Physically-Feasible Reactive Synthesis for Terrain-Adaptive Locomotion via Trajectory Optimization and Symbolic Repair

Ziyi Zhou[1], Qian Meng[2], Hadas Kress-Gazit[2], and Ye Zhao[1]

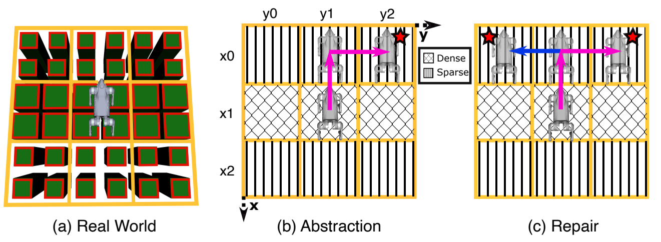
***Abstract*— We propose an integrated planning framework for quadrupedal locomotion over dynamically changing, unforeseen terrains. Existing approaches either rely on heuristics for instantaneous foothold selection–compromising safety and versatility–or solve expensive trajectory optimization problems with complex terrain features and long time horizons. In contrast, our framework leverages reactive synthesis to generate correct-by-construction controllers at the symbolic level, and mixed-integer convex programming (MICP) for dynamic and physically feasible footstep planning for each symbolic transition. We use a high-level manager to reduce the large state space in synthesis by incorporating local environment information, improving synthesis scalability. To handle specifications that cannot be met due to dynamic infeasibility, and to minimize costly MICP solves, we leverage a symbolic repair process to generate only necessary symbolic transitions. During online execution, re-running the MICP with real-world terrain data, along with runtime symbolic repair, bridges the gap between offline synthesis and online execution. We demonstrate, in simulation, our framework's capabilities to discover missing locomotion skills and react promptly in safety-critical environments, such as scattered stepping stones and rebars.**

I. Introduction

Terrain-adaptive locomotion is crucial for enhancing the traversing capabilities of legged robots and advancing beyond blind locomotion [1]–[3]. Existing approaches that enable terrain-adaptive and dynamic locomotion focus on instantaneous foothold adaptation [4], [5] around a nominal reference trajectory. To further enhance the traversability, non-fixed gait pattern has been studied. In [6]–[8], jumping motions are generated offline and triggered through heuristics when facing an obstacle. In [9], [10], the switch between normal walking and jumping is either embedded inside a reduced-order model during sampling [9] or decided by a pre-trained feasibility classifier [10]. However, formal guarantees on locomotion safety [11], [12], considering the robot's physical capabilities for traversing challenging terrains, are rarely explored—despite their importance in safety-critical scenarios like hazardous debris and construction sites.

Mixed-integer program (MIP)-based approaches [13], [14] treat both the contact state and the contact plane selection for each leg at each timestep as binary variables, and directly address the interplay between terrain segments and robot kinematics and dynamics [15]. By relaxing the dynamics and constraints, a global certificate for the approximated convex problem (MICP) exists upon convergence, providing

[1]Ziyi Zhou, and Ye Zhao are with Georgia Institute of Technology. `{zhouziyi, yzhao301}@gatech.edu`.
[2]Qian Meng and Hadas Kress-Gazit are with Cornell University. `{qm34,hadaskg}@cornell.edu`.



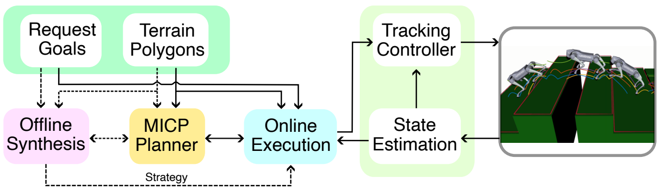


Fig. 1. Upper: terrain abstraction and repair example. (a) Top-down view of the real-world terrain before abstraction. The red polygons denote the segmented terrain polygons. (b) Abstraction of the terrain and robot's skills (pink) moving from one location to another. (c) Repair process to find a new skill (blue). Lower: system architecture overview. The solid lines indicate online communication, while dashed lines represent offline processes. The request goals in the offline phase are user-defined, while those in the online phase are provided by a global planner based on the global goal.

an ideal means to formally determine the locomotion feasibility. However, long time horizon and complex features of surrounding terrains significantly increase the computational burden for MIP-based methods, preventing efficient online deployment. Simplifications have been made to enhance speed, *e.g.*, assuming predetermined contact sequences [16], [17] and timing [18]–[20], or fixing the number of footsteps [21], but still face computational challenges in complicated scenarios with a large number of terrain segments.

To reduce the complexity of navigating challenging terrains, one approach is to break down global, long-horizon tasks into dynamically evolving, robot-centric local environments with intermediate waypoints [4]. However, the corresponding MIP remains computationally expensive because it must consider all surrounding terrains. To address this, we discretize the local environment and restrict the MIP formulation to solving only individual discrete transitions. To efficiently compose these discrete transitions with formal correctness guarantees, we employ Linear Temporal Logic (LTL)-based reactive synthesis [22] to generate a correct-by-construction robot strategy that guides the composition of MIP-based transitions toward the intermediate goals.

In this paper, we combine reactive synthesis and MIP-based approaches to safely and promptly react to dynamically changing environments. We abstract the continuous robot states and local environments into symbolic states, enabling high-level robot actions on the fly toward challenging terrains such as obstacles and large gaps, as shown in Fig. 1. We then solve a MICP to provide a physical feasibility certificate for each symbolic transition. This integrated framework not only bridges the gap between high-level abstraction and low-level physical capabilities but also alleviates the computational burden of MICP. Guided by the synthesis-based symbolic planner, each MICP can consider shorter time horizons and fewer terrain features than traditionally solving a single, long-horizon MICP problem. Another noteworthy feature of our planning framework is its two-stage hierarchy. In the offline synthesis phase, we leverage relatively expensive gait-free MICPs that optimize both contact state and foothold selection to generate the necessary locomotion gaits for symbolic transitions. In the online execution phase, we use efficient gait-fixed MICPs with predefined gaits to produce dynamically feasible motions and footholds on the fly.

Our offline synthesis module tackles two key scalability challenges. First, our task specification has a prohibitively large state space since it needs to encode the surrounding terrain states. Since our synthesis algorithm has exponential time complexity in the number of variables [22], it is inefficient and impractical to directly synthesize a controller for the full specification. To tackle this challenge, we leverage an observation that the robot can continuously reason about its local environment where the terrains and an intermediate goal remain unchanged. Thus, we propose a high-level manager that fixes the terrain and goal information, only keeps the skills needed for the current terrains, and updates them as the robot moves in the environment. In our experiment, the manager reduces the number of variables by $80.9 - 90.1\%$ depending on the number of cells and terrain types, relieving the computation burden of synthesis.

Our second scalability challenge is that, while the size of the MICP for our physical feasibility certificate is much smaller than the pure MIP approach, solving gait-free MICP is still computationally expensive. To mitigate this problem, we leverage a symbolic repair approach [23] to automatically suggest only the missing, yet necessary, symbolic transitions, and only solve MICP for them. In this way, we avoid enumerating and solving the costly MICP for all possible symbolic transitions. In our experiments, the symbolic repair reduces the number of expensive MICP solves by $71.7 - 97.6\%$ depending on the terrains, compared with exhaustively iterating over all possible symbolic transitions, ensuring that we only perform the costly gait-free MICP to generate new locomotion gaits when necessary.

Our online execution module also tackles two key challenges. First, discrepancies in size and shape between offline-checked terrains and real-world ones can lead to mismatches between the desired symbolic transitions and the robot's actual physical capabilities. Second, encountering new terrains or intermediate goals not considered in the offline phase may also make symbolic specifications unrealizable, as the unforeseen scenarios are not handled during the offline phase. As such, our online execution module executes each symbolic transition by solving an online MICP with real-world terrain data and prior locomotion gaits to generate a new nominal trajectory. If a symbolic transition is no longer possible or the current terrains and goal were not considered offline, we perform runtime repair, as done in [24], to generate new robot skills and synthesize an updated robot strategy, allowing continuous traversal on unexpected terrains.

The primary contributions of this paper are as follows:

- We propose an integrated planning framework that combines reactive synthesis with MICP for terrain-adaptive locomotion. Compared to pure MIP approaches, our method reduces computational burden via symbolic guidance and a two-stage hierarchy.
- During offline synthesis, we overcome scalability challenges by leveraging a high-level manager to reduce the size of the specifications based on local environment information, and symbolic repair to minimize the number of calls to the costly gait-free MICP, enabling efficient synthesis of terrain-adaptive locomotion strategies.
- During online execution, we address the disparity between offline synthesis and real-world terrain conditions at both the physical and symbolic levels. Our approach leverages an online MICP solver along with an online symbolic repair process to account for real-world terrain discrepancies and newly encountered conditions, enhancing robustness and motion feasibility at runtime.

II. Preliminaries

Consider a robot equipped with sensors navigating an environment toward a designated global goal $g_{\text{global}} \in \mathbb{R}^2$. This task can be decomposed into a series of local navigation tasks, where the robot moves toward intermediate, local request goals $\mathcal{G}_{\text{local}}$ as guided by a global planner, given surrounding segmented terrain polygons $\mathcal{P}$. We illustrate an example of this local task.

Example 1. *Consider a 2D grid world with nine cells (in yellow) in Fig. 1. The robot is required to move from an initial cell (e.g. the center cell) to a desired goal cell indicated by the star. Each cell is assigned a terrain type, such as Dense or Sparse stepping stones.*

A. Abstractions

Given a set of terrain polygons $\mathcal{P}$, e.g., in Fig. 1(a), we abstract them into a 2D grid of size $n \times m$. For each dimension, we denote each location in that dimension with a symbol, resulting in two sets $\mathcal{X} := \{x_0, \ldots, x_{n-1}\}$ and $\mathcal{Y} := \{y_0, \ldots, y_{m-1}\}$. We define a set of atomic propositions AP, partitioned into sets of inputs $\mathcal{I}$ and outputs $\mathcal{O}$, to describe the symbolic world states and robot actions. The inputs $\mathcal{I}$ consist of the robot inputs $\mathcal{I}_{\text{robot}}$, the request inputs $\mathcal{I}_{\text{req}}$, and the terrain inputs $\mathcal{I}_{\text{terrain}}$ ($\mathcal{I} = \mathcal{I}_{\text{robot}} \cup \mathcal{I}_{\text{req}} \cup \mathcal{I}_{\text{terrain}}$). The robot inputs $\mathcal{I}_{\text{robot}} := \{\pi_x \mid x \in \mathcal{X}\} \cup \{\pi_y \mid y \in \mathcal{Y}\}$ are used to abstract the robot position, the request inputs $\mathcal{I}_{\text{req}} := \{\pi_x^{\text{req}} \mid x \in \mathcal{X}\} \cup \{\pi_y^{\text{req}} \mid y \in \mathcal{Y}\}$ are used to abstract the requested goal cells, and the terrain inputs $\mathcal{I}_{\text{terrain}} := \{n_x^y \mid x \in \mathcal{X}, y \in \mathcal{Y}\}$ describe the terrain type of the grid cells. We consider a finite set of size n_t of terrain types, defined based on their physical characteristics. For

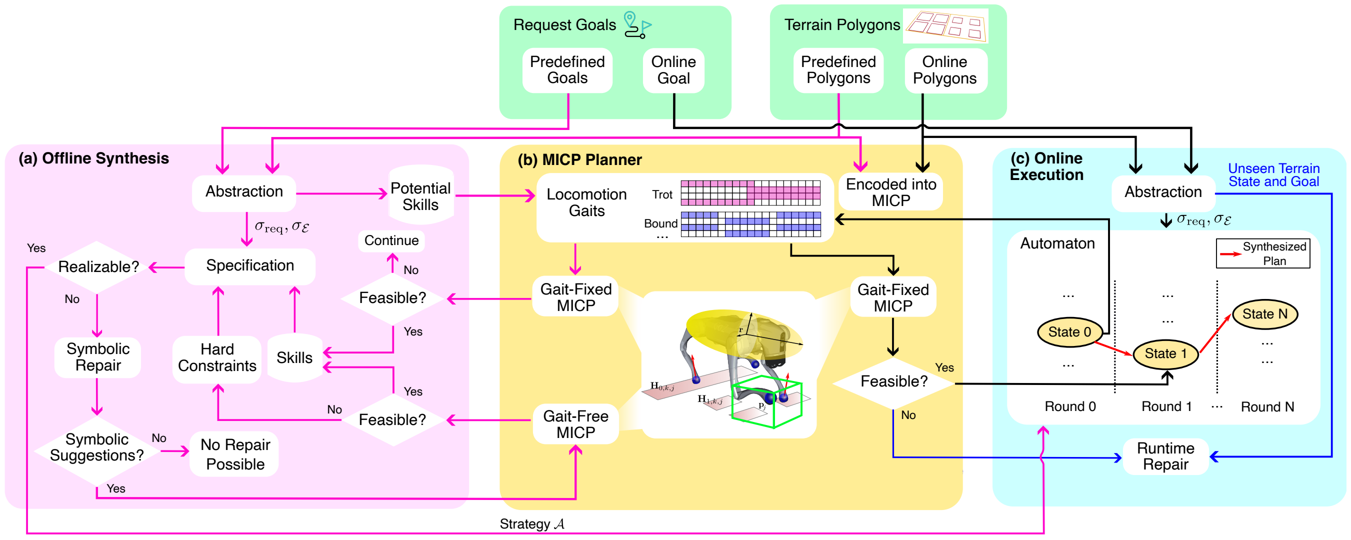


Fig. 2. System overview. During offline synthesis (pink arrows), an initial set of locomotion gaits is provided, and symbolic skills are iteratively generated by solving the MICP. During the online execution (black arrows), MICP is solved again taking online terrain segments and the symbolic state only advances when a solution is found. A runtime repair (blue arrows) is initiated if a solving failure occurs, or an unseen terrain or request goal is encountered.

$z \in \mathbb{N}$, we denote $[z] := \{0, \dots, z-1\}$. We note that terrain inputs $n_x^y \in \mathcal{I}_{\text{terrain}}$ are integer variables belonging to $[n_t]$, and we translate them into a set of Boolean propositions, as done in [25]. An input state $\sigma_{\mathcal{I}}$ is a value assignment function $\sigma_{\mathcal{I}} : \mathcal{I} \to \{\texttt{True}, \texttt{False}\} \cup [n_t]$. Since the robot can only be in one cell at a time, and the goal is a single cell, we require that exactly one π_x, one π_y, one π_x^{req}, and one π_y^{req} be $\texttt{True}$, for each input state. The robot states $\sigma_{\text{robot}} : \mathcal{I}_{\text{robot}} \to \{\texttt{True}, \texttt{False}\}$, request states $\sigma_{\text{req}} : \mathcal{I}_{\text{req}} \to \{\texttt{True}, \texttt{False}\}$, and terrain states $\sigma_{\text{terrain}} : \mathcal{I}_{\text{terrain}} \to [n_t]$ are the projections of $\sigma_{\mathcal{I}}$ on $\mathcal{I}_{\text{robot}}$, $\mathcal{I}_{\text{req}}$, and $\mathcal{I}_{\text{terrain}}$, respectively. We denote the sets of input states as $\Sigma_{\mathcal{I}}$, robot states as Σ_{robot}, request states as Σ_{req}, and terrain states as Σ_{terrain}.

We use a grounding function to describe the physical meaning of input states. Consider a physical state space $\mathcal{Z} \subseteq \mathbb{R}^n$ that represents the physical world, including the robot's position, orientation, and terrain properties. The grounding function $\mathrm{G} : \Sigma_{\mathcal{I}} \to 2^{\mathcal{Z}}$ maps each input state $\sigma_{\mathcal{I}}$ to a set of physical states in $\mathcal{Z}$. For robot states $\sigma_{\text{robot}} \in \Sigma_{\text{robot}}$, we define $\mathrm{G}(\sigma_{\text{robot}}) := \{z \in \mathcal{Z} \mid \exists \pi_x, \pi_y \in \mathcal{I}_{\text{robot}}.\sigma_{\text{robot}}(\pi_x) \wedge \sigma_{\text{robot}}(\pi_y) \wedge \text{robot\_pose}(z) \in \text{cell}(x,y)\}$. Similarly, for request states $\sigma_{\text{req}} \in \Sigma_{\text{req}}$, we define $\mathrm{G}(\sigma_{\text{req}}) := \{z \in \mathcal{Z} \mid \exists \pi_x^{\text{req}}, \pi_y^{\text{req}} \in \mathcal{I}_{\text{req}}.\sigma_{\text{req}}(\pi_x^{\text{req}}) \wedge \sigma_{\text{req}}(\pi_y^{\text{req}}) \wedge \text{request\_goal}(z) \in \text{cell}(x,y)\}$. For terrain states $\sigma_{\text{terrain}} \in \Sigma_{\text{terrain}}$, we define $\mathrm{G}(\sigma_{\text{terrain}})$ to be the set of physical states whose abstracted terrain types are indicated by σ_{terrain}. For an input state $\sigma_{\mathcal{I}} \in \Sigma_{\mathcal{I}}$ whose projections are σ_{robot}, σ_{req}, and σ_{terrain}, $\mathrm{G}(\sigma_{\mathcal{I}}) := \mathrm{G}(\sigma_{\text{robot}}) \cap \mathrm{G}(\sigma_{\text{req}}) \cap \mathrm{G}(\sigma_{\text{terrain}})$. Lastly, we use an inverse grounding function $\mathrm{G}^{-1} : \mathcal{Z} \to \Sigma_{\mathcal{I}}$ to map a physical state $z \in \mathcal{Z}$ to its corresponding input state.

In Example 1, the inputs are $\mathcal{I}_{\text{robot}} := \{\pi_{x_0}, \pi_{x_1}, \pi_{x_2}, \pi_{y_0}, \pi_{y_1}, \pi_{y_2}\}$, $\mathcal{I}_{\text{req}} := \{\pi^{\text{req}} \mid \pi \in \mathcal{I}_{\text{robot}}\}$, and $\mathcal{I}_{\text{terrain}} := \{n_{x_i}^{y_j} \mid i,j \in \{0,1,2\}\}$ where $n_{x_i}^{y_j} = 1$ if the terrain type of the grid cell (x_i, y_j) is Dense, and $n_{x_i}^{y_j} = 0$ if it is Sparse. The input state in Fig. 1(b) is $\sigma_{\mathcal{I}} : \pi_{x_1} \mapsto \texttt{True}, \pi_{y_1} \mapsto \texttt{True}, \pi_{x_0}^{\text{req}} \mapsto \texttt{True}, \pi_{y_2}^{\text{req}} \mapsto \texttt{True}, n_{x_1}^{y_j} \mapsto 1$ for $j \in \{0,1,2\}$. In this paper, we omit the Boolean propositions that are $\texttt{False}$ and integer propositions whose values are 0 in the input states for space.

The outputs $\mathcal{O}$ represent the skills that allow the robot to transit among grid cells, given their corresponding terrain states. A skill $o \in \mathcal{O}$ consists of sets of preconditions $\Sigma_o^{\text{pre}} \subseteq \Sigma_{\text{robot}} \times \Sigma_{\text{terrain}}$, from which the skill is allowed to execute, and postconditions $\Sigma_o^{\text{post}} \subseteq \Sigma_{\text{robot}}$, the resulting grid cell after executing the skill. In Example 1, the skill o_0 moves the robot from the middle to the upper terrain grid cell (Fig. 1b). The precondition of o_0 is $\Sigma_{o_0}^{\text{pre}} = \{(\sigma_{\text{robot}}, \sigma_{\text{terrain}}) : \pi_{x_1} \mapsto \texttt{True}, \pi_{y_1} \mapsto \texttt{True}, n_{x_1}^{y_0} \mapsto 1, n_{x_1}^{y_1} \mapsto 1, n_{x_1}^{y_2} \mapsto 1\}$. The postcondition is $\Sigma_{o_0}^{\text{post}} = \{\sigma_{\text{robot}} : \pi_{x_0} \mapsto \texttt{True}, \pi_{y_1} \mapsto \texttt{True}\}$. In addition, each skill consists of a continuous-level locomotion gait, e.g. a one-second trotting gait L, that physically implements the symbolic transition (Sec. V-A).

B. Specifications

We use the Generalized Reactivity(1) (GR(1)) fragment of linear temporal logic (LTL) [22] to encode task specifications. LTL specifications φ follow the grammar $\varphi := \pi \mid \neg \varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \square \varphi \mid \lozenge \varphi$, where $\pi \in AP$ is an atomic proposition, $\neg$ "not" and $\wedge$ "and" are Boolean operators, and $\bigcirc$ "next", $\square$ "always", and $\lozenge$ "eventually" are temporal operators. We refer the readers to [26] for a detailed description of LTL.

GR(1) specifications are expressed in the form of $\varphi = \varphi_{\text{e}} \to \varphi_{\text{s}}$, where $\varphi_{\text{e}} = \varphi_{\text{e}}^{\text{i}} \wedge \varphi_{\text{e}}^{\text{t}} \wedge \varphi_{\text{e}}^{\text{g}}$ is the assumptions on the behaviors of the possibly adversarial environment, and $\varphi_{\text{s}} = \varphi_{\text{s}}^{\text{i}} \wedge \varphi_{\text{s}}^{\text{t}} \wedge \varphi_{\text{s}}^{\text{g}}$ represents the guarantee of the desired robot's behaviors. For $\alpha \in \{e, s\}$, $\varphi_{\alpha}^{\text{i}}, \varphi_{\alpha}^{\text{t}}, \varphi_{\alpha}^{\text{g}}$ characterize the initial conditions, safety constraints, and liveness conditions, respectively. For symbolic repair (see Sec. V-C), we divide safety constraints $(\varphi_{\text{e}}^{\text{t}}, \varphi_{\text{s}}^{\text{t}})$ into skill constraints $(\varphi_{\text{e}}^{\text{t,skill}}, \varphi_{\text{s}}^{\text{t,skill}})$ and hard constraints $(\varphi_{\text{e}}^{\text{t,hard}}, \varphi_{\text{s}}^{\text{t,hard}})$, i.e., for $\alpha \in \{e, s\}$, $\varphi_{\alpha}^{\text{t}} = \varphi_{\alpha}^{\text{t,skill}} \wedge \varphi_{\alpha}^{\text{t,hard}}$. The skill constraints encode the pre and postcondition of skills (see Sec. V-B) and can be modified by repair, while repair cannot modify the hard constraints.

In practice, the GR(1) specifications are large in size due to the necessity of encoding terrain and task information. Since the exact current terrain and request state information is available at runtime, we can generate a shorter and more efficient version of the specification via *partial evaluation*, which substitutes a subset of the Boolean propositions with their truth values to reduce the state space.

Definition 1. Given an LTL formula φ and two subsets $S^{\texttt{True}}, S^{\texttt{False}} \subseteq AP$, $S^{\texttt{True}} \cap S^{\texttt{False}} = \emptyset$, we define the *partial evaluation* of φ over $S^{\texttt{True}}, S^{\texttt{False}}$, as $\varphi[S^{\texttt{True}}, S^{\texttt{False}}]$, where we substitute propositions $\pi \in AP$ in φ with `True` if $\pi \in S^{\texttt{True}}$ and with `False` if $\pi \in S^{\texttt{False}}$.

III. Problem Statement

Problem 1. Given (i) a global goal g_{global}, (ii) a set of possible local request goals $\mathcal{G}_{\text{local}}$ (iii) a set of predefined terrain polygons $\mathcal{P}_{\text{pre}}$ from user's prior knowledge, and (iv) a set of online terrain polygons $\mathcal{P}_{\text{on}}$ perceived during execution that may differ from the predefined ones, (v) a set of predefined locomotion gaits $\mathcal{L}$; generate controls that enable the robot to navigate the environment and reach the goal.

IV. Approach Summary

To efficiently solve Problem 1, we manage complexity at both symbolic and physical levels. At the symbolic level, we leverage reactive synthesis to decompose the local navigation problem into manageable subproblems whose solutions can be reused by synthesis for different scenarios. Each subproblem corresponds to finding controls for a short-horizon symbolic transition and is solved via a MICP to provide physical feasibility certificates of the transition. Our framework consists of offline synthesis (Sec.V) and online execution (Sec.VI). During the offline phase, we generate a set of potentially useful skills and corresponding locomotion gaits based on predefined terrain states and goals, while leveraging symbolic repair to find missing yet necessary symbolic transitions. At runtime, we leverage runtime repair to identify new symbolic transitions necessary for unforeseen terrain configurations or intermediate request goals.

V. Offline Synthesis

The offline synthesis module generates a strategy that enables the robot to reach local request goals over predefined terrain states. As shown in Fig. 2(a), this module takes in a set of local request goals $\mathcal{G}_{\text{local}}$ and a set of possible terrain polygons $\mathcal{P}_{\text{pre}}$ from prior knowledge of the workspace. In addition, the user provides a set of locomotion gaits $\mathcal{L}$. We first leverage the inverse grounding function G^{-1} to discretize the local environment, obtain a set of possible request states $\Sigma_{\text{req}}^{\text{poss}} \subseteq \Sigma_{\text{req}}$ from the local request goals, and characterize the predefined set of terrain polygons into a set of possible terrain states $\Sigma_{\text{terrain}}^{\text{poss}} \subseteq \Sigma_{\text{terrain}}$. Next, we solve a gait-fixed MICP problem to determine the feasibility of each potential skill given the locomotion gaits (Sec. V-A), and then encode all feasible transitions as skills in a specification (Sec. V-B). Next, we use a high-level manager to generate partial evaluations of the encoded specifications for efficient synthesis. If any partial evaluation is unrealizable, we use a repair process to suggest new robot skills and a gait-free MICP to check the feasibility of the suggested skills, eventually making the specification realizable (Sec. V-C).

$$\min_{\boldsymbol{\phi}, \mathbf{H}_{r,k,j}} \quad \sum_{i=0}^{N-1} \delta\boldsymbol{\phi}[i]^T \mathbf{Q}\, \delta\boldsymbol{\phi}[i] + \boldsymbol{\phi}[i]^T \mathbf{R} \boldsymbol{\phi}[i]$$

$$\text{(Dynamics)} \quad \begin{bmatrix} \mathbf{r}[i+1] \\ \dot{\mathbf{r}}[i+1] \\ m\ddot{\mathbf{r}}[i] \\ \boldsymbol{\theta}[i+1] \\ \dot{\boldsymbol{\theta}}[i+1] \end{bmatrix} = \begin{bmatrix} \mathbf{r}[i] + \Delta t \cdot \dot{\mathbf{r}}[i+1] \\ \dot{\mathbf{r}}[i] + \Delta t \cdot \ddot{\mathbf{r}}[i+1] \\ \sum_j \mathbf{f}_j[i] + m\mathbf{g} \\ \boldsymbol{\theta}[i] + \Delta t \cdot \dot{\boldsymbol{\theta}}[i+1] \\ \dot{\boldsymbol{\theta}}[i] + \Delta t \cdot \ddot{\boldsymbol{\theta}}[i+1] \end{bmatrix} \tag{1a}$$

$$\text{(Foothold)} \quad \forall i \in \mathcal{C}_{k,j}[0], r \in [R], k \in [n_s], j \in [n_f]$$

$$\mathbf{H}_{r,k,j} \Rightarrow \mathbf{A}_r \mathbf{p}_j[i] \le \mathbf{b}_r \tag{1b}$$

$$\mathbf{A}_{\text{eq},r} \mathbf{p}_j[i] = \mathbf{b}_{\text{eq},r}$$

$$\sum_{r=0}^{R-1} \mathbf{H}_{r,k,j} = 1$$

$$\mathbf{H}_{r,k,j} \in \{0, 1\}$$

$$\text{(Frictional)} \quad \mathbf{f}_j[i] \cdot \mathbf{n}(\mathbf{p}_j^{xy}[i]) \ge 0, \ \forall i \in \mathcal{C}_j \tag{1c}$$

$$\mathbf{f}_j[i] \in \mathcal{F}(\mu, \mathbf{n}, \mathbf{p}_j^{xy}[i]), \ \forall i \in \mathcal{C}_j \tag{1d}$$

$$\text{(Contact)} \quad \dot{\mathbf{p}}_j[i] = 0, \ \forall i \in \mathcal{C}_j \tag{1e}$$

$$\mathbf{f}_j[i] = 0, \ \forall i \notin \mathcal{C}_j \tag{1f}$$

$$\text{(Actuation)} \quad \mathbf{J}_j^\top \mathbf{f}_j[i] \le \boldsymbol{\tau}_{\max}, \forall i \in [N] \tag{1g}$$

$$\mathbf{I}\ddot{\boldsymbol{\theta}}[i] \le \boldsymbol{\tau}'_{\max}, \forall i \in [N] \tag{1h}$$

$$\text{(Kinematics)} \forall i \in [N], j \in [n_j] \tag{1i}$$

$$\mathbf{p}_j[i] \in \mathcal{R}_j({}^{\mathcal{B}}\mathbf{p}_j^{\text{ref}}, \mathbf{r}[i], \boldsymbol{\theta}_{\text{ref}}, \mathbf{p}_j^{\max}) \tag{1j}$$

Fig. 3. MICP formulation for physical feasibility checking.

A. Locomotion Gait and Feasibility Checking via MICP

We assume each locomotion gait L comes with a contact sequence $\mathcal{G}$ and corresponding time durations T. Therefore, we define the locomotion gait as $L = \mathcal{M}(\mathcal{G}, T)$. Each skill is assumed to have a unique locomotion gait to determine how the robot moves at the continuous level. As shown in Fig. 2(a), since all the possible request states $\Sigma_{\text{req}}^{\text{poss}}$ and terrain states $\Sigma_{\text{terrain}}^{\text{poss}}$ are given during the offline phase, it is straightforward to first identify all the potential skills at the symbolic level that allow the robots to move freely in all possible local environments. Each of them is evaluated by a gait-fixed MICP using the provided locomotion gaits.

For the MICP formulation shown in Fig. 3, the decision variables $\boldsymbol{\phi}$ include base position $\mathbf{r}$, velocity $\dot{\mathbf{r}}$, acceleration $\ddot{\mathbf{r}}$, orientation $\boldsymbol{\theta}$ parameterized by Euler angle and its first and second-order derivatives $\dot{\boldsymbol{\theta}}, \ddot{\boldsymbol{\theta}}$, individual end-effector (EE) position $\mathbf{p}_j$, velocity $\dot{\mathbf{p}}_j$, and acceleration $\ddot{\mathbf{p}}_j$, and individual contact force $\mathbf{f}_j$ for the foot j. We define binary variables $\mathbf{H}_{r,k,j}$, with r, k, and j expressing the r^{th} convex region, k^{th} footstep, and j^{th} foot. We use N, n_f, n_s, and R to represent the number of timesteps, the number of feet, the number of footsteps specified by the gait configuration for each foot, and the number of convex terrain polygons to be considered, respectively. For example, Fig. 2(b) shows a case with four polygons. The cost function consists of tracking costs and regularization terms governed by diagonal matrices $\mathbf{Q}$ and $\mathbf{R}$. The tracking costs include the deviation from a desired base trajectory interpolating a path between the terrain grid locations defined in each skill.

We encode the system dynamics as a simplified single rigid body in Eq. 1a to keep the dynamics constraint convex. To incorporate safe region constraints to select proper footholds, Eq. 1b restricts the robot's EE to stay within one of the convex polygons. We use $\mathcal{C}_{k,j}$ to denote the set of time steps indicating stance after the k^{th} footstep for the j^{th} foot and the safe region constraint only applies to the first stance time step represented as $\mathcal{C}_{k,j}[0]$. Similarly, the collision avoidance constraint keeps each EE within a collision-free box using additional binary variables, omitting details for brevity. Frictional constraints are defined in Eqs. 1c - 1d, with $\mathbf{n}$ as the normal vector of the terrain corresponding to the position $\mathbf{p}_j^{xy}$ and $\mathcal{C}_j$ as the set of timesteps indicating stance for the j^{th} foot. Contact constraints in Eqs. 1e - 1f check the given contact states and activate at different timesteps. Since the joint angles are omitted in the single rigid body model, we use a fixed Jacobian $\mathbf{J}_j(\mathbf{q}_j^{\text{ref}})$ at a nominal joint pose $\mathbf{q}_j^{\text{ref}}$ for each leg to approximately consider the torque limit constraint in Eq. 1g. In addition, since the translational and angular motions are decoupled, another actuation constraint on the angular acceleration is also added in Eq. 1h, where $\boldsymbol{\tau}_{\max}$ is the joint torque limit, $\boldsymbol{\tau}'_{\max}$ is the torque limit applied on the base, and $\mathbf{I}$ is the moment of inertia for the approximated single rigid body. Lastly, the kinematics constraint in Eq. 1j strictly limits the possible EE movements to assure safety, where $\mathcal{R}_j$ is defined as a 3D box constraint around a nominal foot EE position ${}^{\mathcal{B}}\mathbf{p}_j^{\text{ref}}$ based on the base position and reference orientation $\boldsymbol{\theta}_{\text{ref}}$, and constrained by a maximum deviation $\mathbf{p}_j^{\max}$. Due to the convex nature of this MICP, we can determine the feasibility of a possible symbolic transition by checking if the optimal solution exists.

B. Task Specification Encoding

After acquiring a set of feasible robot skills, we encode the skills into the specification φ as part of the environment safety assumptions $\varphi_{\text{e}}^{\text{t}}$ and system safety guarantees $\varphi_{\text{s}}^{\text{t}}$.

The skill assumptions $\varphi_{\text{e}}^{\text{t,skill}}$ encode the postconditions of the skills. The assumptions ensure that after the skill execution, at least one postcondition of the skill must hold: $\varphi_{\text{e}}^{\text{t,skill}} := \bigwedge_{o\in\mathcal{O}} \square(o \rightarrow \bigvee_{\sigma\in\Sigma_o^{\text{post}}} \bigwedge_{\pi\in\mathcal{I}_{\text{robot}}} \bigcirc(\pi = \sigma(\pi)))$. In Example 1, the postcondition of skill o_0 is defined as $\square(o_0 \rightarrow \bigcirc\pi_{x_0} \wedge \bigcirc\pi_{y_1} \wedge \neg \ldots)$.

The skill guarantees $\varphi_{\text{s}}^{\text{t,skill}}$ encodes the preconditions of the skills. The guarantees impose restrictions on when a skill can be executed. The guarantees only allow the robot to execute a skill if one of the skill's preconditions holds: $\varphi_{\text{s}}^{\text{t,skill}} := \bigwedge_{o\in\mathcal{O}} \square(\neg(\bigvee_{\sigma\in\Sigma_o^{\text{pre}}} \bigwedge_{\pi\in\mathcal{I}_{\text{robot}}\cup\mathcal{I}_{\text{terrain}}} \bigcirc(\pi = \sigma(\pi))) \rightarrow \neg \bigcirc o)$. In Example 1, the precondition of skill o_0 is defined as $\square\big(\neg(\bigcirc x_1 \wedge \bigcirc y_1 \wedge \bigcirc n_{x_1}^{y_0} = 1 \wedge \bigcirc n_{x_1}^{y_1} = 1 \wedge \bigcirc n_{x_1}^{y_2} = 1 \wedge \ldots) \rightarrow \neg \bigcirc o_0\big)$.

The hard constraints $\varphi_{\text{e}}^{\text{t,hard}}$ and $\varphi_{\text{s}}^{\text{t,hard}}$ are constraints that a repair cannot modify (see Sec. V-C). Our system's hard constraints $\varphi_{\text{s}}^{\text{t,hard}}$ only allow the robot to execute one skill at a time: $\square\big(\neg(\bigcirc o \wedge \bigcirc o')\big)$, for any two different skills $o, o' \in \mathcal{O}$. Given that, we encode the following hard assumptions $\varphi_{\text{e}}^{\text{t,hard}}$: (i) the uncontrollable terrain and request inputs cannot change during execution: $\square(\pi \leftrightarrow \bigcirc\pi)$, $\forall\pi \in \mathcal{I}_{\text{terrain}} \cup \mathcal{I}_{\text{req}}$, and (ii) the robot inputs $\mathcal{I}_{\text{robot}}$ remain unchanged if no skill is executed: $\square\big(\bigwedge_{o\in\mathcal{O}} \neg o \rightarrow \bigwedge_{\pi\in\mathcal{I}_{\text{robot}}}(\pi \leftrightarrow \bigcirc\pi)\big)$.

C. High-level Manager and Symbolic Repair

We design a high-level manager to efficiently synthesize controllers for the encoded specification under given sets of terrain and request states. The manager takes in the specification φ, a predefined set of terrain states $\Sigma_{\text{terrain}}^{\text{poss}} \subseteq \Sigma_{\text{terrain}}$, and a predefined set of request states $\Sigma_{\text{req}}^{\text{poss}} \subseteq \Sigma_{\text{req}}$. For every pair of integer terrain and request state $(\sigma_{\text{terrain}}, \sigma_{\text{req}}) \in \Sigma_{\text{terrain}}^{\text{poss}} \times \Sigma_{\text{req}}^{\text{poss}}$, the manager first translates the integer terrain state $\sigma_{\text{terrain}} : \mathcal{I}_{\text{terrain}} \rightarrow [n_t]$ to its corresponding Boolean terrain state $\sigma_{\text{terrain}}^{\mathcal{B}} : \mathcal{I}_{\text{terrain}}^{\mathcal{B}} \rightarrow \{\texttt{True}, \texttt{False}\}$, using a set of Boolean propositions $\mathcal{I}_{\text{terrain}}^{\mathcal{B}}$ to represent the integer terrain inputs $\mathcal{I}_{\text{terrain}}$, as done in [25]. We overload $\sigma_{\text{terrain}}^{\mathcal{B}}$ and σ_{req} to represent the inputs that are `True` in $\sigma_{\text{terrain}}^{\mathcal{B}}$ and σ_{req}. The manager then generates a partial evaluation $\varphi' := \varphi[\sigma_{\text{terrain}}^{\mathcal{B}} \cup \sigma_{\text{req}}, \mathcal{I}_{\text{terrain}}^{\mathcal{B}} \cup \mathcal{I}_{\text{req}} \setminus \sigma_{\text{terrain}}^{\mathcal{B}} \cup \sigma_{\text{req}}]$ (see Definition 1). Next, we remove skills that cannot be executed in the local environment, i.e., their preconditions are evaluated to `False` under the partial evaluation. Since the inputs of φ' only include the robot inputs $\mathcal{I}_{\text{robot}}$ and the outputs of φ' consist of fewer skills, we avoid the exponential blow-up of the synthesis algorithm in the size of the variables. Lastly, the manager synthesizes a strategy $\mathcal{A}_{\varphi'}$ for each φ'.

The partial evaluation φ' can be unrealizable if the gait-fixed MICP-based physical feasibility checking in Sec. V-A does not generate enough skills for the robot to reach the requested goal under the terrain state. To add more robot skills, we will leverage a gait-free MICP which retains all continuous decision variables and constraints from the gait-fixed MICP (formulated in Fig. 3) but modifies the contact state-dependent constraints so that the contact sequence and timing are no longer fixed. Since such a gait-free MICP is computationally expensive, we leverage a symbolic repair tool [23] to generate symbolic suggestions that guide us to perform necessary gait-free MICP only. Given an unrealizable φ', symbolic repair systematically modifies the pre- and postconditions of existing skills to suggest new skills that make φ' realizable, if such skills exist and are physically feasible. Fig. 1(c) illustrates the use of repair in Example 1. To create a skill that reaches the requested grid (x_0, y_0), repair selects the skill transition that moves the robot from the grid (x_1, y_0) to (x_2, y_0), and modifies the postcondition to be (x_0, y_0).

VI. Online Execution

The online execution module takes as input a set of online polygons $\mathcal{P}_{\text{on}}$ and corresponding integer terrain state $\sigma_{\text{terrain}} \in \Sigma_{\text{terrain}}$, an online local goal and its corresponding request state $\sigma_{\text{req}} \in \Sigma_{\text{req}}$, and a set of strategies $\mathcal{A} = \{\mathcal{A}_{\varphi'}\}$ from the offline synthesis module (Sec. V). As shown in Fig. 2(c), if the robot encounters an unseen terrain or request state, i.e.

$(\sigma_{\text{terrain}}, \sigma_{\text{req}}) \notin \Sigma_{\text{terrain}}^{\text{poss}} \times \Sigma_{\text{req}}^{\text{poss}}$, or fails to execute a symbolic transition, we leverage runtime repair to create new robot skills and synthesize a new strategy $\mathcal{A}_{\varphi'}$ (Sec. VI-A). Next, we execute the strategy $\mathcal{A}_{\varphi'}$ corresponding to $(\sigma_{\text{terrain}}, \sigma_{\text{req}})$. We execute each transition in $\mathcal{A}_{\varphi'}$ by solving a MICP with more detailed and accurate terrain information to generate a nominal trajectory (Sec. VI-B). We use a tracking controller to follow the nominal trajectory in real time. (Sec. VI-C). After reaching the request state, the robot resets the strategy with new terrain and request states, and repeats the execution until reaching the final goal state.

A. Runtime Repair

The runtime repair takes in a terrain state $\sigma_{\text{terrain}} \in \Sigma_{\text{terrain}}$, a request state $\sigma_{\text{req}} \in \Sigma_{\text{req}}$, and a Boolean formula $\varphi_{\text{disallow}}$ representing disallowed transitions discovered at runtime. We first update the system hard constraint $\varphi_{\text{s}}^{\text{t}}$ in the specification φ from Sec. V-B to be $\varphi_{\text{s}}^{\text{t}} \wedge \square\varphi_{\text{disallow}}$, so that the updated specification respects the newly discovered disallowed transitions, if any. We then leverage the high-level manager to create a partial evaluation of φ over σ_{terrain} and σ_{req}, and perform symbolic repair to generate new robot skills and a new strategy to handle σ_{terrain} and σ_{req}, as done in Sec. V-C.

B. Strategy Execution

The red path in Fig. 2(c) represents the automaton execution process. We first solve an online MICP before transitioning to a new symbolic state. The automaton advances only if the MICP successfully finds a solution. Slightly different from the offline gait-fixed MICP formulation in Eq. 1, the online MICP takes in online polygons $\mathcal{P}_{\text{on}}$ from a terrain segmentation module, the chosen gait, and initial and final robot states from the symbolic transition. If the MICP is not successful, we set the symbolic transition as a disallowed transition and perform runtime repair (Sec. VI-A).

C. Tracking Control

The tracking control module adopts an MPC-Whole-Body-Control (WBC) hierarchy, ensuring precise end-effector (EE) and centroidal momentum tracking. The MPC tracks the reference trajectory produced by the online MICP using a more accurate model of the robot's centroidal dynamics and kinematics, and is solved via the OCS2 library [27]. The state estimator fuses IMU data, joint encoders, and motion capture inputs to provide accurate body position information.

VII. Results

We present examples of maneuvering across diverse environments in a Gazebo simulation to demonstrate the framework's efficacy, scalability, and generalizability. Case studies on online execution highlight our framework's capability to handle unforeseen terrains and failures.

A. Setup

To demonstrate the generalizability of the proposed framework, we evaluate it on two scenarios with varying terrain types, safe stepping regions, and robot platforms–Unitree Go2 and SkyMul Chotu (a modified Unitree Go1 robot

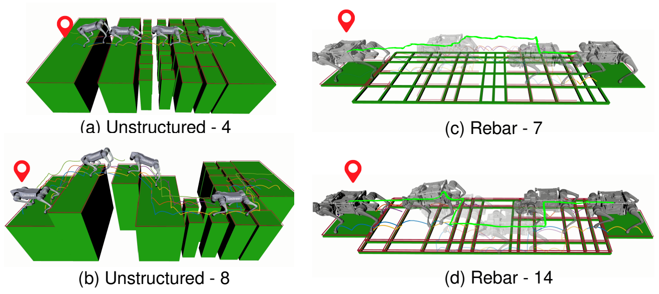


Fig. 4. Unstructured and rebar terrain scenarios.

dedicated for rebar tying tasks). We model obstacles as a distinct terrain type, with obstacle avoidance encoded as hard constraints in $\varphi_{\text{s}}^{\text{t}}$, and the requested waypoint are assumed not to be obstacles. We test both 3×3 and 5×5 grid abstractions. The 5×5 grid has a longer planning horizon and greater decision complexity. We use a 0.8 m cell size for the unstructured terrain and 0.6 m for the rebar, which can be adjusted for practical applications.

1) Unstructured Terrain: We abstract 8 terrain types–*flat terrain*, *high terrain*, *low terrain*, *dense stone*, *sparse stone*, *gap*, *high gap*, and *low gap*–with classification criteria based on polygon heights, counts, and areas relative to the abstracted cells. Note that a terrain is classified as a *gap* when the ratio of its overlapping area with the abstracted cell falls below a threshold. We define two terrain configurations, one with four selected terrain types and another with all eight terrain types, as shown in Figs. 4(a) and 4(b).

2) Rebar Terrain: Inspired by automating labor-intensive rebar tying tasks on construction sites [28], we explore a second case study where the quadrupedal robot navigates a rebar mat. We model each rebar as a rectangular polygon with a 3 cm width. Unlike the stepping stone scenario, this setup offers a larger number of potential stepping polygons due to the higher rebar density. Observing that the robot's traversability depends on rebar sparsity in different directions—whether aligned with the robot's facing direction or not, within a given tolerance—we abstract and classify rebar types based on a combination of horizontal sparsity (perpendicular to the robot's facing direction) and vertical sparsity (parallel to it). By calculating the number and relative spacing of the rebars inside each abstracted cell, we categorize each direction's sparsity as *dense* (0.05 - 0.15 m), *sparse* (0.15 - 0.35 m), or *extreme sparse* (above 0.35 m), *single*, *none*. To reduce the complexity of the problem, we remove some challenging combinations such as *none* or *single* in both directions. As a result, we define two example configurations with 7 and 14 rebar terrain types. The 7-type scenario in Fig. 4(c) generates random rebar sparsity between 0.15 and 0.35 m, excluding the *extreme sparse* type. In contrast, the 14-type scenario in Fig. 4(d) includes rebar sparsity from 0.15 to 0.6 m, potentially covering more challenging combinations including the *extreme sparse* ones.

B. Offline Synthesis Results

We evaluate the offline synthesis module for both *Unstructured* and *Rebar Terrain* by initializing the skills with two

TABLE I
OFFLINE SYNTHESIS AND REPAIR TIME

Scenario	Grid Size	Terrain and Request State Pairs (Success/Total)	Number of Skills (Original/New/Total)	Symbolic Repair Time (s)	Feasibility Check Time (s) (Gait-Fixed/Gait-Free)
Unstructured - 4	3×3	169/169	18/13/64	8.07	18.97/449.94
	5×5	129/158	19/9/64	77.16	20.26/241.42
Unstructured - 8	3×3	250/264	19/20/256	14.90	22.92/382.63
	5×5	179/246	20/16/256	93.21	18.59/196.88
Rebar - 7	3×3	196/196	18/11/196	7.44	15.63/417.55
	5×5	196/196	18/10/196	21.63	14.83/374.89
Rebar - 14	3×3	237/237	20/29/784	10.34	21.92/701.38
	5×5	196/196	20/18/784	24.31	23.06/453.94

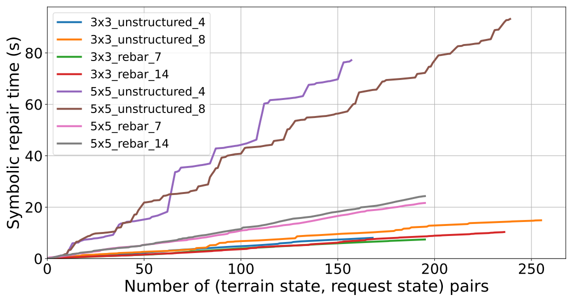


Fig. 5. Symbolic repair time in the size of terrain and request state pairs.

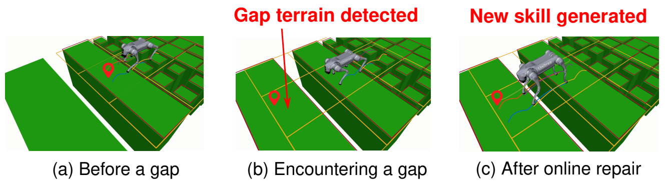


Fig. 6. Runtime repair scenario when encountering a gap.

trotting gaits (2 s and 3 s). Before synthesis, we generate the set of possible terrain states $\Sigma_{\text{terrain}}^{\text{poss}}$ (supposedly provided by the user) by systematically sweeping a local grid across the entire terrain map based on the best available knowledge. The request states $\Sigma_{\text{req}}^{\text{poss}}$ are defined as the top row of the local grid map in the robot's facing direction, along with the two corner cells to allow additional movement directions.

Table I reports (i) the number of terrain and request states pairs $(\sigma_{\text{terrain}}, \sigma_{\text{req}}) \in \Sigma_{\text{terrain}}^{\text{poss}} \times \Sigma_{\text{req}}^{\text{poss}}$, including both successful and total ones, (ii) the number of skills, including the original, the newly discovered, and the total possible ones, and (iii) offline repair and feasibility checking times, including both from gait-fixed and gait-free MICP. We first note that repair solves 93.4% of the terrain and request state pairs. The rest are unrepairable because the request states are unreachable due to obstacles or physical infeasibility. As highlighted in red, the number of newly discovered skills is $71.6 - 97.6\%$ smaller than checking all possible skills, each of which corresponds to solving an expensive gait-free MICP that takes 27.23 s on average (max 145.66 s) for a new locomotion gait. This demonstrates that offline repair effectively minimizes the number of gait-free MICP solves.

To investigate the scalability of offline repair, we perform repair across various sizes of the terrain and request states. As shown in Fig. 5, the repair runtime grows linearly in the size of the terrain and request states for both 3×3 and 5×5 grids. While fewer terrain and request states handled offline reduce checked time, they may lead to more frequent unforeseen states during online execution. Moreover, we note that the slope of 5×5 cases in Fig. 5 are steeper than those of 3×3 as a result of the increased state space. On average, repair takes 478.1% more time for 5×5 grids than 3×3 for one terrain and request states pair. In practice, the perception range and the robot size limit the grid size, so we do not test beyond 5×5 grids, though the user can adjust the resolution.

C. Online Execution Results

Fig. 4 visualizes the snapshots of Go2 and Chotu traversing the terrains during online execution. The robot is tasked with reaching a global waypoint using a naive global shortest path planner. Specifically, the local waypoint in the local grid map is selected by choosing the closest, non-obstacle grid cell toward the global waypoint. Fig. 4(a) shows a maneuver traversing through flat terrain, dense stepping stone, sparse stepping stone, and a gap. Fig. 4(b) shows the robot navigate through all eight terrain types, including elevation variations by adaptively choosing the locomotion gaits. Similarly, Fig. 4(c) and (d) present rebar traversing in separate scenarios with potentially 7 and 14 rebar terrain types. Notably, the robot leverages a set of newly discovered leaping gaits with short aerial phases for transitions from lower to higher terrain or across gaps and extreme sparse rebars as shown in Fig. 4(b) and (d). On average, the online gait-fixed MICP takes 430 ms to solve the unstructured terrain cases and 1.1 s for rebar cases when the collision avoidance is disabled. The rebar scenarios exhibit 155.8% longer solve times due to their overlapping, skewed rebar arrangement, and a larger number of terrain polygons. Additionally, enabling collision avoidance (necessary for *Unstructured - 8* case to handle the elevation change) increases the number of binary variables by 271.0%, increasing the solve time to 2.65 s.

As a case study to demonstrate the capability of handling unforeseen terrains and failures during execution, we highlight the following run-time scenarios:

1) Unforeseen Terrain States: With limited knowledge of the global terrain polygons, the terrain states provided during offline synthesis may be insufficient when encountering new terrain configurations. Fig. 6 illustrates this in an Unstructured - 4 types scenario with a 3×3 grid size, where terrain states involving *gap* terrain are excluded, treating *gap* as an unseen terrain type during offline synthesis. When the robot reaches a new terrain state that includes a *gap*, we trigger runtime repair for the current terrain and request states. A new skill with a leaping gait is generated to successfully cross the gap. The runtime repair takes 12.36 s, where symbolic repair takes 0.18 s, and the gait-free MICP takes 12.18 s to generate the new leaping gait.

2) Solving Failure: Another common runtime failure arises from MICP solving failures. Due to discrepancies between the predefined polygons used in offline synthesis and the actual online terrains—such as variations in shape and size—a skill deemed feasible offline may become infeasible when executed online, even if classified under the same symbolic transition. Originally moving straight ahead, the detours shown as green lines in Fig. 4(c) and (d) illustrate the re-synthesis process triggered by solving failures in both the 7-type and 14-type cases. Solving failure occurs more frequently in the rebar scenario than in the unstructured terrain scenario due to uncertainties in the shapes and sizes of online rebar polygons. The runtime resynthesis takes 0.11 s and 0.07 s for the two cases respectively. Both cases do not trigger runtime repair because other existing skills suffice to navigate the robot to the goals under the solving failures.

VIII. Conclusion

In this work, we presented an integrated planning framework for terrain-adaptive locomotion that combines reactive synthesis with MICP, enabling safe and reactive responses in dynamically changing environments. Our approach not only enhances motion feasibility and safety but also provides a scalable solution for legged robots maneuvering in complex and safety-critical environments. Future work will involve extensive hardware testing to validate the proposed framework and generalization to heterogeneous robot teaming [29]. Additionally, we plan to refine our symbolic repair approach to relieve the computational burden further via learning-based methods such as large language models.

References

[1] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2018, pp. 1–9.

[2] D. Kim, J. Di Carlo, B. Katz, G. Bledt, and S. Kim, "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control," *arXiv preprint arXiv:1909.06586*, 2019.

[3] Z. Zhou, B. Wingo, N. Boyd, S. Hutchinson, and Y. Zhao, "Momentum-aware trajectory optimization and control for agile quadrupedal locomotion," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7755–7762, 2022.

[4] P. Fankhauser, M. Bjelonic, C. D. Bellicoso, T. Miki, and M. Hutter, "Robust rough-terrain locomotion with a quadrupedal robot," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5761–5768.

[5] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter, "Perceptive locomotion through nonlinear model-predictive control," *IEEE Transactions on Robotics*, 2023.

[6] H.-W. Park, P. M. Wensing, and S. Kim, "Online planning for autonomous running jumps over obstacles in high-speed quadrupeds," in *Robotics: Science and Systems*, 2015.

[7] D. Kim, D. Carballo, J. Di Carlo, B. Katz, G. Bledt, B. Lim, and S. Kim, "Vision aided dynamic exploration of unstructured terrain with a small-scale quadruped robot," in *IEEE International Conference on Robotics and Automation*. IEEE, 2020, pp. 2464–2470.

[8] S. Gilroy, D. Lau, L. Yang, E. Izaguirre, K. Biermayer, A. Xiao, M. Sun, A. Agrawal, J. Zeng, Z. Li *et al.*, "Autonomous navigation for quadrupedal robots with optimized jumping through constrained obstacles," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2021, pp. 2132–2139.

[9] J. Norby and A. M. Johnson, "Fast global motion planning for dynamic legged robots," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2020, pp. 3829–3836.

[10] M. Chignoli, S. Morozov, and S. Kim, "Rapid and reliable quadruped motion planning with omnidirectional jumping," in *International Conference on Robotics and Automation*. IEEE, 2022, pp. 6621–6627.

[11] 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," *The International Journal of Robotics Research*, p. 02783649221077714, 2022.

[12] A. Shamsah, Z. Gu, J. Warnke, S. Hutchinson, and Y. Zhao, "Integrated task and motion planning for safe legged navigation in partially observable environments," *IEEE Transactions on Robotics*, 2023.

[13] A. K. Valenzuela, "Mixed-integer convex optimization for planning aggressive motions of legged robots over rough terrain," Ph.D. dissertation, Massachusetts Institute of Technology, 2016.

[14] Y. Shirai, X. Lin, A. Schperberg, Y. Tanaka, H. Kato, V. Vichathorn, and D. Hong, "Simultaneous contact-rich grasping and locomotion via distributed optimization enabling free-climbing for multi-limbed robots," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 13 563–13 570.

[15] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *2014 IEEE-RAS international conference on humanoid robots*. IEEE, 2014, pp. 279–286.

[16] Y. Ding, C. Li, and H.-W. Park, "Kinodynamic motion planning for multi-legged robot jumping via mixed-integer convex program," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 3998–4005.

[17] N. Fey, R. J. Frei, and P. M. Wensing, "3d hopping in discontinuous terrain using impulse planning with mixed-integer strategies," *IEEE Robotics and Automation Letters*, 2024.

[18] B. Ponton, A. Herzog, S. Schaal, and L. Righetti, "A convex model of humanoid momentum dynamics for multi-contact motion generation," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2016, pp. 842–849.

[19] F. Risbourg, T. Corbères, P.-A. Léziart, T. Flayols, N. Mansard, and S. Tonneau, "Real-time footstep planning and control of the solo quadruped robot in 3d environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022, pp. 12 950–12 956.

[20] B. Acosta and M. Posa, "Bipedal walking on constrained footholds with mpc footstep control," in *2023 IEEE-RAS 22nd International Conference on Humanoid Robots*. IEEE, 2023, pp. 1–8.

[21] B. Aceituno-Cabezas, C. Mastalli, H. Dai, M. Focchi, A. Radulescu, D. G. Caldwell, J. Cappelletto, J. C. Grieco, G. Fernández-López, and C. Semini, "Simultaneous contact, gait, and motion planning for robust multilegged locomotion via mixed-integer convex optimization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2531–2538, 2017.

[22] 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.

[23] A. Pacheck and H. Kress-Gazit, "Physically feasible repair of reactive, linear temporal logic-based, high-level tasks," *IEEE Transactions on Robotics*, 2023.

[24] Q. Meng and H. Kress-Gazit, "Automated Robot Recovery from Assumption Violations of High-Level Specifications," in *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*. IEEE, Aug. 2024, pp. 4154–4161.

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

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

[27] F. Farshidian *et al.*, "OCS2: An open source library for optimal control of switched systems," [Online]. Available: https://github.com/leggedrobotics/ocs2.

[28] M. Asselmeier, J. Ivanova, Z. Zhou, P. A. Vela, and Y. Zhao, "Hierarchical experience-informed navigation for multi-modal quadrupedal rebar grid traversal," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 8065–8072.

[29] Z. Zhou, D. J. Lee, Y. Yoshinaga, S. Balakirsky, D. Guo, and Y. Zhao, "Reactive task allocation and planning for quadrupedal and wheeled robot teaming," in *IEEE International Conference on Automation Science and Engineering*. IEEE, 2022, pp. 2110–2117.