Coupling Vision and Proprioception for Navigation of Legged Robots

Zipeng Fu*1  Ashish Kumar*2  Ananye Agarwal1  Haozhi Qi2  Jitendra Malik2  Deepak Pathak1
1Carnegie Mellon University  2UC Berkeley

Abstract
We exploit the complementary strengths of vision and proprioception to develop a point-goal navigation system for legged robots, called VP-Nav. Legged systems are capable of traversing more complex terrain than wheeled robots, but to fully utilize this capability, we need a high-level path planner in the navigation system to be aware of the walking capabilities of the low-level locomotion policy in varying environments. We achieve this by using proprioceptive feedback to ensure the safety of the planned path by sensing unexpected obstacles like glass walls, terrain properties like slipperiness or softness of the ground and robot properties like extra payload that are likely missed by vision. The navigation system uses onboard cameras to generate an occupancy map and a corresponding cost map to reach the goal. A fast marching planner then generates a target path. A velocity command generator takes this as input to generate the desired velocity for the walking policy. A safety advisor module adds sensed unexpected obstacles to the occupancy map and environment-determined speed limits to the velocity command generator. We show superior performance compared to wheeled robot baselines, and ablation studies which have disjoint high-level planning and low-level control. We also show the real-world deployment of VP-Nav on a quadruped robot with onboard sensors and computation. Videos at https://navigation-locomotion.github.io

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, which 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, body orientation, 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

*equal contribution
Walking Policy

Safety Advisor (10 Hz)

Advisor Module, also in simulation, which estimates the safety constraints of the walking policy. It uses proprioception to estimate two

These directly affect the walking robot’s ability to follow navigation of legged robots. At the lowest level, our 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 probabilities: (1) if the robot is in collision, (2) if the robot is about to fall, which are used 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 onboard of the robot.

stables and regions that are hard to traverse, e.g. steps and ramps [13, 81, 83, 86]. However, it is extremely challenging to predict several other terrain properties from vision like how slippery, uneven, granular or deformable the surface is. 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 → trotting → 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 system VP-Nav (Vision and Proprioception for Navigation) in Figure 2. It consists of three subsystems: a velocity-conditioned walking policy, a safety advisor module, and the planning module which together make synergestic 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 [17, 88] and then transfer to the real world via rapid motor adaptation [46] 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 planner also ensures that both the linear and angular commanded velocities are within the feasible range of the 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 onboard of the robot.

**Simulation and Real-World Evaluation:** We evaluate our system VP-Nav 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 at [https://navigation-locomotion.github.io](https://navigation-locomotion.github.io)

In addition, we conduct a series of experiments in simulation. For this we import real-world Matterport 3D [7] maps used in Habitat [68] 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 [17,46]. 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 [46].

**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 proprioceptive state (incl. joint angles, joint velocities, body row angle, body pitch angle, and foot contact indicators) $x_t \in \mathbb{R}^{30}$, command velocities $[v_{cmd}, \omega_{cmd}] \in \mathbb{R}^2$, 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 $\tau = \{(x_0, a_0, r_0), \ldots, (x_{T-1}, a_{T-1}, r_{T-1})\}$ 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 [69] 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 [17]. We 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 $q$, joint velocities as $\dot{q}$, and joint torques as $\tau$.

The reward at time $r_t$ is defined as the sum of the following quantities (see supplementary for specifics):

- **Velocity Matching:** $-|v_x - v_{cmd}| - |\omega_{raw} - \omega_{cmd}|$
- **Energy Consumption:** $-\tau^T \dot{\hat{q}}$
- **Lateral Movement:** $-|v_y|^2$
- **Hip Joints:** $-\|q_{hip}\|^2$

**Training Scheme:** Similar to [46], 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 [46] 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 $z_t$ 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 two safety advisors in simulation: (1) Collision Detector $M_c$ to detect collisions and (2) Fall Predictor $M_f$ to predict future falls, both from proprioception which includes the recent history of states $(x_{t-k:t-1})$ and actions $(a_{t-k:t-1})$ (analogous to [46]). During deployment, the safety advisor module uses the prediction 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 the probability 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 probability is above a threshold (0.5), the safety advisor module adds a fixed size patch of obstacle (9cm x 3cm, about the head size of A1), 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 may be missed by the vision system (e.g. glass walls).

**Fall Predictor** ($M_f$): The fall predictor makes a probability prediction of whether the walking policy is likely to fall within the next 1s using proprioception ($M_f(x_{t-k}, a_{t-k})$). If a fall probability is above a threshold (0.5), the safety advisor module decreases the velocity limit ($v_{max}$) by 0.2 m/s, otherwise it increases the velocity limit by 0.05 m/s. The planner uses $v_{max}$ 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 fashion in simulation. We collect data under randomly sampled environments and commands, and record the binary labels on (1) robot is currently in collision (2) if the policy results in a fall in the next 1s. We then train the safety advisors by minimizing binary cross-entropy loss. Details are in the supplementary.

### 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 Fast Marching Method (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 [37] 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 an open-sourced implementation from Intel RealSense to compute the visual occupancy map [66]. 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) [70] to compute the geodesic distance to the point goal, $d_{goal}(x, y)$ for every starting position $(x, y)$.

**Obstacle Distance Map:** We first compute the signed distance (L1 norm) from the closest obstacle for every point $(d_{diff}(x, y))$, and then compute the obstacle distance map as $\max(0, \alpha_1 - d_{diff}(x, y))$, 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 (otherwise non-differentiable) object boundaries which enables smooth continuous path planning.

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

$$C(x, y) = d_{goal}(x, y) + \alpha_2 \max(0, \alpha_1 - d_{diff}(x, y))$$

(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 [70]. We can compute this optimal heading direction or target orientation of the robot $\theta^\text{target}_t$ 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 $\omega_t$ which is then clipped to the feasible range (specified in supplementary):

$$\omega^\text{cmd}_t = K_p \cdot (\theta^\text{target}_t - \theta_t) + K_d \cdot (\omega^\text{target}_t - \omega_t)$$

(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 $v^\text{cmd}_t$ is $\frac{1}{T}v_0$, where $v_0$ 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.
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 [17,46], 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 α_1 = 0.5m, α_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.

Simulation Environments: We generate top-down view room layouts from room scanning meshes using habitat-sim [68,75]. The meshes are from gibson environment [84] and matterport3D [7]. 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 furthest point from the goal. We then convert the room layout to RaiSim simulation environment [31]. 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 0.8\times0.8m along the path from initial to the goal position. The rough terrain is constructed using the built-in terrain generator by RaiSim [31].
- 2x/4x/8x Inv-Obstacle: we put 2/4/8 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. The rough patches are of z-scale 0.05. The coefficient of friction of slippery patches are sampled from \{0.1, 0.3, 0.5, 0.7, 0.9\}. An 8kg payload (A1 itself is 12kg) is placed on / removed from 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?

Baseline and Metrics: We use the LoCoBot [1] as our
wheeled robot baseline, as it is widely used in visual navigation [4, 8, 9, 21]. We import the PyRobot URDF model [60]. Both VP-Nav and the LoCoBot use a control frequency of 100Hz and a planning frequency of 10Hz. We evaluate our system 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 [17].

**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 VP-Nav 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. In 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:** In Table 2, 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 [17] leads to a 10% lower energy consumption compared to a fixed gait trotting-only policy (Table 3). VP-Nav 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. VP-Nav automatically switches gaits to optimize for stability and energy at different speeds.

**Legs vs. Wheels:** We also compare VP-Nav with LoCoBot on visual navigation in Table 4. On flat terrains, LoCoBot has a slightly lower performance since LoCoBot is more prone to getting stuck in the local minima of the FMM map (see...
supplementary for details). Whereas adding rough terrain (5cm elevation) to the environment leads to a significant drop in goal-reaching performance of the LoCoBot. 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. In short, though energy efficient, wheeled robots struggle on uneven terrains, whereas legged robots are more terrain-agnostic.

**Continuous Velocity Conditioning vs. Discrete:** We compare our continuous planner to a discrete planner typically used in visual navigation \([21, 47, 56, 67]\). 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 VP-Nav 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 (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 rushing 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 [28]. 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 prediction approaches 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 [28]. With proprioceptive collision detector, our robot can reason about these “invisible” objects and update its occupancy map to plan a new path.

**Other Complex Indoor Navigation:** We deploy VP-Nav in challenging settings and compare to baselines 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 mainly studied on wheeled robots by chaining mapping, localizing, and planning. Once a 2D map is created, an optimal path to a goal can be found using graph search techniques [24,44,48,74], level-set methods [43] or potential field methods [39,40] among others. The map is constructed via simultaneous localization and mapping using classical [18,59,77] or learned methods [4,8,12,15,21,38,55,62,82,94,95,97] assuming access to nearly perfect low-level control. In our benchmark, we import maps from the common navigation datasets include Habitat [68], Gibson [84] and Matterport3D [7].

Navigation of Legged Robots: Earlier works decouple locomotion and navigation which restricts the application only to simple terrains [83]. This decoupled framework has been extended to include learned modules for cluttered environment navigation [28,50,79]. [11] describe a coupled navigation and locomotion framework by estimating foothold placements from an elevation map. Foothold scores can be estimated heuristically [13,16,34,42,54,81] or learned [36,45,52,53,80]. Other methods forgo explicit foothold optimization and learn traversibility maps [10,22,86]. Several works complement vision-based state estimation by using contact information [25–27,49]. Instead of relying only on vision, we combine navigation and locomotion via coupling vision and proprioception.

Legged Locomotion: This has conventionally been accomplished using control theory [2,5,6,20,29,32,33,35,41,57,65,73,89] over handcrafted dynamics models. Recently, RL has been successfully used to learn such policies in simulation [19,51,58,69] and in the real world with sim2real methods [23,30,61,63,76,76,78,85]. Alternatively, a policy learned in simulation can be adapted at test-time to work well in real environments [14,17,46,64,71,72,87,90–93,96].

8. Conclusion and Limitations

The use of a legged robot instead of a wheeled one broadens the applicability of visual navigation to complex terrains and environments. In this paper, we combine low-level locomotion with high-level navigation planning to enable goal-reaching for a legged quadruped robot. Our approach, VP-Nav, tightly couples vision and proprioception to exploit their complementary strengths for robust navigation in the presence of disturbances, transparent obstacles and complex terrains which may not be detected by vision alone. VP-Nav is lightweight and only uses the modest onboard computation and storage of the low-cost A1 quadruped robot. One limitation of our system is that low-level locomotion module communicates with navigation planner via safety module, and is not conditioned on the vision directly. Due to this, the robot can walk around obstacles but can not climb or jump over them. We leave vision-guided locomotion for future.

Acknowledgement We thank Aravind Sivakumar, Kenny Shaw and Shivam Duggal for help in real-world experiments. This work is supported by DARPA Machine Common Sense program and in part by Good AI research award.
References

[17] 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