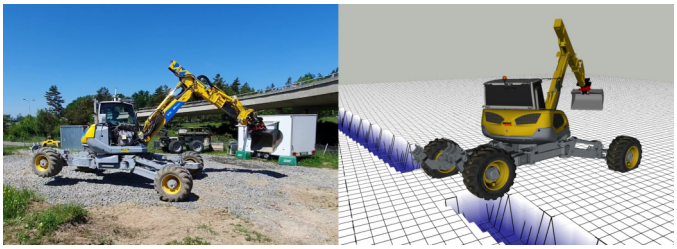
LSTP: Long Short-Term Motion Planning for Legged and Legged-Wheeled Systems

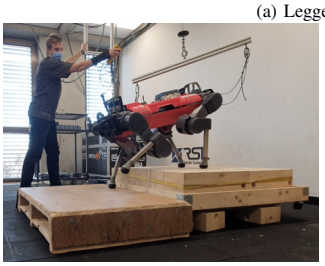
Edo Jelavic*, Kaixian Qu*, Farbod Farshidian and Marco Hutter

***Abstract*—This article presents a hybrid motion planning and control approach applicable to various ground robot types and morphologies. Our two-step approach uses a sampling-based planner to compute an approximate motion which is then fed to numerical optimization for refinement. The sampling-based stage finds a long-term global plan consisting of a contact schedule and sequence of keyframes, i.e., stable whole-body configurations. Subsequently, the optimization refines the solution with a short-term planning horizon to satisfy all nonlinear dynamics constraints. The proposed hybrid planner can compute plans for scenarios that would be difficult for trajectory optimization or sampling planner alone. We present tasks of traversing challenging terrain that requires discovering a contact schedule, navigating non-convex obstacles, and coordinating many degrees of freedom. Our hybrid planner has been applied to three different robots: a quadruped, a wheeled quadruped, and a legged excavator. We validate our hybrid locomotion planner in the real world and simulation, generating behaviors we could not achieve with previous methods. The results show that computing and executing hybrid locomotion plans is possible on hardware in real-time.**

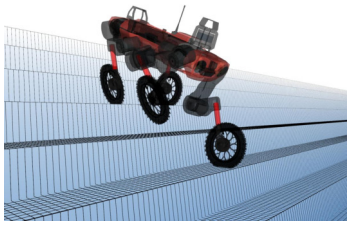
***Index Terms*—Whole-body motion planning, optimization, sampling, legged robot, legged-wheeled robot, legged excavator**



(a) Legged excavator



(b) Quadruped



(c) Wheeled quadruped

Fig. 1: ***a)*** HEAP overcomes a virtual bridge too narrow to drive over it. Hence, the stepping behavior emerges. We did not dig a ditch in our testing field. However, the map supplied to the planner contains it. *Left:* HEAP driving on three wheels over the bridge. *Right:* Visualization of the map the planner sees. ***b)*** ANYmal climbing consecutive steps. ***c)*** ANYmal on wheels stepping up the stairs (plan visualization). All the motions are also shown in the accompanying video https://youtu.be/iQiNAy6sLlo.

I. Introduction

MOTION planning is one of the fundamental problems in robotics, as it enables mobile robots to safely navigate in uncontrolled environments. Among different strategies for mobility, legs combined with wheels are the most promising solution when it comes to tasks that require a high level of mobility (legs) and efficiency (wheels) in challenging terrain [1]–[3].

The motion planning for hybrid platforms, i.e. systems with legs and wheels, is particularly challenging since, on one side, legs and wheels increase the number of Degrees of Freedoms (DoFs) the planner has to handle. On the other hand, motion planning for legged-wheeled robots is particularly complex since the combinatorial nature of stepping with legs (contact schedule) is combined with the non-holonomic nature of wheeled robots (rolling constraints). To tackle this challenging problem, we propose a motion planning framework for legged and legged-wheeled platforms based on different levels of model fidelity and different prediction horizons. In particular, a Numerical Optimization (NO) technique is used for short-term planning with high fidelity (e.g. full kinodynamic model) and randomized sampling for long-term planning with lower fidelity (full kinematics in quasistatic conditions).

NO is frequently used for planning and control with legged robots as it handles well high dimensional joint space, non-holonomic constraints, and nonlinear dynamics constraints. Examples can be found in the early work of [2], [4], [5], which follows the decoupled planning approach for the footholds and the base of the robot. With the advance of solvers and more computing power, researchers have started solving whole body planning problems in a receding horizon fashion fully onboard [6], whereby discrete foothold locations and continuous body motions are jointly optimized. However, a major limitation is the sensitivity to initialization and local optima, which are often caused by terrain constraints (e.g., collision avoidance) [7]. Similarly, researchers tackled the locomotion problem using Reinforcement Learning (RL) methods [8], [9]. Despite the unprecedented robustness and impressive field deployments, RL methods still struggle to achieve precise and coordinated foot placement required to negotiate terrains such as stepping stones [6], [10].

On the other hand, Sampling-Based Planners (SBPs) [11]–[14] can handle very non-convex environments since the randomization allows them to escape local minima. The robotic

This research was supported by the Swiss National Science Foundation (SNSF) as part of project No.188596, by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme grant agreement No 852044 and through the SNSF National Centre of Competence in Digital Fabrication (NCCR dfab).

All authors are with the Robotic Systems Lab, ETH Zurich, 8092 Zurich, Switzerland, contact: edo.jelavic@mavt.ethz.ch. *Edo Jelavic and Kaixian Qu contributed equally.

Manuscript received 2022; revised 2023.

community has been using SBP for planning contacts and whole-body motions on a variety of legged robots [15]–[18]. However, including kinodynamic constraints in a sampling-based planner remains an open research problem. Legged robots have many DoFs which often results in long planning times as SBP runtimes scale exponentially with configuration space dimension.

This work presents Long Short-Term Motion Planner (LSTP) combining the merit of optimization-based and sampling-based methods while generalizing to different robot types. The method is showcased on a quadruped, a wheeled-quadruped, and a legged excavator. The SBP computes contact schedule and keyframe (whole-body state) sequence quickly ($< 1\,\mathrm{s}$) thanks to offline-computed roadmaps. Subsequently, the NO refines the initial solution. Whole-body states from SBP act as attractors for the optimization preventing it from falling into bad local minima. Thus, we can handle both complex terrains (thanks to SBP), many DoFs, and complex system dynamics (thanks to NO). We run the NO in an Model Predictive Control (MPC) fashion, thus ensuring stability and robustness to disturbances [19].

A. Related Work

The robotic community has extensively studied motion planning for legged robots, and NO emerged as one of the most common methods, especially for short-term planning. Examples of optimization-based controllers for quadrupeds can be found in [4], [20] where Hierarchical Optimization (HO) satisfies system constraints while tracking base twist reference. Optimization has also been used to implement active suspension behaviour on legged-wheeled robots [21]–[23]. However, these works do not have a look-ahead horizon in the future and hence cannot overcome challenging obstacles.

Planning allows the robot to prepare for the oncoming obstacles and thus increases the chances of overcoming them. In [24], the authors use collocation to transcribe the problem into an Nonlinear Program (NLP) and solve it using IPOPT solver [25]. The optimization discovers whole-body maneuvers using all DoFs to overcome various obstacles. This approach has been extended to work with a legged-wheeled robot [26] and a legged excavator [7]. Similar examples of using NO for planning can be found in [27]–[29]. The authors above plan offline once and then track the plan using a tracking controller. In [24], [27], the tracking controller is based on HO from [4].

Keeping the contact schedule fixed and allowing the NO to optimize for the whole-body motion is a common practice in the legged-robot community. While NO can also optimize over the contact schedule [7], [24], it often results in a very non-convex NLP that is difficult to solve. This motivates the use of Mixed-Integer Programs (MIPs) and L1 norms for planning [30], [31]. MIPs have been successfully applied in robotics, e.g., for humanoid robots [32]. While this is an attractive problem formulation, the computation times scale exponentially with the planning horizon.

Planning in the receding horizon fashion reduces the need for an accurate tracking controller and increases the system's robustness. Many roboticists use NO in this fashion, and this paradigm is often called MPC. In [2], [5], [20], [33] the authors generate motions in a decoupled fashion (decoupled base trajectory and footholds) using Zero Moment Point (ZMP) criterion. The decoupled planning has also been applied to legged-wheeled robots [2], [34]. The optimization problem can be solved at high frequencies, up to $200\,\mathrm{Hz}$. However, decoupling the problem reduces the controller's generality.

Increased computing power and new methods based on differential dynamic programming [35] have enabled optimizing for base and limbs simultaneously [19]. Many recent works solve a full optimization problem using a centroidal or single-rigid-body model [6], [36]–[39]. Algorithm in [40] solves a hierarchical MPC, where a high-fidelity model (near future predictions) is combined with a lower-fidelity model (predictions further away in time). Optimizing over full dynamics without decoupling makes planned motions stabler [40]. Moreover, the MPC has progressed to the point where it is possible to incorporate the terrain constraints and solve the optimization in real time [41], [42]. [43] shows a two-stage optimization where a motion library is first computed offline and then executed using online MPC. We also follow the approach of using perceptive MPC as a backbone for our refinement planning stage.

While NO is a great tool for dealing with complicated nonlinear dynamics and many DoFs, it inevitably falls prey to local minima. Some researchers have tackled this problem by employing stochastic optimization [44]. While the planner can cope with challenging terrain, the motion plans take up to several minutes to compute. Another stream of research uses different types of planners to cope with local minima; most of them are based on Rapidly-exploring Random Trees (RRTs) [13], [14], Probabilistic Roadmaps (PRMs) [11] or grid-based methods (e.g. A* [45]).

Grid-based methods discretize the workspace and use a graph search algorithm (A* being a popular choice). Early works [46] can generate complex motions by defining a suitable potential field over the workspace. However, motions like crawling narrow passages and ladder climbing come at the cost of high computation times (up to $3\,\mathrm{h}$). More recently, [47], [48] propose motion planning for the legged-wheeled robot Momaro based on A* that runs in real-time by combing handcrafted stepping motions with a carefully designed cost function.

Early works utilizing SBPs generate complex maneuvers, such as free climbing or ladder climbing, yet with very long computation time (e.g., $16\,\mathrm{min}$ for 32 steps) [15], [16], [46]. [49] shows RRT-based whole-whole body motion planning with a Nao humanoid. The work builds upon ideas presented in ([50], [51]), where samples are drawn from pre-computed sets. While gracefully handling whole-body manipulation on flat ground, the approach is unsuitable for traversing obstacles since the set of stable configurations is strongly dependent on the terrain. Recently, more efficient approaches are proposed for humanoid and quadrupedal robots [17], [18], [52], [53]. SBPs can deal with very complex environments, and even compute limb contact schedule [17]. However, creating kinodynamically consistent motion directly in the SBP remains challenging, possibly leading to complex planning pipelines

and long planning times.

Replacing the SBP with an RL policy can be an effective alternative for foothold planning. In [54], RL is used to plan footsteps which are then tracked by the RL based controller. [55] also uses an RL footstep planner with a model-based tracking controller executing the plan. [56] uses a similar strategy except that the model-based tracking is happening in the latent space. While promising, these methods still require retraining for different types of terrains and cannot plan acyclic gait patterns.

Global planning typically relies on a sampling-based or grid-search technique with reduced model order. The resulting high-level planner is then combined with a local planner or controller [47], [57] that are often based on cyclic gait pattern. [58], [59] extends the SBP with neural network to predict the terrain difficulty for the underlying tracking controller. The introduced cost estimator requires re-training of the neural network estimator in case of a controller swap.

Combining sampling and optimization-based planning is a sound strategy to escape the local minima and enforce kinodynamic constraints. One possible combination is to solve a Boundary Value Problem (BVP) for computing connections inside the SBP. BVP can typically be solved either for low DoF systems (e.g., a vehicle) or with an analytical solution (e.g. [60]–[63]). [64], [65] computes transition feasibility for a humanoid robot and successfully transfers the solution to hardware. Another possibility is using sampling or grid-based search to provide an initial guess for the optimization and refine the motion plan. The two-stage approach has been widely used in the literature, most notably for manipulation problems [66]–[68]. In [69], motion plans for a trailer composition are calculated, and in [70], the authors plan whole-body motions for a humanoid robot without planning a contact schedule. We followed the idea of using the two-stage approach, and we extended it to be used for whole-body motion and contact planning.

This work computes approximate whole-body plans with an SBP and then refines them with MPC. We follow the idea of using contact dynamic roadmaps introduced in [52] to speed up the SBP which are an extension of dynamic roadmaps [71] for legged robot usage. We further extend the contact roadmap to handle robots with heavy limbs. Our planner leverages MPC from [19] for tracking and re-planning on a shorter time horizon. We propose a strategy for using kinematic MPC on legged excavators operating in challenging terrain where terrain adaptation is imperative, but precise torque control is often unavailable.

B. Contribution

This work presents LSTP, a combined sampling and optimization-based planner for mobile robots with many DoFs. The planner consists of a long horizon *Initialization Step* with SBP computing a contact schedule and a sequence of keyframe whole-body configurations and a short horizon *Refinement Step* with MPC satisfying dynamics and contact constraints. We introduce the following contributions:

- We extend the dynamic roadmap data structure to be applicable for robots with heavy limbs, such as legged excavators. Compared to [52], we achieve faster roadmap lookup. This allows our SBP planner to rapidly find plans under $0.5\,\mathrm{s}$ over challenging terrains such as stepping stones.
- The proposed LSTP generalizes to a variety of different legged mobile platforms. We exemplarily demonstrate this on a quadruped, a wheeled quadruped with non-steerable wheels, and a legged excavator with steerable wheels and an arm. So far, only generalizations for different legged robots have been shown in the [17]
- We present and apply for the first time a whole body MPC controller to a legged excavator, enabling it to execute motions on par with skilled human operators. We retain all the qualities from the controller presented in [27] and extend it to execute new motions
- We extensively verified the proposed planner in simulation and experimentally on three different robots, including a quadruped (ANYmal), legged-wheeled quadruped (ANYmal on wheels), and a legged excavator (Hydraulic Excavator for Autonomous Purpose (HEAP)).

With respect to our previous work [72], we generalize the SBP planner for different robots, show hardware experiments on multiple platforms and design a whole-body control system for HEAP. Lastly, we remove the need for offline optimization-based refinement and introduce a real-time MPC.

II. Preliminaries

Notations used for robot's limbs are the following: *LF* stands for Left Front, and similarly we have, *RF*, *LH*, *RH* (Right Hind). *ARM* denotes HEAPs arm. Label *EE* means End-Effector. Left superscript denotes the frame (e.g. ${}^{\mathcal{B}}\boldsymbol{p}$ means a position in the base frame $\mathcal{B}$) and defaults to world frame $\mathcal{W}$ if omitted. Right superscript indicates a component of a vector (e.g. $\boldsymbol{p}^z$ is the z component of $\boldsymbol{p}$). $\boldsymbol{T}$ denotes a homogeneous transform composed of translation $\boldsymbol{t}$ and rotation $\boldsymbol{R}$.

Our planner computes a trajectory that is T seconds long, given the starting base pose $\boldsymbol{T}_B(t=0) = \boldsymbol{T}_{B,start}$ and $\boldsymbol{T}_B(t=T) = \boldsymbol{T}_{B,goal}$ where $\boldsymbol{T}_{B,start}$ and $\boldsymbol{T}_{B,goal}$ are given poses of the robot's base. We always command the goal pose in $SE(2)$ since the height, roll, and pitch are well-defined by the terrain around the goal pose (see Sec. III-C1). No additional constraints on the joint configurations are imposed at $\boldsymbol{T}_{B,goal}$. Proposed pipeline is visualized in Fig. 2. The planner is divided into an *Initialization step* computing an approximate long-term plan $\boldsymbol{\Gamma}$ and a *Refinement step* computing a short-term plan satisfying all the constraints and exhibiting smooth motions.

The planner perceives the world via a multilayered map, which can be defined as mapping $f : \mathbb{R}^2 \to \mathbb{R}$ from (x, y) coordinates to various functions $f(x, y)$. It always receives elevation data $h(\cdot, \cdot)$ and then partitions the terrain (map) into *traversable* part $\mathcal{T}$ and *untraversable* part $\neg\mathcal{T}$. Any contact point needs to be in the traversable area $\mathcal{T}$:

$$(\boldsymbol{p}^x, \boldsymbol{p}^y) \in \mathcal{T} \equiv sdf_2(\boldsymbol{p}^x, \boldsymbol{p}^y) \geq \delta \geq 0, \tag{1}$$

where δ is a user-defined parameter and $sdf_2(\cdot, \cdot)$ represents a 2D Signed Distance Field (SDF) with the positive distance

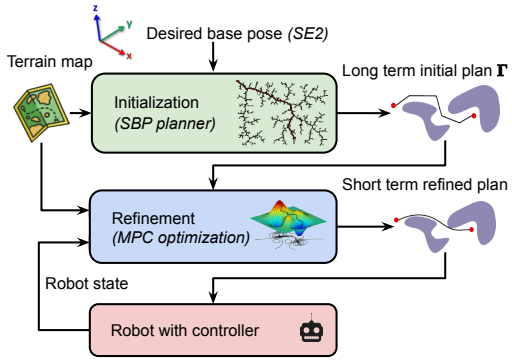


Fig. 2: Planning pipeline overview

meaning the point lies in $\mathcal{T}$. All contact points need to be at least δ away from the $\neg\mathcal{T}$. sdf_2 is calculated using [73] and stored as a layer in the map, relying on the *grid_map* implementation from [74].

A legged robot can in general have both limbs without wheels and limbs with wheels (e.g. HEAP has 4 legs with wheels and an arm). We say that i^{th} limb is in contact when it's close to the surface,

$$|\boldsymbol{p}^z_{ee,i} - h(\boldsymbol{p}^x_{ee,i}, \boldsymbol{p}^y_{ee,i}) - R_w| \leq \epsilon, \quad \epsilon \in \mathbb{R}_+, \tag{2}$$

where $\boldsymbol{p}_{ee,i}$ denotes the position of i^{th} limb End-Effector (EE) and $h(\cdot,\cdot)$ gives the terrain height. R_w is the wheel radius and is set to 0 for limbs without wheels. $\epsilon \in \mathbb{R}_+$ is used to control contact proximity. Eq. 2 assumes that the contact point is directly underneath the wheel's center, which is often violated. However, the tracking controller can compensate for this error (see experiment section).

III. Long Term Planning: Initialization Step

The *Initialization Step* computes an approximate long-term (global) solution to the planning problem. The approximate solution is a sequence of keyframe configurations with a feasible contact schedule. Each keyframe is a statically stable whole-body configuration with contact flags. Simple interpolation between keyframes can violate system constraints (e.g., rolling constraints); hence, the solution is only approximate. The *Initialization Step* uses RRTs [12], [14] to explore the space, but one could also use PRM planners [11]. This chapter first outlines the computation of limb roadmaps that happens offline. Limb roadmaps store the mapping between joint configurations and EE positions and are used to speed up foothold computation during the planning phase. Subsequently, the chapter describes the terrain preprocessing pipeline, triggered whenever the map is updated. The last section combines all the elements in a sampling-based planner.

A. Roadmap Precomputation

A roadmap is a mapping between i^{th} limb joint angles $\boldsymbol{q}_i$ and the EE position in the base frame ${}^{\mathcal{B}}\boldsymbol{p}_{ee,i}$. Each limb has its own base-pose invariant roadmap (expressed in the base frame). Examples of roadmaps are shown in Fig. 3. Generating a roadmap offline and using it online is an idea introduced in [52]. Here we extend the roadmap to store mapping between the i^{th} limb joint angles $\boldsymbol{q}_i$ and the position of its Center of Mass (CoM) ${}^{\mathcal{B}}\boldsymbol{p}_{com,i}$. The CoM location is needed to test the support polygon stability criteria for keyframes. The CoM position of a robot with limbs $\mathcal{I}$ can be computed as:

$${}^{\mathcal{B}}\boldsymbol{p}_{com} = \frac{1}{M}\Big({}^{\mathcal{B}}\boldsymbol{p}_{com,B}\, m_B + \sum_{i\in\mathcal{I}} {}^{\mathcal{B}}\boldsymbol{p}_{com,i}\, m_i\Big), \tag{3}$$

where M denotes the total mass, m_B the mass of the robot base, and m_i the mass of i^{th} limb. Evaluating Eq. 3 is much faster than computing the CoM from scratch, and it allows us to rapidly check the stability criterion during online planning. For robots with heavy limbs, such as HEAP or humanoid robots, evaluating Eq. 3 is essential to ensure stability.

Besides stability, no self-collision should occur for any given configuration of the legs. A simple way to prevent self-collision is to enforce the roadmap vertices to stay in their respective quadrants. For the HEAP arm, preventing all collisions in the roadmap generation might be too restrictive. Hence, the collision-free arm movement is planned during the online planning phase by invalidating portions of the roadmap colliding with legs or the terrain. Choosing the number of vertices in the roadmap involves a tradeoff. Many vertices approximate the workspace well but require more computation for online planning. We found that 4000-5000 vertices yielded an acceptable tradeoff for a legged robot. For HEAP, we used 300 vertices in the leg roadmap and 3000 vertices in the arm roadmap. The connection between neighboring vertices is rejected if it is longer than d_{max}. We used $d_{max} = 0.3\,\mathrm{m}$ for HEAPs legs and $d_{max} = 1\,\mathrm{m}$ for the arm. For the legged robot we used $d_{max} = 0.05\,\mathrm{m}$. Space around the robot is voxelized, and we store vertices within respective voxels. Voxelization accelerates configuration lookup since a good guess of foothold typically exists, e.g., around the default position. Finding candidate footholds takes $9 \pm 9\,\mu\mathrm{s}$ for a quadruped robot. We avoid transforming the terrain into the roadmap frame and search vertices only around the best guess instead of searching the entire roadmap and randomly sampling a feasible one [52]. In implementing their method, just transforming the terrain without any foothold search takes $35 \pm 10\,\mu\mathrm{s}$.

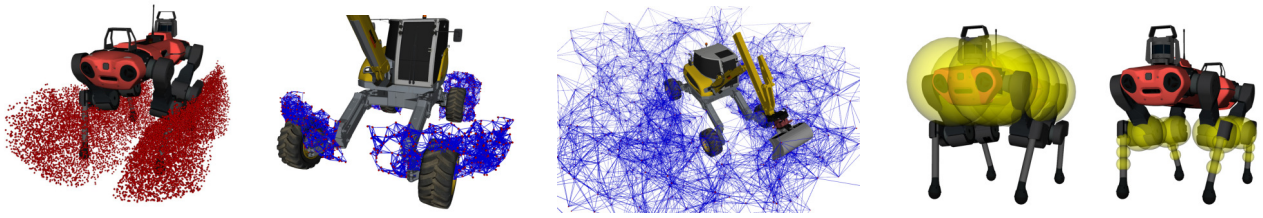
B. Terrain Preprocessing

The only input to the terrain preprocessing module is a raw elevation map (h) implemented using a grid map data structure [74]. From this elevation, we compute additional layers for surface normals, traversability SDF, elevation SDF, and filtered elevation. The relations between different tasks and their usages are illustrated in Fig. 4.

Surface normals are computed by fitting a tangent plane to a neighbourhood within a small radius of each grid cell (see Appendix A).

Traversability SDF is the SDF in 2D storing the distance of any points from the nearest untraversable region (see Fig. 5c). We refer to this field as sdf_2 in further text. More details about how terrain traversability is classified can be found in Appendix A.

Elevation SDF is a signed distance field in 3D which is used



(a) Roadmap ANYmal legs (b) Roadmap HEAP legs (c) Roadmap HEAP arm (d) Collision spheres for ANYmal base and limbs

Fig. 3: Roadmap vertices (red spheres) and edges (blue lines). Each red dot represents an end-effector position in $\mathbb{R}^3$. ***a)*** Roadmap for ANYmal robot (only vertices shown). Each leg has 5000 vertices in the roadmap. ***b)*** HEAP legs have 300 vertices and about 2000 edges in the roadmap. ***c)*** HEAP arm with 1000 vertices and about 8000 edges shown. ***d)*** Collision spheres approximating the geometry of the legged robot. We use ten balls of a radius of 20 cm for the base, two balls of 7 cm radius for each knee, and three balls of 3.5 cm radius for each shank.

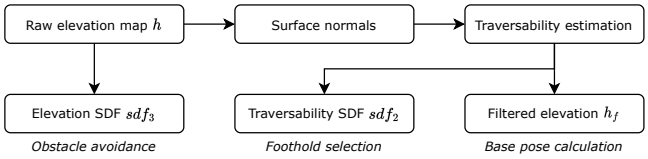


Fig. 4: Terrain preprocessing module overview.

for collision avoidance. The value is positive if the point is outside of the terrain and negative if the point lies inside the terrain. We refer to the 3D SDF field as sdf_3 in further text. sdf_3 is computed using the implementation from *grid_map* package which can be found open source[1].

Filtered elevation, denoted by h_f, is a smoothed version of the terrain height visualized in Fig. 5d. The smooth elevation is used by the base pose selection module (Sec. III-C1). Implementation details can be found in Appendix A.

C. Sampling-Based Planning

With the roadmap computed offline and the terrain preprocessed, we are ready to use the RRT based planner online. The planner attempts to connect $T_{B,start}$ and $T_{B,goal}$ geometrically (only considers the kinematics). The state space $\mathcal{S}$ is defined in Eq. 4 with n_J being the number of limb joints (actuated) and $\{0,1\}^{|\mathcal{I}|}$ contact states of all limbs $\mathcal{I}$. SBPs main components are described below.

$$\mathcal{S} := \mathbb{R}^3 \times SO(3) \times \mathbb{R}^{n_J} \times \{0,1\}^{|\mathcal{I}|}. \tag{4}$$

1) Sampling: Although the planner is a full 3D planner, the sampling module operates in *SE(2)* space which is motivated by the use of legged systems. Since legged robots locomote by interacting with the terrain, their base has to be in the vicinity of the terrain surface. Therefore, we uniformly sample the base's x, y position and the orientation in yaw γ and compute the remaining DoFs from the terrain features. Note that the base pose selection module operates on filtered terrain, which only retains prominent terrain features (see Sec. III-B). Roll α and pitch β are computed from the terrain normal $\boldsymbol{n}$ such that the robot roughly stays parallel to the surface below the base. The z coordinate can then be computed simply as $z = h_{des} + h_f(x, y)$ where h_{des} is a user-defined desired height above the ground and $h_f(x, y)$ is the filtered terrain height.

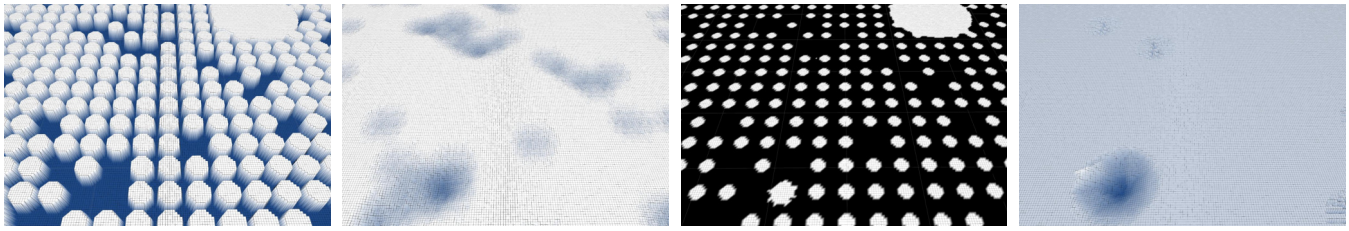
2) Expansion: Expansion refers to connecting the newly drawn sample to the rest of the tree. The maximum connection length is a tuning parameter ($15\,\mathrm{m}$ for HEAP and $5\,\mathrm{m}$ for the legged robot). The connecting path for the base is computed using Reeds-Shepp (RS) curves [75], which give an optimal path between two poses in *SE(2)* while respecting a minimal turning radius constraint. Upon drawing a random sample (x, y, γ), we use RS curve length d_{rs} to determine its nearest neighbors. The RS path is computed without considering any terrain information; it just helps enforce the rolling constraints for robots with steerable wheels. Using straight-line connections might render the problem infeasible in the *refinement step* (see Sec. V). For legged robots, we also use RS curves with small turning radii since they naturally minimize sideways motions, typically less stable and slower than locomoting forwards.

3) Feasibility Checking: SBP checks the connection feasibility before connecting the newly drawn sample to the rest of the tree. Unlike [17] where a *guiding path* is computed first, we do not add the connection into the tree before calculating the entire trajectory and ensuring its feasibility with four criteria.

- *Kinematic Reachability*: A contact must lie inside reachable workspace which is approximated by the roadmap.
- *Collision Avoidance*: Robot's base and limbs must not be in collision with the environment for each state along the proposed connection.
- *Valid Contact Positions*: All contacts need to lie within traversable terrain $\mathcal{T}$.
- *Stability Criteria (relaxed)*: CoM is allowed to deviate at most ϵ_s (parameter) from the support polygon edge. In addition, we impose a lower bound on the support polygon's area. For $\epsilon_s = 0$ the robot is statically stable.

Kinematic reachability can be verified by transforming roadmap vertices into the world frame and then using Eq. 2. This can be evaluated quickly using the pre-computed roadmap and the elevation map. The collision check can be easily enforced using sdf_3 from Sec. III-B. It suffices to ensure that collision spheres $B(\boldsymbol{c}, r)$ (ball of radius r centered at $\boldsymbol{c}$) are outside the terrain, i.e. $sdf_3(\boldsymbol{c}^x, \boldsymbol{c}^y, \boldsymbol{c}^z) \geq r$. Fig. 3d shows an example of collision geometry for the legged robot. Contact validity can be checked using sdf_2 from Sec. III-B and Eq. 1. Lastly, the algorithm checks the support polygon stability criterion, which does not hold in arbitrary terrain configuration, and care needs to be taken to ensure stability. We empirically found that the *refinement step* can stabilize

[1]https://github.com/ANYbotics/grid_map



(a) Raw Elevation (b) Elevated Mean (c) Untraversable (black) (d) Filtered terrain

Fig. 5: Intermediate steps in terrain preprocessing for stepping stones terrain. ***a)*** Raw elevation map input. ***b)*** Elevated mean (see Appendix A). ***c)*** Untraversable terrain shown in black color. ***d)*** Filtered terrain, used by the base pose selector module.

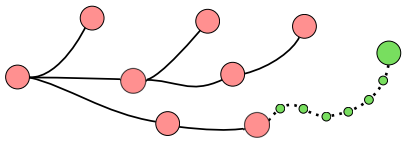


Fig. 6: Addition of a new node into the RRT tree (expansion step). Red circles and full black lines are nodes and paths that make the current RRT tree. The newly sampled node (green) has to be connected to the rest of the tree. For a successful connection, all subnodes on the connecting path (small green circles) have to be valid.

the robot even when the CoM projection is slightly outside the support polygon (controlled by ϵ_s).

Fig. 6 depicts the feasibility checking. Upon drawing a new sample (large green node) as described in Sec. III-C1, we compute a RS connection (dotted line) to the new node, as described in Sec. III-C2. Subsequently, the dotted line is discretized into subnodes (small green circles) using an RS interpolation method [76]. The subnodes do not necessarily be equidistant. Next, for each subnode we generate a full 6 DoF pose using the filtered elevation layer inside the *grid_map*. Finally, each subnode undergoes feasibility checking, ensuring that all criteria above are satisfied. If every subnode is feasible, the new state and the connecting path are added to the tree. For detailed description, refer to Sec. IV.

4) Cost Function: An optimizing SBP in the *Initialization Step* allows us to minimize a user defined cost function. Eq. 5 introduces our cost c, composed of RS path length $d_{rs}(\Gamma)$ and sum of robot-specific costs on each state $L(\boldsymbol{s})$ along the trajectory Γ.

$$c = d_{rs}(\Gamma) + \sum_{\boldsymbol{s}\in\Gamma} L(\boldsymbol{s}), \quad \Gamma \subset \mathcal{S}. \tag{5}$$

Robot specific costs are given with Eq. 6 (HEAP) and Eq. 7 (ANYmal and ANYmal on wheels). For HEAP, we penalize the legs going over untraversable terrain $\neg\mathcal{T}$ to prefer driving over stepping (as shown in our previous work [72]). ${}^{\neg\mathcal{T}}\mathbb{1}_{ee,i}$ is an indicator function with a value 1 if the i^{th} limb EE is inside the untraversable area $\neg\mathcal{T}$. w is the user-defined weight. For the legged robot, we penalize the roll (α) and pitch (β) angles of the base with different weights.

$$L(\boldsymbol{s}) = w \sum_{i\in\mathcal{I}} {}^{\neg\mathcal{T}}\mathbb{1}_{ee,i} \quad \text{for HEAP,} \tag{6}$$

$$L(\boldsymbol{s}) = w_\alpha |\alpha| + w_\beta |\beta| \quad \text{for others.} \tag{7}$$

IV. Feasibility Check

We introduce a subroutine called One-Step Motion (OSM), which does additional discretization between the green subnodes in Fig. 6 to ensure connection feasibility. We describe the algorithm for the legged robot first since it is the most general, and then the special cases for other robots.

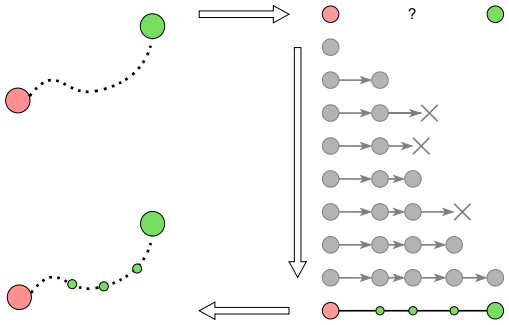


Fig. 7: Path validation using OSMs. Each gray node and arrow refers to an FSS and an OSM, respectively. Crosses represent failed OSM creations. Top to bottom: we first create an FSS for $\boldsymbol{v}_{nn}$, and then successfully generates the first OSM (the first length defined in lentgh candidates $\mathcal{L}$). The second OSM tries two lengths before the third length succeeds. The third OSM tries one length before the second one succeeds. Finally the last OSM connects to $\boldsymbol{v}_{new}$ in one try.

When all feet are grounded, the robot is in a Full-Support State (FSS). OSM is the transition between two FSSs, during which at most one swing phase (corresponding to one step) happens for each leg. We assume the robot constantly moves between two FSSs which will always result in a short full stance phase. This chapter describes how OSM is used inside RRT and subsequently all the subroutines.

A. Using One-Step Motion with RRT

Proposed OSM is used inside RRT expansion step. After sampling the base pose, the new vertex ($\boldsymbol{v}_{new}$) needs to be connected to the tree. We find a nearest neighbor in the tree ($\boldsymbol{v}_{nn}$) and use Alg. 4 (see Appendix B) to check validity between the two samples. If $\boldsymbol{v}_{nn}$ and $\boldsymbol{v}_{new}$ can be connected, Alg. 4 returns a sequence of OSMs that achieve the connection. The maximum length of the edge between $\boldsymbol{v}_{nn}$ and $\boldsymbol{v}_{new}$ can be tailored to different terrains.

OSMs do not have to be the same length. The planner receives a user-defined list of lengths for OSM, $\mathcal{L} = [l_1, l_2, \ldots, l_n]$ which is sorted in descending order. Empirically we found that the planner makes use of longer connections first, which achieve fast progress in easier terrains and resorts to using shorter connections when the terrain gets harder. The process of adding a new vertex with OSMs of different lengths is illustrated in Fig. 7.

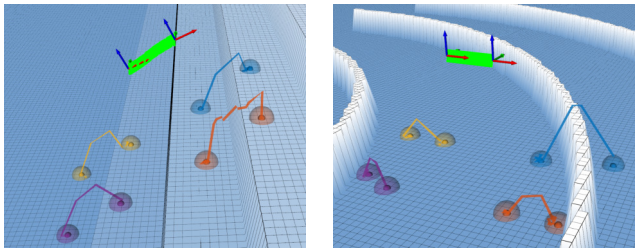


Fig. 8: OSM with length = 20 cm created on stairs ***(left)*** and obstacles ***(right)***. Terrain color represents the height value (white: maximum, blue: minimum). The base trajectory is shown in green, limb trajectories in separate colors. The spheres on the terrain show liftoff and touchdown locations.

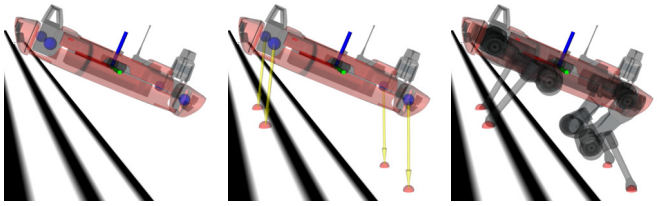


Fig. 9: FSS computation on stairs (white shows traversable terrain, black untraversable). ***Left:*** Base and nominal hip positions (blue spheres). ***Middle:*** Desired footholds (red spheres) computed using nominal hip positions. ***Right:*** FSS with leg configurations.

B. One-Step Motion Creation

OSM creation routine (Alg. 2 in Appendix B) receives a full state as a starting point and a base pose for the goal. It first generates an FSS for the goal base pose, then discretize the connecting RS path, find a feasible contact schedule, and create valid swing motions. Motion duration is computed in the end. The algorithm returns *true* if a feasible OSM is discovered and *false* otherwise. We illustrate an example OSM trajectory on two different terrains in Fig. 8.

1) Full-Support State Computation: FSS creation subroutine receives a base pose and computes joint configuration (one keyframe). We define nominal hip positions[2] which are shown with blue spheres in Fig. 9 on the left. A *desired* foothold is computed as a traversable point on the terrain surface closest to the nominal hip position. Subsequently, joint positions $\boldsymbol{q}$ are created by searching *valid* roadmap vertices in a user-defined neighborhood of the *desired* foothold. A vertex is considered *valid* when foot position is in contact with the traversable terrain $\mathcal{T}$ (Eq. 1 and Eq. 2) and the corresponding leg collision free (see Sec. III-C3). In case no valid vertex is found, FSS creation for this state returns failure. Fig. 9 shows FSS creation on the stairs. One FSS corresponds to a green subnode in Fig. 6 and Fig. 7.

2) Full-Support State Connection: The base path connecting the OSM start and the goal is discretized into subnodes with the detailed algorithm description found in Appendix B. In [52], two FSSs connect with a creep gait, i.e. the robot first moves the limbs and then the base with all 4 limbs on the ground. In contrast, our two-stage approach can accommodate for simultaneous motion of the base and swing limbs, which increases the overall motion speed by $35\,\%$ and allows overcoming larger obstacles.

3) Contact Schedule Computation: Once the OSM connection has been discretized into subnodes, the algorithm attempts to compute a feasible contact schedule. Generally, the robot has to break contact once the foot is at the edge of its workspace. This observation has been used in [17] where FIFO queue is used to decide which limb to recover first. In contrast, we allow for earlier contact breaks giving the planner more freedom to reshape the support polygon in anticipation of obstacles. First we find the latest contact breaking point and earliest contact establishing point for each limb. We define the Latest Contact Break (LCB) for limb i as the latest subnode when limb's swing phase must start (we must break contact because the limb is in the kinematic limits). The counterpart of LCB is the Earliest Contact Creation (ECC), i.e. the earliest subnode that can establish a valid contact with footholds at OSM goal state. Contact schedule is computed by solving a feasibility problem (see Eq. 33) where LCB and ECC define the constraints. Essentially, we seek to compute contact breaks no later than the LCB and contact creations no earlier than ECC. Examples of contact schedule computation are shown in Fig. 10 with detailed description given in Appendix B.

[2]computed by projecting nominal footholds onto the base frame's x-y plane

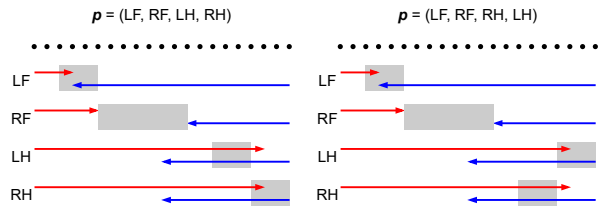


Fig. 10: Contact schedule computation for swing orders (LF, RF, LH, RH) and (LF, RF, RH, LH). The black dots on the top represents the OSM states. Each red arrow points from the start to LCB, and each blue one from the goal to ECC. Swing phases generated from Alg. 1 (see Appendix B) are shown as gray rectangles. There exist many other feasible contact schedules (e.g., move swing phase to the left, increase the size of swing phase).

4) Swing Motion Validation: The last step in OSM creation is to validate swing motion. We look for feasible swing trajectories after finding the contact schedule. Swing motion can be verified by invalidating portions of the roadmap and then performing the graph search, as suggested in [52]. However, in our approach, the roadmap is not stationary during a swing phase because the base moves. We instead search collision-free vertices near the desired swing position that are user-defined distance above the terrain.

5) Timing Computation: Transition duration between two states along the path is computed as follows:

$$\Delta t_l = t(i+1) - t(i) = \frac{1}{v_J}\Big(\|\boldsymbol{q}_J(i+1) - \boldsymbol{q}_J(i)\|_\infty \Big), \quad (8)$$

where $\boldsymbol{q}_J$ is the vector of joint positions and v_J is the expected mean joint velocity (same as [68], [72]).

C. Legged-Wheeled Robot

By modifying the LCB and ECC search, we can reuse the OSM concept for legged-wheeled robots such as ANYmal on wheels [2]. A description on how to find LCB and ECC for a legged-wheeled robot is given in Appendix C. Compared to legged robot duration Δt_l (see Eq. 8), one needs to consider the average driving speed. Transition duration between two states along the path is thus slightly different:

$$\Delta t_{lw} \;=\; \max\Big\{\Delta t_l, \frac{1}{v_B}\big(\|\boldsymbol{r}_B(i+1) - \boldsymbol{r}_B(i)\|_2 \big)\Big\}, \quad (9)$$

where $\boldsymbol{r}_B$ is the position of the base and v_B is the expected mean driving speed.

D. Legged Excavator

OSM method is slightly modified to work for systems like legged excavators, whereby the planner has to handle legs with wheels that can move and an arm that cannot move when in contact. We discretize the connection as described in Sec. III-C3 with maximal distance between the subnodes being $20\,\mathrm{cm}$ (green circles in Fig. 6).

It is most efficient to keep all four legs in contact and drive, which can be seen as FSS for HEAP. The planner allows configurations with minimum three contact limbs (e.g. 2 legs and an arm). When the contact state switches, e.g., HEAP needs to lift Left Front (LF) leg while Right Hind (RH) is airborne, we insert an FSS in between, which is a special case of OSM where the length is not predefined.

When the arm has to move, HEAPs base stops, and we use the roadmap to find a collision-free path similar to [52]. If the arm has to establish contact (e.g., for going over a gap), we find the corresponding contact establishing and contact breaking point. The arm contact position $\boldsymbol{p}^*$ is found by solving

$$\min \|\boldsymbol{p}_{cc} - \boldsymbol{p}_{cb}\| \tag{10}$$

and setting $\boldsymbol{p}^* = (\boldsymbol{p}^*_{cc} + \boldsymbol{p}^*_{cb})/2.0$, where $\boldsymbol{p}_{cb}$ and $\boldsymbol{p}_{cc}$ are positions of valid vertices in arm roadmap at contact break and creation subnodes. Each contact node's arm joint angles are computed solving Inverse Kinematics (IK) $\boldsymbol{q} = \mathrm{IK}(\boldsymbol{p}^*)$ as in [72].

V. Short Term Planning: Refinement Step

Optimization is the backbone of *Refinement step*. It refines the keyframe trajectory $\boldsymbol{\Gamma}$ found in the *Initialization step* to ensure smooth, stable, and feasible motions. $\boldsymbol{\Gamma}$ is a sequence of full body poses and contact states. Full states help the optimization convergence instead of just providing base poses (see Sec. VII-B). Optimization can modify the whole body states but not the Contact Schedule. As opposed to offline Trajectory Optimization (TO) [7], [72], we solve the optimization in an MPC fashion, which brings robustness during execution as it can compensate for small deviations from the reference $\boldsymbol{\Gamma}$. The optimization can deviate from the initial plan if fulfilling systems constraints or rejecting disturbances requires so.

The *Refinement Step* solves an Optimal Control Program (OCP) given below:

$$\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{u}} \quad & \Phi(\boldsymbol{x}(T)) + \int_0^T L(\boldsymbol{x}(t), \boldsymbol{u}(t), t)\, dt \\
\text{s.t.} \quad & \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \\
& \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \\
& \boldsymbol{g}_1(\boldsymbol{x}(t), \boldsymbol{u}(t), t) = \boldsymbol{0} \\
& \boldsymbol{g}_2(\boldsymbol{x}(t), t) = \boldsymbol{0} \\
& \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \geq \boldsymbol{0},
\end{aligned} \tag{11}$$

where L is a time-varying running cost, Φ is the cost at the terminal state, and T is the prediction horizon. The state and input are represented by $\boldsymbol{x}$ and $\boldsymbol{u}$, respectively. A Riccati-based solver solves the optimal control problem in Eq. 11. In particular, we use the Differential Dynamic Programming (DDP) method from [38] and the Direct Multiple Shooting (DMS) scheme from [6]. Both methods handle state-input equality constraints $\boldsymbol{g}_1$ using the projection technique [77]. Pure state equality constraints $\boldsymbol{g}_2$ are enforced using augmented Lagrangian [78]. The state-input inequality constraints $\boldsymbol{h}$ are handled using relaxed barrier functions as proposed in [38]. Transcription implementation, as well as optimization solvers, are publicly available in the OCS2 optimal control toolbox [79].

A. MPC for Legged Robot

OCP given with Eq. 11 has been solved using Sequential Linear Quadratic (SLQ) algorithm presented in [38]. We used prediction horizon of $T = 1\ s$ with the maximal step size of $0.01\ s$. Below we describe the dynamic model and constraints used in the MPC formulation.

1) System Dynamics: Legged robot state is composed of the position of the CoM $\boldsymbol{p}$ in world frame, Euler angles $\boldsymbol{\theta}$ ($Z - Y' - X''$ sequence) of the base, joint position $\boldsymbol{q}_{J,i}$ for leg $i = \{1, 2, 3, 4\}$, and the linear velocity ${}^{\mathcal{B}}\boldsymbol{v}$ and the angular velocity ${}^{\mathcal{B}}\boldsymbol{w}$ of CoM. The total number of joints for the robot is $n_J = 12$. Since the legs are much lighter than the body, CoM is assumed to be configuration invariant. Robot's state can be written as $\boldsymbol{x} = \left(\boldsymbol{p},\ \boldsymbol{\theta},\ \boldsymbol{q}_J,\ {}^{\mathcal{B}}\boldsymbol{v},\ {}^{\mathcal{B}}\boldsymbol{\omega}\right) \in \mathbb{R}^{24}$ and control input $\boldsymbol{u}$ as $\boldsymbol{u} = \left(\boldsymbol{u}_J,\ {}^{\mathcal{B}}\boldsymbol{\lambda}_{ee}\right) \in \mathbb{R}^{24}$, where $\boldsymbol{u}_J$ is the joint velocity and ${}^{\mathcal{B}}\boldsymbol{\lambda}_{ee,i} \in \mathbb{R}^3$ is the contact force at i^{th} EE. We use a kinodynamic model describing the dynamics of a single rigid free-floating body together with the kinematics for each leg. The Equations of Motion can be found in [38].

2) Cost Formulation: MPC should track $\boldsymbol{\Gamma}$, however it is allowed to deviate. Hence, deviation from the reference state is penalized quadratically. The reference twist is computed using finite differences, and the reference forces are equally distributed on stance feet. High-frequency input is also penalized as proposed in [80].

3) Constraints: We use $\mathcal{C}$ to denote the set of contact limbs.

Zero Force or Velocity Constraint: A contact leg cannot change its position while a swing leg cannot generate a contact force,

$$\begin{cases} \boldsymbol{v}_{ee,i} = \boldsymbol{0}, & i \in \mathcal{C}, \\ \boldsymbol{\lambda}_{ee,i} = \boldsymbol{0}, & i \notin \mathcal{C}. \end{cases} \tag{12}$$

Foot Constraint: The foothold of a contact leg must be placed at its reference from $\boldsymbol{\Gamma}$, while for a swing leg, its foot should maintain a certain height to avoid foot scuffing,

$$\begin{cases} \boldsymbol{p}_{ee,i} = \bar{\boldsymbol{p}}_{ee,i}, & i \in \mathcal{C}, \\ \boldsymbol{p}^z_{ee,i} \geq h(\boldsymbol{p}^x_{ee,i}, \boldsymbol{p}^y_{ee,i}) + c(t), & i \notin \mathcal{C}, \end{cases} \tag{13}$$

where $\boldsymbol{p}_{ee,i}$ is the foot position of i^{th} limb and $\bar{\boldsymbol{p}}_{ee,i}$ is the reference. $h(\cdot,\cdot)$ evaluates the terrain height. $c(\cdot)$ is the clearance function used to minimize the constraint violation and thus make the problem more numerically stable for the optimizer (see Fig. 11).

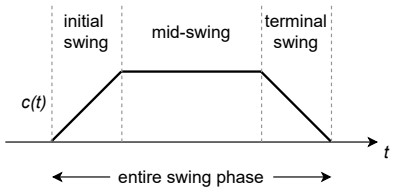


Fig. 11: The ground clearance function. It gradually increases to a user-defined value and then gradually decreases. Such a shape minimizes violation in the initial and terminal swing phases since feet cannot change state instantly.

Friction Cone Constraint: We use the perturbed cone proposed in [38] to avoid differentiability problems around the origin.

$$h_{cone,\epsilon} = \mu_c F_z - \sqrt{F_x^2 + F_y^2 + \epsilon^2} \geq 0, \tag{14}$$

where μ_c is the friction coefficient, ϵ is a control parameter, and $\boldsymbol{F} = [F_x, F_y, F_z]$ is the contact force expressed in the local frame of the terrain surface.

Push Force Constraint: To enforce stability margins, the vertical component of contact forces ${}^{\mathcal{W}}\boldsymbol{\lambda}_{ee,i}^{z}$ for contact legs needs to be larger than a user-defined value,

$${}^{\mathcal{W}}\boldsymbol{\lambda}_{ee,i}^{z} \geq \lambda_{z,\min}, \quad \forall i \in \mathcal{C}. \tag{15}$$

Leg Collision Constraint: sdf_3 is used to prevent collision of the knee and shin with the terrain,

$$sdf_3(\boldsymbol{c}) \geq r \quad \text{for all limb collision spheres } B(\boldsymbol{c}, r). \tag{16}$$

Limb collision spheres for ANYmal can be found in Fig. 3d.

B. MPC for HEAP

The OCP for HEAP is transcribed into a finite-dimensional NLP using DMS [81] and solved with Sequential Quadratic Programming (SQP). We used prediction horizon of $T = 15\,s$. Below we describe the differences in OCP formulation compared to the legged robot.

1) System Dynamics: HEAP state is given with $\boldsymbol{x} = \big(\boldsymbol{p},\ \boldsymbol{\theta},\ \boldsymbol{q}_J\big) \in \mathbb{R}^{31}$. Control input can be written as $\boldsymbol{u}$ as $\boldsymbol{u} = \big(\boldsymbol{v},\ \dot{\boldsymbol{\theta}},\ \dot{\boldsymbol{q}}_J,\ \boldsymbol{\alpha}\big) \in \mathbb{R}^{36}$. Euler angles $\boldsymbol{\theta}$ ($Z - Y' - X''$ sequence) parametrize the rotation, $\boldsymbol{p}$ is the base position, $\boldsymbol{q}_J$ are joint positions and $\boldsymbol{v}$ is the linear velocity of the base. $\boldsymbol{\alpha}$ is a vector of weights used to enforce robust support polygon constraint in Eq. 20. System dynamics can then be described as:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{S}\,\boldsymbol{u}(t), \tag{17}$$

where $\boldsymbol{S}$ is a selection matrix of the form $\boldsymbol{S} = \big[\mathbf{I}_{31\times 31}\ \mathbf{0}_{31\times 5}\big]$. Following our previous work [7], HEAP uses a kinematic model since such a heavy machine is operated in quasi-static conditions.

2) Cost Formulation: The cost function follows the same principle as for the legged robot, i.e., the solution should stay close to $\boldsymbol{\Gamma}$ computed by the SBP. Since HEAP has one hydraulic circuit that spins all wheels at approximately the same velocity, we add cost terms that help bridge the simulation to the hardware gap. The cost term below minimizes slip by penalizing the linear wheel velocity difference between the left and the right side.

$$L(\boldsymbol{x}) = \sum_{i,j\in\mathcal{C}_l} \big\|\boldsymbol{v}_{ee,i} - \boldsymbol{v}_{ee,j}\big\|_2^2, \tag{18}$$

where $\boldsymbol{v}_{ee,i}$ denotes the velocity of the wheel center. We use $\mathcal{C}$ for contact limbs (legs and arm) and $\mathcal{C}_l$ for contact legs only.

3) Constraints: Rolling Constraint prevents lateral slip, i.e. linear velocity of the wheel center stays perpendicular to wheel joint axis.

$${}^{\mathcal{E}_i}\boldsymbol{v}_{ee,i}^{y} = 0, \quad \forall i \in \mathcal{C}_l, \tag{19}$$

with ${}^{\mathcal{E}_i}\boldsymbol{v}_{ee,i}^{y}$ being y component of the wheel center velocity expressed in i^{th} end-effector frame $\mathcal{E}_i$. Optimization is not aware of the wheel joint, it treats the wheel as a moving contact point which reduces number of variables.

Stability Constraint: We use the robust support polygon constraint introduced in [7] which avoids the need to compute half spaces manually:

$$\begin{aligned} &\sum_{i\in\mathcal{C}} \Big(\alpha_i + \frac{\epsilon}{|\mathcal{C}|}\Big)\boldsymbol{p}_{ee,i}^{x,y} = \boldsymbol{p}_{com}^{x,y} \ \in \mathbb{R}^2, \\ &\sum_{i\in\mathcal{C}} \alpha_i = 1 - \epsilon, \quad 0 \leq \epsilon \leq 1,\ \alpha_i \geq 0, \end{aligned} \tag{20}$$

where α_i are part of the input vector $\boldsymbol{u}$, ϵ is a user-defined parameter, and $\boldsymbol{p}_{com}$ is the position of CoM. The bigger the ϵ, the more conservative we are, i.e., the minimum permitted distance of CoM to the edge of the support polygon becomes larger. We set $\epsilon = 0.1$ in this work.

Swing Limb Constraint: We follow the same formulation as for the legged robot. The user-defined height threshold was $0.6\,\mathrm{m}$ for the legs and $0.9\,\mathrm{m}$ for the shovel.

Contact Limb Constraint: We relax the foothold constraint from Eq. 13; MPC can optimize contact position as long as the limb stays grounded within the traversable area (Eq. 1 and Eq. 2).

VI. Control

A. Legged Robot Control

The Low-level Whole Body Controller (WBC) for the legged robot is based on the hierarchical optimization from [4]. It considers full nonlinear rigid body dynamics of the robot and it solves a series of Quadratic Programs (QPs) tasks in a prioritized order through nullspace projection. Table I shows tasks and priorities used. WBCs output torques are augmented with a PD term for stance legs. For more details on the specific WBC version, the reader is referred to [6].

B. HEAP Control

Compared to the legged robot and our previous work [27], we omit the WBC based on HO [4]. Instead, through the use of Virtual Model Controller (VMC) we retain the good qualities from our previous work [27] including the terrain adaptation. The new controller can use the arm as a support limb and it shows higher robustness. Lastly, the presented controller relies on existing tuning of hydraulic cylinders which removes the

TABLE I: WBC task setup. Smaller number indicates higher priority.

Priority	Type	Task
1	=	Floating base equations of motion
1	$\leq$	Joint torque limits
1	$\leq\leq$	Joint position limits
1	$\leq\leq$	Joint velocity limits
1	$\leq$	Friction cone
1	=	Stance legs no contact motion
2	=	Swing leg tracking
3	=	Base orientation tracking
3	=	Base translation tracking
4	=	End-effector force tracking

need for laborious tuning of the low-level torque control loops. Below we describe how VMC is combined with MPC and how each individual joint is controlled.

1) Virtual Model Control: Motion of the CoM is governed by the net resulting force acting on it. Assuming quasi-static conditions ($\ddot{\boldsymbol{p}} = \ddot{\boldsymbol{\theta}} = 0$), virtual force f_v and virtual moment $\boldsymbol{m}_v$ can then be computed by PID control law. This is expressed in Eq. 21.

$$Mg + \sum_i \boldsymbol{\lambda}^z_{c,i} = f_v = \text{PID}(z_{des} - z_{meas}), \tag{21}$$
$$\sum_i \boldsymbol{r}_{c,i} \times \boldsymbol{\lambda}_{c,i} = \boldsymbol{m}_v = \text{PID}(\boldsymbol{\theta}_{des} - \boldsymbol{\theta}_{meas}).$$

$\boldsymbol{p}$ is the position, g is the gravitational acceleration, and $\boldsymbol{\theta}$ is rotation parametrized by Euler angles. $\boldsymbol{\lambda}_{c,i}$ is a contact force at i^{th} contact point and $\boldsymbol{r}_{c,i}$ is the distance from the contact point to the CoM. We only regulate the height in z direction, roll θ_r and pitch θ_p angle of the base. The contact forces required to produce desired virtual force and moment can then be recovered by solving a QP:

$$\begin{aligned} \min_{\boldsymbol{\Lambda}} \quad & \boldsymbol{\Lambda}^T \boldsymbol{W} \boldsymbol{\Lambda} \\ \text{s.t.} \quad & \boldsymbol{\Lambda}_{lb} \leq \boldsymbol{\Lambda} \leq \boldsymbol{\Lambda}_{ub} \\ & \boldsymbol{A}\boldsymbol{\Lambda} = \boldsymbol{b}, \end{aligned} \tag{22}$$

where $\boldsymbol{\Lambda}$ is a stacked vector of contact forces, $\boldsymbol{W}$ is a weighting matrix and the constraint $\boldsymbol{A}\boldsymbol{\Lambda} = \boldsymbol{b}$ arises from the geometry of the contact points w.r.t. to the CoM (Eq. 21). The solution of Eq. 22 yields minimal contact forces which are distributed as equally as possible. The joint torques can then be recovered with $\boldsymbol{J}^T\boldsymbol{\Lambda} = \boldsymbol{\tau}$ where $\boldsymbol{J}$ is the stacked contact Jacobian matrix. Compared to our previous work [82], we include the arm into the control structure as a contact point.

2) HEAP Whole Body Controller: While MPCs purpose is to track the desired plan, VMC makes the system terrain-adaptive and compliant. In practice we cannot acquire a full accuracy map with on-board sensors. Furthermore, such a map has too many details which make the gradients non-smooth. Hence the system needs to plan using simplified geometry and adapt to the unmodelled terrain at runtime. We achieve this by utilizing a minimal set of torque-controlled joints. An important consideration is that hydraulic cylinders have a lot of friction, and in some cases (e.g. tele joint in Fig. 12) the friction is even position dependent. However, if the joint supports lot of weight, the amount of friction torque in the total torque is relatively low and the resulting control performance is better compared to non-load bearing joints. We control Hip Flexion-Extension (HFE) and Telescopic (TELE) joints in torque mode given that these joints bear the most load when corresponding limb is in contact. Since Contact Schedule (CS) allows only configurations with at least three contacts, there is always enough DoFs to control roll, pitch and height.

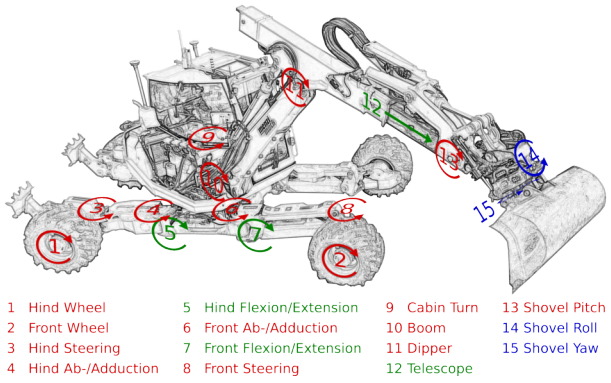


Fig. 12: HEAP joints. Blue joints are always frozen and red joints are always velocity controlled. Green joints are torque controlled when corresponding limb is in contact, velocity controlled otherwise. Figure adapted from [3].

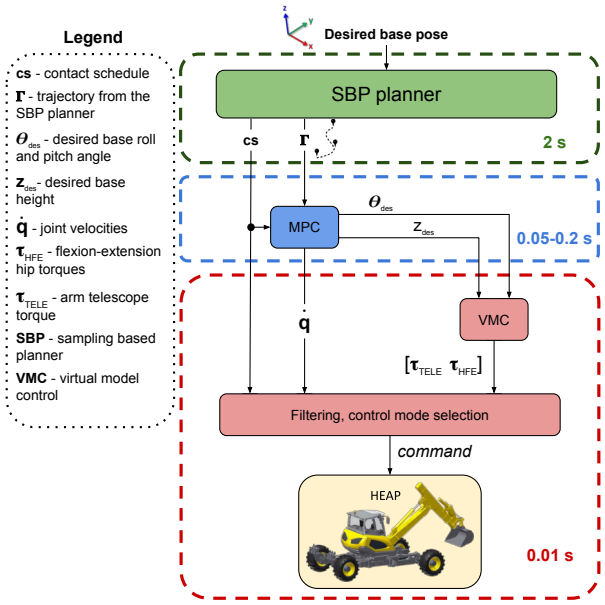


Fig. 13: HEAP control system with layer execution times on the right. MPC tracks trajectory Γ from SBP using a kinematic model from Eq. 17. VMC computes torques for Hip Flexion-Extension and arm Telescopic joint to track roll, pitch and, base height setpoints from the MPC. Joint control mode is selected based on contact schedule.

Fig. 13 shows the structure of combined MPC and VMC controllers. The MPC receives a trajectory $\boldsymbol{\Gamma}$ from the SBP. Optimized roll, pitch and height are fed to VMC which then computes HFE and TELE joint torques to achieve desired base pose. Lower level controller selects joint control mode based on the contact schedule CS. All joints are velocity controlled except for HFE and TELE if the corresponding limb is in contact. Both torque and joint velocity commands are filtered using exponential smoothing to remove small jumps. MPC is tuned to accurately follow the plan from the SBP. MPC uses a prediction horizon of 15 s, which is enough to correct for small deviations around the long term plan $\boldsymbol{\Gamma}$. In contrast, the VMC control loops are tuned for good disturbance rejection since VMC sees the terrain as a disturbance. VMC uses a state machine to handle late touchdown events. A swing

leg that should be in contact (based on contact schedule) is commanded a constant velocity in HFE joint until contact is established. Late touchdowns require dedicated treatment since they may result in instability. We found that early touchdowns are not problematic; they merely degrade the base tracking performance. With dedicated late touchdown handling, we could perform stepping motion in terrain with a roughness of about ± 50 cm.

To execute some of the maneuvers shown in the result section, HEAP needs to control wheel velocities. The lack of wheel speed measurements (no oil flow sensor in the wheel rotary actuators) is mitigated by closing the PI control loop over the base velocity. Given desired wheel velocities ω we can compute linear velocities of the wheel center $v_{center} = \omega R_w$ (R_w is wheel radius). The desired base speed $v_{B,des}$ is approximated by taking the average over grounded wheel linear velocities. Base speed v_B in a local frame can be estimated by differentiating the base position and taking a dot product with the heading of the base. The final estimate is obtained after applying an exponential filter.

Control signal u is computed with a PID control law with feedforward term $k_{ff}\, v_{B,des}$,

$$u = \text{PID}\big(v_{B,des} - v_B\big) + k_{ff}\, v_{B,des}, \tag{23}$$

u is applied to all four wheels since they cannot be controlled individually.

VII. Results

The proposed LSTP is implemented entirely on Central Processing Unit (CPU). All components are implemented in C++ programming language with ROS as a communication middleware. SBP uses the OMPL library [76] and MPC is implemented using OCS2 library [79], both of which are available as open source. OCS2 relies on Pinocchio library [83] for rigid body dynamics and all the derivatives are computed using CppAd framework [84]. Terrain processing is common to all robots. This chapter presents results with a different robot in separate sections. For a better impression please refer to the video[3].

A. Terrain Preprocessing

Normal estimation is a key component in our terrain preprocessing pipeline. It is used by the traversability estimation and later by the base pose selector. Fig. 14 left shows the runtime of normal estimation on a fixed size 8m $\times$ 8m map at 3 cm resolution with varying normal estimation radius R. The algorithm complexity is $O(R^2)$ since the number of points used for estimation is proportional to the neighborhood surface (see Appendix A). The total runtime of our processing pipeline with varying map sizes (resolution is always 3 cm) is shown in the middle. The pipeline can maintain more than 1 Hz real-time rate updates, even for 20×20 meter maps. The right plot shows the percentage of time spent in each submodule, with the vast majority of time spent filtering the terrain. sdf_3 is excluded from Fig. 14 right since it is computed using open-source software; its average runtime is 115 ms for an 8 m $\times$ 8 m map and z coordinate between -0.5 m and 1.9 m.

[3]https://youtu.be/iQiNAy6sLlo

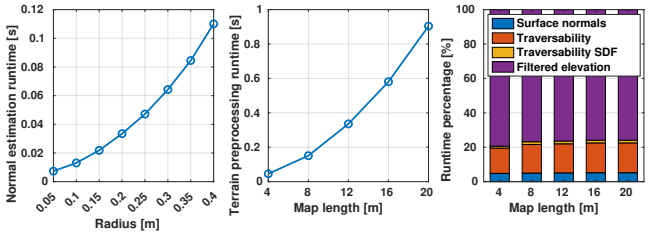


Fig. 14: Terrain processing runtime analysis with 6 threads. ***Left:*** Normal estimation runtimes for 8×8 meters map (resolution = 3 cm) and different radii. ***Middle:*** Total runtimes for different map sizes at a resolution of 3 cm. ***Right:*** Runtime percentage of each component.

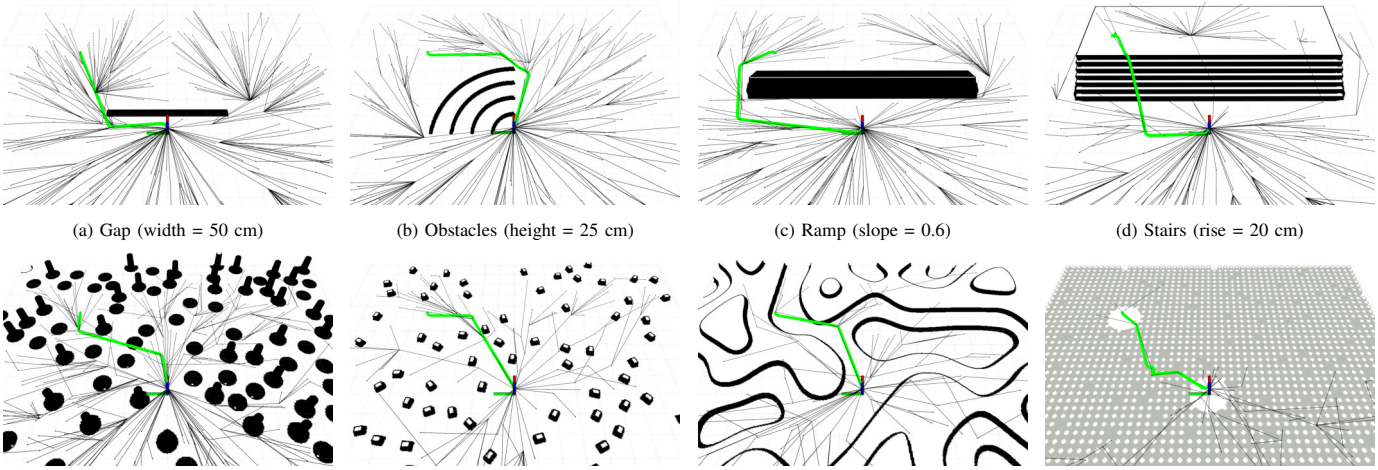
TABLE II: Terrain features for planner evaluation.

Terrain \ Level	Easy	Medium	Hard	Comment
gap width [m]	0.3	0.4	0.5	*0.5 m is too wide for crossing*
obstacle height [m]	0.15	0.2	0.25	*obstacles 6 cm wide*
ramp slope [deg]	11.3	21.8	31	*max traversable slope:* 25°
stair rise [m]	0.1	0.15	0.2	*stair run 30 cm*
num pillars and holes	20	40	60	*position distributed uniformly*
brick height [m]	0.15	0.2	0.25	*100 bricks that robot can step on*
terrace step height [m]	0	0.1	0.2	*Perlin noise with quantization steps*
% stepping stones removed	4	8	12	*grid with some stones removed at random*

B. Legged Robot

We evaluate our planning and control pipeline in simulation first and then validate it on real hardware ANYmal robot. For simulation evaluation we create terrains 20m $\times$ 20m in size (3 cm resolution). Start and goal are fixed *SE(2)* states; $(0, 0, 0)$ and $(5, 5, 0)$, respectively. We generate eight different terrain types with three difficulty levels each. Different terrain types are gaps, obstacles, ramps, stairs, mazes, bricks, terrace, and stepping stones; all shown in Fig. 15. In each map, the white color denotes the fully traversable area, and the fully untraversable area is shown in a dark color (black or olive). The final path after 2 s planning time is denoted in green. Black lines represent the connections inside the RRT tree. Table II describes difficulty levels for each terrain. At *Hard* level, some of the features are too difficult for the robot to traverse, and a detour has to be taken (e.g., slope terrain). This highlights the planner feature of negotiating obstacles to shorten the path if possible and take a detour if not.

1) SBP Success Rate: SBP success rate was tested on a laptop with an Intel i7-10750H@2.60GHz 6-core processor. The success rate is defined as the number of plans that reach the goal over the number of total plans when a feasible path exists. For randomized terrains, we ensure that a feasible path exists by running the planner for 20 seconds. If no path is found, the terrain is re-sampled until a feasible path is found. We consider 6 planning times (0.1, 0.2, 0.5, 1.0, 2.0 and 5.0 seconds), and plan 100 times for every terrain and every planning time. Each randomized terrain is sampled 10 times



(a) Gap (width = 50 cm) (b) Obstacles (height = 25 cm) (c) Ramp (slope = 0.6) (d) Stairs (rise = 20 cm)

(e) Maze (# holes = # pillars = 60) (f) Bricks (height = 25 cm) (g) Terrace (step height = 20 cm) (h) Stepping stones (12% removed)

Fig. 15: ***Top:*** Non-randomized terrains at the highest difficulty level; robot cannot negotiate some terrain features, and a detour must be taken. ***Bottom:*** Four randomized terrain types with the highest difficulty level. We do not know whether a solution exists a priori. If we cannot find a feasible plan within 20 s planning time, the terrain is re-generated until one is found.

TABLE III: SBPs success rate for ANYmal on different terrain types, difficulty levels, and planning times. The top row shows the minimal planning time where no failures occur (e.g., for the hard difficulty level, with 2 s planning time, the SBP finds a solution in all terrains).

Level / Terrain	Easy → 1.0 s			Medium → 1.0 s			Hard → 2.0 s		
	plan time [s]			plan time [s]			plan time [s]		
	0.1	0.2	0.5	0.1	0.2	0.5	0.1	0.2	0.5
gap	.88	.97	1.0	.81	.97	1.0	.50	.82	1.0
obstacles	.82	.96	1.0	.79	.97	1.0	.69	.96	1.0
ramp	.81	.98	1.0	.81	.96	1.0	.42	.78	.98
stairs	.89	.99	1.0	.83	.97	1.0	.70	.84	.99
maze	.87	.93	1.0	.82	.88	.97	.74	.90	.94
bricks	.83	.92	1.0	.75	.86	1.0	.44	.86	.97
terrace	.81	.95	1.0	.82	.92	1.0	.69	.93	.98
st. stones	.79	.94	.98	.74	.87	.99	.67	.83	.94

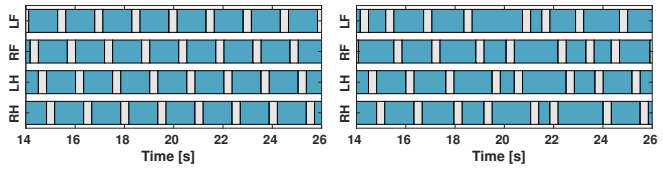


Fig. 16: Contact schedule visualization (cyan: contact phase, white: swing phase). ***Left:*** planner discovers cyclic contact schedule when walking on the flat ground. ***Right:*** planner uses a highly acyclic gait to walk on the stepping stones.

for a fair evaluation. The result is shown in Tab. III. It reads that the success rate drops with decreasing the planning time; the worst-case scenario has a planning success of 94% for 0.5 seconds of planning time. The planner takes at most 1 second to achieve 100 % success rate on the easy and medium terrains, and 2 seconds for the difficult terrains.

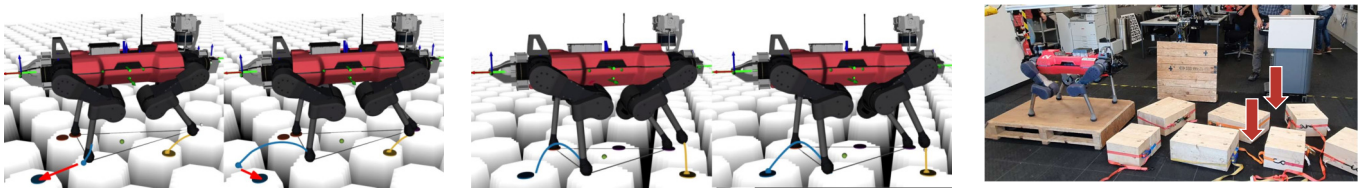
Fig. 16 shows two contact schedules discovered by our planner. These two examples highlight that when the terrain is easy, the planner resorts to using well-understood and efficient cyclic gait patterns. On the other hand, in the presence of obstacles, an acyclic contact schedule can emerge and enable the robot to precisely orient its body in anticipation of difficult terrain features.

2) MPC Convergence Rate: The second scenario evaluates the convergence rate of the combined SBP and MPC, defined as the number of times MPC converges for the entire trajectory over the number of plans executed. For evaluation purposes, the MPC has perfect knowledge of robot dynamics, and the controllers do not influence the success rate. For each non-randomized terrain, ten global plans are executed. For each randomized terrain, we generate two samples for each terrain type and then execute five plans for each sample. The trivial solutions, where the straight-line path on flat ground is enough to connect the start and goal, are removed.

We compare the convergence rate for two different MPC formulations: one with and one without the swing leg joint deviation penalty. We found that the MPC with swing leg penalization always converges, yet the other can diverge on stepping stones with a probability of 30 %. This result demonstrates the benefit of performing whole-body planning in the *initialization step* since the whole-body plans better guide the optimization to the correct local minima. Fig. 17a and Fig. 17b show how the cost function guides the swing leg to the desired foothold.

3) LSTP Physical Simulation: The proposed planning and control framework was tested in the physical simulation together with the WBC. Since running a physical simulation is computationally expensive, we tested the full simulated stack on each terrain three times without failures.

4) LSTP Hardware Tests: We deploy our planning and control framework on ANYmal. ANYmal has two Robosense RS-Bpearl LiDARs, both in front and back. Bpearls are dome-shaped LiDARs with 360°×90° Field of View (FOV). ANYmal has two onboard computers equipped with an Intel i7-8850H@2.60 GHz 6-core processor. The Locomotion PC (LPC) runs MPC at 25 Hz and the WBC at 400 Hz. The controllers are run asynchronously from one another. The Navigation PC (NPC) runs the sampling-based planner and the terrain processing pipeline described in Sec. III-B. Elevation mapping runs at 20 Hz on a Jetson AGX Xavier onboard computer [85]. As a state estimator we use the fusion of leg kinematics and IMU [86] together with the LiDAR odometry [87]. All experiments have been performed with a statically stable gait with the maximum recorded velocity of approxi-



(a) No swing leg deviation penalty (b) With swing leg deviation penalty (c) Gap occlusion

Fig. 17: ***a) & b)*** Improved optimization convergence through penalizing swing leg deviation. We show an example for the LF leg, whose trajectory planned by the MPC is colored in blue. The reference foothold at the end of the swing phase is shown with the blue disk. We show the initial swing trajectory on the left, and the one after some optimization iterations on the right. *a)* Without penalization, the foothold of the optimized trajectory is far away from the reference foothold (distance shown with red arrows). *b)* The optimized swing motion lands smoothly on the reference foothold with penalization. ***c)*** While the LiDAR correctly detects the closer stepping stones, it cannot see holes between the far away stepping stones (marked with red arrows). Hence, the foot placement might be incorrect.

mately 20 cm/s, which is the same velocity we achieved in the simulation.

In our experiments, we let the robot plan in a local map of size $8\,\mathrm{m}$ by $8\,\mathrm{m}$ centered at the robot's current position and snap the goal into this region if needed. We only send a portion of the plan to the MPC for tracking, and once the tracking finishes, SBP re-plans with an updated map. This procedure is repeated until the goal is reached. By letting the SBP re-plan, we can avoid issues like the one shown in Fig. 17c where the holes are occluded, and as a result, the planner could attempt to place the feet there.

A range of terrains used for hardware experiments can be found in Fig. 18. We could cross 36 cm wide gaps, close to the simulation result of 40 cm and more than the manufacturer's specification of 30 cm. The planner discovers an acyclic contact schedule for the gap-crossing maneuver, similar to the stepping stones scenario. Next, we increase the gap width to 47cm (too wide to cross) and add a narrow bridge. The bridge is too narrow to place all feet on it, so the planner has to coordinate the leg placement precisely. Fig. 19 shows the sequence of the robot crossing the bridge. Finally, we also test three different stepping-stone scenarios to showcase the generality of our planner.

Apart from negative obstacles like gaps and stepping stones (see Fig. 18), we tested the planner's capability to negotiate steps. The robot was asked to go up and down a step of 20 cm, more than the stair heights recommended in construction, which vary between 14 cm and 19 cm. Experimenting with the maze environment showcases the planner's ability to escape local minima. We constructed a wall around the room corner and left only one narrow passage allowing the robot to get in. Subsequently, the robot was asked to go to the corner from the other side of the wall, where the straight line connection leads to a dead end. With the initialization from the SBP, the robot can walk around and reach the goal. Such a maneuver would be impossible if only MPC planner was used.

C. Legged-Wheeled Robot

This section shows the planner's capabilities to plan for a legged-wheeled robot with non-steerable wheels. Executing SBPs plans on hardware has been demonstrated in our previous work [43]. For the sake of space, we only show the SBPs evaluation since, in this work, the SBP was generalized to enable base turning and side stepping.

TABLE IV: SBPs success rate for ANYmal on wheels across different terrain types, difficulty levels, and planning times. The top row shows the minimal planning time where no failures occur (e.g., for the hard difficulty level, with $5\,\mathrm{s}$ planning time, the SBP finds a solution in all terrains).

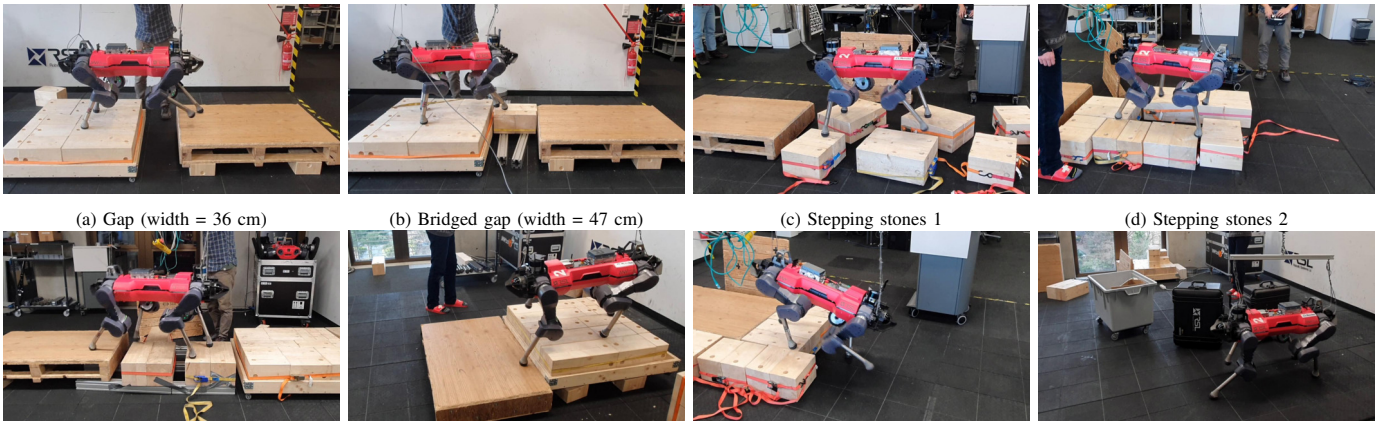
Level / Terrain	Easy → 2.0 s			Medium → 2.0 s			Hard → 5.0 s		
	plan time [s]			plan time [s]			plan time [s]		
	0.1	0.2	0.5	0.1	0.2	0.5	0.1	0.2	0.5
gap	.68	.91	.99	.73	.96	1.0	.44	.73	.99
obstacles	.63	.91	1.0	.71	.86	1.0	.55	.88	.99
ramp	.76	.91	1.0	.72	.91	1.0	.22	.62	.98
stairs	.82	.96	1.0	.67	.86	1.0	.67	.93	.99
maze	.73	.92	.98	.83	.94	.99	.53	.73	.95
bricks	.76	.90	1.0	.70	.90	1.0	.68	.78	.94
terrace	.78	.97	1.0	.71	.90	.99	.65	.82	.99
st. stones	.64	.83	.95	.52	.75	.92	.50	.74	.91

The success rate of the SBP is shown in Tab. IV. Fig. 1 shows an example of a legged wheeled robot navigating stairs using our SBP. The planner is evaluated in the same scenarios as the legged robot. The success rate is comparable to the one for the legged robot shown in Tab. III, but a bit lower. It also takes longer planning times to achieve a 100% success rate for all terrain types. We speculate that this is due to the fact that ANYmal on wheels has a shorter shank compared to ANYmal resulting in shorter reach. The SBP planners use the same tuning for the legged and legged-wheeled robots; we did not need to change any parameters.

D. HEAP

HEAP is used for experimental verification of the proposed planning and control architecture. We ran the planner on a laptop with AMD Ryzen 9 5900HX CPU 3.3 GHz processor. VMC controller, filtering, wheel velocity control loop, and state estimation run on an onboard computer. Leica iCON iXE3 system provides Global Navigation Satellite System (GNSS) with Real-time Kinematic (RTK) corrections. Two-state information filter [88] fuses GNSS measurements with the two Inertial Measurement Unit (IMU) units (from the chassis and the cabin). For more details, please refer to [3]. In all experiments with HEAP, SBP plans once with $2\,\mathrm{s}$ planning time. The plan is then fed to the MPC, which runs between 5 and 20 Hz (depending on the task).

1) LSTP Hardware Tests: We start by testing the combined sampling and optimization planning on hardware. Convergence rate was tested in our previous work [72], and in this work, we focus on hardware results. Fig. 20 shows experiment where HEAP steps over a virtual gap. We did not dig an actual gap in our testing field; however, the height map provided



(a) Gap (width = 36 cm) (b) Bridged gap (width = 47 cm) (c) Stepping stones 1 (d) Stepping stones 2

(e) Stepping stones 3 (f) Stepping up (rise = 20 cm) (g) Stepping down (rise = 20 cm) (h) Maze

Fig. 18: Various terrains tested on the real robot.

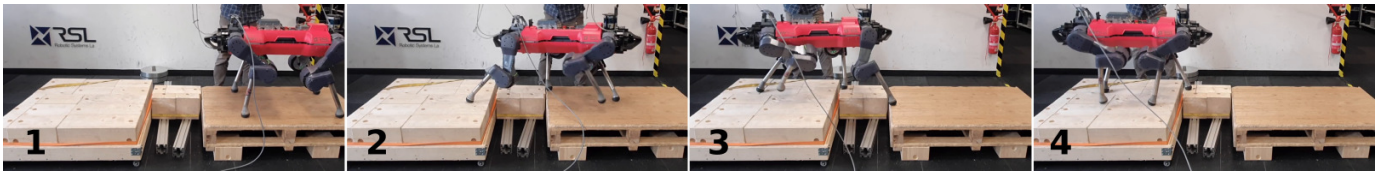


Fig. 19: Anymal using a bridge to cross a 47 cm wide gap. The bridge is too narrow to place all 4 feet on it. Hence the robot has to figure out a contact schedule and coordinate limb motion to avoid this situation.

to the planner contains a gap, and hence the stepping-over behavior emerges. Another maneuver where HEAP crosses the virtual bridge is shown in Fig. 1. These maneuvers require perceptive planning with both SBP and MPC. Stepping over a gap is challenging for the control system since it has to precisely coordinate the movement of the arm kinematics with the movement of the base such that the contact point does not slip. The control system has to be robust since the wheel speed cannot be precisely controlled, and hence we cannot precisely control the base movement. Without high-frequency local re-planning from the MPC, the gap crossing maneuver would prove challenging due to accumulated drift. The total gap crossing duration is $103\,\mathrm{s}$.

2) Long-Horizon MPC: We also experiment with using solely MPC as a planner and controller to verify MPCs ability to plan over long horizons. In this case, MPC receives a goal pose and a contact schedule without any guiding path. The terrains tested are relatively flat since MPC can fail to converge in the presence of obstacles without good initialization [72]. The prediction horizon is set to $25\,\mathrm{s}$. The start and goal pose, along with the route taken by the base and the wheels, is shown in Fig. 21b. The optimization can discover an efficient path that actively utilizes all four steering joints. It involves *crab steering*, a configuration where all four steering joints point in the same direction, resulting in diagonal movement of the base that is not aligned with its heading. The machine is moved $2.53\,\mathrm{m}$ sideways with a trajectory that involves four direction changes (discovered by MPC). The motion duration is $46\,\mathrm{s}$, and a contact schedule was externally provided.

The MPC can also plan stepping motions with a fixed contact schedule. The machine lifts off the legs in the following order: *LH, LF, RH, RF*. Each swing phase lasts $5\,\mathrm{s}$. HEAP is commanded to turn in place multiple times, with approximately 30 degrees yaw increments. Fig. 22 shows snapshots of the machine changing its orientation by approximately 108 degrees. A similar scenario is shown in Fig. 23 where HEAP steps $6.1\,\mathrm{m}$ sideways without driving. Total motion duration is $177\,\mathrm{s}$ for the stepping turn and $266\,\mathrm{s}$ for the sideways stepping.

It is challenging for the optimization to plan in the presence of nonlinear constraints (e.g., support polygon constraint) over long horizons. Furthermore, the planning times should be short enough to run in real-time. In all cases, the MPC plans with the flat ground assumption. Hence, the control system is also tested since it has to compensate for the unmodelled terrain (the whole field is sloped and bumpy).

3) Whole-Body Controller Terrain Adaptiveness: Lastly, Fig. 24 shows an experiment where HEAP is required to adapt to unseen terrain. The planner is given a goal pose ahead of the machine and generates trajectories with flat ground assumption. However, there is a hole on one side of the machine (blue arrow) and a hill on the other (red arrow). In the next frame, one can observe that the machine retracted its left leg and extended its right leg to keep the base as leveled as possible. Base and joint tracking during the experiment is also shown in Fig. 24. The MPC sends base references to the VMC, which tries to track them. Note that there is some offset since the VMC control has not been tuned with the set point following in mind. The HFE leg joints do not follow the planned values from the MPC since the terrain map inside the MPC does not match the reality. Note how the left and right HFE joint move in opposite directions since there is a hill on one side and a hole on the other.

VIII. Discussion

1) Initialization step: We chose an optimizing planner like RRT* since the cost allows shaping the robot's behavior ([72]) at the expense of more computation. Since we sample only a three-dimensional subspace, one could use a grid-based

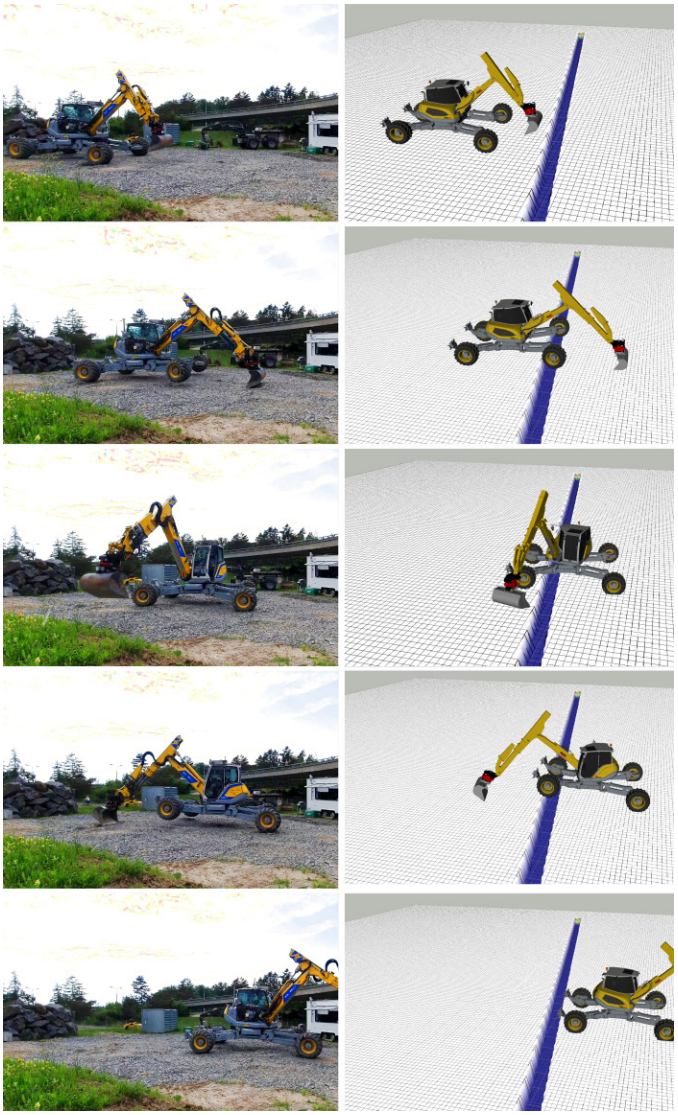
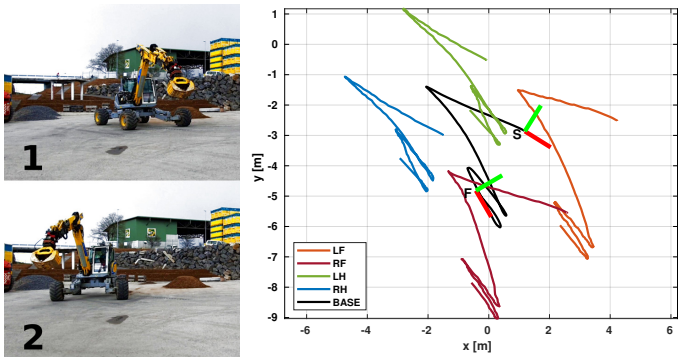


Fig. 20: ***Top to Bottom:*** HEAP stepping over a virtual gap, the timeline goes from top to bottom. ***Left:*** snapshots of the machine performing the maneuver. ***Right:*** terrain representation as seen by the planner and the controller.



(a) Start/Goal (b) Crab Steering Paths

Fig. 21: ***Left:*** Start and end configuration of HEAP performing a driving maneuver. The base is commanded to move sideways, and MPC discovers a motion satisfying the rolling constraint, which involves so-called *crab steering*. It involves all four wheels pointing in the same direction while the machine moves diagonally. ***Right:*** Base and end-effector positions during the crab steering experiment. The lines denote the path taken by each wheel and the base. The coordinate frames denote the base's starting (S) and finishing (F) position of the base, where red color denotes x axis and the green denotes y axis. For HEAP, x axis is aligned with the direction of travel.

method for the base motion planning. We chose RS curves as connections since they elegantly enforce the minimal turning radius for a robot with steerable wheels (HEAP). They are also beneficial for quadrupeds since they naturally limit the robot from walking too much sideways, the least stable way of locomoting. RS curves come with a trade-off: they are far more computationally expensive than straight-line connections (for quadrupeds).

2) Refinement step: While timings are unecessary for quasi-static motions, we compute them because the optimization is formulated as an OCP. Since the MPC only receives and tracks a snippet of the trajectory (e.g., 15 s out of 250 s) in the near future, we trade off some optimality (e.g., cannot speed up the motion) for real-time re-planning; crucial for hardware experiments. We found that using SLQ was possible, however less numerically stable than SQP (especially for HEAP) since SLQ is quite sensitive during the rollout phase.

3) Control: For the torque-controllable legged robot with high bandwidth actuators, we follow the standard approach of using terrain-adaptive whole-body control at the lowest level, which improves robustness. For HEAP, the same method does not work because fine torque control is not possible on all joints. Hence, we run the VMC to be compliant (necessary for hardware experiments) and MPC to track the motion plan. The VMC + kinematic MPC structure avoids a complicated dynamic model inside the MPC, which helps us achieve real-time performance and reduces the amount of tuning for deployment. Optimizing dynamic models for HEAP is numerically challenging because of significant differences in link masses. Moreover, we do not have an accurate dynamic model (e.g., the weight of hoses and oil is not modeled).

IX. Conclusion

This work presented LSTP, a combination of sampling and optimization-based motion planning for complex legged and legged-wheeled machines. The planner is divided into two stages. The long term *initialization step* is backed by an RRT planner, while the short-term *refinement step* is backed by the NO. The initialization step computes a contact schedule and a sequence of keyframe configurations used as an initial guess for the optimization. We found that initializing with whole-body configurations can sometimes improve optimization convergence. SBP presented in this work can find initial solutions in less than 1 second, which makes it suitable for real-time re-planning. Fast computation times are achieved by precomputing limb roadmaps offline and then using them to rapidly check collisions and stability at runtime. Lastly, the SBP can handle three robot types: a legged robot, a legged-wheeled robot with non-steerable wheels, and a legged excavator with steerable wheels.

The planner's second stage uses optimization running in an MPC fashion to refine the initial guess from the sampling-based planner. The constant re-planning allows for robustness which is especially important for the HEAP where the hardware does not allow precise control. The proposed two-stage planning approach has been benchmarked on a variety of terrains requiring MPC to stay out of local minima.

Fig. 22: HEAP performing a stepping turn in place. The $x-y$ position stays approximately the same, while the azimuth of the chassis changes.

Fig. 23: HEAP performing a stepping turn and moving laterally. The chassis keeps the azimuth approximately the same as at the beginning of the maneuver.

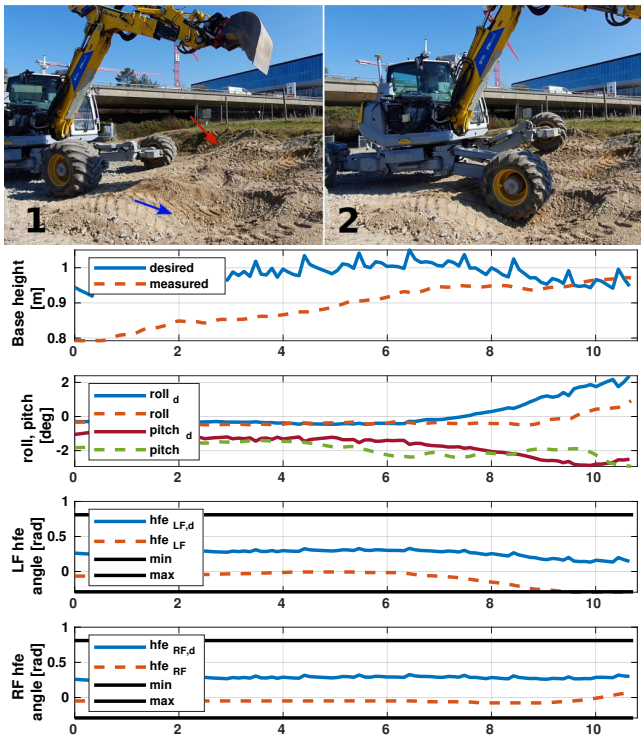


Fig. 24: ***Top:*** HEAP is commanded forward and there is a hole in front of it (blue arrow) and a hill (red arrow). Since the MPC is planning with a flat terrain assumption, VMC has to coordinate the front legs to keep the chassis leveled. ***Bottom:*** Base and joint tracking. While the base is tracking (with some offset) the MPC commands, HFE joints ignore the MPC references to accommodate for the unmodelled terrain.

We validate the proposed planning and control system on hardware for both quadrupedal robot ANYmal and excavator HEAP. We show that our approach can be executed on hardware in realistic conditions. In particular, for HEAP, we show that the proposed MPC controller is terrain-adaptive and does not require precise terrain models.

X. Outlook

The fundamental limitation of our approach is that it only allows for quasi-static motions. The initialization stage of LSTP can only discover static gaits which are not optimized in the second stage. It would be interesting to relax the stability assumption allowing SBP to find more dynamic gaits such as trotting or jumps. Frameworks like [24] allow for gait refinement in the second stage; however, one needs to ensure real-time execution and prevent the optimization from getting stuck. Thereby, hierarchical models [40] or new efficient solvers for switching time optimization [89] could be used.

The other fundamental limitation is the choice of a height map as an environmental representation. Our planner cannot discover contact-rich motions involving overhanging structures (e.g., crawling, ladder climbing), which could be addressed by choosing a different map representation. One big challenge for such confined environments is to devise an efficient strategy for base pose selection.

In addition, the SBPs base pose selection module could be improved. Since a cost function can describe a good base pose, one could use reinforcement learning to train a policy to produce a 3D pose based on the raw terrain observations. This would remove the need for terrain pre-processing, free up computational resources and reduce the amount of tuning. We have already done some preliminary work in this direction.

Lastly, we will implement the SBP running in a separate thread at any time. This way, the MPC could request the new plan before it finishes executing the previous plan, which speeds up maneuver execution.

Appendix

A. Terrain Preprocessing

The normal estimation module fits a plane locally for each cell point $\boldsymbol{p}_i = (x_i, y_i)$ inside the grid map. As a result, terrain normal vectors $\boldsymbol{n}_i = [\boldsymbol{n}_i^x \; \boldsymbol{n}_i^y \; 1]^T$ and a height estimate $\tilde{h}_i$ are computed. The tangent plane at point $\boldsymbol{p}_i$ is a function of height h and local coordinates $(dx, dy) = (x - x_i, y - y_i) \in \mathbb{R}^2$,

$$\boldsymbol{n}_i^x dx + \boldsymbol{n}_i^y dy + (h - \tilde{h}_i) = 0. \tag{24}$$

Assuming that all the points $\boldsymbol{p}_j$ (height denoted by h_j) within radius R from $\boldsymbol{p}_i$ lie on the plane, we can write Eq. 24 in the matrix notation,

$$[\Delta x \; \Delta y \; -1] \cdot \boldsymbol{\theta} = -h_j, \quad \forall j \text{ with } \|\boldsymbol{p}_j - \boldsymbol{p}_i\|_2 < R, \tag{25}$$

where $\Delta x = x_j - x_i$, $\Delta y = y_j - y_i$, and $\boldsymbol{\theta} = [\boldsymbol{n}_i^x \; \boldsymbol{n}_i^y \; \tilde{h}_i]^T$ is the vector of parameters. We can then assemble the data matrix $\boldsymbol{D} \in \mathbb{R}^{M \times 3}$ and height vector $\boldsymbol{h} \in \mathbb{R}^M$ and perform the normal estimation by solving the least squares problem:

$$\boldsymbol{\theta}^* = (\boldsymbol{D}^T \boldsymbol{D})^{-1} \boldsymbol{D}^T \boldsymbol{h}. \tag{26}$$

Normal estimation is used in computing both surface normals and filtered elevation. The difference is that the former one uses a smaller radius R_s ($0.1\,\mathrm{m}$ for ANYmal and $0.3\,\mathrm{m}$ for HEAP) and does not consider traversability. The latter one performs normal estimation using all traversable points within a larger radius of R_l ($0.4\,\mathrm{m}$ for ANYmal and $2.5\,\mathrm{m}$ for HEAP).

A grid cell (x, y) is categorized as untraversable if on a steep slope or if the surrounding terrain is irregular. Terrain is considered irregular at (x, y) when its height deviates too much from its elevated mean value. The elevated mean h_{em} around point $\boldsymbol{p}_i$ is computed in the local neighbourhood with radius R_l,

$$\mathcal{N}_l(R, \boldsymbol{p}_i) \coloneqq \Big\{ \boldsymbol{p}_j \;\Big|\; \|\boldsymbol{p}_j - \boldsymbol{p}_i\|_2 < R_l \Big\}. \tag{27}$$

Define $\mathcal{N}_e(R, \boldsymbol{p}_i)$ as the set of nearby points whose height are above h_{avg}, the average height of $\mathcal{N}_l(R, \boldsymbol{p}_i)$,

$$\mathcal{N}_e(R, \boldsymbol{p}_i) \coloneqq \Big\{ (x, y) \in \mathcal{N}_l(R, \boldsymbol{p}_i) \mid h(x, y) > h_{avg} \Big\}. \tag{28}$$

Then we can compute elevated mean h_{em} using equation Eq. 29 with the height offset defined in Eq. 30.

$$h_{em} = \min\{h_{max},\, h_{avg} + w_o h_o\}, \tag{29}$$

$$h_o = \frac{1}{|\mathcal{N}_e|} \sum_{(x,y)\in\mathcal{N}_e} \Big(h(x, y) - h_{avg} \Big), \tag{30}$$

where h_{max} is the maximum height in $\mathcal{N}_l(R, \boldsymbol{p}_i)$ and w_o is a user-defined weight controlling the influence of elevated mean. Elevated mean is motivated by use case on terrains containing negative obstacles such as ditches and stepping stones. We found elevated mean to help correctly assess traversability for terrains like gaps, holes, and stepping stones without a negative effect on others such as stairs, ramps, and pillars. Visualization of the computing elevated mean can be seen in Fig. 5b. Upon classifying the traversable regions we compute the sdf_2.

Lastly, the algorithm computes filtered elevation for base pose selection by fitting planes using

$$\mathcal{N}_f(R, \boldsymbol{p}_i) \coloneqq \mathcal{N}_l(R, \boldsymbol{p}_i) \cap \mathcal{T}, \tag{31}$$

i.e., all traversable points in $\mathcal{N}_l(R, \boldsymbol{p}_i)$. The rationale behind is that the base of the robot should not adjust itself to the untraversable regions since those are the ones that we want to avoid anyway.

B. Feasibility Check for Point Foot Robot

1) Full-Support State Connection: The base path connecting the start and the goal of OSM is discretized into N_{seg} segments with $N_{seg} + 1$ states. To create a swing phase, we need at least one state to contain a non-grounded leg. Hence we require each swing phase to span across two segments at least, $N_{seg,s} \geq 2$. Larger $N_{seg,s}$ yields finer discretization of swing leg motion in *space* (not in *time*) but decreases the computation speed. The number of segments in OSM cannot be set smaller than $N_{seg,min} \coloneqq K \cdot N_{seg,s}$ with K being the number of the limbs, otherwise we cannot create a feasible contact schedule if all limbs need to have a swing phase. We set N_{seg} slightly larger than $N_{seg,min}$ to allow for longer swing phases if needed.

Algorithm 1 Compute contact schedule for a swing order $\boldsymbol{p}$

```
function GETCONTACTSCHEDULE(p, j_lcb, j_ecc, O)
    j_cb ← [−1, −1, −1, −1]^⊤, j_cc ← [−1, −1, −1, −1]^⊤;
    for k = p.SIZE() − 1 : 0 do
        i ← p[k];                                   ▷ i is k-th swing limb in p
        if k = p.SIZE() − 1 then                    ▷ last swing limb
            j_cc[i] ← N_seg;
        else
            j_cc[i] ← j_cb[p[k + 1]];               ▷ set cc to the next cb
        while true do
            j_cb[i] ← MIN(j_lcb[i], j_cc[i] − N_seg,s);
            if j_cc[i] < j_ecc[i] or j_cb[i] < 0 then
                return false;
            for c = 0 : k − 1 do                    ▷ swing limbs prior to i
                if j_cb[i] < j_ecc[p[c]] then
                    return false;
            t ← UNSTABLEINDEX(p, j_cb[i], j_cc[i], O);
            if t = −1 then
                break;
            else
                j_cc[i] ← t − 1;
    return {j_cb, j_cc};
```

2) Contact Schedule Computation: We refer to set of discretized states within OSM as $\mathcal{O}$, i.e.

$$\mathcal{O} \coloneqq \Big\{ \boldsymbol{o}_j,\; j = 0, \ldots, K \text{ with } K = N_{seg} \Big\}. \tag{32}$$

In Eq. 32, $\boldsymbol{o}_0$ denotes the starting state of OSM where $\boldsymbol{o}_K$ denotes the end of it. Their leg configurations are already determined in FSS computation. To find LCB, we search for every state from 1 to $K-1$ for valid roadmap vertices close to footholds at state o_0. If at state j no such vertex is found, we set $j_{lcb} = j-1$. In case the search reaches index $K-1$ it means that the leg can remain in contact during the entire OSM. On the other hand, ECC are computed by searching from index $K-1$ to $N_{seg,s}$ (backwards). If at index j the footholds in $\boldsymbol{o}_K$ cannot be reached, we set $j_{ecc} = j + 1$. In case contact can remain for all the searched index, we set $j_{ecc} = N_{seg,s}$ because contact break can at earliest happen at 0.

Given the latest contact break $\boldsymbol{j}_{lcb}[i]$ and earliest contact creation $\boldsymbol{j}_{ecc}[i]$ for the i^{th} swing limb, a feasible contact schedule can be computed by solving the feasibility problem below. The algorithm needs to find indices of states where contact break $\boldsymbol{j}_{cb}[i]$ and contact creation $\boldsymbol{j}_{cc}[i]$ happen. The feasibility problem below is solved only for the swing legs $\mathcal{SW}$ of OSM.

$$\begin{aligned}
\text{find}\quad & \boldsymbol{j}_{cb}[i], \boldsymbol{j}_{cc}[i] \text{ for all } i \in \mathcal{SW} \\
\text{s.t.}\quad & \boldsymbol{j}_{cb}[i] \in \{0, 1, \ldots, \boldsymbol{j}_{lcb}[i]\} \\
& \boldsymbol{j}_{cc}[i] \in \{\boldsymbol{j}_{ecc}[i], \boldsymbol{j}_{ecc}[i] + 1, \ldots, N_{seg}\} \\
& \boldsymbol{j}_{cc}[i] - \boldsymbol{j}_{cb}[i] \geq N_{seg,s} \\
& \text{at most one swing leg,} \quad \forall \boldsymbol{o} \in \mathcal{O} \\
& \text{(relaxed) stability constraint,} \quad \forall \boldsymbol{o} \in \mathcal{O}.
\end{aligned} \tag{33}$$

For a fixed swing limb permutation $\boldsymbol{p}$, it is easy to compute the contact schedule. $\boldsymbol{p}$ tells the swing limb order, e.g. limb at position 0 is the one that swings first. An algorithm for finding a feasible contact schedule is shown in Alg. 1.

As inputs Alg. 1 takes a sequence of OSM states $\mathcal{O}$, ECC and LCB indices $(\boldsymbol{j}_{ecc}, \boldsymbol{j}_{lcb})$ for each leg and a swing limb order $\boldsymbol{p}$. The output of Alg. 1 are the indices of contact breaks

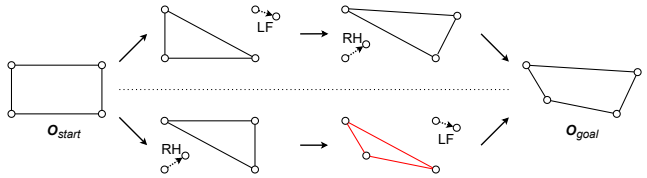


Fig. 25: Different swing orders generate different support polygons in an OSM. In the upper one, the robot swings first the LF leg and then the RH one, which makes the Support Polygon (SP) large enough. The swing limb order is reversed in the lower option, and this makes the SP very small (highlighted in red) once the LF leg breaks contact.

Algorithm 2 Create one-step motion (OSM)

```
1: function CREATEOSM(o_start, o_goal)
2:     O ← DISCRETIZE(o_start, o_goal);                ▷ only base poses
3:     if ¬CREATEFULLSUPPORTSTATE(o_goal) then
4:         return false;
5:     if ¬ISBASECOLLISIONFREE(O) then
6:         return false;
7:     j_lcb ← FINDLATESTCONTACTBREAKS(O);
8:     SW ← GETSWINGLIMBS(j_lcb);
9:     j_ecc ← FINDEARLIESTCONTACTCREATIONS(O, SW);
10:    for all p ∈ Sym(SW) do    ▷ Sym(SW) are permutations of SW
11:        cs ←GETCONTACTSCHEDULE(p, j_lcb, j_ecc, O);    ▷ Alg. 1
12:        if cs ≠ false then
13:            if CREATESWINGMOTION(O, cs) then
14:                return O;
15:    return false;
```

Algorithm 3 Get feasible OSM lengths

```
1: function FEASIBLEOSMLENGTHS(d, d_total)
2:     ret ← {}, d_to_go ← d_total − d;
3:     if d_to_go < MAX(L) then
4:         ret.PUSH_BACK(d_to_go);
5:     for all l ∈ L do
6:         if d_to_go − l > MIN(L) then
7:             ret.PUSH_BACK(l);
8:     return ret;
```

$\boldsymbol{j}_{cb}$ and contact creations $\boldsymbol{j}_{cc}$ for all limbs. If a limb does not change its contact state the corresponding ECC and LCB are set to -1 in our implementation. Alg. 1 returns $false$ if no feasible contact schedule can be computed in line 16. The function UNSTABLEINDEX finds the first index t encountered from $\boldsymbol{j}_{cc}[i]$ to $\boldsymbol{j}_{cb}[i]$ for which stability constraints are violated or -1 if no violation. Two examples of the contact plans found with Alg. 1 are shown in Fig. 10.

For a quadrupedal robot, the number of different swing orders is at most $4! = 24$ and they result in different support polygons. Fig. 25 shows how different swing orders influence on the SPs area. Hence all the combinations are enumerated and we try to compute a feasible contact schedule using the Alg. 1 (this can be done in parallel). Computation terminates as a feasible contact schedule is found.

C. Legged-Wheeled Robot Extension

We say that a leg is in the driving mode if it drives forward in longitudinal direction, without any lateral movement. The robot enters the stepping mode if its base is turning, side-stepping or when the terrain under the nominal hip location is untraversable. For driving legs we can simply establish ground contact same way as during FSS computation (Sec. IV-B2).

Algorithm 4 Motion feasibility validation from $\boldsymbol{s}_{start}$ to $\boldsymbol{s}_{end}$

```
1: function ISMOTIONVALID(s_start, s_end)
2:     traj ← {};          ▷ state sequence between s_start and s_end
3:     d_total ← DISTANCE(s_start, s_end), d ← 0;
4:     while d < d_total do
5:         o_start ← nullptr;
6:         if d = 0 then
7:             o_start ← COPYSTATE(s_start);
8:             CREATEFULLSUPPORTSTATE(o_start);
9:         else
10:            o_start ← COPYSTATE(traj.BACK());
11:        osm_success ← false;
12:        L ← FEASIBLEOSMLENGTHS(d, d_total);          ▷ Alg. 3
13:        for all l ∈ L do
14:            o_goal ← nullptr;
15:            if d + l = d_total then
16:                o_goal ← COPYSTATE(s_end);
17:            else
18:                o_goal ← INTERP(s_start, s_end, d + l);
19:            O ← CREATEOSM(o_start, o_goal);          ▷ Alg. 2
20:            if O ≠ false then
21:                traj.PUSH_BACK(O);
22:                d ← d + l;
23:                osm_success ← true;
24:                break;
25:        if ¬osm_success then
26:            return false;
27:    return true;
```

To find LCB for a legged-wheeled robot we first check the leg mode by iterating over the OSM states. Once the leg enters the stepping mode, it must go through a swing phase before the end of the current OSM. In this case the LCB is searched from the index where the leg switches its state from driving to stepping. Similarly, one can find the ECC for the legged-wheeled robot. The computation can be parallelized for different legs.

REFERENCES

[1] N. Kashiri, L. Baccelliere, L. Muratore, A. Laurenzi, Z. Ren, E. M. Hoffman, M. Kamedula, G. F. Rigano, J. Malzahn, S. Cordasco *et al.*, "Centauro: A hybrid locomotion and high power resilient manipulation platform," *IEEE Robotics and Automation Letters*, 2019.

[2] M. Bjelonic, C. D. Bellicoso, Y. de Viragh, D. Sako, F. D. Tresoldi, F. Jenelten, and M. Hutter, "Keep rollin'—whole-body motion control and planning for wheeled quadrupedal robots," *IEEE Robotics and Automation Letters*, 2019.

[3] D. Jud, S. Kerscher, M. Wermelinger, E. Jelavic, P. Egli, P. Leemann, G. Hottiger, and M. Hutter, "Heap-the autonomous walking excavator," *Automation in Construction*, 2021.

[4] C. D. Bellicoso, C. Gehring, J. Hwangbo, P. Fankhauser, and M. Hutter, "Perception-less terrain adaptation through whole body control and hierarchical optimization," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016.

[5] F. Jenelten, T. Miki, A. E. Vijayan, M. Bjelonic, and M. Hutter, "Perceptive locomotion in rough terrain–online foothold optimization," *IEEE Robotics and Automation Letters*, 2020.

[6] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter, "Perceptive locomotion through nonlinear model predictive control," *submitted to IEEE Transactions on Robotics*, 2022.

[7] E. Jelavic and M. Hutter, "Whole-body motion planning for walking excavators," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

[8] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, 2020.

[9] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, 2022.

[10] P. Brakel, S. Bohez, L. Hasenclever, N. Heess, and K. Bousmalis, "Learning coordinated terrain-adaptive locomotion by imitating a centroidal dynamics planner," *arXiv preprint arXiv:2111.00262*, 2021.

[11] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, 1996.

[12] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[13] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, 2001.

[14] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, 2011.

[15] T. Bretl, "Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem," *The International Journal of Robotics Research*, 2006.

[16] K. Hauser, T. Bretl, J.-C. Latombe, K. Harada, and B. Wilcox, "Motion planning for legged robots on varied terrain," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1325–1349, 2008.

[17] S. Tonneau, A. Del Prete, J. Pettré, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Transactions on Robotics*, 2018.

[18] M. Geisert, T. Yates, A. Orgen, P. Fernbach, and I. Havoutis, "Contact planning for the anymal quadruped robot using an acyclic reachability-based planner," in *Annual Conference Towards Autonomous Robotic Systems*. Springer, 2019.

[19] F. Farshidian, E. Jelavic, A. Satapathy, M. Giftthaler, and J. Buchli, "Real-time motion planning of legged robots: A model predictive control approach," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 2017.

[20] C. D. Bellicoso, F. Jenelten, C. Gehring, and M. Hutter, "Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots," *IEEE Robotics and Automation Letters*, 2018.

[21] W. Reid, F. J. Pérez-Grau, A. H. Göktoğan, and S. Sukkarieh, "Actively articulated suspension for a wheel-on-leg rover operating on a martian analog surface," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016.

[22] F. Cordes, A. Babu, and F. Kirchner, "Static force distribution and orientation control for a rover with an actively articulated suspension system," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[23] P. R. Giordano, M. Fuchs, A. Albu-Schaffer, and G. Hirzinger, "On the kinematic modeling and control of a mobile platform equipped with steering wheels and movable legs," in *2009 IEEE International Conference on Robotics and Automation*, 2009.

[24] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," *IEEE Robotics and Automation Letters*, 2018.

[25] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, 2006.

[26] V. S. Medeiros, E. Jelavic, M. Bjelonic, R. Siegwart, M. A. Meggiolaro, and M. Hutter, "Trajectory optimization for wheeled-legged quadrupedal robots driving in challenging terrain," *IEEE Robotics and Automation Letters*, 2020.

[27] E. Jelavic, Y. Berdou, D. Jud, S. Kerscher, and M. Hutter, "Terrain-adaptive planning and control of complex motions for walking excavators," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020.

[28] O. Melon, M. Geisert, D. Surovik, I. Havoutis, and M. Fallon, "Reliable trajectories for dynamic quadrupeds using analytical costs and learned initializations," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

[29] M. Geilinger, R. Poranne, R. Desai, B. Thomaszewski, and S. Coros, "Skaterbots: Optimization-based design and motion synthesis for robotic creatures with legs and wheels," *ACM Transactions on Graphics (TOG)*, 2018.

[30] D. Ioan, I. Prodan, S. Olaru, F. Stoican, and S.-I. Niculescu, "Mixed-integer programming in motion planning," *Annual Reviews in Control*, 2021.

[31] S. Tonneau, D. Song, P. Fernbach, N. Mansard, M. Taïx, and A. Del Prete, "Sl1m: Sparse l1-norm minimization for contact planning on uneven terrain," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

[32] 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.

[33] A. W. Winkler, C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell, and C. Semini, "Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[34] J. Sun, Y. You, X. Zhao, A. H. Adiwahono, and C. M. Chew, "Towards more possibilities: Motion planning and control for hybrid locomotion of wheeled-legged robots," *IEEE Robotics and Automation Letters*, 2020.

[35] F. Farshidian, M. Kamgarpour, D. Pardo, and J. Buchli, "Sequential linear quadratic optimal control for nonlinear switched systems," *IFAC-PapersOnLine*, 2017.

[36] J.-P. Sleiman, F. Farshidian, M. V. Minniti, and M. Hutter, "A unified mpc framework for whole-body dynamic locomotion and manipulation," *IEEE Robotics and Automation Letters*, 2021.

[37] F. Jenelten, R. Grandia, F. Farshidian, and M. Hutter, "Tamols: Terrain-aware motion optimization for legged systems," *IEEE Transactions on Robotics*, 2022.

[38] R. Grandia, F. Farshidian, R. Ranftl, and M. Hutter, "Feedback mpc for torque-controlled legged robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

[39] K. Ishihara, T. D. Itoh, and J. Morimoto, "Full-body optimal control toward versatile and agile behaviors in a humanoid robot," *IEEE Robotics and Automation Letters*, 2019.

[40] H. Li, R. J. Frei, and P. M. Wensing, "Model hierarchy predictive control of robotic systems," *IEEE Robotics and Automation Letters*, 2021.

[41] O. Melon, R. Orsolino, D. Surovik, M. Geisert, I. Havoutis, and M. Fallon, "Receding-horizon perceptive trajectory optimization for dynamic legged locomotion with learned initialization," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[42] R. Grandia, A. J. Taylor, A. D. Ames, and M. Hutter, "Multi-layered safety for legged robots via control barrier functions and model predictive control," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[43] M. Bjelonic, R. Grandia, M. Geilinger, O. Harley, V. S. Medeiros, V. Pajovic, E. Jelavic, S. Coros, and M. Hutter, "Offline motion libraries and online mpc for advanced mobility skills," *The International Journal of Robotics Research*, 2022.

[44] C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell, and C. Semini, "Motion planning for quadrupedal locomotion: Coupled planning, terrain mapping, and whole-body control," *IEEE Transactions on Robotics*, 2020.

[45] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, 1968.

[46] A. Escande, A. Kheddar, and S. Miossec, "Planning contact points for humanoid robots," *Robotics and Autonomous Systems*, 2013.

[47] T. Klamt and S. Behnke, "Anytime hybrid driving-stepping locomotion planning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017.

[48] ——, "Planning hybrid driving-stepping locomotion on multiple levels of abstraction," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018.

[49] A. Hornung *et al.*, "Humanoid robot navigation in complex indoor environments," Ph.D. dissertation, University of Freiburg, 2014.

[50] I. A. Şucan and S. Chitta, "Motion planning with constraints using configuration space approximations," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012.

[51] J. J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue, "Dynamically-stable motion planning for humanoid robots," *Autonomous robots*, 2002.

[52] A. Short and T. Bandyopadhyay, "Legged motion planning in complex three-dimensional environments," *IEEE Robotics and Automation Letters*, 2017.

[53] S. Tonneau, N. Mansard, C. Park, D. Manocha, F. Multon, and J. Pettré, "A reachability-based planner for sequences of acyclic contacts in cluttered environments," in *Robotics Research*. Springer, 2018.

[54] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning," *IEEE Robotics and Automation Letters*, 2020.

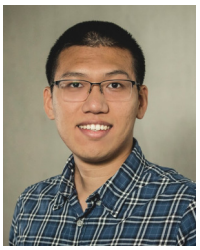
[55] S. Gangapurwala, M. Geisert, R. Orsolino, M. Fallon, and I. Havoutis, "Rloc: Terrain-aware legged locomotion using reinforcement learning and optimal control," *IEEE Transactions on Robotics*, 2022.

[56] T. Li, R. Calandra, D. Pathak, Y. Tian, F. Meier, and A. Rai, "Planning in learned latent action spaces for generalizable legged locomotion," *IEEE Robotics and Automation Letters*, 2021.

[57] M. Wermelinger, P. Fankhauser, R. Diethelm, P. Krüsi, R. Siegwart, and M. Hutter, "Navigation planning for legged robots in challenging

terrain," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.
[58] L. Wellhausen and M. Hutter, "Rough terrain navigation for legged robots using reachability planning and template learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021.
[59] M. Tranzatto, F. Mascarich, L. Bernreiter, C. Godinho, M. Camurri, S. Khattak, T. Dang, V. Reijgwart, J. Loeje, D. Wisth *et al.*, "Cerberus: Autonomous legged and aerial robotic exploration in the tunnel and urban circuits of the darpa subterranean challenge," *arXiv preprint arXiv:2201.07067*, 2022.
[60] J. Hwan Jeon, S. Karaman, and E. Frazzoli, "Anytime computation of time-optimal off-road vehicle maneuvers using the rrt," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, 2011.
[61] C. Xie, J. van den Berg, S. Patil, and P. Abbeel, "Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015.
[62] D. Kim, Y. Kwon, and S.-E. Yoon, "Dancing prm*: Simultaneous planning of sampling and optimization with configuration free space approximation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.
[63] Y. Ding, M. Zhang, C. Li, H.-W. Park, and K. Hauser, "Hybrid sampling/optimization-based planning for agile jumping robots on challenging terrains," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.
[64] P. Fernbach, S. Tonneau, and M. Taïx, "Croc: Convex resolution of centroidal dynamics trajectories to provide a feasibility criterion for the multi contact planning problem," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.
[65] P. Fernbach, S. Tonneau, O. Stasse, J. Carpentier, and M. Taïx, "C-croc: Continuous and convex resolution of centroidal dynamic trajectories for legged robots in multicontact scenarios," *IEEE Transactions on Robotics*, 2020.
[66] S. Dai, M. Orton, S. Schaffert, A. Hofmann, and B. Williams, "Improving trajectory optimization using a roadmap framework," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.
[67] J. Leu, G. Zhang, L. Sun, and M. Tomizuka, "Efficient robot motion planning via sampling and optimization," in *2021 American Control Conference (ACC)*, 2021.
[68] J. Leu, M. Wang, and M. Tomizuka, "Long-horizon motion planning via sampling and segmented trajectory optimization," in *2022 European Control Conference (ECC)*, 2022.
[69] B. Li, L. Li, T. Acarman, Z. Shao, and M. Yue, "Optimization-based maneuver planning for a tractor-trailer vehicle in a curvy tunnel: A weak reliance on sampling and search," *IEEE Robotics and Automation Letters*, 2021.
[70] L. Li, X. Long, and M. A. Gennert, "Birrtopt: A combined sampling and optimizing motion planner for humanoid robots," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016.
[71] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *The International Journal of Robotics Research*, 2002.
[72] E. Jelavic, F. Farshidian, and M. Hutter, "Combined sampling and optimization based planning for legged-wheeled robots," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.
[73] P. F. Felzenszwalb and D. P. Huttenlocher, "Distance transforms of sampled functions," *Theory of computing*, 2012.
[74] P. Fankhauser and M. Hutter, "A universal grid map library: Implementation and use case for rough terrain navigation," in *Robot Operating System (ROS)*. Springer, 2016.
[75] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, 1990.
[76] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, 2012.
[77] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 1999.
[78] J.-P. Sleiman, F. Farshidian, and M. Hutter, "Constraint handling in continuous-time ddp-based model predictive control," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.
[79] F. Farshidian *et al.*, "OCS2: An open source library for optimal control of switched systems," [Online]. Available: https://github.com/leggedrobotics/ocs2.
[80] R. Grandia, F. Farshidian, A. Dosovitskiy, R. Ranftl, and M. Hutter, "Frequency-aware model predictive control," *IEEE Robotics and Automation Letters*, 2019.
[81] H. G. Bock and K.-J. Plitt, "A multiple shooting algorithm for direct solution of optimal control problems," *IFAC Proceedings Volumes*, 1984.
[82] M. Hutter, P. Leemann, G. Hottiger, R. Figi, S. Tagmann, G. Rey, and G. Small, "Force control for active chassis balancing," *IEEE/ASME Transactions on Mechatronics*, 2016.
[83] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2019.
[84] B. M. Bell, "Cppad: a package for c++ algorithmic differentiation," *Computational Infrastructure for Operations Research*, 2012.
[85] T. Miki, L. Wellhausen, R. Grandia, F. Jenelten, T. Homberger, and M. Hutter, "Elevation mapping for locomotion and navigation using gpu," *arXiv preprint arXiv:2204.12876*, 2022.
[86] M. Bloesch, M. Hutter, M. A. Hoepflinger, S. Leutenegger, C. Gehring, C. D. Remy, and R. Siegwart, "State estimation for legged robots-consistent fusion of leg kinematics and imu," *Robotics*, 2013.
[87] S. Khattak, H. Nguyen, F. Mascarich, T. Dang, and K. Alexis, "Complementary multi–modal sensor fusion for resilient robot pose estimation in subterranean environments," in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2020.
[88] M. Bloesch, M. Burri, H. Sommer, R. Siegwart, and M. Hutter, "The two-state implicit filter recursive estimation for mobile robots," *IEEE Robotics and Automation Letters*, 2017.
[89] S. Katayama and T. Ohtsuka, "Whole-body model predictive control with rigid contacts via online switching time optimization," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

Edo Jelavic is a Ph.D. student at the Robotic Systems Lab, ETH Zurich under the supervision of Prof. M. Hutter. His research interests include planning and control for robotic applications. He received his B.Sc. from University of Zagreb in 2012 and his M.Sc. from from ETH Zurich in 2016, both in Electrical Engineering.



Kaixian Qu is a Master student at ETH Zurich, studying Robotics, Systems and Control under the supervision of Prof. Marco Hutter. His research interests lie in planning and control of legged robots. He received his B.Eng. in Mechanical Engineering from Tongji University in 2019.

Farbod Farshidian is a Senior Scientist at Robotic System Lab, ETH Zurich. He received his M.Sc. in electrical engineering from the University of Tehran in 2012 and his PhD from ETH Zurich in 2017, focusing on the planning and control of legged robots. His research interests are in mobile robots' motion planning and control, aiming to develop algorithms and techniques for operating autonomously in real-world applications. Farbod is part of the NCCRs Robotics and Digital Fabrication.

Marco Hutter is Associate Professor for Robotic Systems at ETH Zurich. He received his M.Sc. and PhD from ETH Zurich in 2009 and 2013. He is interested in the development of novel machines and actuation concepts together with the underlying control, planning, and machine learning algorithms for locomotion and manipulation. Marco is part of the NCCR Robotics and NCCR Digital Fabrication and PI in various international projects (e.g. EU NI) and challenges.