# QuadPiPS: A Perception-informed Footstep Planner for Quadrupeds With Semantic Affordance Prediction

Anonymous Authors

***Abstract*—In recent years, locomotion research has showcased agile, difficult maneuvers onboard quadrupedal platforms including parkour, jumping, and backflips through ingenuity at the control level. Despite such motor-level capabilities, quadrupedal navigation and motion planning is still an unsolved problem for quadrupeds, especially in safety-critical environments. In this work, we extend a foothold planning framework powered by multi-modal hierarchical planning to allow for real-world perceptive locomotion. We enable fully perception-informed foothold planning in the perception space by integrating the image space-based semantic egocan representation. The egocan provides a 360° local environment representation and permits a minimal sensor profile for real-time reactive planning. The egocan provides both geometric and semantic environment information to capture unique legged affordances that can be exploited in downstream planning and control. Lastly, a superpixels-based oversegmentation process over the local terrain facilitates deliberate, exhaustive foothold planning. We benchmark against a set of state-of-the-art locomotion baselines, including model-based, model-free, and hybrid approaches using the ANYmal-C quadruped in simulation to better understand the impacts of perception, planning, and control design choices on performance. We deploy this proposed framework onboard a Unitree Go2 quadrupedal hardware platform using a custom computational suite and multi-machine ROS2-based networking setup to validate proposed claims.**

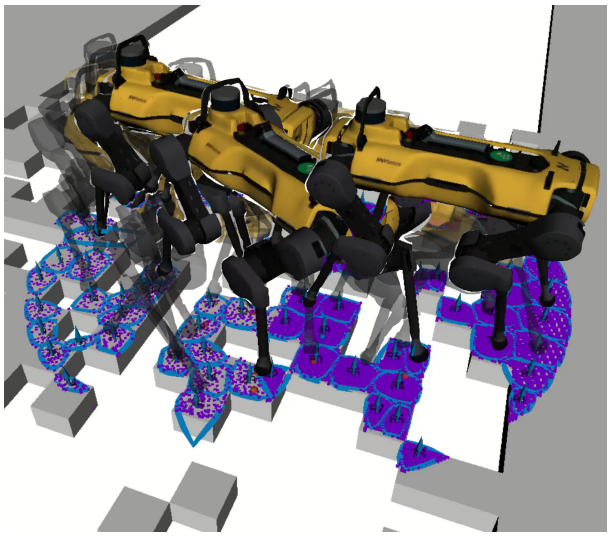


Fig. 1: Visualization of an ANYmal-C planning footholds over sparse terrain. The purple points represent the floor of the egocan environment representation. It is visualized as a pointcloud, but maintained for planning as part of a panoramic image. The blue boundaries and arrows are superpixel clusters which can be understood as planar steppable regions.

## I. Introduction

In recent years, quadrupedal robot platforms have showcased agile, difficult maneuvers including parkour [1], [2], jumping [3], and backflips, through significant efforts at the control level. Despite these motor-level capabilities, quadrupedal navigation and motion planning is still an unsolved problem for quadrupeds, especially in safety-critical environments. To see this, one can look no further than the DARPA Subterranean competition where many teams reported unsalvageable failures on their quadrupedal systems. While joint-level controllers enable spectacular dynamic feats, current quadrupedal navigation frameworks still do not appropriately capture the affordances of legged systems, such as the ability to step over or duck under obstacles.

Many environment models also employ Cartesian space world frame point cloud-based representations which can often act as the computational bottleneck for online frameworks. In this work, we present an extension of our previously shown quadrupedal footstep planning framework. Here, this framework is now fully perception-informed, relying on the perception space-based egocan representation. The egocan provides a 360° local environment representation while permitting low-cost sensor profiles. This egocan enables perception-informed planning in the real world while also maintaining an efficient perception space-based representation. We also supplement the egocan with additional geometric and semantic legged affordances that inform our navigation framework on our ability to plan footsteps at locations in the local environment.

Through this framework, we establish a "feasibility hierarchy" in which as plans escalate down the perception-planning-control hierarchy and increase in resolution and dimensionality, they are checked for increasingly detailed levels of feasibility: geometric checks to ensure footholds are planned on planar regions, kinematic checks to ensure footstep transitions are reachable, and dynamic checks to ensure that transitions can be performed under centroidal dynamics and friction constraints as well as joint position, velocity, and torque limits.

We perform simulation benchmarks against two state-of-the-art baselines: a grid map-based footstep planner to demonstrate the benefits of the PiPS framework, and a perception-informed reinforcement learning policy to demonstrate the importance of footstep planning in safety critical settings. Baselines are evaluated in a series of complex environments including rubble, balance beams, ramps, stepping stones, and stairs. We also perform hardware experiments to demonstrate the framework's ability to handle novel, challenging environments in both indoor laboratory settings and outdoor structured settings. Hardware experiments are enabled through two major components: (1) a custom computational suite including two mini PCs, a portable battery, and a single depth sensor, and (2) a ROS2-based autonomy stack.

In summary, our main contributions are,

1) Introduction of a novel image space-based geometric and semantic environment representation of legged affordances referred to as the semantic egocan
2) Adaptation of a primitive shapes-based synthetic data generation technique to learn to predict legged affordances of steppability, passability, and duckability
3) Adaptation of a superpixels-based oversegmentation approach to planar region extraction to integrate perception into an existing search and optimization foothold planner
4) Experimental validation of the perception-informed foothold planning framework in simulation on the ANYmal-C quadruped against model-based, model-free, and hybrid locomotion baselines
5) Untethered, fully sensor-informed hardware demonstrations on the Unitree Go2 quadruped in both indoor and outdoor settings

## II. Related Works

### A. Legged Affordances

The ways in which legged robot morphologies can interact with the environment are much more diverse than for wheeled or treaded counterparts. It then becomes important to capture these aspects in the local environment representation. One of these affordances is what much of the literature refers to as traversability [4]–[8]. Terrain traversability can be viewed as a continuous score assigned to a subsection of the local environment that reflects the quality of a potential foothold in that location. Traversability is often calculated as a function of terrain properties such as gradients [9], [10], height discontinuities [11], and curvature [12]. Traversability has also been learned through neural networks [13], [14]. This metric is typically stored in a 2D cost map that is either searched [6], [7] or optimized [5] over when it comes to planning. Some approaches do leverage the image space for terrain processing, but purely for the task of terrain class identification [15]. This motivates a deeper investigation into image space-based representations for legged platforms.

While the continuous scoring of terrain is an important metric for footstep planning, the discrete decomposition of the local environment into *steppable* and *non-steppable* regions is also relevant. The level of granularity can differ across approaches, with some just making a binary classification between these two labels [13], [16] while others further decompose the environment into gait- or behavior-specific regions such as jumping [17]. While the label scheme in [17] is similar to the one presented in this proposed work, all methods discussed here maintain a robot-centric 2D grid-based representation. Another common approach is to decompose the environment into a set of polygons [18]–[21] that represent support regions for footholds. Such a representation fits nicely into optimization frameworks for footstep planning.

One final interesting affordance that is unique to legged platforms is the ability to duck or crouch underneath overhanging obstacles [20], [22].

Many of the cited works here demonstrate low-level controllers that can appropriately capture these types of affordances.

### B. Legged Environment Representations

Within an autonomy stack, one of the earliest design decisions is how to represent the robot's local environment. In the literature for legged robots, perhaps the most popular model is that of the 2.5D height map or elevation map [9], [13], [14], [16], [23]–[26]. An elevation map is a grid structure represented in a Cartesian frame, either a world frame or robot-aligned frame, in which each entry of the grid represents the height of the terrain at the given $(x, y)$ position. Here, the 2.5D stems from the notion that for a dense, layered environment, an environment model may capture many different heights: the ground level, a ceiling level, or an intermediate level including obstacles. In choosing which level to capture, some information is lost from the full 3D representation, hence the 2.5 dimensions. This representation has roots in the earliest stages of perception-informed legged locomotion, namely the DARPA Learning Locomotion [27], [28] project where high-resolution height maps were provided to each team. Height maps are popular due to their simple structure, Cartesian frame representation, and open-sourced implementation [29]. However, constructing such a model typically requires point cloud processing which often necessitates GPU parallelization and/or multi-threading to operate in real-time.

Elevation maps can be used to build further representations such as occupancy map [11] through height thresholding or fast collision checking, cost maps [5] which build cost functions based on neighboring height information, signed distance fields [21], [30], [31] which calculate gradient information regarding distance between the environment and the robot, and even neural scenes [32] which use learned networks to fill in gaps in environment representations due to occlusions or sensor artifacts. Elevation maps are also often used to construct planar polygonal regions [14], [16], [18], [19], [21], [33], [34] where point cloud processing steps can be performed to extract planar regions of the environment that are flat enough to support a robot footstep.

For environments where it is important to capture multiple height layers, including floors, obstacles, and ceilings, then 3D environment representations are often deployed such as the voxel grid [35]–[37]. In this case, all three dimensions are discretized, giving the user the freedom to model layers at the cost of memory and computation. For the DARPA SubT competition which occurred in underground settings including tunnels and caves, such representations saw significant usage [38], [39]. Another flavor of voxel grids is that of the Octomap [40] which exploits environmental sparsity to build a more efficient volumetric grid.

### C. Planning in the Perception Space

A perception and planning paradigm that is of key importance to this proposed work is that of Planning in Perception Space (PiPS) [41]. A perception-space approach to planning eschews pre-processing steps for perception data and instead acts directly on the raw data representations that an external sensor provides. In the case of laser or LIDAR scans, this means keeping the range data in an egocentric reference frame

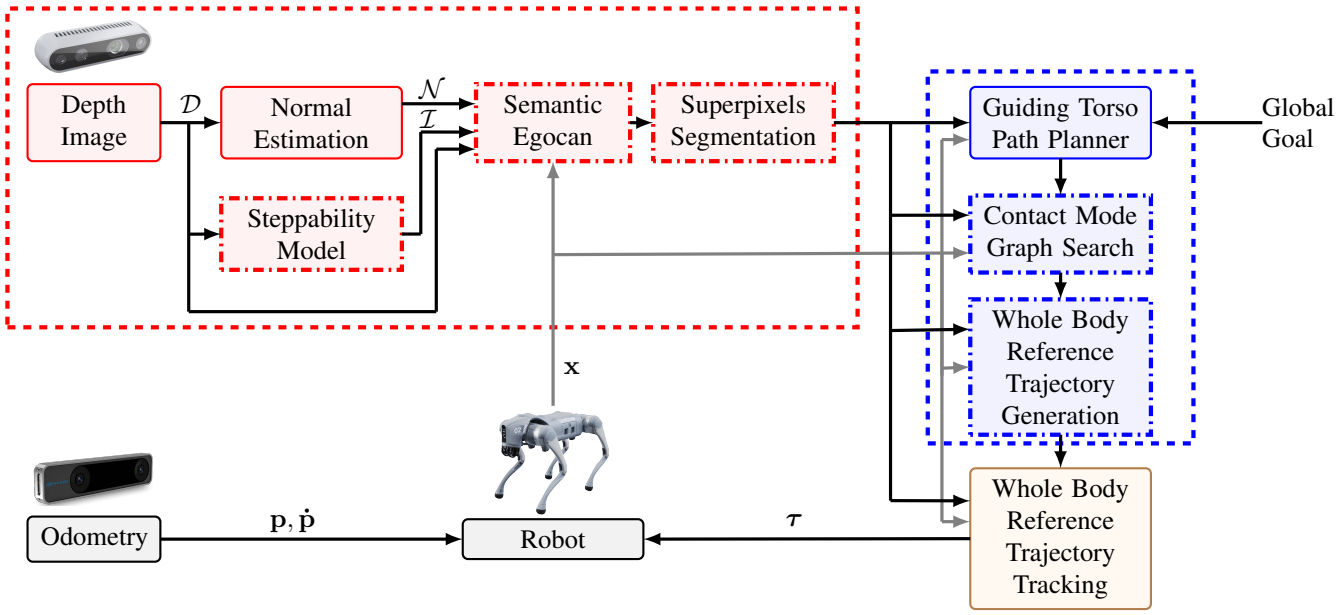



Fig. 2: Workflow for the QuadPiPS framework.

and recasting the task of local perception-informed navigation as a robot-centric decision-making process.

To date, PiPS has provided strong benefits for fast depth space collision-checking [41], collision-free local robot navigation [42]–[44], and task and motion planning [45]. The PiPS approach has also proven effective for both model-based [9], [46] and model-free [47], [48] approaches to legged motion planning. However, a thorough investigation into how the PiPS paradigm should be applied to foothold planning has yet to be performed. To date, torso-level planning and end-to-end approaches have dominated the literature.

While PiPS can faciliate certain local robot processes, the perception space has its drawbacks as well. The rich nearby depth information is offset by sparse environment information at further distances which limits the spatio-temporal horizons in which planning in the perception space is effective.

## III. Overall QuadPiPS Framework

The autonomy stack proposed in this work is open-sourced and is provided on GitHub [1].

The semantic egocan environment representation is comprised of three layers: depth, surface normals, and steppability labels. Depth information is obtained from an incoming stereo image stream. Surface normals are estimated from incoming depth image gradients. Steppability labels are predicted through a semantic segmentation model. New affordance information is inserted into the egocan, and old data is propagated forward in time to enable a 360° view of the local environment despite limited sensor fields of view. Near ground and under foot data is maintained through an *egocan floor* image which is oversegmented into superpixel planar regions which are passed on for foothold planning.

The incoming set of planar regions is used to synthesize a planning graph online for footholds. This graph is built online according to kinematic reachability constraints, allowing the structure and sparsity of the local environment to inform planning. This planning graph is searched over using a simple $A^*$ algorithm using solely a Euclidean distance heuristic to guide the search. Once this search returns a set of kinematically feasible stance configurations that takes the quadruped from start to goal, a set of trajectory optimization subproblems are run to synthesize dynamic whole body trajectories that carry the robot between the sequential stance configurations.

Once this whole body reference trajectory has been generated, it is sent to an MPC/WBC framework to perform tracking. The MPC formulation closely mirrors the formulation used to generate the reference trajectory, while the WBC formulation takes on a hierarchical QP structure, prioritizing dynamic feasibility constraints over less critical terms including reference trajectory tracking.

## IV. Semantic Egocan Representation

### A. Egocan Representation

*1) Egocylinder:* The concept of the egocylinder is introduced in [49] and expounded upon in [43]. The egocan is introduced in [50] for aerial vehicles, but has yet to see use in locomotion. Relevant aspects are briefly included in this work, and interested readers are referred to the prior citations.

The concept of the egocylinder is rooted in the paradigm of Planning in the Perception Space (PiPS). However, traditional depth cameras can only provide a limited field of view of roughly $60° - 90°$ horizontally. This view greatly limits the horizon of planning that one can perform with a single depth image. LIDARs can provide full 360° views of the environment, but this comes at the cost having to process an unordered, dense point cloud. Additionally, for a task such as perceptive locomotion where the environment is being revealed to the robot in real time, it is important to store prior views as the robot translates and rotates throughout

[1] TBD

the environment. To address this need, the egocylinder takes world points from a depth image sensor and projects them onto a virtual cylinder that surrounds the robot. Points on the cylinder are discretized into a panoramic image to leverage the favorable aspects of the image space representation such as parallelization. An example egocylinder image can be seen in Figure 3.

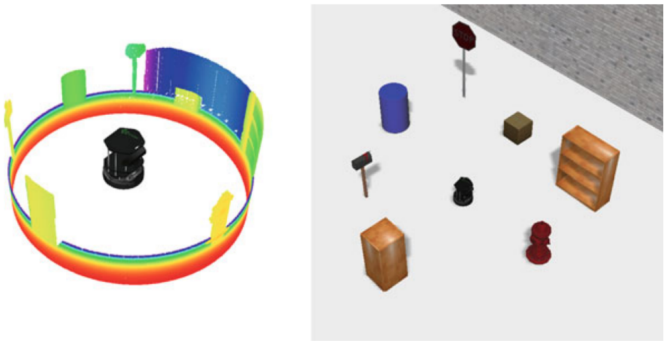


Fig. 3: Example egocylinder (left) and its respective scene (right). Proximity is represented from near to far as red to blue. Transparent parts of the egocylinder are due to depth sensing range limits.

The egocylinder takes as input a depth image $\mathcal{I}_{\text{depth}} \in \mathbb{R}^{w \times h \times 1}$. Each pixel $r = (u, v) \in \mathbb{R}^2$ stored in $\mathcal{I}_{\text{depth}}$ can be parameterized in the camera frame as $\mathbf{p}_{\text{cam}} = (x, y, z) \in \mathbb{R}^3$ where

$$\begin{aligned} x(u,v) &= \frac{u - u_0}{f} \cdot z(u,v) \\ y(u,v) &= \frac{v - v_0}{f} \cdot z(u,v) \end{aligned} \tag{1}$$

and $z(u, v)$ is the depth value read off of $\mathcal{I}_{\text{depth}}$ at pixel $r$. The pixel coordinate $(u_0, v_0)$ represents the center of the image and $f$ represents the focal length of the camera.

New points from incoming sensor streams are projected onto a virtual cylinder that surrounds the ego-robot. This cylinder is propagated forward in time as the robot moves. This cylindrical surface is discretized into a panoramic image $\mathcal{I}_{\text{cyl}} \in \mathbb{R}^{w_{\text{cyl}} \times h_{\text{cyl}} \times 1}$ which stores the egocylindrical range of each point on the cylinder. This range allows the egocylinder to retain information behind the robot. Therefore, incoming Cartesian camera frame information is transformed into egocylindrical coordinates via

$$\mathbf{p}_{\text{cyl}} = \begin{bmatrix} \rho \\ \theta \\ z_{\text{cyl}} \end{bmatrix} = T_{\text{c2e}}(\mathbf{p}_{\text{cam}}) = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \text{Arg}(x + \mathbf{j}z) \\ y \end{bmatrix}. \tag{2}$$

Resulting egocylindrical coordinates are then mapped to egocylindrical image coordinates $r_{\text{cyl}} \in \mathcal{I}_{\text{cyl}}$ using the homogeneous egocylinder projection matrix $K_{\text{cyl}}$:

$$r_{\text{cyl}} = K_{\text{cyl}} \begin{bmatrix} \theta \\ z_{\text{cyl}} \\ 1 \end{bmatrix} \tag{3}$$

where

$$K_{\text{cyl}} = \begin{bmatrix} f_{\text{cyl}} & 0 & u_{\text{cyl},0} \\ 0 & f_{\text{cyl}} & v_{\text{cyl},0} \end{bmatrix} \tag{4}$$

and $f_{\text{cyl}} = \cot(2\pi / w_{\text{cyl}})$, $u_{\text{cyl},0} = w_{\text{cyl}}/2$, and $v_{\text{cyl},0} = h_{\text{cyl}}/2$.

Existing egocylindrical coordinates are mapped back to Cartesian coordinates, propagated forward in time under a Euclidean transform for the induced pose $g_{\text{move}} \in SE(3)$ between the prior and current timesteps, and mapped back to egocylindrical coordinates:

$$\mathbf{p}'_{\text{cyl}} = T_{c2e} \circ g_{\text{move}} \circ T_{e2c}(\mathbf{p}_{\text{cyl}}) \tag{5}$$

where

$$T_{\text{e2c}}(\mathbf{p}_{\text{cyl}}) = \begin{bmatrix} \rho\cos(\theta) \\ z_{\text{cyl}} \\ \rho\sin(\theta) \end{bmatrix}. \tag{6}$$

The resulting egocylindrical coordinates are subsequentlly mapped to egocylindrical image coordinates for storage.

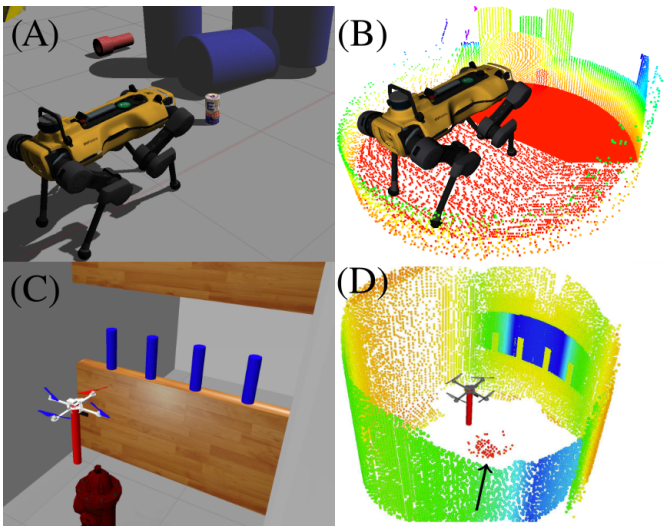



Fig. 4: Example egocan visualized as a point cloud (left) along with its respective depth image (top right) and color image (bottom right).

*2) Egocan:* During egocylindrical propagation, points that do not map onto the panoramic image are lost. This means that points that pass over or under the cylinder are not retained. To address this need, the egocan extends the egocylinder by adding upper and lower egocaps on top and bottom of the panoramic cylinder image to create a closed surface that maps the local environment all around the ego-robot. In the case of AerialLiPS [50], also visualized in Figure 4(C) and (D), the egocaps are used to determine if the UAV can safely increase or decrease its altitude. For this work, visualized in Figure 4(A) and (B), the lower cap is used to capture prior scenes as they travel underfoot, which are later used for footstep planning. Functionally, the egocan floor surface acts much like a height map. However, no point cloud processing is required to build it. The floor surface is also maintained as an image to leverage the benefits of the data structure.

Egocap surfaces map to the images $\mathcal{I}_{\text{lower}}, \mathcal{I}_{\text{upper}} \in \mathbb{R}^{w_{\text{cap}} \times h_{\text{cap}} \times 1}$ where $w_{\text{cap}} = h_{\text{cap}}$. If a new or existing point

$\mathbf{p}_{\text{cyl}}$ does not map onto the cylinder, it is mapped onto $\mathcal{I}_{\text{lower}}$ if $z_{\text{cyl}} > 0$ and $\mathcal{I}_{\text{upper}}$ otherwise. Throughout this work, $\mathcal{I}_{\text{lower}}$ may be referred to as the egocan floor image. The depth data for $\mathbf{p}_{\text{cyl}}$ will be placed at the cap image coordinate $r_{\text{cap}}$ where

$$r_{\text{cap}} = K_{\text{cap}} \begin{bmatrix} x/|y| \\ z/|y| \\ 1 \end{bmatrix} \tag{7}$$

and

$$K_{\text{cap}} = \begin{bmatrix} f_{\text{cap}} & 0 & u_{\text{cap},0} \\ 0 & f_{\text{cap}} & v_{\text{cap},0} \end{bmatrix}. \tag{8}$$

The cap focal length $f_{\text{cap}} = \frac{v_{\text{fov}} w_{\text{cap}}}{4}$ where $v_{\text{fov}}$ is the vertical field of view of the egocylinder which is a user-specified parameter that typically ranges between $\pi/4$ and $\pi/2$. The cap center coordinate $u_{\text{cap},0} = w_{\text{cap}}/2$, and $v_{\text{cap},0} = h_{\text{cap}}/2$.

### B. Image Space Normal Estimation

For the task of perceptive locomotion over complex terrain, additional environmental affordances are required beyond just depth. This proposed work adopts the depth image gradients-based approach [51] for image space-based surface normal estimation. The approach is briefly covered here, and more information can be found in the original paper. The notation $\cdot_{\text{cam}}$ for the camera frame will be temporarily dropped in this section for clarity.

This normal estimation technique relies on building a local planar approximation at the point $\mathbf{p}(u, v)$ along with its directional derivatives $\mathbf{p}_u(u, v)$ and $\mathbf{p}_v(u, v)$. These directional derivatives can be computed as

$$\begin{aligned} \mathbf{p}_u(u,v) &= \left(\frac{\partial x(u,v)}{\partial u}, \frac{\partial x(u,v)}{\partial u}, \frac{\partial z(u,v)}{\partial u}\right) \\ \mathbf{p}_v(u,v) &= \left(\frac{\partial x(u,v)}{\partial v}, \frac{\partial y(u,v)}{\partial v}, \frac{\partial z(u,v)}{\partial v}\right). \end{aligned} \tag{9}$$

The partial derivative terms can then be calculated as

$$\begin{aligned} \frac{\partial x(u,v)}{\partial u} &= \frac{z(u,v)}{f} + \frac{(u-u_0)}{f}\frac{\partial z(u,v)}{\delta u} \\ \frac{\partial y(u,v)}{\partial u} &= \frac{(y-y_0)}{f}\frac{\partial z(u,v)}{\delta u} \\ \frac{\partial x(u,v)}{\partial v} &= \frac{(u-u_0)}{f}\frac{\partial z(u,v)}{\delta v} \\ \frac{\partial y(u,v)}{\partial v} &= \frac{z(u,v)}{f} + \frac{(y-y_0)}{f}\frac{\partial z(u,v)}{\delta v}, \end{aligned} \tag{10}$$

where

$$\begin{aligned} \frac{\partial z(u,v)}{\delta u} &\approx z(u+1,v) - z(u,v) \\ \frac{\partial z(u,v)}{\delta v} &\approx z(v+1) - z(u,v). \end{aligned} \tag{11}$$

Lastly, the terrain normal can be calculated as

$$\mathbf{n}(u,v) = \mathbf{p}_v(u,v) \times \mathbf{p}_u(u,v) \tag{12}$$

and subsequently normalized:

$$\hat{\mathbf{n}}(u,v) = \frac{\mathbf{n}(u,v)}{\|\mathbf{n}(u,v)\|_2}. \tag{13}$$

These pixel-wise normals comprise the normal image $\mathcal{I}_{\text{normal}} \in \mathbb{R}^{w \times h \times 3}$. Example scenes and image space normals are shown in Figure 5.

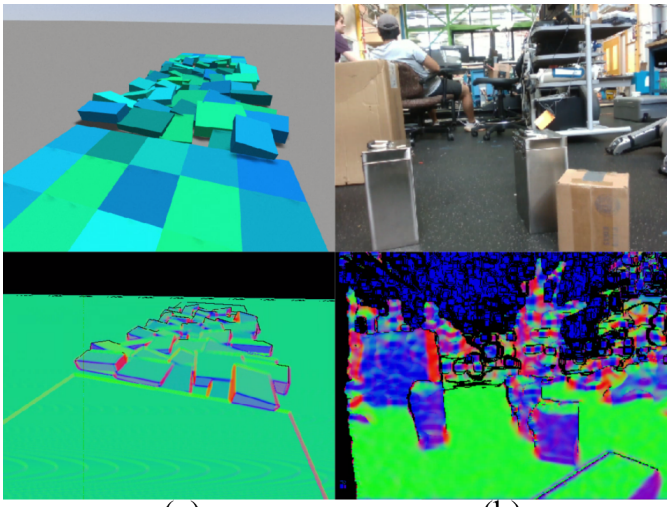



Fig. 5: Example scenes and image space normals. The color scheme of RGB corresponds to the XYZ axes of the camera frame. The absolute value of normal vector values is used to calculate colors, this is why normals pointing both left and right are primarily red and upright normals are primarily green. Column (a) is a simulation scene of scattered stepping stones, and column (b) is a real world scene of foregrounded prismatic objects with backgrounded clutter.

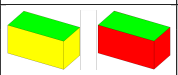
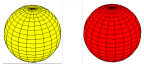
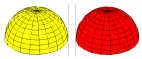
### C. Steppability Dataset Generation

Depth and normal information comprises the geometric affordances present in the semantic egocan representation. However, it is also important to be able to semantically reason about local terrain. In this section, the process in which steppability labels are predicted for the local environment is detailed. First, the simulation scene creation process used to generate synthetic steppability data is discussed. Here, we adapt a primitive shapes-based technique that was previously used for manipulation tasks [52], [53]. In this work, the authors hypothesize that the geometry of graspable objects can be decomposed into a set of primitive shapes where each primitive shape class has a particular family of effective grasps. The ground truth labels of these objects can then be ascertained through the color of the primitives within the simulation scene. We perform a similar process here, but instead utilize class labels concerning steppability. The synthetic scenes used for training data are assembled according to a key set of design parameters that we detail now.

*1) Primitive Shape Classes:* We use nine primitive shape classes to build scenes: *Cuboid*, *Cylinder*, *Ramp*, *Sphere*, *Semisphere*, *Pipe*, *Pole*, *Tube*, and *Floor*. The Cuboid and Ramp classes are parameterized by length $l$, width $w$, and height $h$, the Cylinder class is parameterized by $x$-dimensional radius $r_x$, $y$-dimensional radius $r_y$, and height $h$, the Sphere and Semisphere classes are parameterized by radius $r$, and

the Pipe, Pole, and Tube classes are parameterized by length $l$ and radius $r$. The floor class is non-parametric in that all of its instances are 4 m $\times$ 4 m. Visualizations of each shape class along with design parameter ranges are shown in Table I.

TABLE I: Primitive Shape Classes and Visualizations

| Primitive Shape Class | Geometry Visualizations | Label Visualizations | Parameter Range (unit: m) |
|---|---|---|---|
| Cuboid |  |  | $l \in [0.2, 1.0]$<br>$w \in [0.1, 0.50]$<br>$h \in [0.05, 0.25]$ |
| Ramp |  |  | $l \in [0.2, 1.0]$<br>$w \in [0.1, 0.50]$<br>$h \in [0.05, 0.25]$ |
| Cylinder |  |  | $r_x \in [0.10, 0.50]$<br>$r_y \in [0.10, 0.50]$<br>$h \in [0.05, 0.25]$ |
| Sphere |  |  | $r \in [0.025, 0.05]$ |
| Semisphere |  |  | $r \in [0.025, 0.05]$ |
| Pipe |  |  | $l \in [0.10, 0.50]$<br>$r \in [0.025, 0.05]$ |
| Pole |  |  | $l \in [0.10, 0.50]$<br>$r \in [0.025, 0.05]$ |
| Tube |  |  | $l \in [0.50, 1.0]$<br>$r \in [0.025, 0.05]$ |
| Floor |  |  | N/A |

*2) Primitive Shape Steppability Policies:* Each primitive shape class is assigned a mesh-based steppability policy that defines which faces of the shapes should be assigned which labels. The three labels used are:

- **Steppable (Green):** can support a stable foothold
- **Passable (Yellow):** can not support a stable foothold, but can be stepped over by a foot swing trajectory
- **Non-passable (Red):** can not support a stable foothold and can not be stepped over by a foot swing trajectory

For Cuboids, Ramps, and Cylinders, the top face is labeled as steppable due to its flat horizontal geometry. If the height of the primitive exceeds a maximum swing height for the robot leg, in this work defined as $h_{\max} = 0.10$ m, then all vertical faces are labeled as non-passable. Otherwise, the vertical faces are labeled as passable. For Spheres and Semispheres, the entire primitive is labeled as non-passable if the diameter and radius respectively exceed $h_{\max}$. Otherwise, the entire primitive is labeled as passable. For Pipes, Poles, and Tubes, the entire primitive is labeled as non-passable if the $z-$dimension of its pose exceeds $h_{\max}$. Otherwise, the entire primitive is labeled as passable. Lastly, floors are labeled as completely steppable.

*3) Primitive Shape Pose:* The six-dimensional pose of each primitive shape can be set according to desired scene attributes. For instance, scenes can be set to feature clusters of primitive shapes localized within a particular region of the scene, the primitives can be scattered all throughout the scene, or the poses can be overwritten to accept manually defined entries if the user wants to create more contrived scenes that include structures such as staircases or stepping stones. The $z$-dimension of all shapes is restricted so that all shapes are placed on support surfaces, and the orientations of Cuboids, Ramps, and Cylinders are restricted to ensure that the face labeled as steppable remains as the top face in the scene.

*4) Camera pose:* The six-dimensional pose of the camera placed within each simulation scene can also be parameterized to emulate expected real-world circumstances for onboard sensing. Given that our desired application is quadrupedal locomotion, we set the camera at a height of $z = 0.325$ m and pitch it downwards by 30° to approximate the pose of the depth camera attached to our quadrupedal hardware platform. To capture multiple frames of a single scene, the camera is set to follow a prescribed trajectory that approximates how a quadruped's torso would move through a real world environment. Gaussian noise is also applied to all six pose dimensions to capture the jitter that the camera would experience during locomotion in the real world.

*5) Scene Environment:* Lastly, the overall synthetic environment that the primitive shapes are placed within can also be controlled. In this work, we randomly select between indoor environments that include walls and a ceiling and outdoor environments that instead include an infinite horizon.

From each scene, a depth image $\mathcal{I}_{\text{depth}}$ is extracted along with a ground truth steppability mask $\mathcal{I}^*_{\text{step}}$.

### D. Model Training

For model training, we use the off-the-shelf DeepLabV3+ [54] model from the Detectron2 deep learning library [55]. To construct the dataset, 600 scenes were generated with 5 frames each, making for a total of 3,000 images. In total, dataset generation took roughly 12 hours. The generated data was put into 80%/10%/10% splits of 2,400, 300, and 300 images for training, validation, and test subsets respectively. Model training took 65 minutes on an Intel Xeon W-2223 CPU at 3.60GHz with an NVIDIA T1000 GPU. Plots of training loss and intersection-over-union can be seen in Figure 6. Example scene data including depth images, ground truth labels, and model predictions is shown in Figure 7.

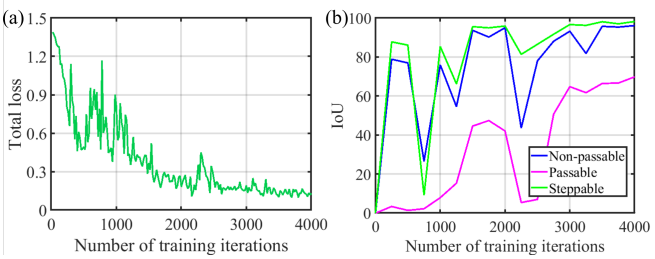



Fig. 6: Results of training. (a) Total training loss over the training iterations. (b) Intersection-over-union (IoU) of all classes over the training iterations.

The output of the model is represented as the predicted steppability mask $\mathcal{I}_{\text{step}}$.

### E. Semantic Egocan Representation

The semantic egocan takes as input a multi-channel affordance image $\mathcal{I}_{\text{afford}} = \begin{bmatrix}\mathcal{I}_{\text{depth}} & \mathcal{I}_{\text{normal}} & \mathcal{I}_{\text{step}}\end{bmatrix}$. The egocan

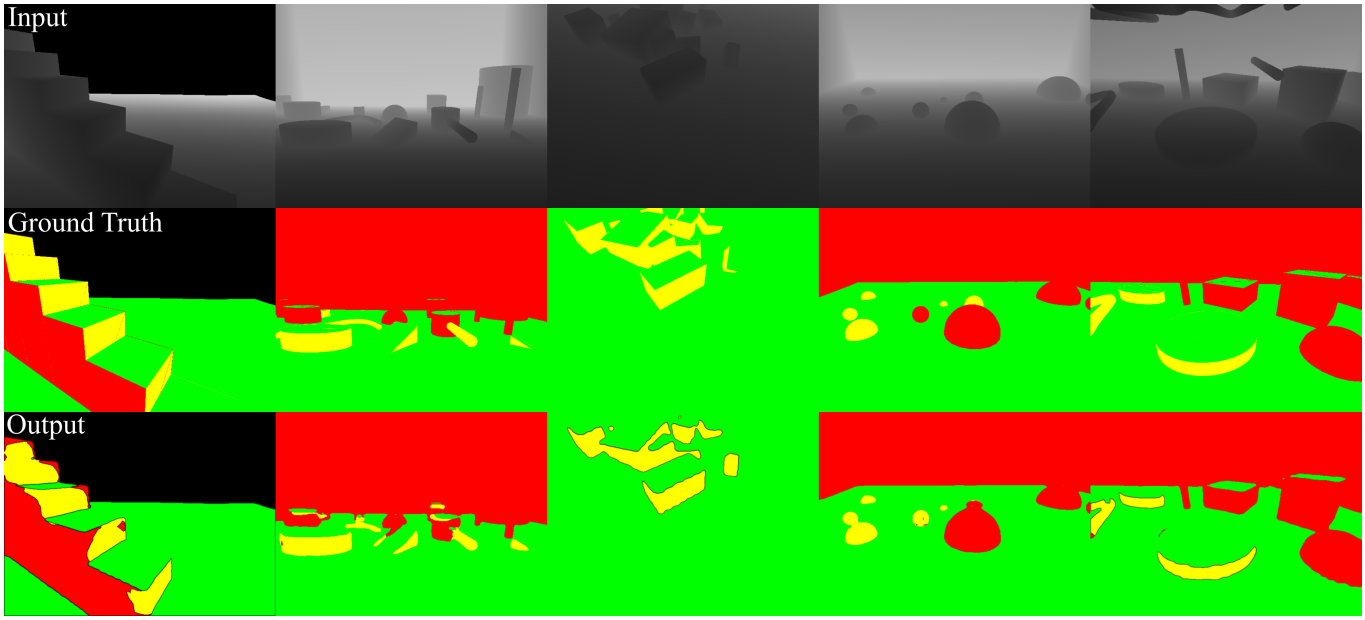



Fig. 7: Example outputs of learned steppability model. Top row: input depth images to the model. Middle row: ground truth steppability labels of the corresponding column's input depth image. Bottom row: model outputs for the corresponding column's input depth image.

propagation step is kept the same, the only difference being that additional normal and steppability information is stored along with depth information. From this point onward, the egocan-related notation including $\mathcal{I}_{\text{lower}}$ is meant to represent the multi-channel images of the semantic egocan.

### *F. Superpixels Oversegmentation*

Once the semantic egocan has been populated, it is still necessary to identify what parts of the terrain can support footholds. It is common to decompose a height map into a set of planar convex regions that can support footholds. While such regions work well for linear optimization constraints, the question of how to search over these regions is somewhat unclear. If the ego-robot is presented with a large, flat floor, one large convex region may be generated. This is not conducive to a search framework. For this reason, the authors adopt an oversegmentation approach via superpixels.

Superpixel algorithms [56], [57] are intended to cluster pixels into perceptually homogeneous regions which are evenly distributed throughout the image space. These homogeneous regions, or superpixels, are meant to modify or replace the uniform gridding of the pixel grid within the image. In the case of this work, superpixels act as an insightful primitive that allows the foothold graph search to not only select which terrain features to plan footholds on, but also decide where within the feature to step.

As input, this superpixel algorithm takes the egocan floor image $\mathcal{I}_{\text{lower}}$. Prior to passing $\mathcal{I}_{\text{lower}}$ in for oversegmentation, a health check is performed on $\mathcal{I}_{\text{lower}}$. Due to the nature of the egocan propagation, the floor image is sparse. Normally, superpixel algorithms are deployed on full RGB or RGB-D images. However, in this use case, a majority of the pixels lack data. First, the image channels are cleaned, where pixels with no data or invalid data are set to zero. Pixels with unstable normals, defined as $\hat{\mathbf{n}}(u,v) \cdot -\hat{\mathbf{e}}_y \leq 0.5$ are also set to zero to avoid building planar regions with them. Pixels with labels that are not assigned steppable are also removed.

When the egocan is initialized, $\mathcal{I}_{\text{lower}}$ is empty. However, this yields no planar regions to plan with. This proposed planning framework makes the assumption that planning begins on flat, obstacle-free terrain. Therefore, regions of the floor image that have not yet been populated by sensor data are initialized to default heights and normals that represent standard ground floor terrain. At this point, the floor image can now be passed to the superpixels algorithm. This algorithm is detailed below and shown in Algorithm 1. The oversegmentation algorithm can be understood as a modified version of the Simple Linear Iterative Clustering (SLIC) algorithm that is designed to handled sparse depth image.

The initial superpixel cluster centers $\mathcal{S}$ are defined by uniformly sampling pixels in $\mathcal{I}_{\text{lower}}$ at regular grid steps $s = \sqrt{N/k}$. Each cluster $S_k \in \mathcal{S}$ contains a pixel $r_k = (u_k, v_k)$, a depth $y_k$, and a normal $\hat{\mathbf{n}}_k$. The value $N = w_{\text{cap}} \times h_{\text{cap}}$ is the number of pixels in the image and $k$ is the desired number of superpixels. These initial cluster also undergo a refinement step in which they are moved to the centroid of the $s/4 \times s/4$ neighborhood surrounding the initial cluster center. This helps in generating smoother superpixel regions.

Then, the proposed algorithm iterates through each cluster and calculates the distance between the current cluster $S_k$ and the current pixel $r$. Only pixels in a $2s \times 2s$ neighborhood are evaluated. The distance metric used in Algorithm 1 scores deviation from the local plane formed by the cluster $S_k$ as well as cluster compactness. The complete distance calculation is

$$d = w_n \frac{d_n}{d_n^{\max}} + w_p \frac{d_p}{d_p^{\max}} + w_w \frac{d_w}{d_w^{\max}} + w_c \frac{d_c}{d_c^{\max}}. \tag{14}$$

The term

$$d_n = 1 - \hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_k \tag{15}$$

measures deviation from the cluster normal $\hat{\mathbf{n}}_k$ and the current pixel's normal $\hat{\mathbf{n}}_i$. The term

$$d_p = |(\mathbf{p}_k - \mathbf{p}_i) \cdot \hat{\mathbf{n}}_k| \tag{16}$$

measures distance between the current pixel's camera frame position $\mathbf{p}_i$ and the cluster plane $(\mathbf{p}_k, \hat{\mathbf{n}}_k)$. The term

$$d_w = \|\mathbf{p}_k - \mathbf{p}_i\|_2, \tag{17}$$

measures Euclidean distance between the current point and the cluster center, and the final term

$$d_c = \sqrt{(u_k - u_i)^2 + (v_k - v_i)^2} \tag{18}$$

measures Euclidean pixel distance between the current pixel and the cluster center. The values $w_\cdot$ are positive scalar values for weighting importance. The notation $d_\cdot^{\max}$ represents the maximum distance for the respective term. These are applied to roughly scale the distance terms.

Once pixel-wise distances are calculated and clusters are assigned, cluster centroids are updated using the newly assigned pixels. Cluster normals are also refined via RANSAC. This assignment and update process is repeated for a user-specified number of iterations $I$. A diagram for the workflow is also shown in Figure 8.

**Algorithm 1:** Depth-based SLIC

**Input:**
$k$ ; /* Desired number of superpixels */
$I$ ; /* Number of iterations */
/* Initialize */
Initialize cluster centers $\mathcal{S}_k$;
/* Adjust */
Move $S_k \in \mathcal{S}$ to centroid in $s/4 \times s/4$ neighborhood;
/* Iterate */
**for** *iteration* $i = 1 : I$ **do**
  Set distance $d(r) \leftarrow \infty$ for each pixel $r$;
  Set label $l(r) \leftarrow \infty$ for each pixel $r$;
  **for** *each cluster* $S_k$ **do**
    **for** *each pixel* $r$ *in* $2s \times 2s$ *region around* $S_k$ **do**
      /* Assign */
      Compute distance $d$ between $S_k$ and $r$;
      **if** $d < d(r)$ **then**
        $d(r) \leftarrow d$;
        $l(r) \leftarrow k$;
      **end**
    **end**
  **end**
  /* Update */
  Compute new cluster centroids;
  /* Refine */
  Refine cluster normals via RANSAC;
**end**

### G. Convex Planar Region Generation

Downstream foothold planning expects a set of convex planar regions as a characterization of the local environment.

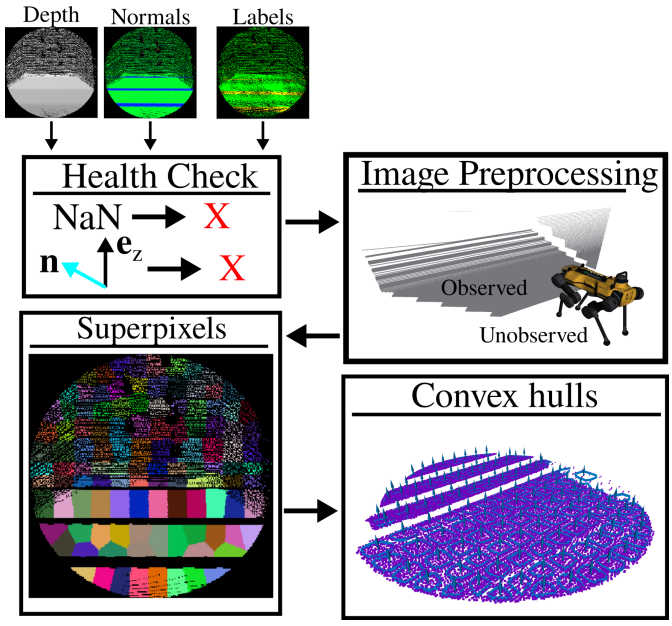



Fig. 8: Information flow for the proposed oversegmentation and planar region extraction method. Inputs are the multi-channel floor image $\mathcal{I}_{\text{lower}}$. First, the image is cleaned to remove all invalid pixels as well as pixels corresponding to non-upright normals and non-steppable labels. Next, regions of the image that have not yet been populated from sensor data are initialized to standard ground plane data. Then, the superpixels algorithm is run to generate a set of pixel clusters, and those clusters are then passed to a convex hullification step to obtain closed planar regions.

Therefore, we must convexify these superpixel regions. It is important to acknowledge here that these regions themselves do not necessarily describe convex regions of planar surface in the world. We rely on the level of granularity of these regions to be high enough so that when they are convexified, they are not falsely classifying large sections of unsafe terrain as planar regions.

First, all points in each superpixel cluster $S_k$ are projected onto the plane described by $(\mathbf{p}_k, \hat{\mathbf{n}}_k)$ from the region's centroid. Then, this set of 2D points is passed to the Graham Scan algorithm [58] to compute the convex hull. The set of 2D points that define the hull are then passed on to foothold planning in the form of a set of planar regions $\mathcal{P}$. Each region $P \in \mathcal{P}$ is defined by a pose $\mathbf{p}_0 \in \mathbb{R}^6$ and a list of 2D boundary points $\{\mathbf{b_0}, \mathbf{b_1}, \ldots, \mathbf{b_{n_b}}\} \in \mathbb{R}^{2 \times n_b}$ that lie in the plane defined by $\mathbf{p}_0$. The number of boundary points $n_b$ can vary from region to region.

## V. Quadrupedal Motion Planning

### A. Problem Statement

Consider a quadrupedal robot with a configuration space $\mathcal{Q} \subset \mathbb{R}^{6+n_j}$. The robot configuration $\mathbf{q} \in \mathcal{Q}$ is comprised of a floating base pose $\mathbf{q}_b \in \mathbb{R}^6$ and a sequence of $n_j$ joints $\mathbf{q}_j \in \mathbb{R}^{n_j}$. The pose $\mathbf{q}_b$ can be further decomposed into a position $\mathbf{p}_b = \begin{bmatrix} x_b & y_b & z_b \end{bmatrix}$ and orientation $\boldsymbol{\theta}_b = \begin{bmatrix} \alpha_b & \beta_b & \gamma_b \end{bmatrix}$. The proposed planning framework must travel from a starting configuration $\mathbf{q}_{\text{start}}$ to a goal region $\mathcal{Q}_{\text{goal}} = \{\|\mathbf{q}_b - \mathbf{q}_{\text{goal}}\| < \epsilon\}$. In this work, the environment is entirely unknown and partially observable. Therefore, the robot must its onboard sensing to reveal the environment online. As stated before, this

framework assumes access to a depth image sensor providing $\mathcal{I}_{\text{depth}}$. Multiple camera streams are supported, but only one is used in this work.

### B. Torso Search

First, a coarse initial search is performed to construct a guiding torso path through the local environment. This initial planning problem is framed as a simple graph search over the floating base pose $\mathbf{q}_b \in \mathbb{R}^6$. The graph itself will be represented as $\mathcal{G}_{\text{torso}} = (\mathcal{N}_{\text{torso}}, \mathcal{E}_{\text{torso}})$. For each planar region $P \in \mathcal{P}$, a set of nodes $\{N_{P,0}, \dots N_{P,K}\}$ are created at the region pose $\mathbf{q}_0$ plus an offset. The pose of each node is offset in the direction of the region normal by a nominal torso height $H$ and in the region yaw by $\Delta\gamma$. An edge $E$ is added between nodes with a Euclidean distance under $d^{\max}_{\text{torso}}$ and a yaw difference under $\Delta\gamma_{\max}$.

The node closest to the global goal is marked as the goal node. The typical A* search algorithm [59] is employed with Euclidean distance plus yaw difference as the edge weight metric and cost-to-go heuristic. The sequence of torso nodes is passed onto the foothold search in the form of a guiding torso path $\mathcal{T} = \{\mathbf{q}_{b,0}, \dots, \mathbf{q}_{b,M}\}$ where the furthest pose $\mathbf{q}_{b,M}$ is taken as the local waypoint $\mathbf{q}_{\text{LG}}$ to be reached during footstep planning.

### C. Foothold Search

This proposed planning approach adapts its core hierarchical philosophy from the Augmented Leafs with Experience on Foliations (ALEF) framework [60]. Here, the task of foothold planning is decomposed into its discrete and continuous planning spaces. The discrete space consists of the potential footholds, and the continuous space consists of the whole body configurations that the robot can assume to perform said footholds. In this section, the discrete foothold space will be discussed. More information is provided in prior work [61].

Foothold planning is treated as a search problem over the lower-dimensional discrete planning space. This search employs the graph $\mathcal{G}_{\text{foot}} = (\mathcal{N}_{\text{foot}}, \mathcal{E}_{\text{foot}})$ in which each node $N_{\text{foot}} \in \mathcal{N}_{\text{foot}}$ represents a contact mode family. A contact mode family is a partial stance in which a proper subset of the quadruped's feet are in contact with a unique combination of planar regions in the environment. For a single contact mode family, each stance foot must be in contact with a different region. Each edge $E_{\text{foot}} \in \mathcal{E}_{\text{foot}}$ represents a transition between two partial stances, meaning that the edge itself is a full stance in which all feet are in contact. The next section details how this mode family transition graph is created.

*1) Mode Family Transition Graph Construction:* To initialize the mode family transition graph, a starting mode family $N_{\text{start}}$ is constructed from the current robot state and inserted into a priority queue $Q$. This proposed framework employs lazy graph construction: during the search, the highest priority node $N_{\text{high}}$ in $Q$ is popped and its neighboring nodes and their respective edges are only added to the graph once $N_{\text{high}}$ is expanded. The process in which nodes and edges are added to graph is detailed below.

**Contact Sequence:** First, a contact sequence constraint is applied. While this constraint is not strictly necessary and relaxing it would enable acyclic planning, it does help to reduce the number of potential transitions. Whichever contact phase the current node is in, the candidate nodes must satisfy the following phase of a user-defined gait.

**Reachability:** Once the next set of stance feet has been decided, a set of reachable regions $\mathcal{R} = \{\mathcal{R}_{\text{FL}}, \mathcal{R}_{\text{FR}}, \mathcal{R}_{\text{RL}}, \mathcal{R}_{\text{RR}}\}$ are compared against the current set of perceived planar regions $\mathcal{P}$ to determine which reachable planar regions $\mathcal{P}_{\text{reach}} = \{\mathcal{P}_{\text{FL}}, \mathcal{P}_{\text{FR}}, \mathcal{P}_{\text{RL}}, \mathcal{P}_{\text{RR}}\}$ can be transitioned to for the current swing feet. The reachable regions are defined as superquadrics [62], an expressive family of 3D volumes. The set of points that fall within the superquadric centered at $(x_0, y_0, z_0)$ are defined as

$$S = \left\{(x,y,z) \in \mathbb{R}^3 \;\middle|\; \left|\frac{x-x_0}{A}\right|^a + \left|\frac{y-y_0}{B}\right|^b + \left|\frac{z-z_0}{C}\right|^c \leq 1\right\}, \tag{19}$$

where scalars $A, B, C$ and $a, b, c$ control dimensions and curvature, respectively. Visualizations of the reachable volumes used for the Go2 and Anymal can be seen in Figure 9.

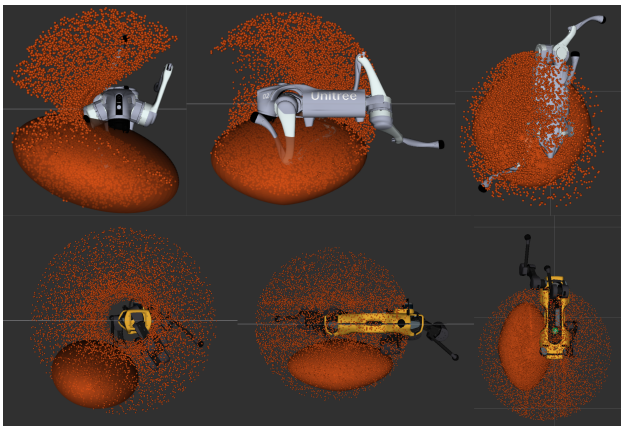


Fig. 9: Visualizations of reachable volumes for Unitree Go2 (top row) and ANYmal C (bottom row). Only the reachable volumes for the front right foot are visualized here. Four volumes in total are generated, one for each foot.

For each planar region $P \in \mathcal{P}$, the planar region pose $\mathbf{q}_0$ is evaluated in Equation 19 to determine if it can be reached for a given foot. Then, this graph expansion step can iterate through $\mathcal{P}_{\text{reach}}$. Each combination of reachable regions defines a candidate node $N_{\text{cand}}$ that must be evaluated for addition to the graph.

**Stance stability:** The stance formed by the edge between $N_{\text{curr}}$ and $N_{\text{curr}}$ is checked for stability. A nominal torso pose $\mathbf{q}_{\text{nom}}$ is estimated from this candidate edge, and the width $w_{\text{front}}, w_{\text{rear}}$ and length $l_{\text{left}}, l_{\text{right}}$ of the stance feet positions are evaluated against a set of ideal stance positions taken at stand up. The stance positions are calculated as

$$\begin{aligned} w_{\text{front}} &= \|R^T(\boldsymbol{\theta}_{\text{nom}}) \cdot (\mathbf{p}_{\text{FR}} - \mathbf{p}_{\text{FL}}) \cdot \hat{\mathbf{e}}_y\|_2 \\ w_{\text{rear}} &= \|R^T(\boldsymbol{\theta}_{\text{nom}}) \cdot (\mathbf{p}_{\text{RR}} - \mathbf{p}_{\text{RL}}) \cdot \hat{\mathbf{e}}_y\|_2 \\ l_{\text{left}} &= \|R^T(\boldsymbol{\theta}_{\text{nom}}) \cdot (\mathbf{p}_{\text{FL}} - \mathbf{p}_{\text{RL}}) \cdot \hat{\mathbf{e}}_x\|_2 \\ l_{\text{right}} &= \|R^T(\boldsymbol{\theta}_{\text{nom}}) \cdot (\mathbf{p}_{\text{FR}} - \mathbf{p}_{\text{RR}}) \cdot \hat{\mathbf{e}}_x\|_2 \end{aligned}, \tag{20}$$

and stability is represented as

$$\begin{aligned} |w. - w_{\text{ideal}}| &< \epsilon_{\text{width}} \\ |l. - l_{\text{ideal}}| &< \epsilon_{\text{length}} \end{aligned} \tag{21}$$

where $\hat{\mathbf{e}}_x$ and $\hat{\mathbf{e}}_y$ are basis vectors in the base frame and $\epsilon_{\text{width}}, \epsilon_{\text{length}}$ are user-defined error thresholds.

**Swing distance:** Lastly, a constraint is applied to the distance between the liftoff and touchdown positions of the swing feet for the candidate node.

$$\|\mathbf{p}_{\text{liftoff}} - \mathbf{p}_{\text{touchdown}}\|_2 \leq d_{\max}, \tag{22}$$

where $d_{\text{swing}}^{\max}$ is a user-defined maximum swing distance.

If all constraints are met, the candidate node $N_{\text{cand}}$ is constructed if it does not already exist. An edge is added between $N_{\text{curr}}$ and $N_{\text{cand}}$ if the edge does not already exist as well.

*2) Mode Family Transition Graph Search:* To plan a discrete sequence of footholds, we perform a search over the mode family transition graph $\mathcal{G}_{\text{foot}}$. The graph search itself follows a standard $A^*$ [59] implementation.

*3) Edge weight:* For a transition between source node $N_i$ and destination node $N_{i+1}$, the graph edge $E = (N_i, N_{i+1})$ is assigned the weight

$$\Delta c(N_i, N_{i+1}) = d_{\text{torso}}(N_i, N_{i+1}) \tag{23}$$

where the term $d_{\text{torso}}(N_i, N_{i+1})$ is the Euclidean distance between nominal torso positions for $N_i$ and $N_{i+1}$.

*4) Cost-to-go:* For the A* search, a node $N$ is assigned the search heuristic value

$$g(N) = d_{\text{CoM}}(N, \mathbf{q}_{\text{LG}}) \tag{24}$$

to ensure an admissible search heuristic and therefore provide optimal foothold sequences with respect to edge weights.

This graph search returns a discrete sequence of footholds that are then passed to a whole body trajectory optimization (TO) program to generate a full trajectory.

### D. Swing Trajectory Optimization

*1) Trajectory Optimization Subproblem Formulation:* Our lower-level TO problem solves over the robot state $\mathbf{x} = [\mathbf{h}, \mathbf{q}_b, \mathbf{q}_j]$, and the robot input $\mathbf{u} = [\mathbf{f}, \mathbf{v}_j]$. The term $\mathbf{h} = [\mathbf{k}, \mathbf{l}]^T \in \mathbb{R}^6$ is the centroidal momentum which includes linear $\mathbf{k} \in \mathbb{R}^3$ and angular $\mathbf{l} \in \mathbb{R}^3$ momentum, $\mathbf{f} \in \mathbb{R}^{12}$ are the contact forces for all four feet, and $\mathbf{v}_j \in \mathbb{R}^{12}$ are the joint velocities. The TO formulation for generating a transition trajectory between two stance configurations is written as:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{u}} \quad & \|\mathbf{x}[N] - \mathbf{x}^{\text{des}}[N]\|_{Q_f}^2 + \\ & \sum_{k=0}^{N-1} \left( \|\mathbf{x}[k] - \mathbf{x}^{\text{des}}[k]\|_Q^2 + \|\mathbf{u}[k]\|_R^2 \right) \\ \text{subject to} \quad & \\ \text{(Planar Region)} \quad & F_l(\mathbf{q}[k], P_l) = \mathbf{0} \qquad \forall l \in \mathcal{C}_i \qquad (25a) \\ \text{(Dynamics)} \quad & \dot{\mathbf{x}}[k] = f_{\text{cent}}(\mathbf{x}[k], \mathbf{u}[k]) \qquad (25b) \\ \text{(Friction)} \quad & \mathbf{f}_l[k] \in \mathcal{F}_l(\mu, \mathbf{q}[k]) \qquad \forall l \in \mathcal{C}_i \qquad (25c) \\ & \mathbf{f}_l[k] = \mathbf{0} \qquad \forall l \notin \mathcal{C}_i \qquad (25d) \\ \text{(Feasibility)} \quad & \mathbf{q}_j \in \mathcal{Q}_{\text{feas}} \qquad (25e) \\ & \mathbf{v}_j \in \mathcal{V}_{\text{feas}} \qquad (25f) \end{aligned}$$

where $f_{\text{cent}}$ are the centroidal dynamics described in [61] and [63], $\mathcal{F}_l(\mu, \mathbf{q})$ represents the friction cone which depends on the friction coefficient $\mu$ and robot pose $\mathbf{q}$. The planar region constraint function $F_l(\mathbf{q}[k], P_l)$ is represented as a set of half-space constraints defined by the boundary of the planar region $P_l$ for swing foot $l$. The set $\mathcal{C}_i$ represents the set of stance feet for the contact mode family, or node, $N_i$. The cost matrices $Q$ and $R$ are defined through user-defined hyperparameters, and $Q_f$ is computed from a linear-quadratic regular (LQR) solution at a nominal state and input.

*2) Implementation Considerations:* The above TO formulation is implemented as a Sequential Quadratic Program (SQP) and solved through the OCS2 library [64]. A time horizon of $T = 0.5$ seconds and $N = 50$ knot points with maximal 25 iterations is employed for each swing trajectory subproblem. The Pinocchio library [65] is utilized for kinematics and dynamics calculations. The desired state trajectory $\mathbf{x}^{\text{des}}$ is generated in two steps. First, a contact projection step is run to project a randomly sampled target configuration into contact satisfying the source and destination contact mode families. If this first step is successful, cubic splines are then synthesized for the swing feet to build out the remainder of $\mathbf{x}^{\text{des}}$.

The solutions to the sequence of swing trajectory subproblems are appended to form a long-horizon reference trajectory. This reference trajectory is subsequently run through a refinement TO problem to smoothen any irregularities between subproblems. After refinement, the reference trajectories $\mathbf{x}^{\text{ref}}$ and $\mathbf{u}^{\text{ref}}$ are sent to an MPC module for real-time tracking.

## VI. Motion Tracking Control

### A. Reference Trajectory Publication Monitoring

The proposed motion planner publishes the reference trajectory for tracking and subsequently monitors the robot state for events that signal a re-plan. One such event is that the robot completes the previously sent reference trajectory. Another such event is that the quadruped begins to tip or becomes unstable, signaled by an abnormally large roll or pitch or abnormally large joint velocities or torques. If so, the quadruped is commanded to abort the prior reference trajectory and maintain stability before re-planning.

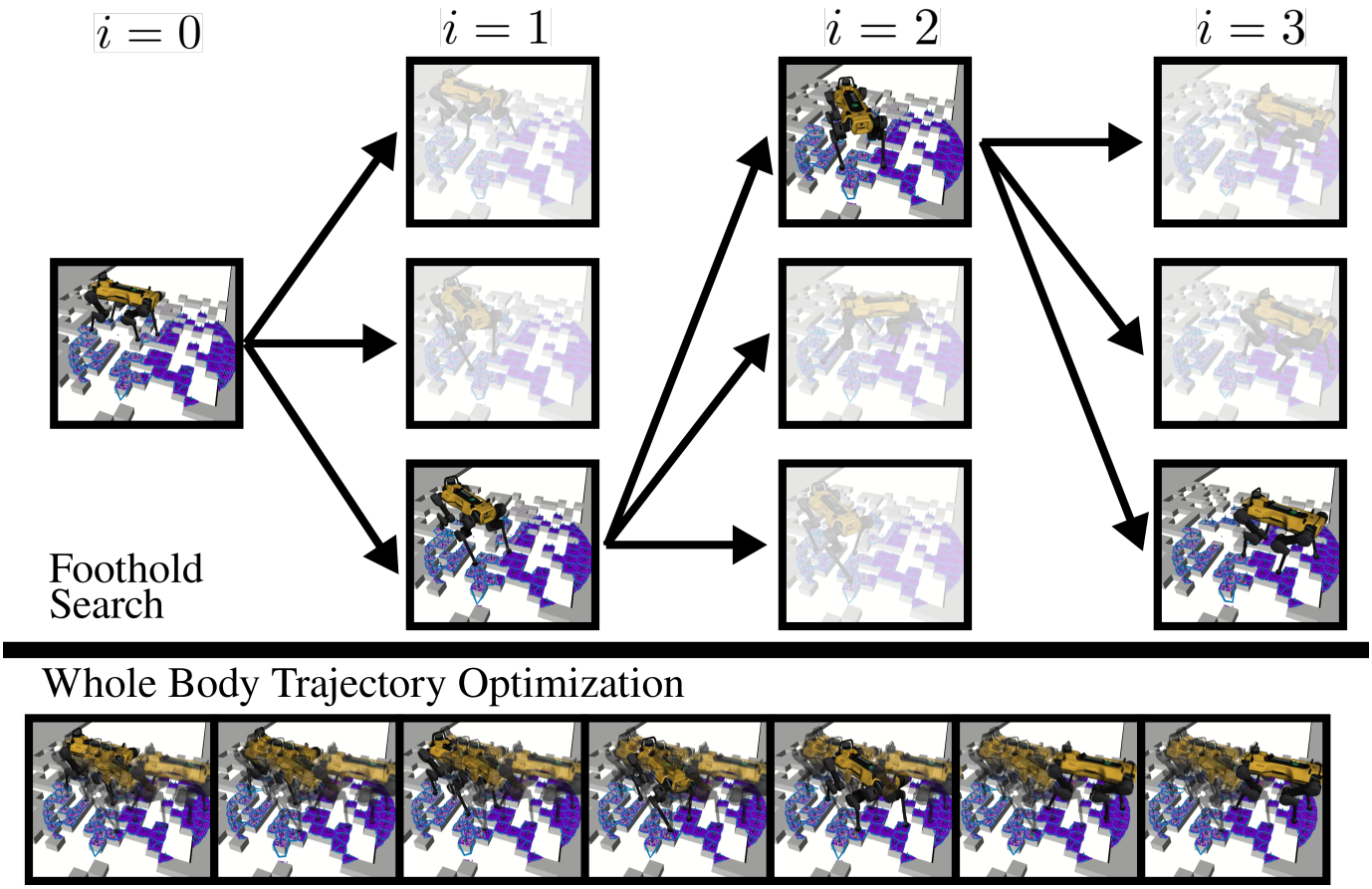



Fig. 10: Overall diagram of planning framework. A graph search (Section V-C) is performed over contact mode family transitions defined through a series of geometric and kinematic constraints. Then, the suggested contact sequence is passed to a long-horizon trajectory optimization (Section V-D) problem to synthesis the whole body reference trajectory.

### B. MPC Trajectory Tracking

The Model Predictive Controller (MPC) follows the exact same formulation as that given in Equation 25. For the MPC, a timestep $\Delta t$ of 0.015 seconds is employed over a time horizon of 1.0 seconds. The solver is also warm-started for each new solution using the prior solution and subsequently run for only 5 SQP iterations. At this level, the reference trajectory $\mathbf{x}^{\text{ref}}$ encourages the MPC to step on the selected regions from the graph search, but this is not forced and the MPC has the authority to assign swing feet to different regions if this is necessary to maintain stability. The MPC also synthesizes new swing trajectories depending on the current foot positions to afford the solver more flexibility to deviate from and return to the reference trajectory when necessary.

### C. WBC

The optimal state $\mathbf{x}^*$ and input $\mathbf{u}^*$ from the MPC are passed on to a whole-body controller (WBC) to synthesize joint torque commands. This WBC takes the form of a hierarchical quadratic program (QP) adapted from [66].

## VII. Experimental Results

### A. Image Pre-processing

*1) Simulation:* A bilateral filter is applied to the incoming depth image to aid in normal estimation. This filtered depth image is then passed along to the egocan.

*2) Hardware:* A series of filters are applied from the Realsense API [67]. These are displayed in order of application in Table II. Visualization of the effects of this post-processing are depicted in Figure 11.

TABLE II: Active Depth Filters

| Filter | Description |
|---|---|
| Decimation | Reduces depth scene complexity by scaling down image in proportion to aspect ratio |
| Depth-to-Disparity | Transforms image into disparity domain (1 / Distance) |
| Spatial | Smooths over reconstructed data while preserving edges |
| Temporal | Improves depth data persistency according to prior image frames |
| Disparity-to-Depth | Transforms image back into depth domain (1 / Disparity) |
| Hole Filling | Fills in missing data using neighboring pixel information |

### B. Experimental Setup - Software

This proposed foothold planner is integrated into our `legged_software` codebase which is an extension of [68].

*1) State estimation:* In simulation, experiments are run with ground truth state estimation so that we can purely

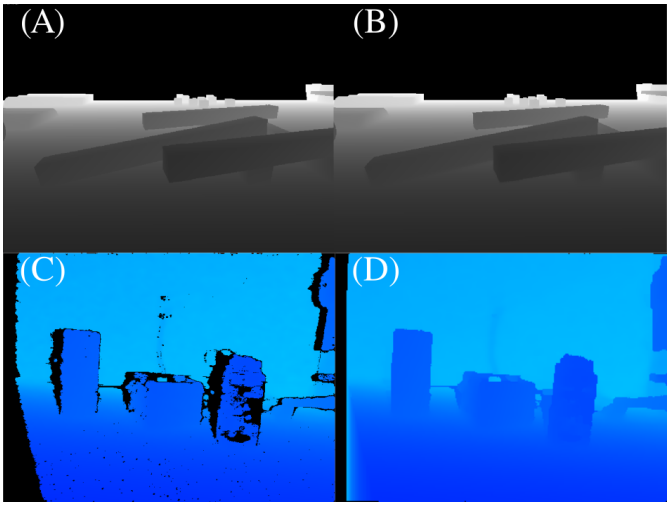



Fig. 11: Example depth images. (A) Raw depth image from simulation, and (B) Pre-processed depth image from simulation. (C) Raw depth image from the real world, and (D) Pre-processed depth image from the real world.

evaluate the performance of the perception, planning, and control. On hardware, `legged_software` supports both proprioceptive and exteroceptive methods of state estimation. Hardware experiments are run with the Contact-Aided Kalman Filter [69] along with an Intel RealSense T265 Localization Camera. The torso orientation $\theta_{\text{torso}}$ is observed directly from the onboard IMU. The KF is used to estimate $x_{\text{torso}}$ and $y_{\text{torso}}$ while the T265 is used to estimate the $z_{\text{torso}}$.

*2) Finite state machine:* Within our autonomy stack, we have a set of modes that the quadruped can operate within:

1) Passive
2) Recovery Stand
3) Perceptive Locomotion MPC

The passive mode sends zero torque commands to the motors to have the robot go prone. This mode is useful for ensuring safety during start-up and deployment. The Recovery Stand mode performs an open-loop joint-level tracking routine that stands the quadruped up from an initial laying down position. This mode sends high P-gain commands to ensure the quadruped stays stiff. This is useful for resetting and transporting the quadruped between hardware runs. Lastly, the Perceptive Locomotion mode is where the previously detailed MPC and WBC modules are deployed to allow command velocity or reference trajectory tracking through the provided Unitree handheld controller.

*3) Safety Observer:* The final notable module within our autonomy framework is a safety observer which previews the robot state as well as torque commands. If the robot's state enters manually defined unsafe regimes such as an abnormally large roll or pitch or a commanded joint configuration that lies beyond the limits, the observer forces a switch into the Passive mode. The same applies to abnormally large contact forces or joint torques. The module also forces a switch to Passive if the MPC or WBC modules fail to find a solution.

### C. Experimental Setup - Hardware

The previously discussed autonomy stack is deployed onboard the Unitree Go2 quadrupedal hardware platform. To exploit proper hardware advantages and enable minimal communication delay, we run this stack through a multi-machine setup that communicates through ROS2 [70] via Cyclone DDS middleware [71]. The Unitree Go2 itself employs DDS-based communication for internal communication which motivated the choice to configure ROS2 to communicate through DDS.

*1) ROS2 networking:* There are two PCs onboard the quadruped that collectively run the autonomy stack. First, an NVIDIA Jetson Orin NX 16GB with 8-core Arm Cortex A78AE v8.2 64-bit CPU 2MB L2 + 4MB L3 runs all of the perception modules. An Intel RealSense D435i camera streams in depth images with 640x480 dimensions at roughly 60 Hz in simulation and 45 Hz on hardware. This Jetson device then runs depth image normal estimation, egocan propagation, and superpixel oversegmentation all on CPU while running steppability segmentation on GPU. Second, a Minisforum UM890 Pro Mini PC with AMD Ryzen 9 8945HS + 32 GB RAM runs all the core autonomy framework along with planning and control. A third device on the quadruped handles internal communication, sensor readings, and API requests. This third machine sends out joint state readings and receives joint torque commands over ROS2+DDS. Lastly, the user performs remote visualization and monitoring and sends high-level waypoint commands on a Dell XPS13 laptop (add specs) over ROS2+DDS as well. This networking setup is visualized in Figure 12. Communication between the Jetson, MiniPC and Go2 internal PC is all done through ethernet for minimum latency. Communication between the MiniPC and the laptop is done over WiFi on a private network to ensure real-time performance.

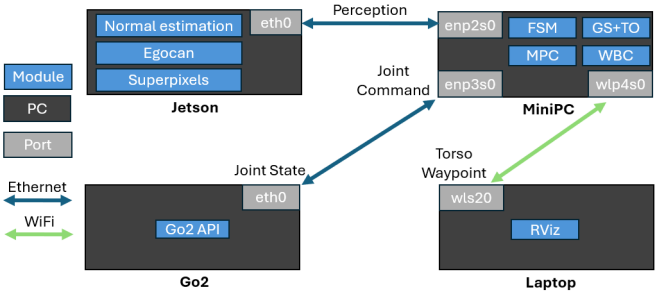



Fig. 12: Networking diagram for hardware setup onboard Unitree Go2.

*2) Computational Suite:* To mount all of the previously mentioned computing devices along with the necessary sensors onboard the Go2, we designed and fabricated a custom mounting setup. This setup is visualized in Figure 13. A machined aluminum rail plate can be screwed into the two horizontal mounting rails that sit on top of the Jetson device that comes equipped onto the Unitree Go2 Edu version we use in this work. Then, a Vesa mount is screwed into the bottom of the Mini PC and fixed to the rail plate with screws and nuts. A riser is 3D printed from PLA and fixed to the rail plate as well with screws and nuts. This riser creates some room to slide zip ties underneath and over the battery we use to power the

Mini PC. While the zip ties may not be a permanent design, they provide an easy way to quickly attach and remove the battery for charging and testing.

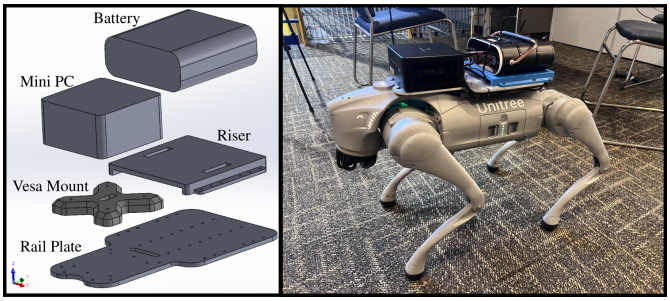



Fig. 13: Left: exploded view of the CAD assembly for our custom mounting setup. Right: Custom mounting setup onboard the Unitree Go2 quadruped.

### D. Simulation Benchmarking

To contextualize our proposed foothold planning approach amongst the state of the art, we benchmark our work against several other methods for footstep planning.

*1) Baselines:* At the perception level, we benchmark against a suite of elevation map-based representations. At the planning and control levels, we benchmark against an MPC-style approach that generates heuristic foothold positions and locally adjusts them according to the perceived terrain, an RL-based end-to-end approach that performs vision-to-action-style joint torque control, and a hybrid approach that combines the benefits of the prior two methods. Implementations for these three baselines are adapted from [72]. We test with two types of our approached work: (1) an MPC-style approach using the superpixels pipeline to allow for pure perception pipeline evaluation, and (2) the full QuadPiPS foothold planner. More information is provided below.

**Elevation map + Model Predictive Control (EM+MPC) [21]:** This baseline performs elevation mapping [14] to obtain a local 2.5D grid representation that is then passed on for additional steps including filtering, classification, segmentation, and precomputation. Inpainting and median filters are applied to account for occlusions and reduce noise. Binary steppability classification is performed to ignore regions of the grid map that can not support a foothold. Then, planar region segmentation is performed via connected component labelling and contour extraction. Lastly, an SDF is computed over the map for collision detection. The MPC formulation follows the same design as Equation 25. This is done so that the perception pipelines can be compared with all downstream planning and control set to be the same.

**Elevation map + Reinforcement Learning (EM+RL) [73]:** This baseline employs the same elevation mapping pipeline, but passes a downsampled map along with a twist command and proprioceptive state information into a multi-layer perceptron which in turn outputs a leg phase offset and residual joint position target. This policy is trained through a student-teacher learning framework run in parallelized simulation environments.

**Elevation map + Deep Tracking Control (EM+DTC) [74]** This baseline employs the same elevation mapping pipeline, but employs both a model-based foothold planner as well as a model-free tracking controller. A terrain-aware trajectory generation scheme [75] is employed to provide desired footholds, desired base poses, twists, and accelerations, and desired joint positions. Then, a learned RL policy tracks the desired state and outputs target joint positions. The policy is trained through a custom PPO implementation along with massively parallel simulations using a terrain curriculum.

**Superpixels + Model Predictive Control (S+MPC):** This baseline employs the proposed perception pipeline in this work. This oversegmented set of planar regions is passed into the MPC formulation defined in Section VI-B. The MPC formulation then tracks a reference torso trajectory extrapolated one second into the future over the perceived terrain. Footholds are generated through a heuristic calculation and local adjustment based on nearby planar regions.

**Quadrupedal footstep Planning in the Perception Space (QuadPiPS):** The entire proposed framework in this work.

*2) Environments:* We benchmark in 10 different environments designed to evaluate distinct terrain attributes: *Ramp*, *Stairs*, *Rubble*, *Pegboard*, *Balance beam*, *Ramped balance beam*, *Ramped stepping stones*, and *Sparse stones*. All environments can be seen in Figure 14.

*3) Perception Timing:* Average timing metrics are taken across 10 runs in the Ramp simulation environment. A module-by-module breakdown on timing for the proposed perception timeline is given in Figure 15. The same breakdown for the elevation mapping pipeline is given in Figure 16.

While the steppability segmentation is not able to run in real-time given GPU hardware constraints onboard the quadruped, we do not restrict the overall update rate of the perception pipeline to this bottleneck. When a new segmentation mask is available, that information is included in the egocan update, but otherwise, new points are added with an unknown steppability label and prior points with labels are propagated forward. This allows us to run the perception pipeline at 15 Hz.

The elevation mapping pipeline update rate is comparable to the egocan update rate as the two base environment representations. However, the convex planar region decomposition pipeline can run at roughly four times the speed of the superpixel over-segmentation. This is the price we pay for performing this over-segmentation, but this level of granularity pays off in performance as will be seen in Section VII-D4.

*4) Benchmarking Results:* For each baseline/environment combination, 10 trials are run. For each trial, the robot is placed at the same start position and commanded to walk to the same goal position. All baselines outside of QuadPiPS take a twist command as input. For these baselines, we simply send a twist in the XY plane from the current robot position to the goal position. For QuadPiPS, the global goal position is provided to the foothold planner. All trials are run on an ANYmal-C platform due to the limited availability of learned baselines across quadrupedal platforms. Pre-trained models for the EM-RL and EM-DTC were available for the ANYmal-C platform, and the proposed work is comprised of model-based planners that can be adapted to any quadrupedal platform with minimal tuning. For simulation trials, ground truth odometry is

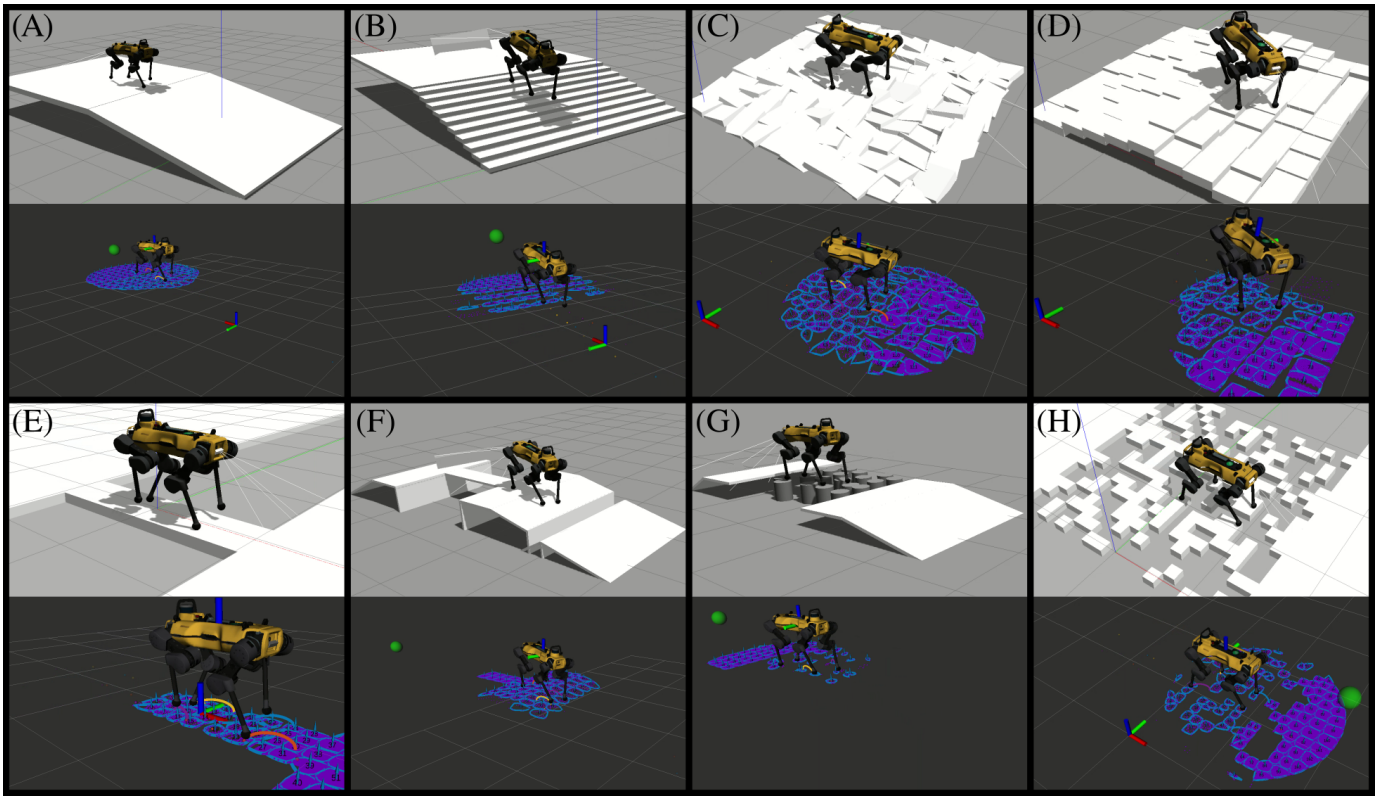



Fig. 14: Visualization of environments with corresponding superpixels. (A) Ramp, (B) Stairs, (C) Rubble, (D) Pegboard, (E) Balance Beam, (F) Ramped Balance Beam, (G) Ramped Stepping Stones), and (H) Sparse Stones.

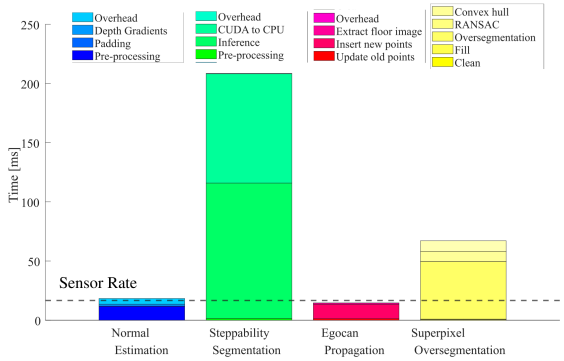



Fig. 15: Average timing for each module in the proposed perception pipeline.

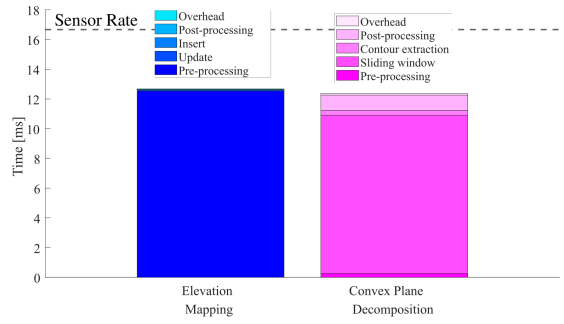



Fig. 16: Average timing for each module in the grid map-based perception pipeline.

provided. This is done so that solely the perception, planning, and control modules can be compared and their performance can be maximized. For each trial, baselines have a maximum of two minutes to reach the goal. If they fail to reach the goal in time, the trial is marked as a timeout. If the quadruped falls over or steps off of the evaluated terrain elements, the trial is marked as a failure. Otherwise, if the baseline reaches the goal under the allotted amout of time, the trial is marked as a success.

**Ramp:** Baseline results for the ramp environment are depicted in Figure 17(A), and a visualization of the environment itself along with a sample set of superpixels is provided in Figure 14(A). All trials across all baselines were successful, except for one EM-MPC trial. During this trial, the whole-body controller hit the maximum number of working set recalculations during its QP solution and returned a poorly conditioned set of joint torques. This caused the quadruped to collapse. While somewhat of an outlier, this trial does highlight a key element to the model-based and model-free control tradeoff. Model-based optimization problems have larger variance in solve times due to features such as initial conditions and time-varying constraints. Model-free controllers represented as neural networks have more consistent inference times which can be seen in the clustered runs for EM-RL and EM-DTC.

**Stairs:** Baseline results for the stairs environment are depicted in Figure 17(B), and a visualization of the environment itself along with a sample set of superpixels is provided in Figure 14(B). All trials across all baselines were successful,

except for one QuadPiPS trial. This trial failed when a reference trajectory was received by the MPC module at a slight time delay due to the nature of ROS2 networking. Tracking was delayed and the robot lurched forward in attempt to catch up to the reference trajectory, causing the robot to fall. This trial highlights the brittle nature of model-based planners. Even small perturbations from assumed conditions such as this can be enough to cause failures.

**Rubble:** Baseline results for the rubble environment are depicted in Figure 17(C), and a visualization of the environment itself along with a sample set of superpixels is provided in Figure 14(C). The EM-MPC baseline sustained one failure whereas the S-MPC baseline sustained two failures. These failures arose from unexpected contact with the environment. In the case of EM-MPC, this was due to the smoothed grid map reporting incorrect terrain heights. In the case of S-MPC, these arose from inaccurate swing trajectory tracking causing contact with the side of stones as opposed to the top of stones. This is another example of the type of small perturbation from expected conditions, including assumed contact sequences, that model-predictive controllers can struggle to reject.

**Pegboard:** Baseline results for the pegboard environment are depicted in Figure 17(D), and a visualization of the environment itself along with a sample set of superpixels is provided in Figure 14(D). Three failures are observed for both the EM-MPC and QuadPiPS baselines. EM-MPC failures stemmed from the large step up required to initially mount the pegboard. As can be seen from Figure 17(D), the failures are clustered towards the start of the pegboard. As for QuadPiPS, observed failures largely came from instability during tracking. The exhaustive nature of the graph search requires intermittent stopping for re-planning. However, for environments such as this, maintaining forward momentum can be crucial to staying upright. In the case of the pegboard, pausing at precarious stances caused the ANYmal to tip over.

**Balance beam:** Baseline results for the balance beam environment are depicted in Figure 17(E), and a visualization of the environment itself along with a sample set of superpixels is provided in Figure 14(E). In this setting, the EM-RL baseline exhibited two failures whereas the S-MPC baseline exhibited one failure. This is the first environment in which precise foothold planning becomes important. In prior settings, if a desired foothold was not accurately tracked, the environment is structured in such a way that there is always neighboring terrain where the robot can step. In the case of the balance beam, an errant step while the robot is along the beam will almost certainly mean failure. The EM-RL failures arose from the baseline's lack of explicit foothold planning. For the S-MPC baseline, the quadruped approached the beam at a slight offset and the MPC struggled to transition all four feet onto the beam, leading to a slip. The QuadPiPS foothold planner is able to execute lateral steps to align the robot with the beam before stepping onto the beam which proved to be useful in a handful of trials.

**Ramped balance beam:** Baseline results for the ramped balance beam environment are depicted in Figure 17(F), and a visualization of the environment itself along with a sample set of superpixels is provided in Figure 14(F). In this setting, the EM-RL baseline reports no successful trials whereas the EM-MPC and EM-DTC trials each report one successful trial. The S-MPC baseline reports nine successes whereas the QuadPiPS baseline report eight successes. For the elevation mapping baselines, most failures and concentrated on the balance beam. The two failure modes here were either mounting the balance beam at an offset and being unable to transition all four feat onto the beam, or getting onto the beam but losing stability and tipping over. A select few trials failed on the ramp up to the beam. The divot in the ramp presented a non-flat dip in terrain that some baselines stepped in and buckled under which led to a fall. The superpixels-based perception pipeline presented a significant benefit in this environment. The more granular region over-segmentation provided a natural decomposition over the balance beam that facilitated both the MPC and QuadPiPS planners with adjusting their desired footholds inwards onto the beam.

**Ramped stepping stones:** Baseline results for the ramped stepping stones environment are depicted in Figure 17(G), and a visualization of the environment itself along with a sample set of superpixels is provided in Figure 14(G). All elevation mapping baselines reported no successful trials in this environment. Moreover, these baselines made little to no progress over the stepping stones themselves. This largely came from the grid map representation. The continuous nature of the grid map representation led to the height information of neighboring stones being merged together, giving the false impression of one large stepping stone that can support footholds. This was true even when infilling and smoothing filters were removed. With this terrain representation, all baselines planned footholds over unsafe terrain and fell off the stepping stones. The superpixels representation generate clean, disparate regions for each stepping stone which enabled stable foothold planning across the terrain. The superpixels baselines were the only planners capable of making progress across the stepping stones, and the observed failures often took place towards the end of the set of stones. These failures arose from slightly inaccurate foothold tracking which caused the robot to slip off the desired stepping stones and fall.

**Sparse stones:** Baseline results for the sparse stones environment are depicted in Figure 17(H), and a visualization of the environment itself along with a sample set of superpixels is provided in Figure 14(H). In this setting, the elevation mapping baselines all reported no successful trials, the S-MPC baseline reported three successful trials, and the QuadPiPS baseline reported eight successful trials. The elevation map representation exhibited similar issues to what was discussed for the Ramped Stepping Stones environment in which nearby stepping stones were merged into incorrectly large planar regions. As was the case in the Ramped Stepping Stones environment, these baselines were unable to make any progress through the stepping stones in this setting. The S-MPC was able to complete some trials, but the nominal foothold positions generated by the MPC proved to be inexpressive enough to provide consistently stable trajectories. All failures observed for S-MPC were from slipping off of stones due to an inability to accurately track the heuristically generated footholds. The search-based foothold planning in the QuadPiPS baseline proved to be

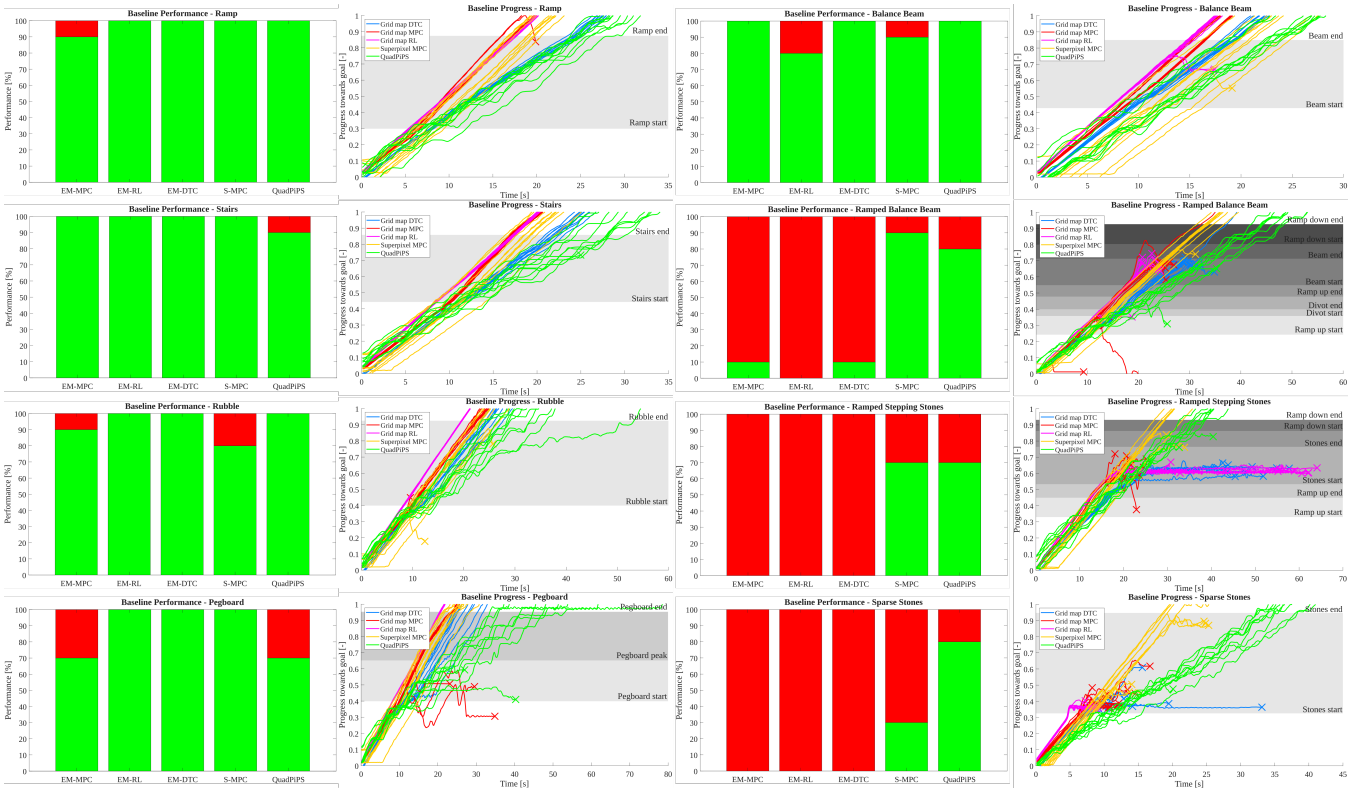



Fig. 17: .

enormously beneficial in finding stable stance configurations in this environment. QuadPiPS took longer to find a path to the goal, and these paths were often more circuitous compared to other baselines. However, for an environment such as this where terrain is sparse and precise foothold planning is critical, this tradeoff is heavily preferred.

## VIII. Conclusion

In this work, we introduce a novel image space-based method of predicting the steppability properties of the local environment for footstep planning. We adapt primitive shapes-based techniques from the domain of dexterous manipulation in order to efficiently generate diverse synthetic data for training a semantic segmentation model to perform steppability prediction, and we deploy this steppability model in our existing interleaved search and optimization contact planner in the form of a deep visual search heuristic. Through offline planning trials and online reactive navigation, we demonstrate how this steppability heuristic allows for expedited offline experience accumulation, proactive collision avoidance, and reactive recovery from disturbances.

In the future, we intend to run image-based perception elements such as depth img normal estimation and super-pixel oversegmentation on the GPU to further exploit the perception space representations. We also aim to convert the steppability model to C++ to enable faster inference times. We also aim to do propagation and warm-starting for superpixel generation to enable faster runtimes. Also would be worth doing some reconstruction. On the planning side, we aim to convert the proposed foothold planner into an interleaved search + optimization framework to be able to evaluate the dynamic feasibility of graph edges. We also plan on running the foothold planner in a receding horizon fashion so that the quadruped does not need to pause between plans, even if the pauses are minimal.

## REFERENCES


[1] X. Cheng, K. Shi, A. Agarwal, and D. Pathak, "Extreme Parkour with Legged Robots," Sep. 2023, arXiv:2309.14341 [cs, eess]. [Online]. Available: http://arxiv.org/abs/2309.14341

[2] Z. Zhuang, Z. Fu, J. Wang, C. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao, "Robot Parkour Learning," Sep. 2023, arXiv:2309.05665 [cs]. [Online]. Available: http://arxiv.org/abs/2309.05665

[3] Y. Yang, G. Shi, C. Lin, X. Meng, R. Scalise, M. G. Castro, W. Yu, T. Zhang, D. Zhao, J. Tan, and B. Boots, "Agile Continuous Jumping in Discontinuous Terrains," Sep. 2024, arXiv:2409.10923 [cs]. [Online]. Available: http://arxiv.org/abs/2409.10923

[4] Y.-C. Lin and D. Berenson, "Humanoid Navigation Planning in Large Unstructured Environments Using Traversability - Based Segmentation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018, pp. 7375–7382, iSSN: 2153-0866.

[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*, vol. 5, no. 4, pp. 5370–5376, Oct. 2020.

[6] A. Agha, K. Otsu, B. Morrell, D. D. Fan, R. Thakker, A. Santamaria-Navarro, S.-K. Kim, A. Bouman, X. Lei, J. Edlund, M. F. Ginting, K. Ebadi, M. Anderson, T. Pailevanian, E. Terry, M. Wolf, A. Tagliabue, T. S. Vaquero, M. Palieri, S. Tepsuporn, Y. Chang, A. Kalantari, F. Chavez, B. Lopez, N. Funabiki, G. Miles, T. Touma, A. Buscicchio, J. Tordesillas, N. Alatur, J. Nash, W. Walsh, S. Jung, H. Lee, C. Kanellakis, J. Mayo, S. Harper, M. Kaufmann, A. Dixit, G. Correa, C. Lee, J. Gao, G. Merewether, J. Maldonado-Contreras, G. Salhotra, M. S. Da Silva, B. Ramtoula, Y. Kubo, S. Fakoorian, A. Hatteland, T. Kim, T. Bartlett, A. Stephens, L. Kim, C. Bergh, E. Heiden, T. Lew, A. Cauligi, T. Heywood, A. Kramer, H. A. Leopold, C. Choi, S. Daftry, O. Toupet, I. Wee, A. Thakur, M. Feras, G. Beltrame, G. Nikolakopoulos, D. Shim, L. Carlone, and J. Burdick, "NeBula: Quest for Robotic Autonomy in Challenging Environments; TEAM CoSTAR at the DARPA Subterranean Challenge," Oct. 2021, arXiv:2103.11470.

[7] N. Kottege, J. Williams, B. Tidd, F. Talbot, R. Steindl, M. Cox, D. Frousheger, T. Hines, A. Pitt, B. Tam, B. Wood, L. Hanson, K. L. Surdo, T. Molnar, M. Wildie, K. Stepanas, G. Catt, L. Tychsen-Smith, D. Penfold, L. Overs, M. Ramezani, K. Khosoussi, F. Kendoul, G. Wagner, D. Palmer, J. Manderson, C. Medek, M. O'Brien, S. Chen, and R. C. Arkin, "Heterogeneous robot teams with unified perception and autonomy: How Team CSIRO Data61 tied for the top score at the DARPA Subterranean Challenge," Feb. 2023, arXiv:2302.13230 [cs]. [Online]. Available: http://arxiv.org/abs/2302.13230

[8] Z. Yoon, L. Y. Zhu, L. Gan, and Y. Zhao, "STATE-NAV: Stability-Aware Traversability Estimation for Bipedal Navigation on Rough Terrain," Jun. 2025, arXiv:2506.01046 [cs]. [Online]. Available: http://arxiv.org/abs/2506.01046

[9] I. D. Miller, F. Cladera, A. Cowley, S. S. Shivakumar, E. S. Lee, L. Jarin-Lipschitz, A. Bhat, N. Rodrigues, A. Zhou, A. Cohen, A. Kulkarni, J. Laney, C. J. Taylor, and V. Kumar, "Mine Tunnel Exploration using Multiple Quadrupedal Robots," Feb. 2020, arXiv:1909.09662 [cs]. [Online]. Available: http://arxiv.org/abs/1909.09662

[10] A. Dixit, D. D. Fan, K. Otsu, S. Dey, A.-A. Agha-Mohammadi, and J. W. Burdick, "STEP: Stochastic Traversability Evaluation and Planning for Risk-Aware Navigation; Results from the DARPA Subterranean Challenge," *IEEE Transactions on Field Robotics*, pp. 1–1, 2024, conference Name: IEEE Transactions on Field Robotics. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10779483

[11] K. Stepanas, J. Williams, E. Hernández, F. Ruetz, and T. Hines, "OHM: GPU Based Occupancy Map Generation," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 078–11 085, Oct. 2022.

[12] H. Biggie, E. R. Rush, D. G. Riley, S. Ahmad, M. T. Ohradzansky, K. Harlow, M. J. Miles, D. Torres, S. McGuire, E. W. Frew, C. Heckman, and J. S. Humbert, "Flexible Supervised Autonomy for Exploration in Subterranean Environments," *Field Robotics*, vol. 3, no. 1, pp. 125–189, Jan. 2023, arXiv:2301.00771 [cs]. [Online]. Available: http://arxiv.org/abs/2301.00771

[13] 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)*, Sep. 2021.

[14] T. Miki, L. Wellhausen, R. Grandia, F. Jenelten, T. Homberger, and M. Hutter, "Elevation Mapping for Locomotion and Navigation using GPU," Apr. 2022.

[15] L. Wellhausen, A. Dosovitskiy, R. Ranftl, K. Walas, C. Cadena, and M. Hutter, "Where Should I Walk? Predicting Terrain Properties From Images Via Self-Supervised Learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1509–1516, Apr. 2019.

[16] P. Karkowski and M. Bennewitz, "Prediction Maps for Real-Time 3D Footstep Planning in Dynamic Environments," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 2517–2523.

[17] D. Kim, D. Carballo, J. Di Carlo, B. Katz, G. Bledt, B. Lim, and S. Kim, "Vision Aided Dynamic Exploration of Unstructured Terrain with a Small-Scale Quadruped Robot," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 2464–2470.

[18] R. J. Griffin, G. Wiedebach, S. McCrory, S. Bertrand, I. Lee, and J. Pratt, "Footstep Planning for Autonomous Walking Over Rough Terrain," in *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, Oct. 2019, pp. 9–16.

[19] S. Bertrand, I. Lee, B. Mishra, D. Calvert, J. Pratt, and R. Griffin, "Detecting Usable Planar Regions for Legged Robot Locomotion," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA: IEEE, Oct. 2020, pp. 4736–4742.

[20] R. Buchanan, L. Wellhausen, M. Bjelonic, T. Bandyopadhyay, N. Kottege, and M. Hutter, "Perceptive whole-body planning for multilegged robots in confined spaces," *Journal of Field Robotics*, vol. 38, no. 1, pp. 68–84, 2021.

[21] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter, "Perceptive Locomotion through Nonlinear Model Predictive Control," in *arXiv*, Aug. 2022.

[22] T. Miki, J. Lee, L. Wellhausen, and M. Hutter, "Learning to walk in confined spaces using 3D representation," Feb. 2024, arXiv:2403.00187 [cs]. [Online]. Available: http://arxiv.org/abs/2403.00187

[23] S. Thrun, "Learning Occupancy Grid Maps with Forward Sensor Models," *Autonomous Robots*, vol. 15, no. 2, pp. 111–127, Sep. 2003. [Online]. Available: https://doi.org/10.1023/A:1025584807625

[24] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, "ROBOT-CENTRIC ELEVATION MAPPING WITH UNCERTAINTY ESTIMATES," in *Mobile Service Robotics*. Poznan, Poland: WORLD SCIENTIFIC, Aug. 2014, pp. 433–440. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/9789814623353_0051

[25] P. Fankhauser, M. Bloesch, and M. Hutter, "Probabilistic Terrain Mapping for Mobile Robots With Uncertain Localization," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3019–3026, Oct. 2018.

[26] S. Behnke, M. Schwarz, T. Rodehutskors, D. Droeschel, M. Schreiber, A. Topelidou-Kyniazopoulou, D. Schwarz, C. Lenz, S. Schuller, J. Razlaw, I. Ivanov, N. Araslanov, and M. Beul, "Team NimbRo Rescue at DARPA Robotics Challenge Finals," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. Seoul, South Korea: IEEE, Nov. 2015, pp. 554–554. [Online]. Available: http://ieeexplore.ieee.org/document/7363587/

[27] P. D. Neuhaus, J. E. Pratt, and M. J. Johnson, "Comprehensive summary of the Institute for Human and Machine Cognition's experience with LittleDog," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 216–235, Feb. 2011, publisher: SAGE Publications Ltd STM. [Online]. Available: https://doi.org/10.1177/0278364910390538

[28] J. Pippine, D. Hackett, and A. Watson, "An overview of the Defense Advanced Research Projects Agency's Learning Locomotion program," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 141–144, Feb. 2011.

[29] "ANYbotics/elevation_mapping," Aug. 2024, original-date: 2014-05-08T13:26:47Z. [Online]. Available: https://github.com/ANYbotics/elevation_mapping

[30] M. Zucker, N. Ratliff, M. Stolle, J. Chestnutt, J. A. Bagnell, C. G. Atkeson, and J. Kuffner, "Optimization and learning for rough terrain legged locomotion," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 175–191, Feb. 2011. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364910392608

[31] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant Hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, Aug. 2013, publisher: SAGE Publications Ltd STM. [Online]. Available: https://doi.org/10.1177/0278364913488805

[32] D. Hoeller, N. Rudin, C. Choy, A. Anandkumar, and M. Hutter, "Neural Scene Representation for Locomotion on Structured Terrain," Jun. 2022.

[33] 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)*, Jun. 2021.

[34] B. Acosta and M. Posa, "Bipedal Walking on Constrained Footholds with MPC Footstep Control."

[35] H. Oleynikova, Z. Taylor, M. Fehr, J. Nieto, and R. Siegwart, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 1366–1373, arXiv:1611.03631 [cs]. [Online]. Available: http://arxiv.org/abs/1611.03631

[36] A.-a. Agha-mohammadi, E. Heiden, K. Hausman, and G. S. Sukhatme, "Confidence-rich grid mapping," *The International Journal of Robotics Research*, vol. 38, no. 12-13, pp. 1352–1374, Oct. 2019, arXiv:2006.15754 [cs]. [Online]. Available: http://arxiv.org/abs/2006.15754

[37] J. Frey, D. Hoeller, S. Khattak, and M. Hutter, "Locomotion Policy Guided Traversability Learning using Volumetric Representations of Complex Environments," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2022, pp. 5722–5729, arXiv:2203.15854 [cs]. [Online]. Available: http://arxiv.org/abs/2203.15854

[38] T. Hines, K. Stepanas, F. Talbot, I. Sa, J. Lewis, E. Hernandez, N. Kottege, and N. Hudson, "Virtual Surfaces and Attitude Aware Planning and Behaviours for Negative Obstacle Navigation," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 4048–4055, Apr. 2021, conference Name: IEEE Robotics and Automation Letters. [Online]. Available: https://ieeexplore.ieee.org/document/9376244

[39] M. Tranzatto, F. Mascarich, L. Bernreiter, C. Godinho, M. Camurri, S. Khattak, T. Dang, V. Reijgwart, J. Loeje, D. Wisth, S. Zimmermann, H. Nguyen, M. Fehr, L. Solanka, R. Buchanan, M. Bjelonic, N. Khedekar, M. Valceschini, F. Jenelten, M. Dharmadhikari, T. Homberger, P. De Petris, L. Wellhausen, M. Kulkarni, T. Miki, S. Hirsch, M. Montenegro, C. Papachristos, F. Tresoldi, J. Carius, G. Valsecchi, J. Lee, K. Meyer, X. Wu, J. Nieto, A. Smith, M. Hutter, R. Siegwart, M. Mueller, M. Fallon, and K. Alexis, "CERBERUS: Autonomous Legged and Aerial Robotic Exploration in the Tunnel and Urban Circuits of the DARPA Subterranean Challenge," Jan. 2022, arXiv:2201.07067 [cs]. [Online]. Available: http://arxiv.org/abs/2201.07067

[40] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, Apr. 2013. [Online]. Available: http://link.springer.com/10.1007/s10514-012-9321-0

[41] J. S. Smith and P. Vela, "PiPS: Planning in perception space," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 6204–6209.

[42] R. Xu, S. Feng, and P. A. Vela, "Potential Gap: A Gap-Informed Reactive Policy for Safe Hierarchical Navigation," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 8325–8332, 2021.

[43] J. S. Smith, S. Feng, F. Lyu, and P. A. Vela, "Real-Time Egocentric Navigation Using 3D Sensing," in *Machine Vision and Navigation*, O. Sergiyenko, W. Flores-Fuentes, and P. Mercorelli, Eds. Cham: Springer International Publishing, 2020, pp. 431–484. [Online]. Available: https://doi.org/10.1007/978-3-030-22587-2_14

[44] J. S. Smith, R. Xu, and P. Vela, "egoTEB: Egocentric, Perception Space Navigation Using Timed-Elastic-Bands," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 2703–2709.

[45] D. Driess, O. Oguz, J.-S. Ha, and M. Toussaint, "Deep Visual Heuristics: Learning Feasibility of Mixed-Integer Programs for Manipulation Planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 9563–9569.

[46] S. Feng, Z. Zhou, J. Smith, M. Asselmeier, Y. Zhao, and P. A. Vela, "GPF-BG: A hierarchical vision-based planning framework for safe quadrupedal navigation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[47] M. Sorokin, J. Tan, C. K. Liu, and S. Ha, "Learning to Navigate Sidewalks in Outdoor Environments," Sep. 2021.

[48] A. Agarwal, A. Kumar, J. Malik, and D. Pathak, "Legged Locomotion in Challenging Terrains using Egocentric Vision," Sep. 2022.

[49] L. Matthies, R. Brockers, Y. Kuwata, and S. Weiss, "Stereo vision-based obstacle avoidance for micro air vehicles using disparity space," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 3242–3249, iSSN: 1050-4729. [Online]. Available: https://ieeexplore.ieee.org/document/6907325

[50] J. S. Smith and P. Vela, "AeriaLPiPS: A Local Planner for Aerial Vehicles with Geometric Collision Checking," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, May 2023, pp. 4092–4098. [Online]. Available: https://ieeexplore.ieee.org/document/10160852/?arnumber=10160852

[51] Y. Nakagawa, H. Uchiyama, H. Nagahara, and R.-I. Taniguchi, "Estimating Surface Normals with Depth Image Gradients for Fast and Accurate Registration," in *2015 International Conference on 3D Vision*, Oct. 2015, pp. 640–647. [Online]. Available: https://ieeexplore.ieee.org/document/7335535/?arnumber=7335535

[52] M. Nieuwenhuisen, J. Stueckler, A. Berner, R. Klein, and S. Behnke, "Shape-Primitive Based Object Recognition and Grasping," in *ROBOTIK 2012; 7th German Conference on Robotics*, May 2012, pp. 1–5.

[53] Y. Lin, C. Tang, F.-J. Chu, and P. A. Vela, "Using synthetic data and deep networks to recognize primitive shapes for object grasping," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 10 494–10 501.

[54] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," Aug. 2018.

[55] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick, "Detectron2," 2019.

[56] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, Nov. 2012, conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence. [Online]. Available: https://ieeexplore.ieee.org/document/6205760/?arnumber=6205760

[57] P. Neubert and P. Protzel, "Compact Watershed and Preemptive SLIC: On Improving Trade-offs of Superpixel Segmentation Algorithms," in *2014 22nd International Conference on Pattern Recognition*, Aug. 2014, pp. 996–1001, iSSN: 1051-4651. [Online]. Available: https://ieeexplore.ieee.org/document/6976891/?arnumber=6976891

[58] R. Graham, "An efficient algorith for determining the convex hull of a finite planar set," *Information Processing Letters*, vol. 1, no. 4, pp. 132–133, Jun. 1972. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0020019072900452

[59] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968, publisher: Institute of Electrical and Electronics Engineers (IEEE). [Online]. Available: https://doi.org/10.1109/tssc.1968.300136

[60] Z. Kingston and L. E. Kavraki, "Scaling Multimodal Planning: Using Experience and Informing Discrete Search," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 128–146, Feb. 2023.

[61] M. Asselmeier, J. Ivanova, Z. Zhou, P. A. Vela, and Y. Zhao, "Hierarchical Experience-informed Navigation for Multi-modal Quadrupedal Rebar Grid Traversal," *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[62] 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)*, May 2021, pp. 9805–9811.

[63] D. E. Orin, A. Goswami, and S.-H. Lee, "Centroidal dynamics of a humanoid robot," *Autonomous Robots*, vol. 35, no. 2, pp. 161–176, Oct. 2013.

[64] "OCS2: An open source library for Optimal Control of Switched Systems." [Online]. Available: https://github.com/leggedrobotics/ocs2

[65] 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 *IEEE International Symposium on System Integrations (SII)*, 2019.

[66] C. Dario 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)*, Nov. 2016, pp. 558–564, iSSN: 2164-0580. [Online]. Available: https://ieeexplore.ieee.org/document/7803330/

[67] A. Grunnet-Jepsen and D. Tong, "Depth Post-Processing for Intel® RealSense™ D400 Depth Cameras."

[68] Qiayuan Liao and others, "{legged_control}: NMPC, WBC, state estimation, and sim2real framework for legged robots based on OCS2 and ros-controls," [Online]. Available: \url{https://github.com/qiayuanl/legged_control}.

[69] R. Hartley, M. Ghaffari, R. M. Eustice, and J. W. Grizzle, "Contact-aided invariant extended Kalman filtering for robot state estimation," *The*

*International Journal of Robotics Research*, vol. 39, no. 4, pp. 402–430, 2020.

[70] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, May 2022, publisher: American Association for the Advancement of Science. [Online]. Available: https://www.science.org/doi/10.1126/scirobotics.abm6074

[71] G. Pardo-Castellote, "OMG Data-Distribution Service: architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, May 2003, pp. 200–206. [Online]. Available: https://ieeexplore.ieee.org/document/1203555

[72] lnotspotl, "lnotspotl/tbai_ros," Nov. 2025, original-date: 2024-04-04T13:17:18Z. [Online]. Available: https://github.com/lnotspotl/tbai_ros

[73] 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*, vol. 7, no. 62, p. eabk2822, Jan. 2022, arXiv:2201.08117 [cs]. [Online]. Available: http://arxiv.org/abs/2201.08117

[74] F. Jenelten, J. He, F. Farshidian, and M. Hutter, "DTC: Deep Tracking Control," *Science Robotics*, vol. 9, no. 86, p. eadh5401, Jan. 2024, arXiv:2309.15462 [cs, eess]. [Online]. Available: http://arxiv.org/abs/2309.15462

[75] F. Jenelten, R. Grandia, F. Farshidian, and M. Hutter, "TAMOLS: Terrain-Aware Motion Optimization for Legged Systems," Jul. 2022, arXiv:2206.14049. [Online]. Available: http://arxiv.org/abs/2206.14049