\mathcal{T}_{\text{switch}}$. To safely recover from such large perturbations, we employ a variant of the capture point formulation [39, 54] to redesign the

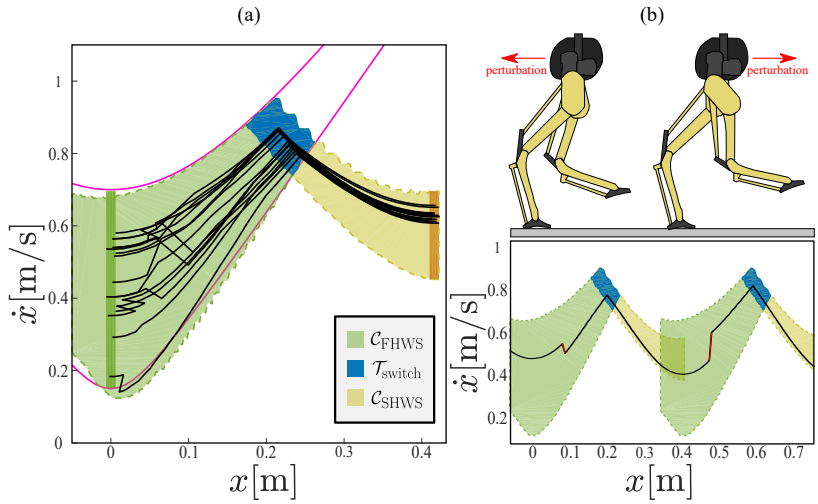



Figure 3.6: Results of OWS robust PSP. (a) shows a 15 random keyframe transitions with bounded disturbances, where $\mathcal{T}_{\text{OWS}} = (0.416 \text{ m}, [0.45, 0.7] \text{ m/s})$. (b) Composition of controllable regions of OWS. Here, we demonstrate that the synthesized controller is able to handle the perturbed CoM trajectory, shown as a black solid line, inside the superimposed controllable regions and successfully complete multiple steps when controllable regions are composed as proposed in in [41]

next foot position $x_{\text{foot},n}$ while maintaining the desired $v_{\text{apex},n}$ via the following formula:

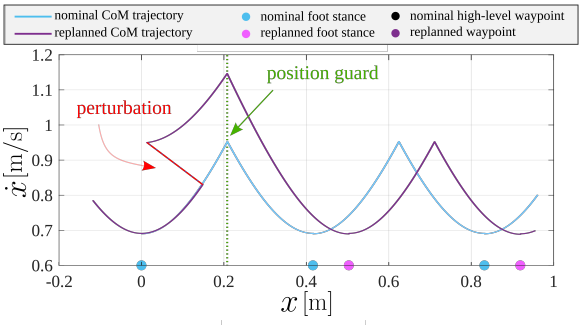
$$x_{\text{foot},n} = x_{\text{switch}} + \frac{1}{\omega}(\dot{x}^2_{\text{switch,dist}} + v^2_{\text{apex},n})^{1/2} \tag{3.3}$$

where $x_{\text{switch}}$ is determined analytically based on the nominal transition, and $\dot{x}_{\text{switch,dist}}$ is the post-disturbance sagittal CoM velocity at switch instant and computed through a position guard $x = x_{\text{switch}}$ shown as the vertical dashed line in Figure 3.7 (a). The nominal foot position is determined by the high-level waypoint. In case that the new foot location lands in a different fine cell, the *online* integration mechanism between the high-level and middle-level will update the action planner for a new waypoint location as shown in Figure 3.7(b) and (c). The action planner reacts to to the perturbation by replanning $d$ and $\Delta\theta$ in $a_{\text{HL}}$, which further induces a waypoint change at the next walking step. In particular, the nondeterministic transition flag $t_{nd} = \{\textsf{nominal}, \textsf{forward}, \textsf{backward}\}$ indicates the perturbation direction. The automata shown in Figure 3.7 (c) is a fragment from the larger action planner consisting of 21447 nodes. The navigation planner automaton has 20545 nodes.

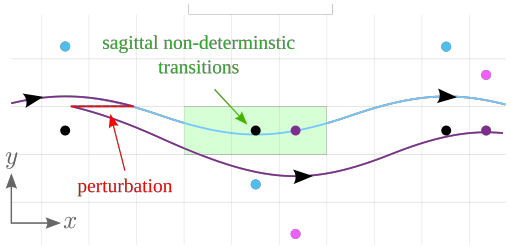
Table 3.3: Success rate of perturbed OWS transitions

| $v_{\text{apex},n}$ margin | Success Rate | | |
|---|---|---|---|
| | $d_1$ | $d_2$ | $d_2$ |
| $[0.2, 0.45]$ m/s | 90.2% | 91.6% | 92.5% |
| $[0.45, 0.7]$ m/s | 91.8% | 92.2% | 93.6% |

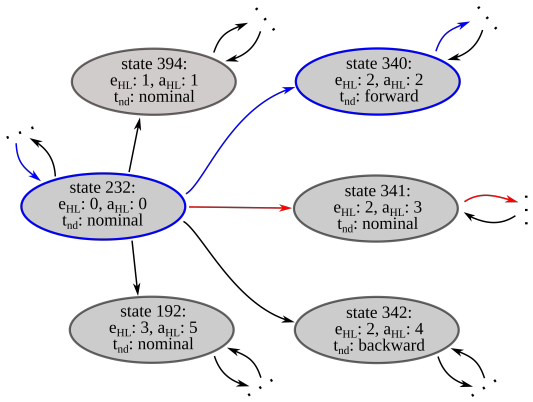
Online resynthesis of these planner automata is computationally intractable, and thus we incorporate the nondeterministic transition flag $t_{nd}$ into the automaton offline synthesis and employ them online for action replanning.




(a) Sagittal phase-space




(b) CoM trajectory

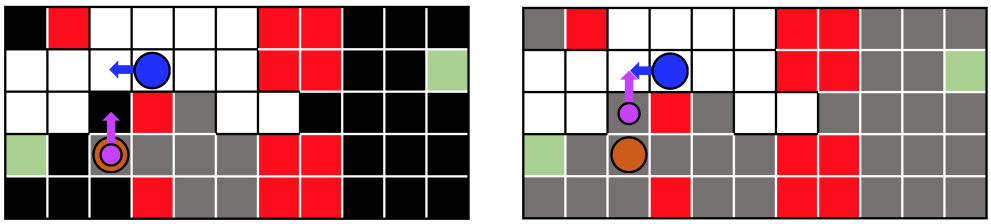



(c) Navigation Automaton Fragment with Replanning

Figure 3.7: Safe recovery from a large perturbation. (a) shows the sagittal phase-space plan, where a position guard is used to determine a safe replanned foot location to recover from the perturbation. (b) shows the CoM trajectory in Cartesian space and the *online* integration of the high-level action planner and the middle-level PSP for a waypoint modification. (c) shows a fragment of the synthesized action planner automaton capturing modeled nondeterministic transitions (with the associated flag $t_{nd}$). For each next state of the environment ($e_{\text{HL}}$), there is a set of game states corresponding to all possible $t_{nd}$. Blue transitions capture the replanned execution when the robot CoM is perturbed forward while red transitions depict a nominal execution without any perturbation. Numerical values for $e_{\text{HL}}$ and $a_{\text{HL}}$ index distinct environment state and robot action sets in the algorithm implementation.

### 3.6.4 Belief Space Planning

The belief abstraction in the navigation planner is successful in tracking and bounding nonvisible obstacles as can be seen in Figure 3.3. The tracked belief enables the robot to navigate around static obstacles while guaranteeing that the dynamic obstacles are not in the immediate non-visible vicinity. Figure 3.8 depicts a snapshot of a simulation where the robot must navigate around such an obstacle to reach its goal states. The gridworld environment is abstracted into 6 distinct belief regions resulting in 64 possible belief states. A successful strategy can be synthesized only when using a belief abstraction. Without explicitly tracking possible non-visible obstacle locations, the task planner believes the obstacle could be in any non-visible cell when it is out of sight, including the adjacent visible cell in the next turn of the game. That means the planner can not guarantee collision avoidance and is not able to synthesize a strategy that would allow the robot to advance. Figure 3.8b depicts a potential collision that could occur in pink. This comparison underlines the significance of the belief abstraction approach.



(a) With explicit belief tracking (b) No explicit belief tracking

Figure 3.8: A snapshot of the coarse-level navigation grid during a simulation where the robot (blue circle) is going between the two goal states (green cells), while avoiding a static obstacle (red cells) and a dynamic obstacle (orange circle). White cells are visible while grey and black cells are non-visible. Gray cells represent the planner's belief of potential obstacle locations. The closest the obstacle could be to the robot, as believed by the planner, is depicted by the pink circle.

The belief abstraction provides additional information for deciding long-horizon navigation actions beyond guaranteeing immediate collision avoidance. In the simulation shown in Figure 3.4, it is challenging to navigate around the vision occluding static obstacles at the lower-level (including the walls and a multi-stair platform). The synthesized

strategy reacts to the additional information about the dynamic obstacle provided by belief tracking in three distinct ways. Based on the belief, the robot either (i) continues on the most direct route to the goal location; (ii) loops around to the right and positions itself to be able to go around either side of the static obstacle; or (iii) stops and waits until the dynamic obstacle disappears (see the result in the simulation video). The planner can choose any of these three strategies as long as all safety specifications are met. This nondeterministic mechanism offers the task planner flexibility in choosing safe navigation actions.

Generating global navigation task planners for two dynamic obstacles using a joint belief abstraction requires only 40% of the synthesis time as that of independently tracking the belief state of each obstacle. Specifically, synthesizing a strategy for the scene in Figure 3.4 with two dynamic obstacles took 34 mins using joint

### 3.6.5 Discussion and limitations

Belief tracking expands the guaranteed safe navigation actions available to the navigation planner. Merging the belief of multiple dynamic obstacles into one abstract state captures less information than individual obstacle tracking by design. This reduces computational complexity while providing the same guarantees of capturing dynamic obstacle locations. One path to enhance the proposed framework in the future is to model small obstacles in the action planner so that an entire coarse cell containing such obstacles are still accessible to the robot in the navigation game. Additionally, the library of turning sequences can be expanded to incorporate more aggressive navigation decisions while obeying the safety criteria proposed in subsection 2.1.3 and to include fine-level obstacle avoidance maneuvers.

# CHAPTER 4

# HETEROGENEOUS MULTI-AGENT COLLABORATION FOR ENVIRONMENT ASSUMPTION VIOLATION RESOLUTION AT RUNTIME

The planning framework discussed in chapter 3 relies on environment assumptions to strictly hold true to guarantee a correct run for the system using the synthesized automaton. It is not possible to always ensure that assumptions used during offline synthesis hold true at runtime, in fact it is preferable to relax the assumptions if possible to enable for more robust planning. Others have studied how to directly relax the assumptions in the synthesized controller and use a metric of satisfaction to meet the specifications as closely as possible [25]. Such an approach is favorable when a single agent is operating in an environment and completes a task to the best of its abilities. However, when multiple heterogeneous agents are operating in the same environment it is possible to utilize capabilities of one agent to fix environment assumption violations for another agent to allow the continued execution of it's automaton when the agent that is being helped can not do so themselves. For example if a door that was assumed to be open blocks the path for a quadcopter another agent with manipulation capabilities can open the door, or if liquid is spilled on the floor that prevents a bipedal robot from safely traversing an area that was assumed to be traversable a mobile robot could clean up the spill. The planning framework presented here is particularly amenable to such solutions as the dynamics of a specific robot are taken into account only in the action planner, meaning a new action planner can be synthesized and swapped into the framework to safely plan for a number of different types of agents.

Here we will study a particular control synthesis problem where the environment violates an assumption at runtime preventing an agent in a heterogenous multi-agent team from complete its tasks. Formally, this occurs because an unmodeled environment behavior causes the specification to become unrealizable. We are specifically interested in

scenarios where the broken environment assumption results in the specification of an agent to become unrealizable, i.e. there exist no valid transitions in the automata that eventually result in the objective being met, and therefore requires that the broken assumption be fixed before the system can continue operation. The work presented here has been outlined in [55].

## 4.1 problem formulation

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 $\varphi_o^p$ and $\varphi_s^p$, respectively.

At the high-level, the environment is modeled using a coarse abstraction that divides the workspace into a set of $N$ discrete regions $\mathcal{L} = \{l_0, l_1, ..., l_{N-1}\}$. 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, static irresolvable obstacles $\mathcal{O} \subset \mathcal{L}$ are accounted for as the set of safety specifications

$$\varphi_s^p := \bigwedge_{s \in \mathcal{O}} \Box \neg l. \tag{4.1}$$

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”.

Static obstacles that are resolvable but not known a priori are modeled as another subset $\mathcal{R} \subset \mathcal{L}$. 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

$$c_r \in \bigcup_{p \in \mathcal{P}} C_p, \quad L_r \subset (\mathcal{L} \setminus \mathcal{O}). \tag{4.2}$$

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 $\varphi_a^p$, which contains the same specifications as $\varphi_s^p$ but in addition contains all of the additional safety specifications originating from unknown, resolvable obstacles:

$$\varphi_a^p := \varphi_s^p \bigwedge_{r \in R \mid c_r \notin C_p} \square \neg r \tag{4.3}$$

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

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

## 4.2 Controller Synthesis

To leverage the formal guarantees afforded by LTL, we synthesize navigation planners for each agent based on the planning framework detailed in [9]. 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 [45, 46, 47] adopted here is summarized as follows. To synthe-

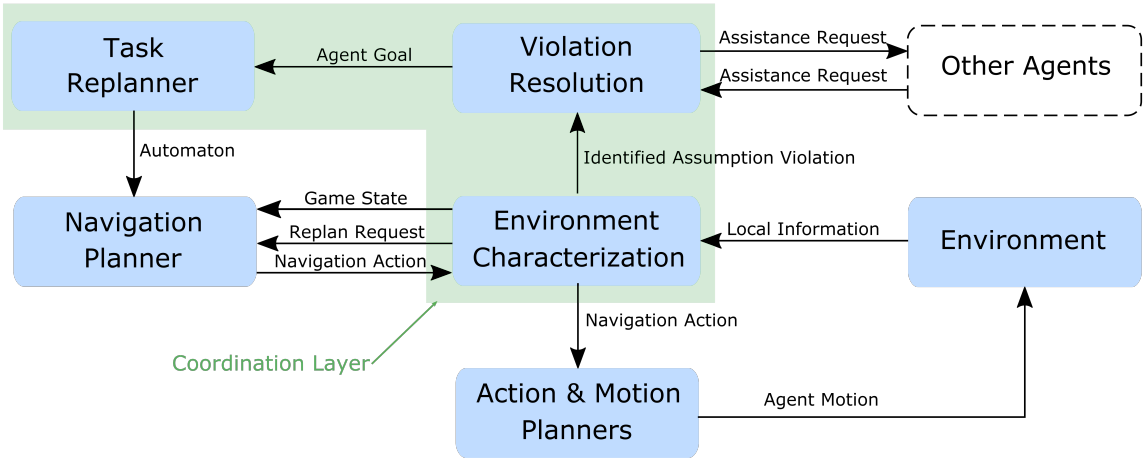



Figure 4.1: Block diagram of collaborative task and motion planning framework. A coordination layer verifies that the desired actions generated by an offline-synthesized navigation planner are still safe based on environment information observed at runtime. If an environment assumption is violated, the 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.

size 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 [10] 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 [49] are implementable in our planning framework.

## 4.3 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 4.1.

### 4.3.1 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 $\varphi_a^p$ 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.

### 4.3.2 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.

### 4.3.3 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) $L_r$ 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.

### 4.3.4 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.

### 4.3.5 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 2.

**Algorithm 2:** 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 l_o then
        if c_r ∉ C_{p_r} 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
```

### 4.3.6 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 3.

**Algorithm 3:** 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_{p_r} 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
```

## 4.4 Results

In this paper, we implement and evaluate the framework described in section 4.2, which was primarily built to synthesize controllers for the bipedal walking robot platform Cassie, designed by Agility Robotics [10]. 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 $180^\circ$ 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 [9, 33], 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 [39].

Resolvable obstacles are also implemented within the simulated environment. We recall from section 4.1 that each resolvable obstacle $r$ has an associated state $L_r$ 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 $\varphi_s^p$, is used for the initial synthesis, and then the resulting controller is applied to the environment instance containing the resolvable obstacles, which corresponds to $\varphi_a^p$. At runtime, if an agent enters a state containing a resolvable obstacle $r$ that it is unable to resolve (i.e., violates $\varphi_a^p$), 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 $\mathcal{L} = \{l_0, l_1, ..., l_{90}\}$ such that $l_0$ 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 Figure 4.2-Figure 4.6.

For each case study, we consider a team of agents consisting of $P = \{\texttt{quadcopter}, \texttt{Cassie}\}$ with unique capabilities $C_{\text{quad}} := \{\texttt{sense}\}$, $C_{\text{Cassie}} := \{\texttt{push}\}$. 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: $r = \texttt{door}$ and $r = \texttt{uncertainty}$. A resolvable obstacle of type $r = \texttt{door}$, if found in $s_i$, has properties

$$L_r = \{l_i\},\ c_r = \texttt{push}, \tag{4.4}$$

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

$$L_r = \{l_{\text{north}}, l_{\text{east}}, l_{\text{south}}, l_{\text{west}}\},\ c_r = \texttt{sense}, \tag{4.5}$$

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 $l_{\text{A}}, l_{\text{B}} \in \mathcal{L}$, with an AP $\texttt{scout}$ unique to each agent, which initializes to $\mathsf{False}$, to track which region the agent should

head towards. The set of objective specifications is thus given as

$$\begin{aligned}\text{Patrol}_p(l_\text{A}, l_\text{B}) =& \square\lozenge(l_\text{A} \wedge \neg\texttt{scout}_p) \\ &\wedge \square((l_\text{A} \wedge \neg\texttt{scout}_p) \Rightarrow \bigcirc\texttt{scout}_p) \\ &\wedge \square\lozenge(l_\text{B} \wedge \texttt{scout}_p) \\ &\wedge \square((l_\text{B} \wedge \texttt{scout}_p) \Rightarrow \bigcirc\neg\texttt{scout}_p)\end{aligned} \tag{4.6}$$

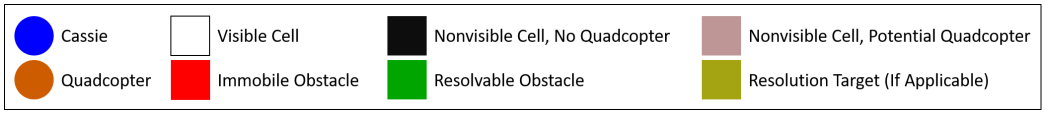
Each implementation of these case studies utilizes the resynthesis method detailed in subsection 4.3.5 due to its overall lower synthesis time, but it should be noted that case study 1 is fully implementable using the non-resynthesis method outlined in subsection 4.3.6. As case studies 2 and 3 require multiple locations to be visited and resolved, additional work is required to enable the non-resynthesis method to work in these cases.

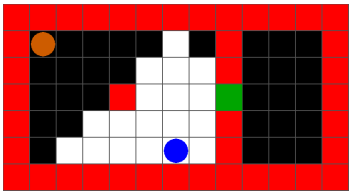
Additionally, while the figures in this section mainly feature abstractions of the environment in order to easily illustrate the behaviors of each agent, the computed control actions are applicable to a real 3D simulation environment, as shown in Figure 4.3. Those two subfigures show the real-world interactions resulting from the behavior in case study 1.

### 4.4.1 Case Study 1: Opening A Door

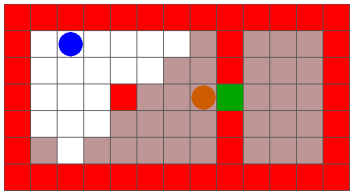
The first case study leverages Cassie's manipulation capability in the environment with higher dexterity and power than the quadcopter. The quadcopter is tasked with patrolling between $l_\text{homeQ}$ and $l_\text{awayQ}$, where $l_\text{homeQ}$ is a region in the left room and $l_\text{awayQ}$ is in the right room. Cassie is tasked with patrolling between $l_\text{homeC}$ and $l_\text{awayC}$, where both $l_\text{homeC}$ and $l_\text{awayC}$ are in the left room:

$$\begin{aligned}\varphi_o^\text{quad} &:= \text{Patrol}_\text{quad}(l_\text{homeQ}, l_\text{awayQ}), \\ \varphi_o^\text{Cassie} &:= \text{Patrol}_\text{Cassie}(l_\text{homeC}, l_\text{awayC}).\end{aligned} \tag{4.7}$$

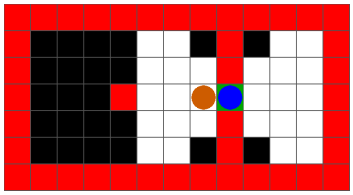


(a) Initial Configuration



(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.

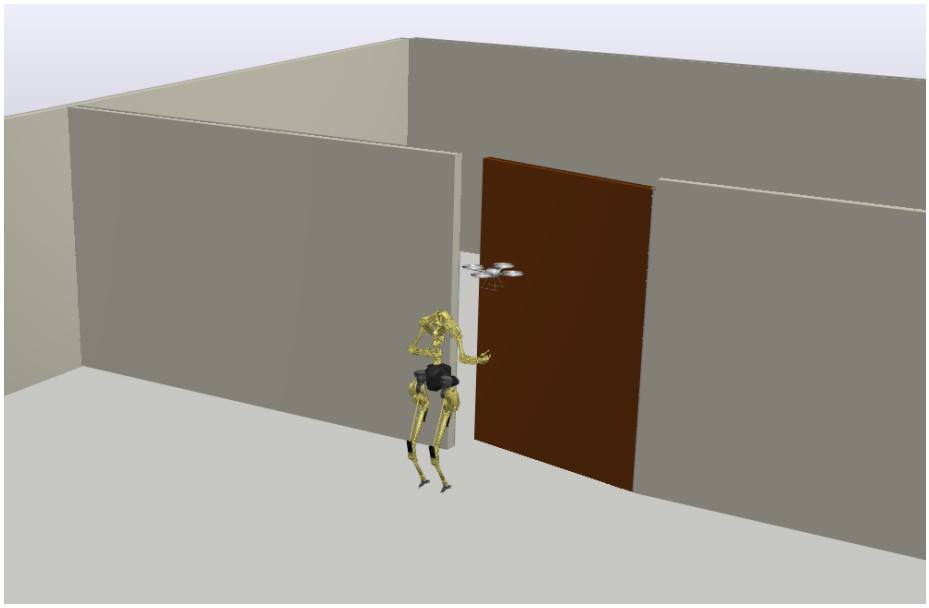
Figure 4.2: Execution of case study 1 leveraging Cassie's higher strength and manipulation abilities to open up the path for the quadcopter. Cassie's objective is to patrol the left room, while the quadcopter's objective is to deliver something to the right room. However, the quadcopter discovers a closed 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.

At runtime, the quadcopter discovers an obstruction at $l_{\text{door}} = l_{47}$ while in $l_{\text{safe}} = l_{46}$ 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:

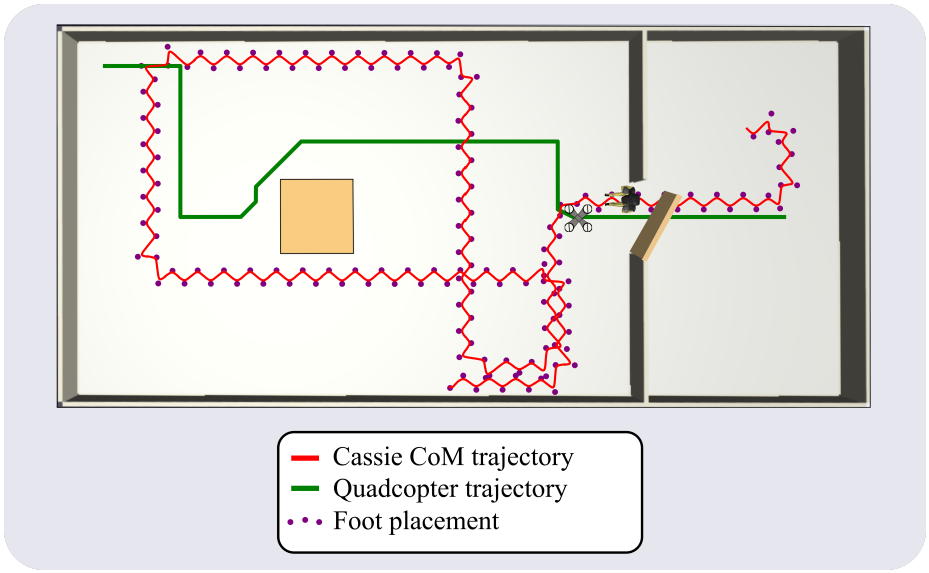
$$\begin{aligned} \varphi_o^{\text{quad}} &:= \text{Patrol}_{\text{quad}}(l_{\text{safe}}, l_{\text{safe}}), \\ \varphi_o^{\text{Cassie}} &:= \text{Patrol}_{\text{Cassie}}(l_{\text{homeC}}, l_{\text{door}}). \end{aligned} \tag{4.8}$$

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 4.2. For this case study, we also used low-level planners to generate safe motions for the quadcopter and Cassie, including CoM trajectories and foot placements. The 3D visualization can be

seen in Figure 4.3.

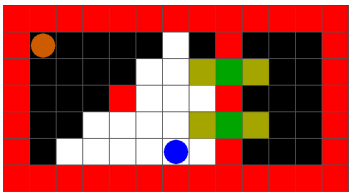


(a) Cassie opening the door for the quadcopter

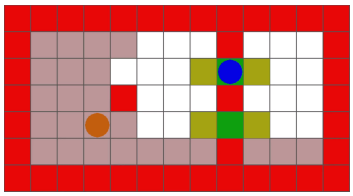



(b) A bird's eye view of Cassie's and the quadcopter's trajectories

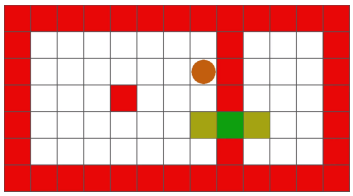
Figure 4.3: 3D rendering of case study 1 in Drake simulation [51]. The quadcopter approaches an unknown door obstacle preventing it from achieving its goal. The quadcopter requests assistance from Cassie. Accordingly, Cassie opens the door and both agents resume their original task.



(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.

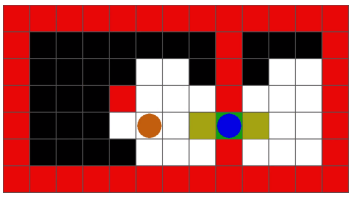
Figure 4.4: Partial execution of case study 2 leveraging the quadcopter's more powerful sensory capabilities to find a traversable path for Cassie. The quadcopter's objective is to patrol the left room, while Cassie's objective is to deliver something to the right room. However, Cassie encounters an uncertain region, prompting the quadcopter to observe the region and determine that it is nontraversable.

### 4.4.2 Case Study 2: Scouting Ahead

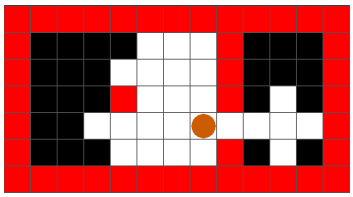
The second case study involves Cassie encountering several states and not knowing whether each state is safe to traverse on foot, requiring the help of the quadcopter's heightened sensing capabilities. To this end, the quadcopter is set to patrol between $l_{\text{homeQ}}$ and $l_{\text{awayQ}}$, where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

$$\begin{aligned} \varphi_o^{\text{quad}} &:= \text{Patrol}_{\text{quad}}(l_{\text{homeQ}}, l_{\text{awayQ}}), \\ \varphi_o^{\text{Cassie}} &:= \text{Patrol}_{\text{Cassie}}(l_{\text{homeC}}, l_{\text{awayC}}). \end{aligned} \tag{4.9}$$

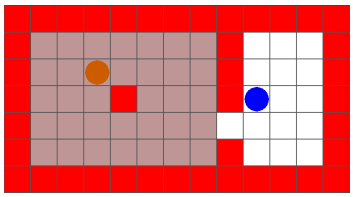
At runtime, Cassie encounters region $l_{\text{uncertain1}} = l_{34}$ while at $l_{\text{safe1}} = l_{33}$, and it is unsure about its ability to traverse this region. A resynthesis is triggered, where the quadcopter is tasked with observing the uncertain region by visiting any of the adjacent regions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



(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.

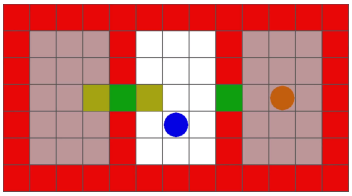
Figure 4.5: Latter half of the execution of case study 2. The quadcopter and Cassie are executing their original objectives when Cassie encounters another uncertain region. The quadcopter observes the new uncertainty, this time determining that the region is traversable, thus allowing both agents to fully complete their original objectives.

case, we select the region $l_{\text{uncertain1W}}$ directly west of $l_{\text{uncertain1}}$):

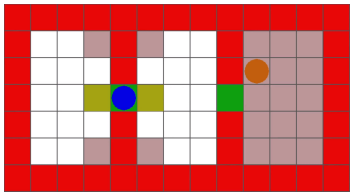
$$\begin{aligned} \varphi_o^{\text{quad}} &:= \text{Patrol}_{\text{quad}}(l_{\text{homeQ}}, l_{\text{uncertain1W}}), \\ \varphi_o^{\text{Cassie}} &:= \text{Patrol}_{\text{Cassie}}(l_{\text{safe1}}, l_{\text{safe1}}). \end{aligned} \tag{4.10}$$

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, $l_{\text{uncertain1W}}$ is found to be untraversable.

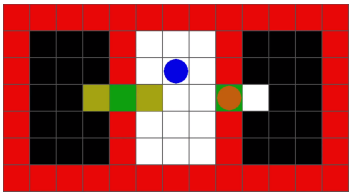
The two agents are returned to their original objectives, outlined in (Equation 4.9), before Cassie encounters another uncertain state at $l_{\text{uncertain2}} = l_{60}$ while in $l_{\text{safe2}} = l_{59}$, where the process repeats. The quadcopter is tasked with visiting $l_{\text{uncertain2W}}$, directly west of $l_{\text{uncertain2}}$, while Cassie is instructed to stay at $l_{\text{safe2}}$. 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 Figure 4.4 and Figure 4.5.



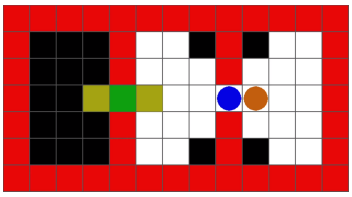
(a) Initial Configuration



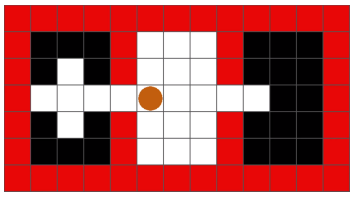
(b) Cassie's controller encounters an uncertain region, so it elects to have Cassie wait in the preceding region until it is resolved.



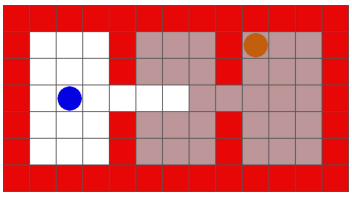
(c) On the way to sense the region that Cassie is uncertain about, the quadcopter encounters a closed door and waits on the other side.



(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.

Figure 4.6: 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.

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

Initially, Cassie is tasked with patrolling between $l_{\text{homeC}}$ in the leftmost room and $l_{\text{awayC}}$ in the center room, while the quadcopter is tasked with patrolling between $l_{\text{homeQ}}$ and

$l_{\text{awayQ}}$, both in the rightmost room:

$$\begin{aligned} \varphi_o^{\text{quad}} &:= \text{Patrol}_{\text{quad}}(l_{\text{homeQ}}, l_{\text{awayQ}}), \\ \varphi_o^{\text{Cassie}} &:= \text{Patrol}_{\text{Cassie}}(l_{\text{homeC}}, l_{\text{awayC}}) \end{aligned} \tag{4.11}$$

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

$$\begin{aligned} \varphi_o^{\text{quad}} &:= \text{Patrol}_{\text{quad}}(l_{\text{homeQ}}, l_{\text{uncertainE}}), \\ \varphi_o^{\text{Cassie}} &:= \text{Patrol}_{\text{Cassie}}(l_{\text{safeC}}, l_{\text{safeC}}) \end{aligned} \tag{4.12}$$

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

$$\begin{aligned} \varphi_o^{\text{quad}} &:= \text{Patrol}_{\text{quad}}(l_{\text{safeQ}}, l_{\text{safeQ}}), \\ \varphi_o^{\text{Cassie}} &:= \text{Patrol}_{\text{Cassie}}(l_{\text{safeC}}, l_{\text{door}}) \end{aligned} \tag{4.13}$$

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 4.6.

# CHAPTER 5
# CONCLUSION

The proposed task and motion planning framework in this thesis takes the first step towards locomotion task and motion planning that incorporates multi level safety guarantees. A reduced order PIPM is used to derive centroidal dynamics that govern trajectories between consecutive walking steps. Continuous trajectories between temporally discrete keyframes are planned by adhering to analytically derived safety theorems by one walking step phase space planning. At each keyframe a high-level task planner generates a action set that is safe with respect to both balancing and navigation safety and is guaranteed to lead to the eventual completion of the system's navigation goal. The high level planner uses two layers of abstraction to generate coarse global reactive navigation plans in partially observable environments in the presence of possibly adversarial dynamic obstacles and local action plans to ensure safe motion planning between keyframes. Safety is not only reasoned about at each planning level but is ensured between the layers of this hierarchical framework. To the author's knowledge this is the first work that captures low-level locomotion dynamics in the high-level specification design to formally guarantee navigation safety while maintaining balancing safety. Phase space planning constraints are directly and indirectly encoded enforced in the action planner through the use of turning sequence libraries and specification design. The action planner guarantees the desired transitions in the coarse global navigation game by design and accounts for collision avoidance with obstacles in adjacent coarse cells by capturing $b_{\text{safety}}$.

Capturing a detailed model and set of constraints in the action and navigation planners makes it challenging to synthesize strategies that meet all safety and task completion requirements. Belief space planning and relaxed action planner state transitions are used to address these challenges. Allowing for state transitions in the action planner to not

be deterministically set by a given action set relaxes the initial state-based constraints on turning sequences. Belief space planning method is designed and used in the global navigation planner to explicitly track the evolution of abstract dynamic obstacle belief states, this expands the set of navigation actions that are guaranteed safe for a given navigation game state, making it possible to synthesize strategies that meet all requirements that would otherwise not be possible. This belief tracking methodology is refined for multi-obstacle tracking with reduced computational complexity when compared to directly scaling the individual obstacle belief tracking method.

The effectiveness of the hierarchical framework has been demonstrated by simulating the system completing a pick and place task in a logistics environment with non-flat terrain in the presence of two mobile robots that are assumed to be adversarial. The navigation planner assumes the responsibility of collision avoidance and commands safe navigation actions based on the location of visible obstacles and its belief of non-visible obstacles. The action planner generates actions at each keyframe that the phase space planner is able to safely execute, the online integration allows the action planner to update the waypoint at several instances and reactively plan new locomotion actions. The pick and place task is successfully completed while maintaining navigation and balancing safety, and it is shown that in certain environments a successful planner can not be synthesized without the use of belief tracking.

The bipedal locomotion planning framework is extensible to other types of agents from mobile robots to drones, this can be done by swapping the action planner to plan appropriate actions for a given agent. While this work does not specifically create action planners for many agent types that consider agent's dynamics it uses this extensibility to increase the robustness of the global navigation planner via multi-agent collaboration. The presented approach specifically addresses the issue of environments mismatching the model and assumptions used for control strategy synthesis at runtime. This is achieved in four steps: 1) The environment is characterized at runtime and it is verified whether the next state in

the controller automaton would satisfy or violate any safety specifications based on runtime observations, (2) the immediate control action is replanned by backtracking states in the automaton and replacing unsafe actions with known safe actions, 3) a resolution is identified and assigned to another agent, 4) involved agents replan to eliminate violation via online resynthesis or goal substitution. This approach maintains all planning guarantees afforded by the bipedal locomotion planning framework yet extends it's robustness. This solution is unique to existing literature as it uses the capabilities of a team of agents to resolve environment anomalies at runtime, allowing agents to continue their tasks when they otherwise would not physically be able to. The successful operation of this method has been demonstrated in three case studies, including chained conflict resolution, and has been shown to integrate with the overall locomotion planning framework in a 3D simulation.

There are multiple possible directions to further improve the solutions detailed in this work. The current framework only considers static and dynamic obstacles in the navigation planner, however it should be possible to take obstacles into account when locally planning actions. A separate fine abstraction and associated action planner can be constructed for each coarse cell that captures detailed obstacle locations, an additional layer would be needed to correctly switch between action planners on the fly. To reason about dynamic obstacles at the local level it may be possible to utilize concepts from [31] to calculate occlusion cost metrics and guarantee local collision avoidance only when navigation planner's belief is such that it is possible for a dynamic obstacle to appear locally. Lastly further work can be done to expand the action planning turning library to include maneuvers beyond 90-deg turns and to include more non-deterministic action planner state transitions to account for perturbations. Heterogeneous multi-agent collaboration can be extended by 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.

In conclusion, the planning framework presented here leverages the expressive nature and formal guarantees of LTL to generate provably correct controllers for complex robotic systems. The use of belief space planning for dynamic obstacle belief tracking and heterogeneous robot capabilities to assist one another when environment assumptions are violated allows the planning framework to reduce the conservativeness traditionally associated with using formal methods for robot planning.

# Appendices

# APPENDIX A

# ANALYTICAL SOLUTION FOR PIPM DYNAMICS

When the CoM motion is constrained within a piece-wise linear surface parameterized by $h = a(x - x_{\text{foot}}) + h_{\text{apex}}$, the reduced-order model becomes linear and an analytical solution exists:

$$p(t) = Ae^{\omega t} + Be^{-\omega t} + p_{\text{foot}} \tag{A.1}$$

$$\dot{p}(t) = \omega(Ae^{\omega t} - Be^{-\omega t}) \tag{A.2}$$

where $\omega = \sqrt{\frac{g}{h_{\text{apex}}}}$, $A = \frac{1}{2}((p_0 - p_{\text{foot}}) + \frac{\dot{p}_0}{\omega}$, $B = \frac{1}{2}((p_0 - p_{\text{foot}}) - \frac{\dot{p}_0}{\omega}$. manipulate Equation A.1-Equation A.2 gives

$$p + \frac{\dot{p}}{\omega} - p_{\text{foot}} = 2Ae^{\omega t} \tag{A.3}$$

which renders

$$t = \frac{1}{\omega}\log(\frac{p + \frac{\dot{p}}{\omega} - p_{\text{foot}}}{2A}) \tag{A.4}$$

To find the dynamics, $\dot{p} = f(p)$, which will lead to the switching state solution, remove the $t$ term by plugging Equation A.4 into Equation A.1.

$$\frac{1}{2}(p - \frac{\dot{p}}{\omega} - p_{\text{foot}}) = \frac{2AB}{p + \frac{\dot{p}}{\omega} - p_{\text{foot}}} \tag{A.5}$$

$$(p - p_{\text{foot}})^2 - (\frac{\dot{p}}{\omega})^2 = 4AB \tag{A.6}$$

which yields

$$\dot{p} = \pm\sqrt{\omega^2((p - p_{\text{foot}})^2 - (p_0 - p_{\text{foot}})^2) + \dot{p}_0^2} \tag{A.7}$$

If the apex height is constant, then $\omega$ is constant. According to the constrain that sagittal velocity should be continuous, the saggital switching position is obtained by

$$x_{\text{switch}} = \frac{1}{2}\left(\frac{C}{x_{\text{foot},n} - x_{\text{foot},c}} + (x_{\text{foot},c} + x_{\text{foot},n})\right) \tag{A.8}$$

where

$$\begin{aligned} C =& (x_{\text{apex},c} - x_{\text{foot},c})^2 - (x_{\text{apex},n} - x_{\text{foot},n})^2 \\ &+ \frac{v_{\text{apex},n}^2 - v_{\text{apex},c}^2}{\omega^2} \end{aligned} \tag{A.9}$$

## APPENDIX B

## PROOF OF THEOREM 2.1.1

*Proof.* First, the sagittal switching position can be obtained from the analytical solution in Appendix Appendix A:

$$x_{\text{switch}} = \frac{1}{2}\left(\frac{C}{x_{\text{foot},n} - x_{\text{foot},c}} + (x_{\text{foot},c} + x_{\text{foot},n})\right) \tag{B.1}$$

where $C = (x_{\text{apex},c} - x_{\text{foot},c})^2 - (x_{\text{apex},n} - x_{\text{foot},n})^2 + (\dot{x}^2_{\text{apex},n} - \dot{x}^2_{\text{apex},c})/\omega^2$. This walking step switching position is required to stay between the two consecutive CoM apex positions, i.e.,

$$x_{\text{apex},c} \leq x_{\text{switch}} \leq x_{\text{apex},n} \tag{B.2}$$

which introduces the sagittal apex velocity constraints for two consecutive keyframes as follows.

$$\begin{aligned} &\omega^2(x_{\text{apex},n} - x_{\text{apex},c})(x_{\text{apex},c} + x_{\text{apex},n} - 2x_{\text{foot},n}) \\ &\qquad \leq \; v^2_{\text{apex},n} - v^2_{\text{apex},c} \leq \\ &\omega^2(x_{\text{apex},n} - x_{\text{apex},c})(x_{\text{apex},c} + x_{\text{apex},n} - 2x_{\text{foot},c}) \end{aligned} \tag{B.3}$$

Given this bounded difference between two consecutive CoM apex velocity squares, the corresponding safe criterion for straight walking can be expressed as Equation 2.3. □

# APPENDIX C

# PROOF OF THEOREM 2.1.2

*Proof.* First, for the sagittal phase-space, the sagittal velocity is required to be above the asymptote:

$$\dot{x}_{\text{apex},c} \geq \omega \cdot x_{\text{foot},c} \tag{C.1}$$

Initiating a heading angle change introduces a new local sagittal coordinates as seen in Figure 2.2. Therefore Equation C.1 becomes

$$v_{\text{apex},c} \cdot \cos \Delta\theta \geq \omega \cdot \Delta y_{2,c} \cdot \sin \Delta\theta \tag{C.2}$$

As for the lateral phase-space, the lateral velocity is required to be below the asymptote in the new coordinate as follows

$$v_{\text{apex},c} \cdot \sin \Delta\theta \leq \omega \cdot \Delta y_{2,c} \cdot \cos \Delta\theta \tag{C.3}$$

Combining Equation C.2-Equation C.3 results in the steering safety criterion in Equation 2.4. □

## REFERENCES


[1] T. Koolen, T. De Boer, J. Rebula, A. Goswami, and J. Pratt, "Capturability-based analysis and control of legged locomotion, part 1: Theory and application to three simple gait models," *The international journal of robotics research*, vol. 31, no. 9, pp. 1094–1113, 2012.

[2] S. Heim and A. Spröwitz, "Beyond basins of attraction: Quantifying robustness of natural dynamics," *IEEE Transactions on Robotics*, vol. 35, no. 4, pp. 939–952, 2019.

[3] J. Luo, Y. Su, L. Ruan, Y. Zhao, D. Kim, L. Sentis, and C. Fu, "Robust bipedal locomotion based on a hierarchical control structure.," *Robotica*, vol. 37, no. 10, pp. 1750–1767, 2019.

[4] N. Bohórquez, A. Sherikov, D. Dimitrov, and P.-B. Wieber, "Safe navigation strategies for a biped robot walking in a crowd," in *IEEE-RAS International Conference on Humanoid Robots*, 2016.

[5] A. Pajon and P.-B. Wieber, "Safe 3d bipedal walking through linear mpc with 3d capturability," in *International Conference on Robotics and Automation*, IEEE, 2019, pp. 1404–1409.

[6] V. Vasilopoulos, W. Vega-Brown, O. Arslan, N. Roy, and D. E. Koditschek, "Sensor-based reactive symbolic planning in partially known environments," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 1–5.

[7] O. Arslan and D. E. Koditschek, "Sensor-based reactive navigation in unknown convex sphere worlds," *The International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 196–223, 2019.

[8] V. Vasilopoulos, G. Pavlakos, S. L. Bowman, J. D. Caporale, K. Daniilidis, G. J. Pappas, and D. E. Koditschek, "Reactive semantic planning in unexplored semantic environments using deep perceptual feedback," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4455–4462, 2020.

[9] J. Warnke, A. Shamsah, Y. Li, and Y. Zhao, "Towards safe locomotion navigation in partially observable environments with uneven terrain," in *2020 59th IEEE Conference on Decision and Control (CDC)*, IEEE, 2020, pp. 958–965.

[10] A. Robotics, *Cassie simulators.*

[11] 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.

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

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

[14] 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*, 2020, pp. 7719–7725.

[15] 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*, 2019, pp. 195–201.

[16] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[17] 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.

[18] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, IEEE, 2005, pp. 2020–2025.

[19] Y. Zhao, Y. Li, L. Sentis, U. Topcu, and J. Liu, "Reactive task and motion planning for robust whole-body dynamic locomotion in constrained environments," *arXiv preprint arXiv:1811.04333*, 2018.

[20] S. Kulgod, W. Chen, J. Huang, Y. Zhao, and N. Atanasov, "Temporal logic guided locomotion planning and control in cluttered environments," in *American Control Conference*, IEEE, 2020.

[21] J. A. DeCastro, J. Alonso-Mora, V. Raman, D. Rus, and H. Kress-Gazit, "Collision-free reactive mission and motion planning for multi-robot systems," in *Robotics research*, Springer, 2018, pp. 459–476.

[22] E. Plaku and S. Karaman, "Motion planning with temporal-logic specifications: Progress and challenges," *AI communications*, vol. 29, no. 1, pp. 151–162, 2016.

[23] A. Wu and J. P. How, "Guaranteed infinite horizon avoidance of unpredictable, dynamically constrained obstacles," *Autonomous robots*, vol. 32, no. 3, pp. 227–242, 2012.

[24] S. Sarid, B. Xu, and H. Kress-Gazit, "Guaranteeing high-level behaviors while exploring partially known maps," Jul. 2012.

[25] M. R. Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative temporal motion planning for hybrid systems in partially unknown environments," in *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC, Association for Computing Machinery, 2013, pp. 353–362.

[26] 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*, IEEE, 2012, pp. 5163–5170.

[27] S. C. Livingston, P. Prabhakar, A. B. Jose, and R. M. Murray, "Patching task-level robot controllers based on a local $\mu$-calculus formula," in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 4588–4595.

[28] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *ACM International Conference on Hybrid Systems: Computation and Control*, 2010.

[29] S. Ragi and E. K. Chong, "Uav path planning in a dynamic environment via partially observable markov decision process," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 4, pp. 2397–2412, 2013.

[30] C. Fulgenzi, A. Spalanzani, and C. Laugier, "Dynamic obstacle avoidance in uncertain environment combining pvos and occupancy grid," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, IEEE, 2007, pp. 1610–1616.

[31] W. Chung, S. Kim, M. Choi, J. Choi, H. Kim, C.-b. Moon, and J.-B. Song, "Safe navigation of a mobile robot considering visibility of environment," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 3941–3950, 2009.

[32] S. Bouraine, T. Fraichard, and H. Salhi, "Provably safe navigation for mobile robots with limited field-of-views in dynamic environments," *Autonomous Robots*, vol. 32, no. 3, pp. 267–283, 2012.

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

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

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

[36] 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*, 2011, pp. 211–220.

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

[38] 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.

[39] 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.

[40] Y. Zhao and L. Sentis, "A three dimensional foot placement planner for locomotion in very rough terrains," in *IEEE-RAS International Conference on Humanoid Robots*, IEEE, 2012, pp. 726–733.

[41] A. Shamsah, J. Warnke, Z. Gu, and Y. Zhao, "Integrated task and motion planning for safe legged navigation in partially observable environments," *arXiv preprint arXiv:2110.12097*, 2021.

[42] K. He, A. M. Wells, L. E. Kavraki, and M. Y. Vardi, "Efficient symbolic reactive synthesis for finite-horizon tasks," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 8993–8999.

[43] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[44] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive (1) designs," *Journal of Computer and System Sciences*, vol. 78, no. 3, pp. 911–938, 2012.

[45] 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.

[46] 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.

[47] 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.

[48] R. Ehlers and V. Raman, "Slugs: Extensible gr (1) synthesis," in *International Conference on Computer Aided Verification*, Springer, 2016, pp. 333–339.

[49] 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, no. 4, pp. 801–824, 2018.

[50] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *International Conference on Computer Aided Verification*, Springer, 2000, pp. 154–169.

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

[52] S. B. Akers, "Binary decision diagrams," *IEEE Transactions on computers*, vol. 27, no. 06, pp. 509–516, 1978.

[53] Y. Li and J. Liu, "Rocs: A robustly complete control synthesis tool for nonlinear dynamical systems," in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control*, 2018, pp. 130–135.

[54] J. Englsberger, C. Ott, M. A. Roa, A. Albu-Schäffer, and G. Hirzinger, "Bipedal walking control based on capture point dynamics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 4420–4427.

[55] M. E. Cao, J. Warnke, Y. Zhao, and S. Coogan, "Leveraging heterogeneous capabilities in multi-agent systems for environmental conflict resolution,"