From Pixels to Legs: Hierarchical Learning of Quadruped Locomotion

Deepali Jain
Robotics at Google
jaindeepali@google.com

Atil Iscen
Robotics at Google
atil@google.com

Ken Caluwaerts
Robotics at Google
kencaluwaerts@google.com

Abstract: Legged robots navigating crowded scenes and complex terrains in the real world are required to execute dynamic leg movements while processing visual input for obstacle avoidance and path planning. We show that a quadruped robot can acquire both of these skills by means of hierarchical reinforcement learning (HRL). By virtue of their hierarchical structure, our policies learn to implicitly break down this joint problem by concurrently learning High Level (HL) and Low Level (LL) neural network policies. These two levels are connected by a low dimensional hidden layer, which we call latent command. HL receives a first-person camera view, whereas LL receives the latent command from HL and the robot's on-board sensors to control its actuators. We train policies to walk in two different environments: a curved cliff and a maze. We show that hierarchical policies can concurrently learn to locomote and navigate in these environments, and show they are more efficient than non-hierarchical neural network policies. This architecture also allows for knowledge reuse across tasks. LL networks trained on one task can be transferred to a new task in a new environment. Finally HL, which processes camera images, can be evaluated at much lower and varying frequencies compared to LL, thus reducing computation times and bandwidth requirements. Video illustrations of our learned policies are available at this link[1].

Keywords: Hierarchical Reinforcement Learning, Vision Based Locomotion

1 Introduction

Legged robots have the potential to traverse many types of terrains while demonstrating a diverse set of agile skills. However, control of legged robots is challenging due to the dynamic nature of the problem. When incorporating visual inputs in the control loop, the task at hand becomes more difficult as it requires perceiving the environment while simultaneously handling the fast and contact-rich dynamics of the robot's legs.

One solution is to split the problem into independent modules for vision and dynamics. However, this approach is typically limited by the high-level features and low-level behaviors that are independently designed or learned. It may be impossible to come up with a single feature space that is optimal for all given tasks, or the low-level behaviors needed might be different for each given task. We tackle the two problems of vision processing and fast dynamics by designing a hierarchical architecture with a high level (HL) and low level (LL) subsystem, which are concurrently trained. This framework does not require design decisions beyond a standard RL setup. HL handles vision with variable frequency and outputs a latent command, which is passed on to LL. LL runs at a higher frequency and handles control of the legs.

We build on the hierarchical architecture presented in a related work by Jain et al [1] and incorporate vision processing to learn to navigate environments while concurrently discovering legged locomotion skills. The architecture is trained using Evolutionary Strategies (ES) [2]. The main contributions of our research are as follows:

[1] https://rb.gy/jacqsb

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

- Our HRL solution implicitly learns a complete pipeline from pixels to motor commands for quadruped legged locomotion without the need to design or learn low-level behaviors. We show that a hierarchical policy with more than 10^5 parameters can be successfully trained using evolutionary strategies.
- Separation of observations into hierarchical levels allows knowledge reuse, because behaviors learned by LL are largely task agnostic and transferable to tasks of a similar nature. We show that data efficiency can be further improved by transferring LL from previously solved tasks, even if they were trained in a different environment.
- The high level runs at a variable frequency computed by the high level's neural network. The result is that visual inputs are processed at much lower frequency ($1.5\,\mathrm{Hz} - 10\,\mathrm{Hz}$) compared to the low level ($500\,\mathrm{Hz}$). This leads to more efficient exploration and reduced training times, as illustrated by our experiments.
- We illustrate how locomotion primitives emerge and can be selected as a low dimensional latent command. This creates an information bottleneck in which HL learns to extract only the useful visual information and LL learns only primitives relevant to the environment and the task at hand. We provide a detailed analysis of these specific learned behaviors.

To test our method, we use a highly realistic simulation model of the Laikago robot, a quadruped with 12 degrees of freedom. The model is created in PyBullet [3] software and carefully tuned based on the physical robot. Despite our temporary inability to validate the results on hardware, we are confident in transferring our learned policies. In prior work, we have demonstrated successful deployment of policies learned in simulation on the robot. Additionally, we have performed validation experiments for the HL by processing real world depth images and verifying the computed latent commands. More details are provided in Appendix A. We test our framework on 3 visual navigation tasks and compare our policies with a non-hierarchical baseline. Our method outperforms the baseline and achieves increased sample efficiency and a lower wall-clock training time. Furthermore, we show that by running HL at low frequency, the inference time and computation cost of the learned policy is reduced with minimal effect on task performance.

2 Background and Related Work

Hierarchical Reinforcement Learning. HRL decomposes complex decision making into subproblems. Well-known HRL frameworks in literature are based on Options [4], MAXQ value decomposition [5] and Hierarchical Abstract Machines [6]. In these frameworks, a HL policy typically outputs temporally extended actions which are executed by LL for a specified amount of time. Designing or training good LL policies quickly becomes challenging for complicated tasks such as those encountered in robotics. Some papers try to learn LL by imitating from reference data [7, 8, 9]. This approach relies heavily on high quality data-sets, which are often hard to obtain.

Recognizing the challenges of pre-training LL, we focus on a framework for learning both levels concurrently from scratch; many approaches have been proposed for this. In one class of methods, a sub-goal conditioned LL is trained to reach a point in observation space specified by HL [10, 11, 12]. However, this interface is not suitable in the case of high-dimensional observation spaces, such as camera images [13]. Some methods design auxiliary rewards to promote diversity in LL skills [14, 15, 16]; however, by doing so, the RL agent may be forced to learn many skills irrelevant to the given task, leading to inefficiency in learning. Bacon et al [17] use an intrinsic reward function to learn LL. On the contrary, by training our policy with gradient-free policy search, we are able to train the whole hierarchical architecture from the main task reward. Thus we avoid imposing any external priors on LL behavior through intrinsic or auxiliary rewards. Some methods learn a finite set of options [17, 18, 19]. In our solution, we adopt the hierarchical policy structure proposed in [1], which uses a vector space for modulating LL, allowing it to learn a continuum of skills. This is required to solve agile locomotion tasks.

Legged Locomotion. Reinforcement Learning (RL) has been successfully applied to the problem of learning basic locomotion skills [20, 21, 22]. Complex locomotion tasks have also been addressed using RL [9, 23, 7, 24, 25]. Often, as complexity grows, only a part of the pipeline uses RL. Domain knowledge is used to constrain the problem, usually through a hand-crafted hierarchical solution [9, 10]. This is especially true for locomotion on real robots [22]. Peng et al [9] use HRL to learn locomotion in physics-based character animation. The two levels are learned separately by

means of RL. Li et al [10] learn locomotion on the hexapod robot named Daisy. In their solution, HL does model planning to select one out of a few pre-trained LL skills. In this work, we solve vision-based legged locomotion without designing or pre-training LL.

Perception in Robotics. Solving robotics tasks from vision input is an important and well-researched topic [26, 27, 28, 29]. Our work focuses on learning legged locomotion and necessary navigation skills from vision. Prior work has considered HL kinematic or wheeled navigation from vision [29, 30, 31]. Our method learns LL legged locomotion directly from vision input, which involves processing vision to obtain navigation directives and learning dynamic legged locomotion skills to execute those directives. Some solutions for vision-based legged locomotion have been proposed that use domain-specific, hand-designed pipelines [32, 33].

Evolutionary Strategies in Robotics. Many RL algorithms for continuous control are available for training policies to solve robotics tasks; policy gradient and actor-critic methods are especially popular. However, given the architecture of our hierarchical policy, training with derivative-free approaches is the reasonable choice. Recently, evolutionary methods [2] have been successfully applied in robotics [21, 34, 35, 36]. We use an evolutionary algorithm called Augmented Random Search (ARS) [37] to optimize our neural network policies.

3 Method

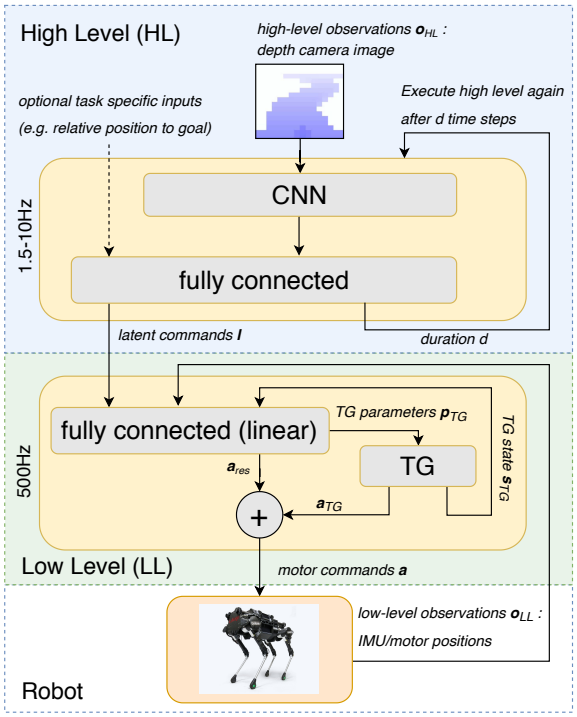


Figure 1: **Hierarchical policy.** The high level (HL) is a CNN with parameters $\boldsymbol{\theta}_{HL}$. The HL receives depth camera observations $\boldsymbol{o}_{HL}$ and outputs a *latent command vector* $\boldsymbol{l}$ and a duration d. Optionally, task specific inputs can be fed into the HL's fully connected output layer. The low level (LL) is a linear network with parameters $\boldsymbol{\theta}_{LL}$. It computes motor actuation commands $\boldsymbol{a}_{res}$ and trajectory generator parameters $\boldsymbol{p}_{TG}$ based on $\boldsymbol{l}$, trajectory generator state $\boldsymbol{s}_{TG}$, and low-level observations $\boldsymbol{o}_{LL}$ (IMU sensor values and motor angles). $\boldsymbol{a}_{res}$ is added to the motor commands from trajectory generator $\boldsymbol{a}_{TG}$ and applied to the robot motors. The HL is only evaluated every d steps. In our experiments, the low level runs at the environment simulator's frequency of $500\,\text{Hz}$, while the high level policy runs at $1.5\,\text{Hz} - 10\,\text{Hz}$. The HL and LL level networks are trained concurrently by an evolutionary algorithm.

The hierarchical policy structure introduced in [1] that we use for our solution is illustrated in Fig. 1. The HL is a convolution neural network (CNN) while the LL is a linear fully-connected neural network. The policy interacts with the robot, which is controlled by combining the output of a trajectory generator (TG) with values computed by the LL's fully connected layer. A TG serves as a parameterized function that computes cyclic leg positions. The LL neural network continuously modulates the TG's phase and amplitude and adjusts the leg trajectories with residuals as needed. More details about the usage of TGs for learning locomotion can be found in [21].

Algorithm 1 shows how an episode is executed using a hierarchical policy in which the HL network ($f_{\boldsymbol{\theta}_{HL}}$) and LL network ($f_{\boldsymbol{\theta}_{LL}}$) have weights $\boldsymbol{\theta}_{HL}$ and $\boldsymbol{\theta}_{LL}$ respectively. The HL receives task specific exteroceptive observations ($\boldsymbol{o}_{HL}$), such as the vision input in our tasks, and issues commands as a latent vector ($\boldsymbol{l}$) to LL. HL also decides the duration (d) until its next execution. Note that the HL can also optionally receive task-specific inputs (e.g. relative position to goal). The LL receives

Algorithm 1 Executing a Hierarchical Policy

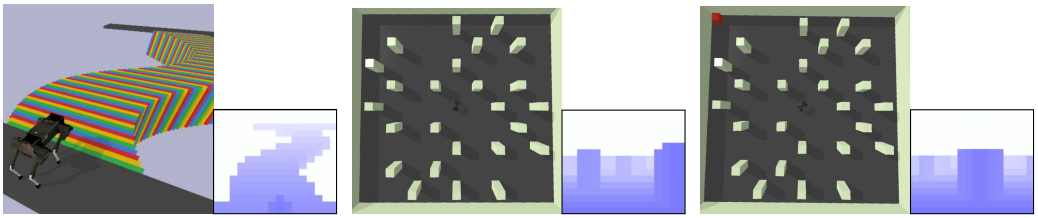
procedure RUNHRLPOLICY($\boldsymbol{\theta}_{HL}, \boldsymbol{\theta}_{LL}$) ▷ HRL policy weights
  $\{R, d, \boldsymbol{s}_{TG}\} \leftarrow \{0, 0, \mathbf{0}\}$ ▷ Return, HL duration, TG state
  while not end of episode **do**
    if $d = 0$ **then**
      $\boldsymbol{o}_{HL} \leftarrow$ Most recent depth image ▷ HL observation
      $\{d, \boldsymbol{l}\} \leftarrow f_{\boldsymbol{\theta}_{HL}}(\boldsymbol{o}_{HL})$ ▷ HL execution
    $\boldsymbol{o}_{LL} \leftarrow$ IMU & motor positions ▷ LL observation
    $\{\boldsymbol{a}_{res}, \boldsymbol{p}_{TG}\} \leftarrow f_{\boldsymbol{\theta}_{LL}}(\boldsymbol{l}, \boldsymbol{s}_{TG}, \boldsymbol{o}_{LL})$ ▷ LL execution
    $\{\boldsymbol{a}_{TG}, \boldsymbol{s}_{TG}\} \leftarrow f_{TG}(\boldsymbol{p}_{TG})$ ▷ Trajectory generator execution
    $r \leftarrow$ ExecuteAction($\boldsymbol{a}_{res} + \boldsymbol{a}_{TG}$)
    $d \leftarrow d - 1$
    $R \leftarrow R + r$
  return R

proprioceptive observations ($\boldsymbol{o}_{LL}$) that include IMU (roll, pitch, roll rate and pitch rate) and motor angles. LL also processes the current latent command ($\boldsymbol{l}$) and the current TG state, $\boldsymbol{s}_{TG}$. The LL outputs TG parameters $\boldsymbol{p}_{TG}$ and residual motor actuation commands $\boldsymbol{a}_{res}$, which are added to TG output $\boldsymbol{a}_{TG}$ and executed on the hardware. The environment returns the task reward (r) for the robot's action. The HL is invoked again after duration d and the process repeats.

A reinforcement learning problem can be modeled as a Markov Decision Process (MDP) with state space $\mathcal{S}$, action space $\mathcal{A}$, a state transition function $P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ and a reward function, $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$. A policy $\pi_\Theta(\boldsymbol{s})$, parameterized by a weight vector $\boldsymbol{\Theta}$, maps states $\boldsymbol{s}$ to actions $\boldsymbol{a}$. For a hierarchical policy, $\boldsymbol{\Theta}$ is the collection of parameters from all levels ($\boldsymbol{\Theta} = \{\boldsymbol{\theta}_{HL}, \boldsymbol{\theta}_{LL}\}$). The policy interacts with the MDP for an episode of T timesteps at a time. To jointly learn the parameters $\boldsymbol{\theta}_{HL}$ and $\boldsymbol{\theta}_{LL}$ of the two levels, we maximize the expected total reward (return) at the end of an episode.

We use an evolutionary algorithm called Augmented Random Search (ARS) [37] to maximize the return. The algorithm proceeds by iteratively estimating gradient of return w.r.t. policy parameters and performing gradient ascent.

During training, LL skills automatically emerge and are invoked by HL through latent commands ($\boldsymbol{l}$) to solve a task. A trained LL can also be transferred to new tasks in unseen environments. This allows sharing of primitive skills across problems and is faster than learning from scratch on each task. LLs can be transferred by keeping $\boldsymbol{\theta}_{LL}$ fixed after training on the original task and re-initializing $\boldsymbol{\theta}_{HL}$. Then, during new training only $\boldsymbol{\theta}_{HL}$ is updated by ARS.



(a) Curved cliff environment. (b) Maze traversal task. (c) Goal finding task in maze.

Figure 2: Task environments and vision inputs from depth camera.

4 Experimental Setup

We use the Laikago quadruped robot from Unitree[2]. This robot is 60 cm tall, has 12 degrees of freedom (3 per leg) and weighs about 22 kg. The swing and extension of each leg is controlled by a PD position controller provided with the robot. We train our policies in simulation using PyBullet [3, 38]. Our tasks are set up in two environments: a curved cliff and a maze.

[2] unitree.cc

Curved Cliff Environment. In this environment, the robot starts from the origin and a curved cliff lies ahead of it. The robot can observe the environment through a first-person depth-camera view, angled down slightly to see the cliff. Fig. 2a shows a still from this environment and a sample camera input. The robot's task in this environment is to progress forward as fast as possible. To accomplish this, it needs to learn to steer in order to follow the curves of the cliff and avoid falling off the edge. The shape of the cliff curve is randomized for each episode. The reward function is specified as the capped (v_{cap}) velocity of the robot along the x direction:

$$f_{v_{cap}}(r) = \max(-v_{cap}, \min(r, v_{cap})) \tag{1}$$

$$r_{cc}(t) = f_{v_{cap}}(x(t) - x(t-1)). \tag{2}$$

Maze Environment. For the maze environment, the robot is placed in the middle of a walled $13 \times 13\,\mathrm{m}^2$ arena uniformly filled with pillars. An episode starts with randomly orientated robot observing the world with a depth-camera looking straight ahead. This environment and a sample camera image is shown in Fig. 2b. We set up two tasks in this environment: maze traversal and goal finding. For the maze traversal task, the robot needs to keep going further away from its starting point (origin). The optimal behavior constitutes stable forward walking and steering to avoid colliding with pillars and boundary walls. For this task, the robot also observes its position and orientation relative to the origin, $\mathbf{x}$. The reward function for this task is as follows:

$$r_{mt}(t) = f_{v_{cap}}(\|\mathbf{x}(t)\| - \|\mathbf{x}(t-1)\|). \tag{3}$$

In the goal finding task, the robot needs to reach a goal randomly placed in one of the 4 corners of the maze for each episode (see Fig. 2c). Along with the camera input, it observes its position and orientation relative to the goal. To successfully find the goal it needs to learn to align itself in the direction of the goal along with all the skills for maze traversal. The reward function is given by:

$$r_{gf}(t) = f_{v_{cap}}(\|\mathbf{x}(t-1) - \mathbf{g}\| - \|\mathbf{x}(t) - \mathbf{g}\|) \tag{4}$$

$$r(t) = \omega r_{gf}(t) + (1-\omega) r_{mt}(t); \quad \omega = \|\mathbf{x}(t)\| \,/\, \|\mathbf{g}\|\,, \tag{5}$$

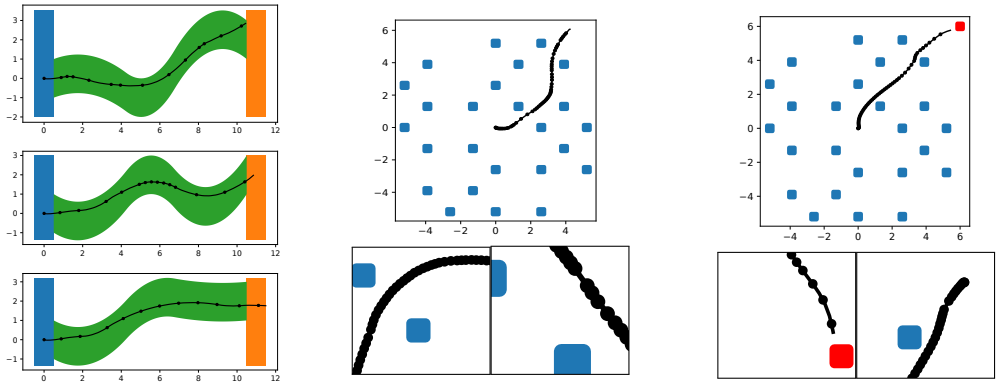
where $\mathbf{g}$ is the position of the goal. The reward is a weighted average of the maze traversal reward term r_{mt}, and a term for progression towards the goal position, r_{gf}, based on ω. The variable ω corresponds to the fraction of the distance travelled relative to the total distance from the goal to the origin. The reward is dominated by r_{mt} when the robot is close to the origin and becomes more defined by r_{gf} as it gets closer to the goal. This reward function encourages the robot to learn locomotion skills in the early stages of training. Without r_{mt}, the robot doesn't experience any positive reinforcement for stable walking unless it happens to walk in the goal direction.

In all tasks, the episode terminates if the robot loses its balance, falls off a cliff, collides with a pillar or boundary wall, or if the episode reaches 6000 LL robot control time steps ($12\,\mathrm{s}$). Additionally, in the goal finding task, the episodes terminates when the robot comes within $0.5\,\mathrm{m}$ of the goal position. We set v_{cap} to $0.002\,\mathrm{m}$ in all experiments, which corresponds to $1\,\mathrm{m\,s^{-1}}$.

Solution Implementation details. The high level contains a CNN that receives a $16 \times 16 \times 1$ depth camera input. It has 3 convolutional layers of 3×3 filters with output channels $4, 8$, and 8, followed by a pooling layer with filter of size 2×2 applied with a stride of 2. Output from the pooling layer is flattened and transformed into a $10D$ feature vector through a fully-connected layer with $\tanh$ activation. If present, the task-specific HL inputs (relative position in the maze environment) are concatenated with the feature vector. It is then fed into a fully-connected layer to produce an output clipped between -1 and 1. For most of our experiments we use a $3D$ output, with the first dimension corresponding to the HL duration (d) and the rest to the latent command ($\boldsymbol{l}$). The duration is calculated by linearly scaling the output to a value between 50 - 300 time-steps ($\approx 1.5\,\mathrm{Hz} - 10\,\mathrm{Hz}$). The latent command concatenated with IMU, motor angles, and trajectory generator (TG) state is fed to LL linear fully-connected network to output the residual motor commands and PMTG parameters. HL network has around 3000 parameters and LL has around 300. For comparison, we also train a non-hierarchical CNN with same convolutional and pooling layers as above. The feature vector is concatenated with all other sensor observations and fed to 2 fully connected layers (hidden layer size is 10) before producing the actions.

The trajectory generator (TG) is based on the *Policies Modulating Trajectory Generators* (PMTG) architecture, which has shown success at learning diverse primitive behaviors for quadruped robots [21]. As mentioned above, LL observes the PMTG state (s_{TG}), which specifies the position along a periodic leg trajectory, and updates the PMTG parameters at every time-step.

We train the policies using a distributed ARS implementation. For each optimization iteration, we evaluate policy perturbations on 64 parallel workers. Since all our tasks are randomized, we take average of return from 3 environment episodes to evaluate each perturbation. The number of perturbations evaluated, gradient step size, number of top perturbations used for gradient estimation, and the standard deviation for generating new perturbations are all determined by hyper-parameter tuning using a gaussian process bandits approach [39].



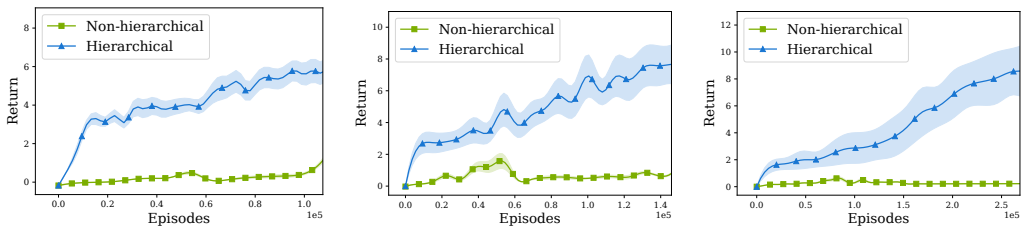
(a) Curved cliff task. (b) Maze traversal task. (c) Goal Finding Task.

Figure 3: Robot trajectories in simulated environments. Dots indicate HL execution (d). Axes in m.

5 Experimental Results

Our hierarchical policies are able to complete all 3 visual navigation tasks described in Sec. 4 by learning locomotion directly from vision input. Fig. 3 shows the trajectories of the trained robot in simulation for the 3 tasks. Dot markers, along the trajectories, show the points at which HL becomes active and computes the next latent command ($\boldsymbol{l}$) and duration (d). Notice that for solving the curved cliff task (Fig. 3a), the HL takes decisions more frequently (small d) when sharp turns are made to avoid falling off the cliff. In straighter regions, HL executions are sparser (large d). For the goal finding task, HL takes sparser decisions when the goal is close (Fig. 3c). The robot efficiently turns in-place to face the goal using dynamic leg movements which are difficult to hand-design and tune.

We compare the learning curves for our hierarchical policies described in Sec. 3 with non-hierarchical CNN policies on these 3 tasks (refer to Fig. 4). The plot lines show the average return across ≈ 450 environment episodes. The shaded region denotes the standard deviation. The return is plotted against the total training episodes. We see that in all 3 cases our method largely outper-



Curved cliff task. Maze traversal task. Goal finding task.

Figure 4: Learning curves for HRL and CNN policies.

forms the baseline, completing each task at the end of the training. Though we trained policies with approximately 10^3 weights (Sec. 4 for details) in most of our experiments, we show that larger CNNs with over 10^5 weights can be trained using ARS. Fig. 5 shows the learning curve for the maze traversal and goal finding tasks. The image resolution is 32×32 and we use 2 additional $32D$ fully connected layers at the end of the CNN.

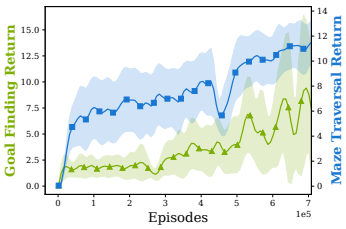
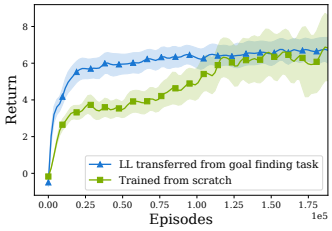
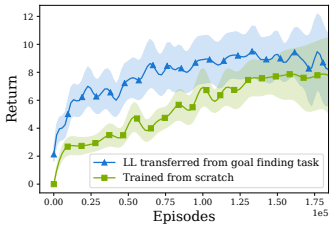


Figure 5: Learning HRL policies with 10^5 parameters.



Curved cliff.

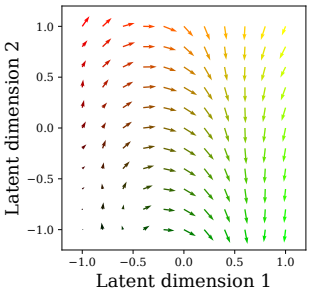


Maze traversal.

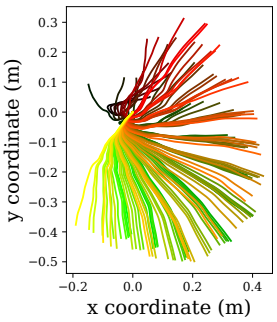
Figure 6: Transferring LL from goal finding task.

Transferring Low Level Policies between Tasks. LLs from learned policies can be reused for new tasks and environments as shown in Fig. 6. We first trained a policy with $2D$ latent commands for the goal finding task. The LL only has access to proprioceptive sensors, which forces it to learn generic steering and turning-in-place primitives. We reuse these skills for training a new policy in the maze traversal and curved cliff tasks based on the goal finding LL. LL weights are initialized from a pre-trained policy and frozen during the new HL policy training. In both of these cases we observe that this improves the learning efficiency. As can be expected, LL policies trained on the simpler and more restricted cliff task did not yield good performance when transferred to the maze traversal or goal finding tasks. In future work, we plan to explore fine-tuning transferred LL weights so that it can adapt to new tasks if previously learned skills do not suffice.

6 Analysis of Hierarchical Policies



2D latent command space.



LL trajectories.

Figure 7: Latent command space analysis.

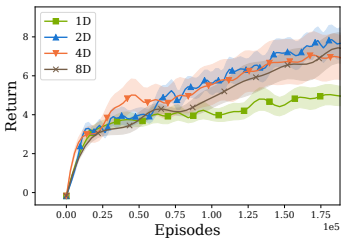
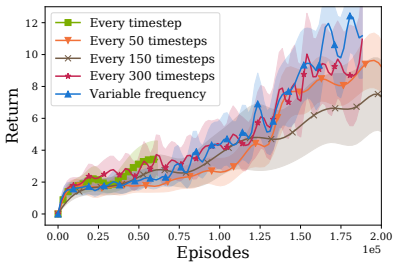


Figure 8: Latent command dimensions comparison.

Analysing 2D Latent Command Space. To better understand the behavior of the learned policy, we visualize a $2D$ latent command space for the goal finding task (Fig. 7). For a learned hierarchical policy, we evaluated the low level with artificial latent command inputs taken from a uniform grid over the whole command space. Those behaviors are shown on the right of the figure in the form of robot trajectories running for 1 s in the XY coordinate plane. On the left we show the latent command space points for which these behaviors emerged in corresponding colors. For each point, we generate a vector summarizing the LL trajectory. Vector direction shows the movement direction of the whole LL trajectory and the length is proportional to the distance covered. This visualization shows how the latent space is used to smoothly transition between robot steering behaviors of varying velocities that have emerged automatically.



High level frequency (every n timesteps)	Inference time (s)	Effective policy size	Training speed (timesteps/s)
1	0.03000	3256	210.2
50	0.00070	359.1	1977.3
150	0.00030	319.7	1689.8
300	0.00020	309.8	1803.8
Variable freq.	0.00055	328.2	1774.2

Figure 9: Comparing hierarchical policies with HL running at different frequencies. Left: Running HL at lower frequencies has minimal effect on performance. Right: Without temporal abstraction, the inference and training speed is low.

Influence of Latent Command Dimension. The comparison between hierarchical policies with $1, 2, 4$, and 8 dimensional latent command space (LCS) is shown in Fig. 8. These policies are trained on the curved cliff task. It is clear that the $1D$ LCS is too restrictive for this task and the policy is not able to achieve optimal performance. The $2, 4$, and $8D$ LCS perform similarly. It is promising to see $2D$ LCS reaching optimal performance, since low dimensional LCS has many benefits: they can be easily visualized and interpreted, they are easy to control and hence amenable to transfer, and they reduce the network size making it easier to train.

Impact of High Level Frequency on Training and Inference. To study the effect of temporal abstraction, we compare policies with different HL execution durations, trained on the goal finding task (see Fig. 9). On the left, we show the learning curves for policies with the HL running once every 1, 50, 150, 300, and d time-steps, where d is the variable time interval output by the HL. We see that all variants are able to learn the task with minor differences in performance. However, notice that HL running at every time-step has made comparatively little progress. This difference in training speed is captured in Column 4 of the table on the right. It is clearly inefficient to run the HL every time given that we can achieve the optimal return much faster with the temporally abstracted policies. The exact inference times are recorded in Column 2. Inference on temporally abstracted policies is $\approx$ 100 times faster, which will also facilitate deployment on hardware. Column 3 calculates the effective size of the policies over time due to variation in HL frequency.

7 Discussion and Conclusion

We presented an HRL technique to solve visual navigation for a quadruped from pixels to leg motions. Our method outperformed non-hierarchical baselines on 3 navigation tasks and achieved higher data efficiency and a lower wall-clock training time. However, the advantages of using our approach extend beyond performance improvements. First, by decoupling the high level from the low level, we were able to run them at different frequencies. Indeed, the high level learns when it has to process a new image. This has practical applications as processing vision input synchronously with the low level control loop is often impractical. Secondly, we analyzed low level policies and demonstrated that they can be transferred between tasks. This is important as it is non-trivial to define skills - encoded by the latent command space - that are both robust and exploit the full range of robot capabilities. Transfer of low level policies makes it possible to use the learned skills as a continuous low-level action space for other learning algorithms, a research direction we intend to pursue in future work.

Note on Hardware Evaluation. Due to COVID-19 restrictions, we were unable to include hardware results. However, we are planning to implement our hierarchical policies on a real Laikago robot once we regain access to our lab spaces. We have previously validated learned low-level policies similar to those found by our algorithm on multiple real robots, and we are therefore confident in the transfer of our HRL policies to hardware. We have also already tested our vision stack in combination with a predefined set of low-level skills on a legged robot. Finally, we found that similar latent commands were computed by the high level network when presented with real depth camera images and simulated ones in an environment with obstacles. More details can be found in Appendix A.

References

[1] D. Jain, A. Iscen, and K. Caluwaerts. Hierarchical reinforcement learning for quadruped locomotion. *IROS*, pages 7551–7557, 2019.

[2] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[3] E. Coumans. Bullet Physics SDK. https://github.com/bulletphysics/bullet3, 2013.

[4] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[5] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

[6] R. Parr and S. J. Russell. Reinforcement learning with hierarchies of machines. In *Advances in neural information processing systems*, pages 1043–1049, 1998.

[7] J. Merel, A. Ahuja, V. Pham, S. Tunyasuvunakool, S. Liu, D. Tirumala, N. Heess, and G. Wayne. Hierarchical visuomotor control of humanoids. *arXiv preprint arXiv:1811.09656*, 2018.

[8] X. B. Peng, M. Chang, G. Zhang, P. Abbeel, and S. Levine. MCP: Learning composable hierarchical control with multiplicative compositional policies. *ArXiv*, abs/1905.09808, 2019.

[9] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4): 41, 2017.

[10] T. Li, N. G. Lambert, R. Calandra, F. Meier, and A. Rai. Learning generalizable locomotion skills with hierarchical reinforcement learning. *ArXiv*, abs/1909.12324, 2019.

[11] A. Levy, R. Platt, and K. Saenko. Hierarchical reinforcement learning with hindsight. In *International Conference on Learning Representations*, 2019.

[12] O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3307–3317, 2018.

[13] O. Nachum, S. Gu, H. Lee, and S. Levine. Near-optimal representation learning for hierarchical reinforcement learning. *arXiv preprint arXiv:1810.01257*, 2018.

[14] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

[15] C. Florensa, Y. Duan, and P. Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

[16] J. D. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, and S. Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *ICML*, 2018.

[17] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[18] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017.

[19] C. Daniel, G. Neumann, O. Kroemer, and J. Peters. Learning sequential motor tasks. In *2013 IEEE International Conference on Robotics and Automation*, pages 2626–2632. IEEE, 2013.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[21] A. Iscen, K. Caluwaerts, J. Tan, T. Zhang, E. Coumans, V. Sindhwani, and V. Vanhoucke. Policies modulating trajectory generators. In *CoRL*, pages 916–926, 2018.

[22] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), 2019.

[23] X. B. Peng, G. Berseth, and M. Van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):81, 2016.

[24] N. Heess, G. Wayne, Y. Tassa, T. P. Lillicrap, M. A. Riedmiller, and D. Silver. Learning and transfer of modulated locomotor controllers. *CoRR*, abs/1610.05182, 2016.

[25] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. van de Panne. Iterative reinforcement learning based design of dynamic locomotion skills for cassie. *arXiv preprint arXiv:1903.09537*, 2019.

[26] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[27] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 79–86. IEEE, 2017.

[28] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[29] X. Pan, T. Zhang, B. Ichter, A. Faust, J. Tan, and S. Ha. Zero-shot imitation learning from demonstrations for legged robot visual navigation. *ArXiv*, abs/1909.12971, 2019.

[30] C. Li, F. Xia, R. M. Martin, and S. Savarese. HRL4IN: Hierarchical reinforcement learning for interactive navigation with mobile manipulators. In *CoRL*, 2019.

[31] G. Blanc, Y. Mezouar, and P. Martinet. Indoor navigation of a wheeled mobile robot along visual routes. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 3354–3359. IEEE, 2005.

[32] S. Bazeille, V. Barasuol, M. Focchi, I. Havoutis, M. Frigerio, J. Buchli, C. Semini, and D. G. Caldwell. Vision enhanced reactive locomotion control for trotting on rough terrain. In *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, pages 1–6. IEEE, 2013.

[33] O. A. V. Magana, V. Barasuol, M. Camurri, L. Franceschi, M. Focchi, M. Pontil, D. G. Caldwell, and C. Semini. Fast and continuous foothold adaptation for dynamic locomotion through cnns. *IEEE Robotics and Automation Letters*, 4(2):2140–2147, 2019.

[34] W. Gao, L. Graesser, K. Choromanski, X. Song, N. Lazic, P. Sanketi, V. Sindhwani, and N. Jaitly. Robotic table tennis with model-free reinforcement learning. *arXiv preprint arXiv:2003.14398*, 2020.

[35] K. Choromanski, A. Pacchiano, J. Parker-Holder, Y. Tang, D. Jain, Y. Yang, A. Iscen, J. Hsu, and V. Sindhwani. Provably robust blackbox optimization for reinforcement learning. In *Conference on Robot Learning*, pages 683–696, 2020.

[36] X. Song, K. Choromanski, J. Parker-Holder, Y. Tang, W. Gao, A. Pacchiano, T. Sarlos, D. Jain, and Y. Yang. Reinforcement learning with chromatic networks. *arXiv preprint arXiv:1907.06511*, 2019.

[37] H. Mania, A. Guy, and B. Recht. Simple random search of static linear policies is competitive for reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 1805–1814, Red Hook, NY, USA, 2018. Curran Associates Inc.

[38] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi:10.15607/RSS.2018.XIV.010.

[39] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM, 2017.

[40] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese. Gibson Env: real-world perception for embodied agents. In *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE, 2018.

A Feasibility of Deploying Our Learned Policies on the Robot

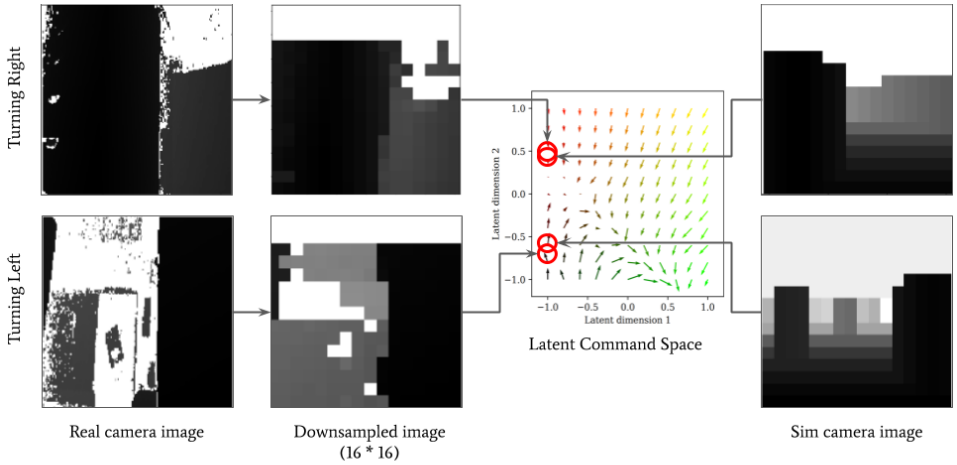


Figure 10: Real depth camera images processed by high level.

We trained a hierarchical policy on goal finding task in simulation and evaluated the learned HL on images from a real depth camera (Intel RealSense L515). We compared the downsampled (16×16) real world camera images with similar looking simulated camera images from our experiments. Both types of images result in similar latent commands, supporting compatibility of HL with real depth camera images (Fig. 10).

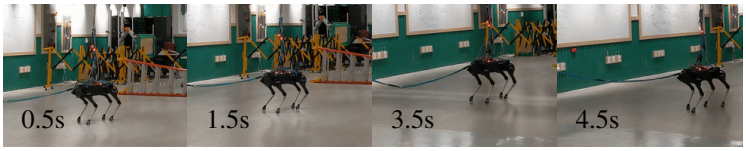
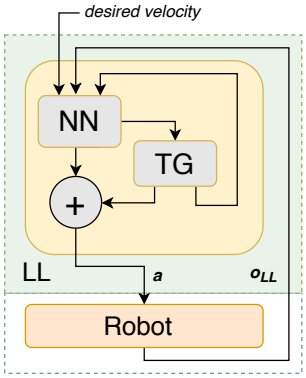


Figure 11: Forward walking policy deployed on hardware.

We have previously deployed a forward walking policy (trained in simulation) that tracks a desired velocity to the real Laikago robot (Fig. 11). The hierarchical policies presented in our experiments are trained with a similar infrastructure. Hence, we expect that our policies will transfer to hardware.



Finally, we trained our policies in simulated 3D spaces with realistic visuals from the Gibson dataset [40]. After training, our policies were able to transfer to a new space (Fig. 12).

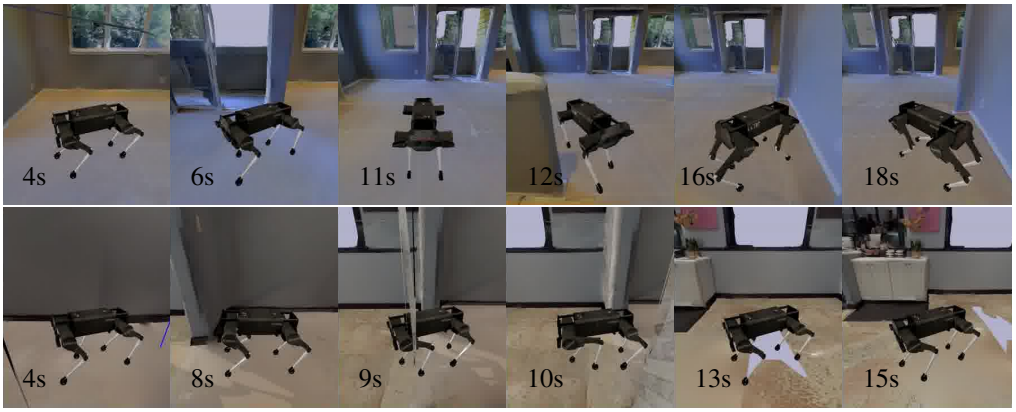


Figure 12: Navigating Gibson environments with hierarchical policies.