# Coupling Vision and Proprioception for Navigation of Legged Robots

Zipeng Fu[*1] Ashish Kumar[*2] Ananye Agarwal[1] Haozhi Qi[2] Jitendra Malik[2] Deepak Pathak[1]
[1]Carnegie Mellon University [2]UC Berkeley

## Abstract

*We exploit the complementary strengths of vision and proprioception to achieve point goal navigation in a legged robot. Legged systems are capable of traversing more complex terrain than wheeled robots, but to fully exploit this capability, we need the high-level path planner in the navigation system to be aware of the walking capabilities of the low-level locomotion policy on varying terrains. We achieve this by using proprioceptive feedback to estimate the safe operating limits of the walking policy, and to sense unexpected obstacles and terrain properties like smoothness or softness of the ground that may be missed by vision. The navigation system uses onboard cameras to generate an occupancy map and a corresponding cost map to reach the goal. The FMM (Fast Marching Method) planner then generates a target path. The velocity command generator takes this as input to generate the desired velocity for the locomotion policy using as input additional constraints, from the safety advisor, of unexpected obstacles and terrain determined speed limits. We show superior performance compared to wheeled robot (LoCoBot) baselines, and other baselines which have disjoint high-level planning and low-level control. We also show the real-world deployment of our system on a quadruped robot with onboard sensors and compute. Videos at* https://navigation-locomotion.github.io/camera-ready

## 1. Introduction

Gibson has famously remarked, "we see in order to move and we move in order to see." Although, it would be more accurate to say that we *see* and *feel* in order to move. Vision and proprioception are complementary senses. Vision is a distance sense, it allows us to avoid static and dynamic obstacles. However, vision is slow and cannot directly sense physical properties of terrains such as softness vs hardness, smooth vs rough. Proprioception (knowledge of agent's own body like joint angles, foot contacts, etc) is fast and gives a direct measurement of physical environment characteristics. In this paper, we will focus on exploiting the complementary strengths of vision and proprioception for navigation of legged robots. The goal is to train a legged robot by developing both low-level control of its motor joints to walk on terrains (i.e., locomotion) as well as high-level path planning to reach some goal location by autonomously avoiding any obstacles along the way (i.e., navigation).

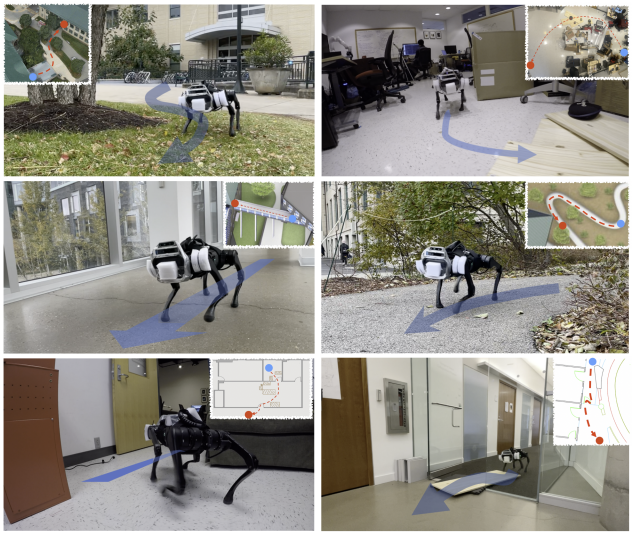


Figure 1. Example deployment scenarios for our proposed point goal navigation system for legged robots. The varying terrains on the way to the goal require the planner to be aware of the robots locomotion capabilities. The proprioceptive coupling between the locomotion controller and the navigation planner allow the robot to sense properties of the environment which the vision might miss (slippery terrain, glass obstacle, etc).

**Locomotion and Navigation** Traditionally, locomotion and navigation are studied as separate problems and then put together on a robot as individual modules [27, 48, 78, 82]. However, to truly support dynamic goal reaching in complex terrains, the planner should know about the walking ability of the robot in different terrains. For instance, a robot navigating to a goal through a slippery patch may either lower its walking speed or walk around it altogether depending on its locomotion ability. To facilitate such communication between high-level and low-level, prior works generally infer a cost map for the planner from an onboard vision

*equal contribution

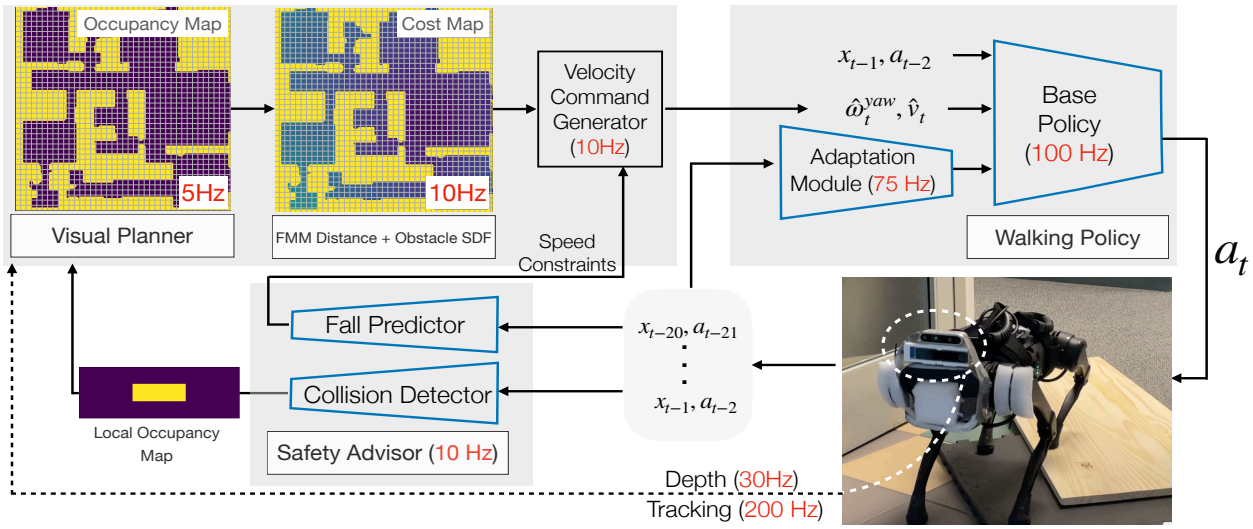



Figure 2. Our navigation system consists of a velocity-conditioned walking policy, a safety advisor module, and a planning module. The velocity-conditioned walking policy takes as input the command velocity and the proprioceptive robot state to output the actions needed to walk in a variety of complex settings. Once we have learned the walking policy in simulation, we then train a Safety Advisor Module, also in simulation, which estimates the safety constraints of the walking policy. It uses proprioception to estimate two bits of information (1) if the robot is in collision, (2) if the robot about to fall, and uses this to update the map and velocity estimates to walk safely in its environment. The planner uses on board cameras to compute a navigation cost map for an input point goal and takes in the safety constraints from the Safety Advisor module to compute desired walking velocity and direction. All modules run asynchronously on the onboard compute of the robot.

sensor which is only capable of detecting clearly visible obstacles and regions that are hard to traverse, e.g. steps and ramps [14, 80, 83, 86]. However, it is extremely challenging to predict several other terrain properties from vision like how slippery, uneven, granular or deformable is the surface. These directly affect the walking robot's ability to follow the plan. Furthermore, the environment could also contain obstacles that are invisible to a vision-only planner as shown in Figure 1 and Figure 5, e.g., glass walls or uneven bumps on ground — things which a robot can readily *feel* as it walks through them.

**Proprioceptive Feedback** Our insight is to leverage this robot's on-ground *feeling* as observed via *proprioception* to bridge the gap and continually update the high-level navigation plan in accordance with low-level locomotion. Furthermore, this coupling of locomotion with navigation improves locomotion efficiency as well. For instance, a planner aware of locomotion ability can direct the robot to switch low-level gaits (walking $\rightarrow$ trotting $\rightarrow$ galloping) for increasing its speed whenever the path is straight and switch other way round to decrease speed on winding paths. We posit that the adaptation of navigational plan from vision and proprioception must occur online in real time. But, how?

**Coupled Vision and Proprioception** We show a high-level illustration of our overall algorithm in Figure 2. It consists of three subsystems: a velocity-conditioned walking policy, a safety advisor module, and the planning module which together make synergistic use of vision and proprioception for navigation of legged robots. At the lowest level, our velocity-conditioned locomotion controller is trained via reinforcement learning to allow the robot to walk at different speeds and in different directions. It takes the commanded linear and angular velocity as input along with the robot's proprioception state to predict the target joint angles directly without using any hand-engineered control primitives. We train this base controller in simulation via energy-based reward to allow for seamless gait switching at different speeds [19, 87] and then transfer to the real world via rapid motor adaptation [45] that estimates environment extrinsics using an adaptation module trained in simulation. Once we have learned the walking policy, which includes the base policy and the adaptation module, in simulation, we freeze it and train a Safety Advisor (SA) Module, also in simulation, which learns to estimate the safety constraints of the walking policy. It uses proprioception to estimate(1) if the robot is in collision to a visually undetected object such as glass walls (2) what is a safe velocity limit for the robot to walk in the current terrain which could be soft, slippery, bumpy, etc. During deployment, the walking policy (base policy and adaptation module) and the SA (safety advisor) module are kept frozen and interact with the planner as shown in Figure 2. The planner uses on board cameras to compute a navigation cost map for an input to the point goal and takes in the two bits of safety constraints from the safety advisor module to compute the target linear and angular velocity which is given to the walking policy to track. This process ensures that both linear and angular com-

manded velocities are within the feasible range of walking policy. The planning module continually updates the cost map and safety constraints to generate the target velocity for the walking velocity as the robot moves. All the modules run asynchronously on the onboard compute of the robot.

**Simulation and Real-World Evaluation** We evaluate our system in challenging navigation settings (e.g., Figure 1) with difficult terrains, invisible glass obstacles, slippery surfaces, deformable ground and challenging outdoor scenarios. Please see videos in supplementary as well as website [1].

In addition, we conduct a series of experiments in simulation. For this we import real-world Matterport 3D [7] maps used in Habitat [66] and Gibson [84] into RaiSim to create a simulation benchmark for controlled study of joint navigation and legged locomotion. We find that the proposed system is 7% - 15% better than baselines with disjoint planning and control loop in different terrains and in settings with invisible obstacles. We find that minimizing time to goal can lead to more energy consuming behaviours which can be compensated for by the use of efficient locomotion policy with emergent gaits. We also additionally show the importance of legged systems over wheeled counterparts in traversing challenging terrains, and empirically demonstrate that continuous velocity-conditioned policy is more time efficient than its discrete counterpart.

## 2. Velocity-Conditioned Walking Policy

Our velocity-conditioned walking policy is an implementation of the approach in [19, 45]. We present a review here to make this paper self-contained. The walking policy contains a base policy which takes the command velocity and the robot state as input and predicts the target joint angles. It additionally takes the extrinsics vector as input which is estimated by the adaptation module and enables rapid online adaptation to varying environment conditions [45].

**Base Policy:** We first train a base policy to walk in simulation on varying terrains and track a commanded linear and angular velocity. The base policy $\pi$ takes the current propriopceptive state $x_t \in \mathbb{R}^{30}$, command velocities $[v^{\text{cmd}}, \omega^{\text{cmd}}] \in \mathbb{R}^2$, previous action $a_{t-1} \in \mathbb{R}^{12}$ and the extrinsics vector $z_t \in \mathbb{R}^8$ to predict the target joint positions $a_t$, which are converted to torques by a PD controller. The extrinsics vector $z_t$ is an encoding of the environment conditions (like payload, friction, etc) which enables the base policy to adapt to different environment conditions instead of being blind to it. The extrinsics vector $z_t$ is generated by an environment encoder $\mu$ from privileged environment information $e_t \in \mathbb{R}^{19}$, as follows: $z_t = \mu(e_t)$ and $a_t = \pi(x_t, a_{t-1}, z_t)$.

We jointly train both $\pi$ and $\mu$ end-to-end with model-free reinforcement learning to maximize discounted expected return $J(\pi) = \mathbb{E}_{\tau \sim p(\cdot|\pi)} \left[ \sum_{t=0}^{T-1} \gamma^t r_t \right]$, where $\tau = \{(x_0, a_0, r_0), ..., (x_{T-1}, a_{T-1}, r_{T-1})\}$ is a sampled trajectory of the robot when executing policy $\pi$ in the simulation, and $p(\tau|\pi)$ represents the likelihood of the trajectory under $\pi$. We use PPO [68] to maximize this objective.

**RL Reward:** Reward encourages the policy to accurately track a commanded linear and angular velocity while penalizing a higher energy consumption [19]. Let's denote the linear velocity as $v$, the orientation as $\theta$ and the angular velocity as $\omega$, all in the robot's base frame. We additionally define the joint angles as $\boldsymbol{q}$, joint velocities as $\dot{\boldsymbol{q}}$, and joint torques as $\boldsymbol{\tau}$. The reward at time $r_t$ is defined as the sum of the following quantities (see supplementary for specifics):

- Velocity Matching: $-|v_x - v^{\text{cmd}}| - |\omega_{\text{yaw}} - \omega^{\text{cmd}}|$
- Energy Consumption: $-\boldsymbol{\tau}^T \dot{\boldsymbol{q}}$
- Lateral Movement: $-|v_y|^2$
- Hip Joints: $-\|\boldsymbol{q}_{\text{hip}}\|^2$

**Training Scheme:** Similar to [45], we train our agent on fractal terrains without any additional artificial rewards for foot clearance or external pushes. For target velocities, we sample from one of the two settings: jointly track linear and angular velocity (curve following), or turning in place. Turning in place is important to handle very cluttered environments. See supplementary for range details.

**Adaptation Module:** Since we don't have the privileged environment information during deployment, we use RMA [45] to train an adaptation module $\phi$ in simulation itself to estimate the extrinsics $z_t$ from proprioceptive state, which is available during deployment. Concretely, the adaptation module uses the recent history of robot's states $x_{t-k:t-1}$ and actions $a_{t-k:t-1}$ to generate $\hat{z}_t$ which is an estimate of the true extrinsics vector $z_t$. This is trained via supervised learning because we have access to both proprioceptive history and the true extrinsics vector in simulation.

## 3. Safety Advisor Module

The safety advisor module captures the constraints which enable the robot to walk safely. For this, we train the two safety advisors in simulation: (1) to detect a collision ($M_c$) and (2) to predict a future fall ($M_f$), both from proprioceptive input which includes the recent history of robot's states ($x_{t-k:t-1}$) and actions ($a_{t-k:t-1}$) (analogous to [45] ). During deployment, the safety advisor module uses the output of these two advisors to inform the planner of the safe operating constraints of the walking policy.

**Collision Detector ($M_c$):** The collision detector estimates a binary value of whether the robot is currently in collision, using proprioception ($M_c(x_{t-k:t-1}, a_{t-k:t-1})$). If a collision is detected, the safety advisor module adds a fixed size patch

[1] Videos at https://navigation-locomotion.github.io/camera-ready

of obstacle (9cm x 3cm), where the side with 3cm is in the current direction of robot, to the cost map in front of the current position of the robot to indicate an obstacle which was potentially missed by the vision system (e.g. glass walls).

**Fall Predictor ($M_f$):** The fall predictor makes a binary prediction whether the walking policy is likely to fall within the next 1s using proprioception ($M_f(x_{t-k:t-1}, a_{t-k:t-1})$). If a fall is predicted, the safety advisor module decreases the velocity limit ($v_t^{max}$) by 0.2 m/s, otherwise it increases the velocity limit by 0.05 m/s. The planner uses $v_t^{max}$ to generate the linear velocity command for the walking policy. This enables the planner to slow the robot down in dangerous settings like soft or slippery terrains, heavy payload, etc.

**Module Training:** We train both the safety advisors $M_f$ and $M_c$ in a self-supervised way in simulation itself. We unroll the velocity-conditioned locomotion policy under randomly sampled payload, friction and roughness of the terrain, obstacles and a randomly sampled command, and record the binary labels on (1) if the policy results in a fall in the next 1s (2) robot is currently in collision. We then use this paired data to learn the safety advisors using supervised learning.

# 4. Visual Planner

The visual planner uses the onboard cameras to generate a top down 2D cost map and uses it to plan a path to the goal. It additionally uses the safety constraints estimated by the safety advisor to generate the command velocities which are fed into the walking policy. Concretely, the visual planner consists of (1) a mapping module which generates a top down 2D occupancy map from onboard cameras, (2) cost map generation step using FMM and signed distance field (3) PID based planner to use the cost map and safety constraints from the safety advisor module to generate linear and angular velocity commands for the walking policy.

## 4.1. Visual Occupancy Map

We first generate a top down 2D visual occupancy map by incrementally accumulating point clouds from an onboard Intel RealSense D435 depth camera [35] as the robot moves. The point clouds are transformed into the world reference frame using pose information from an onboard tracking camera (Intel RealSense T265). The transformed point clouds are capped by a maximum height of interest and then dynamically projected into a horizontal 2D frame to form an occupancy map where each grid has a value from 0 to 1 to indicate the probability of being free space. The occupancy map is binarized for the path planning using a threshold of 0.5. We use a open-sourced implementation from Intel RealSense to compute the visual occupancy map [64]. We convert it to a configuration space by modeling the robot size as a square and dilating the occupancy map.

## 4.2. Cost Map Generation

The 2D cost map is a sum of goal distance map (geodesic distance to the goal) and obstacle distance map (to maintain a safety margin from obstacles). Following the direction of steepest descent from any starting point in this cost map gives an obstacle free path to the goal.

**Goal Distance Map:** We use Fast Marching Method (FMM) [69] to compute the geodesic distance to the point goal, $d^{\text{goal}}(x, y)$ for every starting position $(x, y)$.

**Obstacle Distance Map:** We first compute the signed distance (L1 norm) from the closest obstacle for every point ($d^{\text{sdf}}(x, y)$), and then compute the obstacle distance map as $\max(0, \alpha_1 - d^{\text{sdf}}(x, y))$, where $\alpha_1$ is a distance threshold. We only penalize the robot when it is within $\alpha_1$ to an obstacle. This inverse signed distance field serves two purposes: 1) it penalizes the robot for being too close to obstacles; 2) gives a smooth differentiable cost map even at at (otherwise non-differentiable) object boundaries which enables smooth continuous path planning.

**Cost Map:** The final cost map is

$$C(x, y) = d^{\text{goal}}(x, y) + \alpha_2 \max(0, \alpha_1 - d^{\text{sdf}}(x, y)) \quad (1)$$

Here, $\alpha_2$ is a scaling factor to trade off the two costs. During deployment, the safety advisor module asynchronously adds an additional local obstacle to the cost map if the collision detector ($M_c$) predicts a collision.

## 4.3. Velocity Command Generation

Given the robot's current position $(x_t, y_t)$, heading (yaw) $\theta_t$ and cost map $C(x, y)$, the optimal heading direction is the direction of steepest descent in the cost map [69]. We can compute this optimal heading direction or target orientation of the robot $\theta_t^{\text{target}}$ as the normalized negative gradient of the cost map $-\nabla C(x_t, y_t)$.

**Angular Velocity:** We use a PD controller to compute the command angular velocity (3) which is then clipped to the feasible range (specified in supplementary):

$$\omega_t^{\text{cmd}} = K_p \cdot (\theta_t^{\text{target}} - \theta_t) + K_d \cdot (\omega_t^{\text{target}} - \omega_t) \quad (2)$$

**Linear Velocity:** We do a linear search in the cost map starting from the robot current position $(x_t, y_t)$ in the direction of $\theta$ to get a short-term target position $(x_t', y_t')$. The key insight is that the robot should go in its current direction as far as possible as long as the cost keeps decreasing (Figure 3). The target linear velocity $v^{\text{cmd}}$ is $\frac{1}{T}\alpha_0$, where $\alpha_0$ is obtained from the optimization problem in Figure 3a. A larger $T$ will lead to a more conservative target linear velocity, whereas a small $T$ will be more aggressive. The command sent to the robot is an exponentially smoothed average of the target speed. We maintain a separate exponential moving average for speed-up and slow-down.

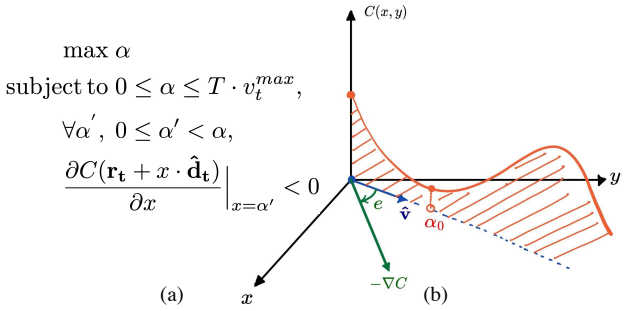



Figure 3. The optimal direction is along the direction of steepest descent in the cost map $-\nabla C$. The angular velocity is computed by PD control on the error $e$ between optimal direction and current direction $\hat{\mathbf{v}}$. The magnitude of linear velocity is determined by finding the furthest point $\alpha_0$ along the current direction such that the cost keeps decreasing. ($\mathbf{r_t}$: robots current position, $\hat{\mathbf{d}}_\mathbf{t}$: unit vector in direction $\theta_t$, $v_t^{max}$: maximum linear walking speed from the fall predictor $M_f$, and $T$: lookahead time)

## 5. Experimental Setup

**Physical Hardware:** We use the A1 robot from Unitree with 18-DoF (12 actuatable). Its proprioception sensors include joint motor encoders, roll and pitch from the IMU sensor and binarized foot contact indicators. We additionally mount Intel RealSense depth D435 and tracking T265 cameras. The deployed policy uses joint position control.

**Locomotion Policy:** For locomotion policy, we use similar architecture and training details as [19, 45], and list the exact policy and training details in the supplementary.

**Safety Advisor Module:** Similar to the adaptation module, both the collision detector and fall predictor module share the same architecture and embed states and actions into a 32-dim vector using a linear layer. Then, we use 3 layers of 1D convolutions with input channels, output channels and strides $[32, 32, 8, 4]$, $[32, 32, 5, 1]$, $[32, 32, 5, 1]$. The flattened features are then passed through a 2-layer MLP with 8 hidden units to get 1 sigmoid output as the predicted probability value. We train the module in an online fashion by rollouts in environments with randomly sampled invisible obstacles, frictions, terrain roughness and payload values (see supplementary for ranges). At simulation test time, we run both the collision detector and fall predictor at 5Hz, whereas for deployment on robot we train a lightweight version using only the last 0.2s of observation history and run it at 10Hz. More details are in the supplementary.

**FMM Planner and PID Controller:** During cost map generation we choose $\alpha_1 = 10, \alpha_2 = 0.5$. For controlling the angular velocity we set gains $K_p = 1, K_d = 0.02$ with $\omega^{\text{target}}$ set to 0. At run time, we clip the linear speed supplied by the line search to the maximum command velocity determined by the fall predictor. To facilitate in-place turning, if the linear speed is less than 0.2, we clip the angular velocity to the range $[0.4, 0.8]$. The planner runs at 10Hz in simulation and robot.

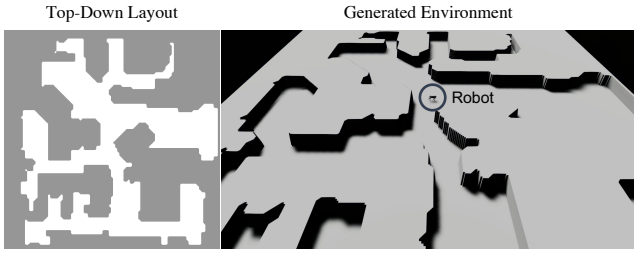



Figure 4. An example of the top-down view room layout and the corresponding generated simulation environment.

**Simulation Environments:** We generate top-down view room layouts from room scanning meshes using habitat-sim [67, 74]. The meshes are from gibson environment [84] and matterport3D [8]. We then select 200 challenging room layouts for navigation as our validation set. For each room layout, we sample 10 navigation goals and set the initial point to be the farthest point from the goal. We then convert the room layout to RaiSim simulation environment [30]. The resolution is 0.1m per pixel. We show an example of the top-down layouts and the generated environment in figure 4.

To demonstrate our navigation system on complex terrains, we construct the following variations:

- Flat: flat surface with coefficient of friction $\mu = 0.8$.
- RoughTerrain: we put eight patches of z-scale 0.05 and size 0.8m×0.8m along the path from initial to the goal position. The rough terrain is constructed using the built-in terrain generator by RaiSim [30].
- 2x/4x/8x Inv-Obstacle: we put 2/4/8 0.2m×0.2m obstacles that cannot be detected by the vision sensor.
- Randomized: we put 8 rough and slippery patches along the path from initial to goal position. And a 8kg payload is placed/removed on top of the robot every 5s.

## 6. Experimental Results

We test our approach both in simulation and in the real world.

### 6.1. Simulation Experiments

In simulation we assume that the agent has access to the ground-truth occupancy map, and we only vary the terrains and the navigation strategy. The purpose of our simulation experiments is to answer the following questions:

- How much does proprioception feedback help?
- Minimizing time to goal requires more aggressive walking and more energy. Can a varying gait policy compensate for some of the energy consumed?

We additionally evaluate the following broader questions:

- Does legged locomotion improve goal reaching?
- Is continuous velocity conditioning better than discrete?

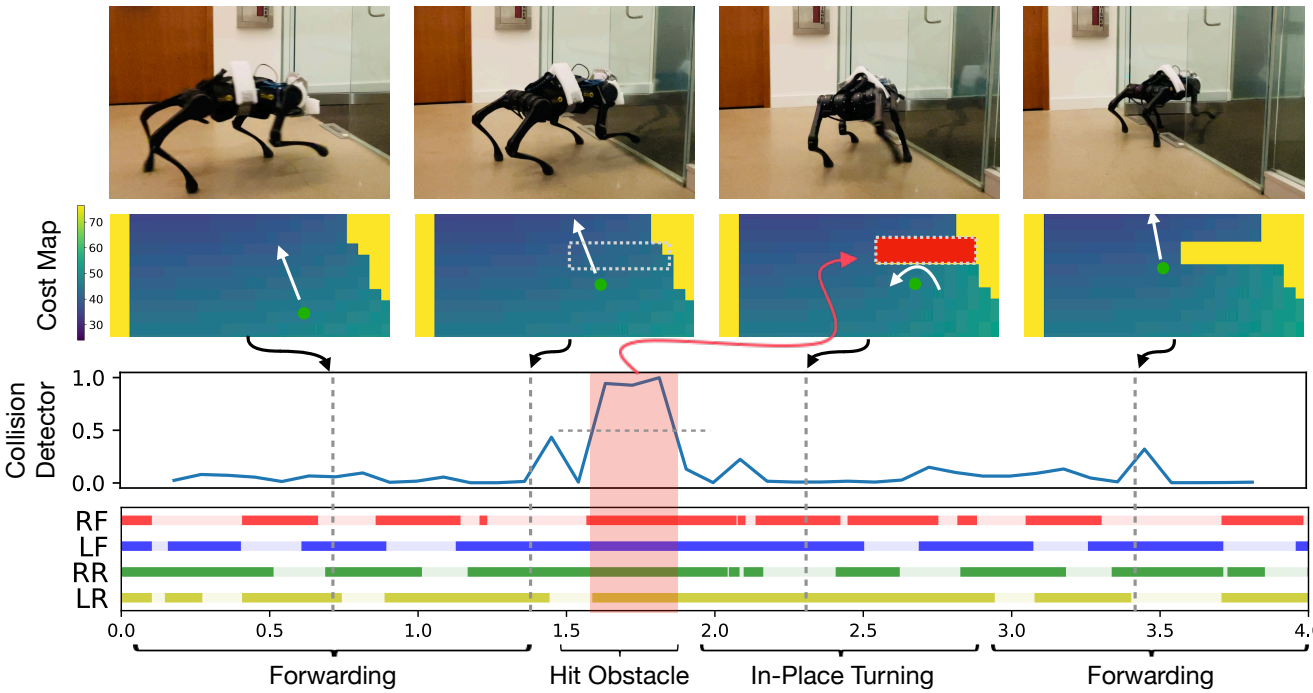



Figure 5. Collision Detector: The top row shows the deployed robot, the second row shows the state of the occupancy map and the bottom two rows show the predictions of the collision detector and the gait plot of the robot. The robot collides with the glass wall which is missed by the onboard cameras, after which, the collision detector detects this from proprioception and indicates a missed obstacle. The map is updated locally to indicate this and the robot replans its path around it. The gait plot shows that the robot is stuck for a fraction of the second before the collision detector senses the glass wall and updates the map. Our method bypasses the glass wall with a 100% (8 out of 8) success rate, whereas a vision only baseline fails to cross it even once.

**Baseline and Metrics:** We use the LoCoBot [1] as our wheeled robot baseline, as it is widely used in visual navigation [4, 9, 10, 23]. We import the PyRobot URDF model [58]. Both our method and the LoCoBot use a control frequency of 100Hz and a planning frequency of 10Hz. We evaluate our method using the following metrics: 1) Success Rate 2) Success weighted by (normalized inverse) Path Length (SPL) [3] 3) Average time used to achieve the goal. If the agent fails to reach the goal, we add a constant timeout penalty (220s) for the failure episodes; 4) Average energy consumption over the successful episodes [19].

**Improvements with Proprioceptive Coupling:** We separately analyze the importance of the two safety advisors (collision detector and fall predictor).

*Collision Detector:* We uniformly place 2/4/8 0.2m×0.2m obstacles along the path from initial and goal positions, and run our method with/without proprioceptive feedback. The obstacles are not marked in the top-down view map to simulate the glass or other objects that an imperfect vision sensor fails to capture. From Table 1 we note that adding invisible obstacles makes the navigation task very challenging as evident from the performance drop of all the methods. Using the proprioceptive collision detector module improves the success rate by 5.7 points over the baseline method which does not use it. The performance improvements are even larger when the environment becomes more challenging with up to 15 points improvement over baseline.

*Fall Predictor:* We show that learned fall predictor enables safe navigation in challenging environments involving a combination of slippery surfaces, rough terrains, and payload changes. We put eight 2.4m×2.4m patches with uneven slippery surfaces along the path from initial and the goal positions. An 8kg payload is placed / removed to the robot every 5 second. Using the proprioceptive fall prediction to adjust the speed of the robot gives 7 points higher goal-reaching success rate over the baseline without proprioception.

**Compensating for higher energy consumption induced by minimizing time to goal:** Minimizing time to goal leads to aggressive locomotion behaviours and increased energy consumption. To compensate for some of the increase in energy consumption, we show that a policy with efficient gaits [19] leads to a 10% lower energy consumption compared to a fixed gait trotting-only policy (See Table 3). Our method also has a slightly higher success rate because it switches to a more stable gait at low speeds when traversing complex settings, as compared to a fixed gait policy. Our method automatically switches gaits to optimize for stability and energy at different speeds.

**Legs vs Wheels:** We also compare our method with LoCoBot on visual navigation in Table 4. The LoCoBot has

| | Navigation System | Terrain Type | Success | SPL | Time (s) |
|---|---|---|---|---|---|
| (a) | w/o Proprio | Flat | 95.20 | 0.79 | 80.28 |
| (b) | w/o Proprio | 2x Inv-Obstacle | 68.45 | 0.57 | 119.80 |
| (c) | Ours | 2x Inv-Obstacle | 74.15 | 0.61 | 111.93 |
| (d) | w/o Proprio | 4x Inv-Obstacle | 45.85 | 0.38 | 152.39 |
| (e) | Ours | 4x Inv-Obstacle | 59.20 | 0.49 | 134.70 |
| (f) | w/o Proprio | 8x Inv-Obstacle | 24.35 | 0.20 | 184.07 |
| (g) | Ours | 8x Inv-Obstacle | 39.25 | 0.32 | 164.95 |

Table 1. **Proprioceptive Feedback help navigation with invisible obstacles.** With proprioceptive feedback, the Success Rate is improved by more than 5 points when two invisible obstacles present. In more challenging environment, the performance improvement is increased to 15 points.

| | Navigation System | Terrain Type | Success | SPL | Time (s) |
|---|---|---|---|---|---|
| (a) | w/o Proprio | Flat | 95.20 | 0.79 | 80.28 |
| (b) | w/o Proprio | Randomized | 80.25 | 0.66 | 105.68 |
| (c) | Ours | Randomized | 87.40 | 0.73 | 117.65 |

Table 2. **Proprioceptive Feedback help navigation with challenging terrains.** Without proprioceptive feedback, the success rate is decreased by 7 points in the presence of a combination of slippery, rough surfaces, and abrupt payload changes, which cannot be inferred from a vision-only system. But with propriception, the planner can readily "feel" the terrain property and the payload changes and plan with a safer velocity.

a slightly lower performance since LoCoBot is more prone to getting stuck in the local minima of the FMM map (see supplementary for details). Adding rough terrain (5cm elevation) to the environment leads to a sudden drop in goal reaching performance of the Locobot, which is expected. We additionally try the planning scheme which plans around rough terrains while assuming ground truth access to their locations. Although the success rate improves, the time cost is still significantly worse than our legged robot baseline, which is able to maintain similar success rate and time to goal because of its robust walking capabilities.

**Continuous Velocity Conditioning vs Discrete:** We compare our continuous planner to a discrete planner typically used in visual navigation [23, 46, 54, 65]. Our discrete planner only commands four actions: 1) forward with 0.6 m/s; 2) turn left with 0.8 rad/s; 3) turn right with 0.8 rad/s; 4) stop, whereas planning over the continuous range of linear and angular velocities enables smoother trajectory and shorter time to goal. In Table 5, we see that our system is 27% more time efficient than a discrete planner as the robot can simultaneously turn and go forward.

### 6.2. Real-World Experiments

**Invisible Obstacles:** We tested the collision detector with invisible obstacles like glass doors, humans that abruptly walk into the robot's path, walls and boxes without textures

| | Navigation System | Terrain Type | Success | SPL | Energy (K) |
|---|---|---|---|---|---|
| (a) | Trot Only | Flat | 93.80 | 0.77 | 252.56 |
| (b) | Ours | Flat | 95.20 | 0.79 | 233.05 |

Table 3. **Energy efficiency**. Our policy with varying gaits consumes less energy compared with the single-gait policy.

| | Navigation System | Terrain Type | Success | SPL | Time (s) |
|---|---|---|---|---|---|
| (a) | LoCoBot-Proceed | Flat | 90.65 | 0.81 | 102.98 |
| (b) | Ours | Flat | 95.20 | 0.79 | 80.28 |
| (c) | LoCoBot-Proceed | RoughTerrain | 15.70 | 0.14 | 215.28 |
| (d) | LoCoBot-Avoid | RoughTerrain | 69.10 | 0.60 | 146.85 |
| (e) | Ours | RoughTerrain | 95.05 | 0.79 | 80.87 |

Table 4. **The importance of legs for goal-reaching.** LoCoBot cannot easily pass rough terrains even when its height is only 5cm. The success rate drops to only 15.7 (LoCoBot-Proceed). Even when the LoCoBot has access to location of rough terrain patches and can plan to avoid it (LoCoBot-Avoid), the success rate is still significantly lower than ours with a higher time cost.

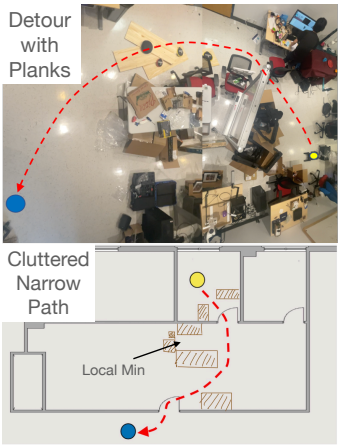
| | Navigation System | Terrain Type | Success | SPL | Time (s) |
|---|---|---|---|---|---|
| (a) | LoCoBot-Dis | Flat | 86.45 | 0.77 | 178.27 |
| (b) | LoCoBot-Cts | Flat | 90.65 | 0.81 | 102.98 |
| (c) | Ours-Dis | Flat | 95.35 | 0.80 | 110.27 |
| (d) | Ours-Cts | Flat | 95.20 | 0.79 | 80.87 |

Table 5. **Comparison between discrete planner (-Dis) and continuous planner (-Cts)**. Using the continuous planner makes the navigation system spend less time to reach the goal.

(Fig 5, 6). We find that feedback from the safety module gives higher success rate in all these settings. The glass wall, which is invisible to the onboard cameras is detected by the proprioceptive feedback once the robot collides with the door. The missed obstacle is then updated in the map at the place of the collision and the robot replans its path around it. Humans abruptly walking into the robot's path are similarly missed by the camera, and then later block the robot's cameras to be detected by them (depth camera's near distance is around 30cm). Such obstacles that suddenly appear from outside into the field-of-view render the trajectory prediction approaches useless [27]. With proprioceptive collision detector, our robot can reason about these "invisible" objects and update its occupancy map to plan a new path.

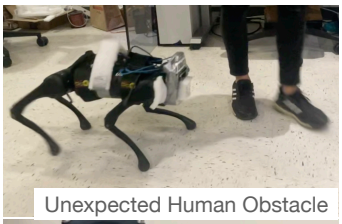
**Rough Slippery Terrains:** We tested the fall detector with challenging terrains including movable planks scattered on the floor and slippery terrain, shown in Figure 6 and in the supplementary. On rough slippery terrain, the fall predictor uses proprioception to estimate the risk of falling and accordingly decreases the velocity to ensure safety.

**Other Complex Indoor Navigation:** We deploy our method in challenging settings and compare to baselines




| | Success Rate | # Obstacles Hit |
|---|---|---|
| Vision + Proprio | 100% | 0.2 |
| Vision | 80% | 0.2 |

| | Success Rate | # Obstacles Hit |
|---|---|---|
| Vision + Proprio | 80% | 0.6 |
| Vision | 60% | 0.6 |




| | Success Rate | Time to Recover After Blocked (s) |
|---|---|---|
| Vision + Proprio | 100% | 0.9 |
| Vision | 0% | $\infty$ |


| | Success Rate | # Slips |
|---|---|---|
| Vision + Proprio | 100% | 3.2 |
| Vision | 80% | 4.0 |

Figure 6. Real-World Experiments: We compare our method with a pure vision approach (without proprioceptive feedback from safety advisor) and evaluate for 5 trials in several challenging settings. Our method gives a higher success rate in all these settings. On the left, we have 2 indoor tasks with planks scattered on the floor and cluttered narrow paths. In both settings, texture-less walls, transparent panels, large brown packaging boxes can be missed by vision. With proprioceptive feedback from safety advisor, we update the occupancy map and replan, despite hitting same number of obstacles. On the right, we tested with a fast human obstruction. With proprioceptive feedback from safety advisor, the robot recovers within a second. Additionally, on challenging terrains as shown on the bottom right, a likely fall detected by the predictor can be used to decrease the safe velocity limit, and improve the stability and success rate.

which use pure vision without fall prediction and collision detection from proprioceptive feedback, and evaluate for 5 trials in all settings (Figure 6). We find that using vision and proprioception for coupled navigation and locomotion gives a higher success rate in all these settings. In the left of Figure 6, we have 2 indoor tasks which require taking a detour with planks scattered on the floor and maneuvers through a cluttered narrow path. In both settings, there are objects that can easily be missed by the vision system, including white walls with no texture, transparent desktop side panels and large brown packaging boxes in dim light. With the proprioceptive safety advisor, our robot can reason about these "invisible" objects and update its occupancy map to replan for a new viable path, despite hitting the same number of obstacles. The robot also slows down on unstable planks that are scattered on the ground.

## 7. Related Work

**Visual Navigation:** Visual navigation is classically done by chaining mapping, localizing, and planning. Once a 2D map is created, optimal path to goal can be found using graph search techniques [26, 43, 47, 73], level-set methods [41] or potential field methods [37, 38] among others. The map is constructed via simultaneous localization and mapping using classical [20, 57, 76] or learned methods [4, 9, 13, 16, 23, 36, 53, 60, 81, 93, 94, 96] assuming access to nearly perfect low-level control. Common benchmark datasets include Habitat [66], Gibson [84] and Matterport3D [8], from which, we import the maps in our Raisim benchmark.

**Navigation in Legged Robots:** Attempts which decouple locomotion and navigation problem work in simple terrains [82]. This decoupled framework has been extended to include learned modules for cluttered environment navigation [27, 48, 78]. [12] describe a coupled navigation and locomotion framework which reasons about foot placement by estimating foothold costs from an elevation map. Foothold scores can be estimated heuristically [14, 18, 32, 40, 52, 80] or learnt [34, 44, 50, 51, 79]. Other methods forgo explicit foothold optimization and learn whether a given section of terrain is traversible [11, 24, 86]. Instead of using just vision, we combine planning module and locomotion policy via vision and proprioception.

**Legged Locomotion:** This has conventionally been accomplished using control theory [2, 5, 6, 22, 28, 31, 33, 39, 55, 63, 72, 88] over handcrafted dynamics models. Recently, RL has been successfully used to learn such policies in simulation [21, 49, 56, 68] and in the real world with sim2real methods [25, 29, 59, 61, 75, 75, 77, 85]. Alternatively, a policy learnt in simulation can be adapted at test-time to work well in real environments [15, 19, 45, 62, 70, 71, 89–92, 95].

## 8. Conclusion and Limitations

The use of a legged robot instead of a wheeled one makes the problem of visual navigation additionally interesting. In this paper, we developed an approach which shows how tightly coupling locomotion and navigation leads to better performance on various dimensions. This is made possible by a tight coupling between proprioception and vision, which is of course common in biological systems. One limitation of our work is that the robot can walk around obstacles but not over them. That is another interesting possibility created by the use of walking robots compared to wheels. We hope the reader is convinced that there are many fun problems associated with the use of vision for walking.

**Acknowledgement** We would like to thank Aravind Sivakumar, Kenny Shaw and Shivam Duggal for their help in recording the real-world experiments. This work is supported by DARPA Machine Common Sense program and in part by Good AI research grant.

## References

[1] Locobot. http://www.locobot.org/. 6

[2] Aaron D Ames, Kevin Galloway, Koushil Sreenath, and Jessy W Grizzle. Rapidly exponentially stabilizing control lyapunov functions and hybrid zero dynamics. *IEEE Transactions on Automatic Control*, 2014. 8

[3] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018. 6

[4] Somil Bansal, Varun Tolani, Saurabh Gupta, Jitendra Malik, and Claire Tomlin. Combining optimal control and learning for visual navigation in novel environments. In *Conference on Robot Learning*. PMLR, 2020. 6, 8

[5] Monica Barragan, Nikolai Flowers, and Aaron M. Johnson. MiniRHex: A small, open-source, fully programmable walking hexapod. In *Robotics: Science and Systems Workshop on "Design and Control of Small Legged Robots"*, 2018. 8

[6] Gerardo Bledt, Matthew J Powell, Benjamin Katz, Jared Di Carlo, Patrick M Wensing, and Sangbae Kim. Mit cheetah 3: Design and control of a robust, dynamic quadruped robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018. 8

[7] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017. 3

[8] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. In *3DV*, 2017. 5, 8

[9] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. *arXiv preprint arXiv:2004.05155*, 2020. 6, 8

[10] Devendra Singh Chaplot, Dhiraj Prakashchand Gandhi, Abhinav Gupta, and Russ R Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. *Advances in Neural Information Processing Systems*, 33, 2020. 6

[11] R Omar Chavez-Garcia, Jérôme Guzzi, Luca M Gambardella, and Alessandro Giusti. Learning ground traversability from simulations. *IEEE Robotics and Automation letters*, 3(3), 2018. 8

[12] Joel Chestnutt. *Navigation planning for legged robots*. Carnegie Mellon University, 2007. 8

[13] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2), 2019. 8

[14] Annett Chilian and Heiko Hirschmüller. Stereo camera based navigation of mobile robots on rough terrain. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4571–4576. IEEE, 2009. 2, 8

[15] Ignasi Clavera, Anusha Nagabandi, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations*, 2019. 8

[16] Samyak Datta, Oleksandr Maksymets, Judy Hoffman, Stefan Lee, Dhruv Batra, and Devi Parikh. Integrating egocentric localization for more realistic point-goal navigation agents. *arXiv preprint arXiv:2009.03231*, 2020. 8

[17] Gregory Dudek and Michael Jenkin. *Computational principles of mobile robotics*. Cambridge university press, 2010. 13

[18] Péter Fankhauser, Marko Bjelonic, C Dario Bellicoso, Takahiro Miki, and Marco Hutter. Robust rough-terrain locomotion with a quadrupedal robot. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5761–5768. IEEE, 2018. 8

[19] Zipeng Fu, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Minimizing energy consumption leads to the emergence of gaits in legged robots. In *CoRL*, 2021. 2, 3, 5, 6, 8

[20] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial intelligence review*, 43(1):55–81, 2015. 8

[21] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*. PMLR, 2018. 8

[22] Hartmut Geyer, Andre Seyfarth, and Reinhard Blickhan. Positive force feedback in bouncing gaits? *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 2003. 8

[23] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 6, 7, 8

[24] Jérôme Guzzi, R Omar Chavez-Garcia, Mirko Nava, Luca Maria Gambardella, and Alessandro Giusti. Path planning with local motion estimations. *IEEE Robotics and Automation Letters*, 5(2), 2020. 8

[25] Josiah Hanna and Peter Stone. Grounded action transformation for robot learning in simulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017. 8

[26] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. 8

[27] David Hoeller, Lorenz Wellhausen, Farbod Farshidian, and Marco Hutter. Learning a state representation and navigation in cluttered and dynamic environments. *IEEE Robotics and Automation Letters*, 6(3):5081–5088, 2021. 1, 7, 8

[28] Marco Hutter, Christian Gehring, Dominic Jud, Andreas Lauber, C Dario Bellicoso, Vassilios Tsounis, Jemin

Hwangbo, Karen Bodie, Peter Fankhauser, Michael Bloesch, et al. Anymal-a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016. 8

[29] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 2019. 8

[30] Jemin Hwangbo, Joonho Lee, and Marco Hutter. Per-contact iteration method for solving contact dynamics. *RA-L*, 2018. 5

[31] Dong Jin Hyun, Jongwoo Lee, SangIn Park, and Sangbae Kim. Implementation of trot-to-gallop transition and subsequent gallop on the mit cheetah i. *The International Journal of Robotics Research*, 2016. 8

[32] Fabian Jenelten, Takahiro Miki, Aravind E Vijayan, Marko Bjelonic, and Marco Hutter. Perceptive locomotion in rough terrain–online foothold optimization. *IEEE Robotics and Automation Letters*, 5(4), 2020. 8

[33] Aaron M Johnson, Thomas Libby, Evan Chang-Siu, Masayoshi Tomizuka, Robert J Full, and Daniel E Koditschek. Tail assisted dynamic self righting. In *Adaptive Mobile Robotics*. World Scientific, 2012. 8

[34] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, and Stefan Schaal. Learning locomotion over rough terrain using terrain templates. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009. 8

[35] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, and Achintya Bhowmik. Intel realsense stereoscopic depth cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017. 4

[36] Arbaaz Khan, Clark Zhang, Nikolay Atanasov, Konstantinos Karydis, Vijay Kumar, and Daniel D Lee. Memory augmented control networks. *arXiv preprint arXiv:1709.05706*, 2017. 8

[37] Oussama Khatib. The potential field approach and operational space formulation in robot control. In *Adaptive and Learning Systems*, pages 367–377. Springer, 1986. 8

[38] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986. 8

[39] Mahdi Khoramshahi, Hamed Jalaly Bidgoly, Soroosh Shafiee, Ali Asaei, Auke Jan Ijspeert, and Majid Nili Ahmadabadi. Piecewise linear spine for speed–energy efficiency trade-off in quadruped robots. *Robotics and Autonomous Systems*, 2013. 8

[40] Donghyun Kim, D Carballo, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, Bryan Lim, and Sangbae Kim. Vision aided dynamic exploration of unstructured terrain with a small-scale quadruped robot. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020. 8

[41] R Kimmel and JA Sethian. Fast marching methods for robotic navigation with constraints. *Center for Pure and Applied Mathematics Report, University of California, Berkeley*, 1996. 8

[42] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015. 12

[43] Sven Koenig and Maxim Likhachev. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3), 2005. 8

[44] J Zico Kolter, Mike P Rodgers, and Andrew Y Ng. A control architecture for quadruped locomotion over rough terrain. In *2008 IEEE International Conference on Robotics and Automation*. IEEE, 2008. 8

[45] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. RMA: Rapid Motor Adaptation for Legged Robots. In *Robotics: Science and Systems*, 2021. 2, 3, 5, 8, 12

[46] Ashish Kumar, Saurabh Gupta, David Fouhey, Sergey Levine, and Jitendra Malik. Visual memory for robust path following. *arXiv preprint arXiv:1812.00940*, 2018. 7

[47] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998. 8

[48] Tianyu Li, Roberto Calandra, Deepak Pathak, Yuandong Tian, Franziska Meier, and Akshara Rai. Planning in learned latent action spaces for generalizable legged locomotion. *IEEE Robotics and Automation Letters*, 6(2), 2021. 1, 8

[49] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016. 8

[50] Octavio Antonio Villarreal Magana, Victor Barasuol, Marco Camurri, Luca Franceschi, Michele Focchi, Massimiliano Pontil, Darwin G Caldwell, and Claudio Semini. Fast and continuous foothold adaptation for dynamic locomotion through cnns. *IEEE Robotics and Automation Letters*, 4(2), 2019. 8

[51] Carlos Mastalli, Michele Focchi, Ioannis Havoutis, Andreea Radulescu, Sylvain Calinon, Jonas Buchli, Darwin G Caldwell, and Claudio Semini. Trajectory and foothold optimization using low-dimensional models for rough terrain locomotion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017. 8

[52] Carlos Mastalli, Ioannis Havoutis, Alexander W Winkler, Darwin G Caldwell, and Claudio Semini. On-line and on-board planning and perception for quadrupedal locomotion. In *2015 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, pages 1–7. IEEE, 2015. 8

[53] Lina Mezghani, Sainbayar Sukhbaatar, Thibaut Lavril, Oleksandr Maksymets, Dhruv Batra, Piotr Bojanowski, and Karteek Alahari. Memory-augmented reinforcement learning for image-goal navigation. *arXiv preprint arXiv:2101.05181*, 2021. 8

[54] Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. Learning to navigate in cities without a map. *Advances in Neural Information Processing Systems*, 31:2419–2430, 2018. 7

[55] Hirofumi Miura and Isao Shimoyama. Dynamic walk of a biped. *The International Journal of Robotics Research*, 1984. 8

[56] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 2016. 8

[57] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002. 8

[58] Adithyavairavan Murali, Tao Chen, Kalyan Vasudev Alwala, Dhiraj Gandhi, Lerrel Pinto, Saurabh Gupta, and Abhinav Gupta. Pyrobot: An open-source robotics framework for research and benchmarking. *arXiv*, 2019. 6, 13

[59] Ofir Nachum, Michael Ahn, Hugo Ponte, Shixiang Shane Gu, and Vikash Kumar. Multi-agent manipulation via locomotion using hierarchical sim2real. In *Conference on Robot Learning*. PMLR, 2020. 8

[60] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017. 8

[61] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018. 8

[62] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. In *Robotics: Science and Systems*, 2020. 8

[63] Marc H Raibert. Hopping in legged systems—modeling and simulation for the two-dimensional one-legged case. *IEEE Transactions on Systems, Man, and Cybernetics*, 1984. 8

[64] Intel RealSense. librealsense. https://github.com/IntelRealSense/librealsense. 4

[65] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. *arXiv preprint arXiv:1803.00653*, 2018. 7

[66] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019. 3, 8

[67] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *ICCV*, 2019. 5

[68] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 3, 8, 12

[69] James A Sethian. Fast marching methods. *SIAM review*, 41(2):199–235, 1999. 4

[70] Laura Smith, J Chase Kew, Xue Bin Peng, Sehoon Ha, Jie Tan, and Sergey Levine. Legged robots that keep on learning: Fine-tuning locomotion policies in the real world. *arXiv preprint arXiv:2110.05457*, 2021. 8

[71] Xingyou Song, Yuxiang Yang, Krzysztof Choromanski, Ken Caluwaerts, Wenbo Gao, Chelsea Finn, and Jie Tan. Rapidly adaptable legged robots via evolutionary meta-learning. In *International Conference on Intelligent Robots and Systems (IROS)*, 2020. 8

[72] Koushil Sreenath, Hae-Won Park, Ioannis Poulakakis, and Jessy W Grizzle. A compliant hybrid zero dynamics controller for stable, efficient and fast bipedal walking on mabel. *The International Journal of Robotics Research*, 2011. 8

[73] Anthony Stentz. Optimal and efficient path planning for partially known environments. In *Intelligent unmanned ground vehicles*. Springer, 1997. 8

[74] Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. In *NeurIPS*, 2021. 5

[75] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics: Science and Systems*, 2018. 8

[76] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002. 8

[77] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017. 8

[78] Joanne Truong, Denis Yarats, Tianyu Li, Franziska Meier, Sonia Chernova, Dhruv Batra, and Akshara Rai. Learning navigation skills for legged robots with learned robot embeddings. *arXiv preprint arXiv:2011.12255*, 2020. 1, 8

[79] Lorenz Wellhausen and Marco 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 2021)*, 2021. 8

[80] Martin Wermelinger, Péter Fankhauser, Remo Diethelm, Philipp Krüsi, Roland Siegwart, and Marco Hutter. Navigation planning for legged robots in challenging terrain. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1184–1189. IEEE, 2016. 2, 8

[81] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. *arXiv preprint arXiv:1911.00357*, 2019. 8

[82] David Wooden, Matthew Malchano, Kevin Blankespoor, Andrew Howardy, Alfred A Rizzi, and Marc Raibert. Autonomous navigation for bigdog. In *2010 IEEE international conference on robotics and automation*. Ieee, 2010. 1, 8

[83] David Wooden, Matthew Malchano, Kevin Blankespoor, Andrew Howardy, Alfred A. Rizzi, and Marc Raibert. Autonomous navigation for bigdog. In *2010 IEEE International Conference on Robotics and Automation*, pages 4736–4741, 2010. 2

[84] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *CVPR*, 2018. 3, 5, 8

[85] Zhaoming Xie, Xingye Da, Michiel van de Panne, Buck Babich, and Animesh Garg. Dynamics randomization re-

visited: A case study for quadrupedal locomotion. *arXiv preprint arXiv:2011.02404*, 2020. 8

[86] Bowen Yang, Lorenz Wellhausen, Takahiro Miki, Ming Liu, and Marco Hutter. Real-time optimal navigation planning using learned motion costs. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9283–9289. IEEE, 2021. 2, 8

[87] Yuxiang Yang, Tingnan Zhang, Erwin Coumans, Jie Tan, and Byron Boots. Fast and efficient locomotion via learned gait transitions. In *5th Annual Conference on Robot Learning (CoRL)*, 2021. 2

[88] KangKang Yin, Kevin Loken, and Michiel Van de Panne. Simbicon: Simple biped locomotion control. *ACM Transactions on Graphics (TOG)*, 2007. 8

[89] Wenhao Yu, Visak C. V. Kumar, Greg Turk, and C. Karen Liu. Sim-to-real transfer for biped locomotion. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2019. 8

[90] Wenhao Yu, C Karen Liu, and Greg Turk. Policy transfer with strategy optimization. In *International Conference on Learning Representations*, 2018. 8

[91] Wenhao Yu, Jie Tan, Yunfei Bai, Erwin Coumans, and Sehoon Ha. Learning fast adaptation with meta strategy optimization. *IEEE Robotics and Automation Letters*, 2020. 8

[92] Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In *Robotics: Science and Systems*, 2017. 8

[93] Jingwei Zhang, Lei Tai, Ming Liu, Joschka Boedecker, and Wolfram Burgard. Neural slam: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*, 2017. 8

[94] Xiaoming Zhao, Harsh Agrawal, Dhruv Batra, and Alexander G Schwing. The surprising effectiveness of visual odometry techniques for embodied pointgoal navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16127–16136, 2021. 8

[95] Wenxuan Zhou, Lerrel Pinto, and Abhinav Gupta. Environment probing interaction policies. In *7th International Conference on Learning Representations, ICLR 2019*, 2019. 8

[96] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017. 8

# A. Locomotion Policy Details

**Base Policy & Env-Factor Encoder Architecture:** We follow the implementation of [45]. The base walking policy is a multi-layer perceptron (MLP) with 3 hidden layers. The input is the current state $x_t \in \mathbb{R}^{30}$, previous action $a_{t-1} \in \mathbb{R}^{12}$ and the extrinsics vector $z_t \in \mathbb{R}^8$ and the output is 12-dim target joint angles. The dimension of hidden layers is 128. The extrinsics vector $z_t$ is estimated by an environment factor encoder. The environment factor encoder is a 3-layer MLP (256, 128 hidden layer sizes) and encodes $e_t \in \mathbb{R}^{17}$ into $z_t \in \mathbb{R}^8$.

**Adaptation Module Architecture:** The adaptation module first embeds states and actions into 32-dim vector using a 2-layer MLP. Then, a 3-layer 1-D CNN convolves the representations across the time dimension to capture temporal correlations in the input. The input channel number, output channel number, kernel size, and stride of each layer are $[32, 32, 8, 4], [32, 32, 5, 1], [32, 32, 5, 1]$. The flattened CNN output is linearly projected to estimate $\hat{z}_t$.

**Learning the Walking Policy:** We jointly train the base policy and the environment encoder network using PPO [68] for $15,000$ iterations (1.2B sample, 24 hours) each of which uses batch size of $80,000$ split into $4$ mini-batches. We then train the adaptation module using supervised learning with on-policy data. We run the optimization process for $1000$ iterations ($80$M samples, 3 hours) and use Adam optimizer [42] to minimize MSE loss. The batch size is $80,000$ split up into $4$ mini-batches.

**Reward Function:** The reward at time $r_t$ is defined as the sum of the following quantities:

- Velocity Matching: $-|v_x - v_x^{\text{cmd}}| - |\omega_{\text{yaw}} - \omega_{\text{yaw}}^{\text{cmd}}|$
- Energy Consumption: $-\boldsymbol{\tau}^T \dot{\boldsymbol{q}}$
- Lateral Movement: $-|v_y|^2$
- Hip Joints: $-\|\boldsymbol{q}_{\text{hip}}\|^2$

The corresponding scalings are 20, 0.075, 1 and 0.2. The survival bonus is set by a simple rule as $10 + 20(v_x^{\text{cmd}} + \omega_{\text{yaw}}^{\text{cmd}})$.

We list the ranges of command linear velocity and angular velocity in Table 6. We re-sample the command velocities within a single episode with probability $0.004$.

# B. Safety Advisor Training Details

**Network Structure:** Similar to the adaptation module, both the collision detector and fall predictor module share the same architecture and embed states and actions into a 32-dim vector using a linear layer. Then, we use 3 layers of 1D convolutions with input channels, output channels and strides $[32, 32, 8, 4], [32, 32, 5, 1], [32, 32, 5, 1]$.

**Obstacle Detector:** The flattened features are then passed through a 2-layer MLP with 8 hidden units and 1 sigmoid

| Task | Command Linear Velocity Range (m / s) | Command Angular Velocity Range (rad / s) |
|---|---|---|
| Curve Following | [0.15, 1.0] | [-0.4, 0.4] |
| In-Place Turning | [0, 0.15] | [-0.6, 0.6] |

Table 6. Command velocity range for curve following and in-place turning.

output unit to get the predicted probability value, indicating whether the robot collides with an obstacle. We train the module in an online fashion by collecting data from robot walking/colliding with the obstacles.

**Fall Predictor:** To train the fall detector, we collect a dataset of observations and train the module in a supervised fashion $\{(x_t^i, x_{t+1}^i, \ldots, x_{t+49}^i), y^i\}$ where $y^i$ is 1 if the robot falls at time $t + 100$ and 0 otherwise (note that one simulation time-step is 0.01s). The module is also trained in an online fashion by rollouts with a random command linear/angular velocity in environments with randomly sampled frictions, terrain roughness and payload values from the following list:

- Coefficient of Friction: $[0.1, 0.6, 1.1, 1.6, 2.1]$.
- Payload: $[1.2, 2.4, 3.6, 4.8, 6.0]$ (kg).
- Rough Terrain z-scale: $[0.01, 0.08, 0.14, 0.23]$.
- Linear Velocity: $[0, 0.5, 1.0]$ (m/s).
- Angular Velocity: $[-0.4, 0.0, 0.4]$ (rad/s).

We train for 145k iterations with a batch size of 1000. At simulation test time, we run both the collision detector and fall predictor at 5Hz whereas for deployment on robot we train a lightweight version using only the last 20 timesteps of observation history and run it at 10Hz.

## C. Visual Planner Details

We command the angular velocity for our robot and the baseline LoCoBot using the following equation:

$$\omega_t^{\text{cmd}} = K_p \cdot (\theta_t^{\text{target}} - \theta_t) + K_d \cdot (\omega_t^{\text{target}} - \omega_t) \quad (3)$$

where $K_p = 1$, $K_d = 0.02$, $\omega^{\text{target}}$ is set to 0. The command angular velocity is clipped to the range in Table 6 before being sent to the locomotion policy in order to be consistent with the training setting. We also observe that when the linear speed is low (less than 0.1m/s), the locomotion policy is unable to make in-place turns with a small commanded angular velocity, due to the imperfection of our locomotion policy. Thus in this case we clip the absolute value of the commanded angular velocity to be at least 0.4 to compensate this imperfection. We empirically observe a higher performance even when the command is sub-optimal, mainly because our planning algorithm operates in a relatively high frequency and can soon correct the angular velocity command as soon as the linear velocity becomes large enough.

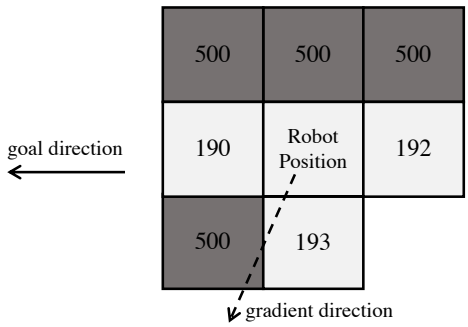



Figure 7. An example of local minima produced by the map. The gray square represents the non-traversible areas. The white square represents the traversible regions. The number in each square represents the cost on that point. At the current position, the robot will orientes to the bottom left which the linear velocity commands will command 0 velocity, in which case the robot got stuck in the local minima.

## D. LoCoBot Baseline Details

We import the PyRobot URDF model [58]. Both our method and the LoCoBot use a control frequency of 100Hz and a planning frequency of 10Hz. We follow [17] to convert commanded linear and angular velocity to the angular speed of the left and right wheel of the LoCoBot. The low level controller is also a PD controller with $K_p = 10$, $K_d = 0.05$. The controller gain is adjusted so that no obvious motion jerk happens during movement. Since the control of wheeled robot is simpler and more accurate, we do observe the LoCoBot being more likely to stuck in local minima in the cost map (an illustration is shown in Figure 7). For our robot, since the locomotion policy is not perfect and the legged robot is harder to control compared with LoCoBot, it sometimes can get out of the local minima due to the noisy movement, which is the reason why we perform better in the perfect flat ground (Table 4 (a) and (b) in the main text). However, we want to emphasize again that our point here is not to show our robot performs slightly better than baseline in the flat ground. Instead, what we show is the ability to traverse and navigate over difficult terrains where LoCoBot easily fail (Table 4 (c), (d), and (e) in the main text).