Leveraging Engineering Expertise in Deep Reinforcement Learning

by

Liam J. Ackerman

S.B. in Electrical Engineering and Computer Science,
Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

Author .....
Department of Electrical Engineering and Computer Science
August 5th, 2022

Certified by .....
Sangbae Kim
Professor
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Leveraging Engineering Expertise in Deep Reinforcement Learning

by

Liam J. Ackerman

Submitted to the Department of Electrical Engineering and Computer Science
on August 5th, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Deep reinforcement learning has been used to craft robust and performant control policies for legged robotics. However, the engineering processes to create these policies are often plagued by long training times that slow down engineering iteration. This thesis suggests that model-based controllers offer a wealth of successful computation that may be used within reinforcement learning control pipelines to improve learning efficiency. Two ideas incorporate this engineering expertise to increase reinforcement learning efficiency. First, successful model-based computations are pre-processed and incorporated directly into network observations. Introducing these terms into the reinforcement learning architecture is shown to increase learning speeds and policy performance dramatically. Next, inspired by model-based task hierarchies, more structure is added to the reinforcement learning objective function to activate and deactivate reward terms based on an agent's state. This structure is intended to avoid local minima which impede learning. This reward restructure is shown to avoid local minima during training but degrades final policy performance at edge-cases.

Thesis Supervisor: Sangbae Kim
Title: Professor

Acknowledgments

It takes a village. It takes a village of support, love, feedback, and motivation to do anything in a life where nothing happens in a vacuum. I am happy to have this chance to acknowledge just a small portion of the village which helped me to keep pushing, growing, and appreciating the world around me as I traversed MIT.

It is my pleasure and duty to thank the ones who brought me into this world and never stopped supporting me as I grew in it. To my parents, thank you so much for your unfathomable and consistent love and support. I could not have asked for a better team and would never even want to. I love you to the moon and back.

To Professor Kim, thank you for accepting me into your group and taking the time to converse with me this year. You have helped me to broaden my perspective on how to apply my skills as an engineer for a more profound life experience. Whether its the physics of the tennis swing and racket, lab employee management, picking up apples, or the torque-speed curves of human muscles, your analyses have been insightful and provided excellent guidance on how to better process the world around me. Thank you for creating such a superb laboratory environment for us all to grow in.

To my lab mates in the Biomimetic Robotics Lab, I appreciate each of you. Your company in both a social and technical sense made my year enjoyable and valuable. I have easily learned as much from all of you as I have from my traditional academic work this year. Thank you for being available in the lab this year.

Finally, to my friends and siblings, I appreciate you all being there to hear me out as I went through incredible highs and occasional lows. In particular, to my sister and friend Francesca who is going through the MIT graduate school process with me, I wish you all the best and would love to be there for you as much as you were there for me. To Joush, I appreciate spending this year working through life with you. You're an awesome friend who kept me afloat as I hope I did for you as well.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Legged robots provide a valuable middle-ground between wheeled platforms and flying drones. Their articulated legs offer the flexibility to traverse most terrains while contact with the ground enables a single robot to handle medium sized payloads. These capabilities make them the perfect candidates for replacing humans in safety critical operations like search and rescue operations [53] or any application requiring agents to navigate more extreme terrains while interacting with the environment [16].

The deployment of legged systems has been successfully demonstrated in a number of research and industry applications. For real-world deployment, the Robotics Systems Lab's Anymal legged platform [1] won the 2021 DARPA Urban underground challenge [52] and was successfully able to complete a mountain hike [40]. Boston Dynamic's robot Spot [2] is currently available for purchase and is used for continuous spot checks on industrial plants [3]. When it comes to studying robotic agility, the Biomimetic Robotics Lab's Mini Cheetah [31] can successfully complete dynamic maneuvers like jumps and flips [18]. The Cassie robotic platform [4] developed by Agility Robotics and OSU's Dynamic Robotics Laboratory can leverage its proprioceptive sensors to climb a staircase [48] without any vision.

Enabling these systems to perform such important or complex tasks requires enormous engineering creativity, implementation, debugging, and tuning [55]. Two control

paradigms that dominate the legged robotic control space are model-based optimal control and deep reinforcement learning. Each method offers unique (and occasionally complimentary) benefits, drawbacks, and opportunities for improvement. By leveraging an understanding of both approaches, related work has shown that hybrid solutions can bolster the strengths of each [29, 39, 22].

1.1.1 Model-Based Optimal Control

Model-based control poses legged locomotion as an optimal control problem [33]. Optimal control problems generally have three parts: decision variables, an objective function, and constraints. The solver or optimizer for optimal control problems will attempt to choose values for the decision variables to minimize the objective function while satisfying the constraints. For legged locomotion, the decision variables are usually joint torques, joint position targets, or footstep locations [35, 33, 20]. The objective function is usually hand-crafted and tuned according to human intuition and style preferences. The constraints usually contain an approximate model of the forward dynamics of the robot and its environment [21]. Deploying a model-based control pipeline requires real-time optimal solutions to be generated at high frequencies [14].

Achieving fast solve rates is the key goal in posing optimal control problems for legged robots. To accomplish this, faster solvers can be used [17, 5], constraints can be relaxed [28], approximate models can be further simplified [21, 37], and problems may be broken down into a hierarchy of cooperating modules [19]. Achieving balance among these engineering decisions can make or break a legged robotic controller. Making better engineering decisions requires constant iteration, testing, and tuning. Thus, significant engineering infrastructure has been built up around model-based solutions. Engineers have developed custom dynamics simulators [6, 7, 51] using efficient rigid body simulation algorithms [24]. These simulation software stacks offer a safe and accurate environments for measuring the performance of model-based controllers enabling shorter engineering feedback loops which are critical to the success of model-based solutions.

Unfortunately, model-based approaches are susceptible to humans' ability to understand and cover edge-cases in high-dimensional state and action spaces [25]. Also, any measured model inaccuracies or unmodeled behavior can significantly degrade optimal control solutions leading to a sim-to-real gap. While complex terrains often present the exact types of environments and disturbances which are difficult to model [52, 40], hand-crafted control heuristics have been used to cover many edge cases on which traditional model-based controllers fail [21, 20, 13]. These solutions are highly customizable to the human engineers who build them.

1.1.2 Deep Reinforcement Learning

Rather than using approximate models based on physical analysis, deep reinforcement learning [42] optimizes legged robotic controllers using collected data. Just like model-based optimal control, deep RL also attempts to optimize a solution for a human-designed optimal control problem; though, it uses a different method. This method deploys a control policy neural network on simulated agents and observes the rewards collected. These observations are combined into a training dataset containing state-action-reward trajectories. The control policy neural network uses the collected data to make better control decisions to maximize expected returns when the policy is redeployed. This learning process is still technically constrained to physical dynamics, yet the optimizer can only see the data it has collected rather than the human engineered approximate model of the system. Thus, the optimizer gets to choose how it models the control policy without direct human intervention and tractability assumptions.

Deep RL techniques have been used to craft control policies with increased robustness to unmodeled disturbances and with performance where model-based controllers (subject to lower-order approximations) may fail to deploy [50]. However, reinforcement learning pipelines are often inefficient and make engineering iteration difficult. The primary engineering challenge when working with deep RL is that training models requires significant amounts of time to converge. Additionally, large policy networks may require too many computations to operate in real time or may occupy too much

storage onboard a deployed system.

1.2 Hypothesis

1.2.1 Making Deep Reinforcement Learning More Efficient Using Engineering Expertise

For model-based control pipelines, the effects of tuning a controller parameter or changing a model can be qualitatively seen on the order of seconds or minutes. Quick feedback enables intuition and tunability to model-based controllers. Deep RL architectures lack this near-instantaneous system deployment. Each time a policy is trained by a neural network, it must be randomly initialized. It then relearns from scratch which can take minutes or hours to converge to an analysis-ready steady state. This order of magnitude difference in time occurs due to a core trade-off between classical engineering modeling and data-based approximation. In the case of model-based control, it is possible to iterate and tune without deleting older variations of the model. Model-free methods will need to relearn the physics of the system every time any change needs to be made.

Thus, if model-free pipelines were correctly modified in order to access existing engineering knowledge, then model-free methods may choose to incorporate this expertise resulting in less redundant work per policy training run. Leveraging these control architectures together should promote the fast iteration benefits of model-based control with the robustness of model-free learning.

1.3 Related Work

Leveraging prior knowledge about a problem space is not a new idea. Related works have shown success in improving deep reinforcement learning efficiency by including domain specific knowledge into learning pipelines. When applying deep RL to robotic applications, there are three main ways to change a specific architecture: (1) improv-

ing the quality of policy observations, (2) adjusting the action space of the policy, and (3) reformulating the reward functions.

1.3.1 Observations

Many works have created methods to estimate high-quality environment and system information. These works demonstrate that improving the quality of data is a key to learning better control policies. The student-teacher architecture used in [36] enables the estimation of better data by training its RL policy in two steps. First, a 'teacher' network is trained in simulation while given privileged information about the environmental parameters. Once the 'teacher' policy training converges, the 'student' network is then trained to make the same control decisions as the teacher network without the privileged information. The work reports that converged policy performance of the student network outperforms naively trained student networks by 33%.

The student-teacher architecture in [36] is an example of a knowledge distillation method [26]. Knowledge distillation can be used to estimate un-observable parameters, but they can also be used to compress and combine large policies into one smaller policy. Legged robotic control problems are usually goal conditioned: where commands from a human or joystick are included in the observations of the policy. Results from [55] show that different walking behaviors may emerge at different commanded planar velocity ranges. Informed by this engineering expertise, this work trains many sub-actor networks which learn to locomote at restricted commanded velocity ranges. Policy (knowledge) distillation is then used to combine each small range actor network together. By carefully considering final desired behaviors and their goal observations, this approach enables faster training for each restricted-velocity network and better performance for the final combined actor policy.

The two prior observational techniques improve policy performance by adding more learning stages to the RL pipeline. Instead, it is also possible to access better observation data in parallel. For example, the authors of [30] add another network to the learning architecture to concurrently estimate privileged information like end-effector positions and contact forces typically not available to deployed systems. This

work reports that adding this information increases policy performance and reduced learning instability.

Finally, it is also possible to choose better representations of existing observation data. Earlier work in reinforcement learning [34] found success by using a Fourier embedding for existing state information. The paper suggests that converged policies performed better at example tasks and converged quicker than methods that did not consider these pre-computed non-linearities. Another study [11] found that using the quaternion representation for three dimensional orientation led to better learned policy outcomes.

1.3.2 Actions

Other related works change the output interface of the actor network. Changing the policy output is often inspired by engineering best practices and requires the implementation of some model-based module within the model-free architecture.

For example, experiments performed in [45] show that it is a best practice in legged robotics to output joint position targets rather than joint torques directly. This change works because neural networks and real-time optimization tools typically operate on the order of 100s of Hz [33, 54], whereas the joint PD controllers update torque set-points at 10s of kHz. Rolling back the actor policy interface from torques to joint positions requires implementation of these PD joint regulators within the learning architecture. This extra engineering is worth the effort: network performance and learning speed is shown to improve when using soft gain and damped position residuals with only little trade off in neural network expressiveness.

The related works below continue this trend, finding a hybrid position between pure end-to-end learning and the more consistent/tunable model-based control modules.

Continuing to change the actor control interface, the authors of [22] incorporated an end-effector position controller into their learning pipeline. Here, the actor network could output task-space end-effector targets for the Cassie robotic platform. The work claims that letting the actor network explore in a more reasonable (task) space while

model-based control regulates the joints led to a four times increase in learning speed. On the Cassie system, reasoning in task space allowed a walking behavior to emerge within 8 hours of training rather than 32 hours in joint space.

A more extreme example of changing the action policy interface is seen in [39] where the authors let the reinforcement learning architecture output higher-level desired velocities and gaits to the Mini Cheetah system. Here, the actor network has no control over joint positions, but instead automates human joystick operation. Unfortunately, while learning in this space is quick, network performance is limited by the model-based lower level. The model-based controllers used here are shown in [33]. This control stack uses an optimal control formulation with tractability tradeoffs like small angle assumptions on the body orientation and angular velocity. More end-to-end learning solutions do not require these limiting assumptions.

1.3.3 Rewards

Finally, it is also possible to adjust the reward functions of model-free learning in order to boost performance and speed. Recall that reinforcement learning is just a different method for solving optimal control problems; thus, it is possible to make connections to more traditional optimal control solution expertise. Model-based pipelines will often form their objective functions to work well with the optimizing tools themselves. Some optimizers are best at solving convex problems [8], where others can solve highly non-linear optimal control problems [17]. Recognizing the importance of correctly formulating optimal control problems, the BMRL has completed work to better pose optimal control problems for real-time operation in [13, 21, 20].

Just like in more traditional optimal control, it is possible to change the objective function - equivalently, rewards - of a reinforcement learning policy to increase sample efficiency or learning speed. The most basic reward structure for an optimal control problem in RL is a sparse reward. Sparse rewards trigger a positive signal when some state is satisfied like being in a room or achieve a desired height. Using a sparse reward, an RL agent would explore randomly until the reward state was found and then update its policy to take actions which achieve that state as quickly as

possible. However, as the dimensionality of state and action spaces increase or become continuous, exploration can become intractable.

To alleviate long training times due to possibly infinite random exploration, sparse rewards may be substituted with dense rewards. Dense rewards smear the optimal epsilon reward ball across a greater portion of the state space. This way, an agent is more likely to see a non-zero reward states while exploring. Witnessing a positive signal will cause the optimizer to update the policy and value for reaching a specific state. State of the art learned methods have used tuned dense rewards to successfully and quickly train good control policies [30, 22, 40, 55]. However, dense reward functions may cause incorrect values to be assigned to specific states which bias the optimal policy and may introduce unnecessary local minima. These local minima impede the learning process and may be impossible to escape. The method in [43] addresses this issue by formulating dense rewards using a potential-based shaping method. The resulting shaped rewards maintain the sample efficiency benefits of hand-tuned dense reward formulations without incorrectly biasing optimal values which may create local minima.

Though potential-based reward shaping can achieve quick learning without adding local minima, it is not popular in state of the art systems for robotic control. Rather, researchers often elect to simply hand-tune their reward functions. This route is chosen because humans will often desire behaviors which "look better" but are sub-optimal rather than the actual optimal behavior for a system. The control pipelines presented in [44, 23] present an extreme example of this trend which discards traditional reward structures in favor of behavioral imitation of biological or stylized behavior. This line of work indicates that it is still important to have a human somewhere in the loop when it comes to designing near-anthropomorphic robotic behaviors.

It is still possible to shape dense rewards that are crafted by humans. Additional reward structuring in [27] uses a bi-level optimization to decide when some area of a dense reward is not useful and then turns off that signal. This allows the final policy performance to get closer to the optimal behavior while maintaining some of

the sample efficiency boosts gained by using the dense reward formulations. However, this approach may still remove important human-preferred behaviors while passing through the optimization black box.

Returning to model-based experience, these architectures occasionally achieve desired results by crafting a task hierarchy or prioritization to specify which task is most important to achieve. In [33, 12], authors establish some primary task and secondary task for a redundant articulated system. Here, the primary task can be solved in many ways. Within this null space of the solution to the primary task, the optimizer is attempts to solve lower priority task. In this optimization structure, the super-task can be solved without being obscured by sub-task gradient information.

1.4 Proposed Approach and Contribution

The Biomimetic Robotics Laboratory (BMRL) actively develops both model-based and model-free legged system controllers. This thesis explores two ideas which incorporate model-based engineering expertise into deep RL pipelines to improve the model-free engineering process by increasing learning speed.

1.4.1 Augmentations

For robotic applications using PPO, the actor network must be able to represent possibly non-linear optimal control laws and the critic network must accurately predict the value of reward trajectories constrained by kinematics and physical laws. Many studies [11, 34, 22, 39] show that choosing better information representations, adding privileged data, and extracting work out of the actor network can all contribute to boosting model-free learning efficiency. However, existing methods often require additional learning infrastructure or remove some expressive power from the neural network. This work attempts to achieve this same performance increase while neither limiting the expressive power of the actor network nor adding dramatically more engineering infrastructure. To accomplish the task, this work presents an architectural change shown in figure 1-1. Here, dynamics and control terms collected from

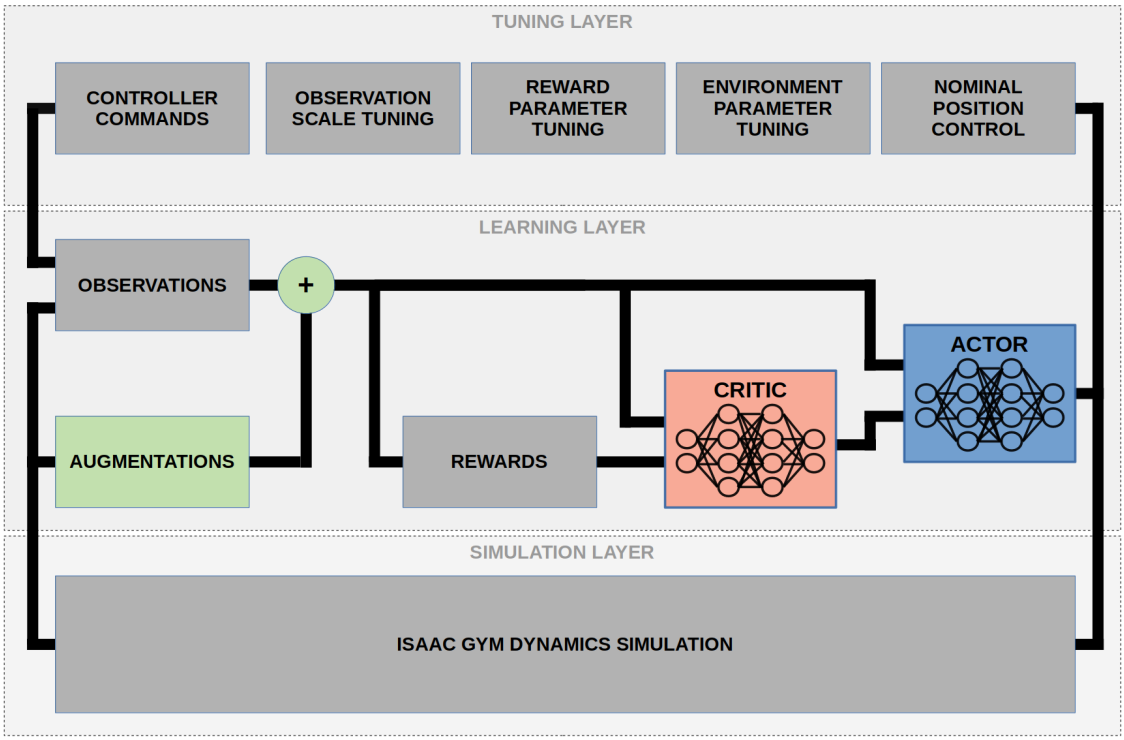

Figure 1-1: A traditional model-free learning pipeline includes three major layers: (1) a tuning layer where humans control the major scales, weights, and parameters of the learning environment and algorithm, (2) the learning layer where model-free algorithms optimize value estimates and make control decisions, and (3) a physics simulator where policies can be tested. This thesis presents the increment highlighted in green where non-linearities from successful model-based controllers are pre-processed for use by both the actor and critic networks.

already successful model-based controllers are pre-processed to be used to augment the observations of the both actor and critic networks

This thesis presents the methods and results of completing experiments to determine whether these pre-processed observation augmentations successfully improve learning efficiency. First, observation augmentations are introduced into a low-order test system: the cartpole. Initial testing on this system is performed to indicate which groups of augmentations perform best in improving learning speeds and predicting expected performance improvements when deployed on a full-order system. Next, augmentations are used to train a dynamics simulated version of the BMRL's Mini Cheetah legged robot. Experiments are completed on this system to determine learning performance with augmentations included.

1.4.2 Coupled Rewards

Traditional dense rewards increase sample efficiency and human-tunability, yet they can bias the exploration of an optimal policy leading to incorrect learned behaviors. One toy problem that illuminates this issue is the control of a car. Here, the goal for the car is to achieve a desired speed while minimizing the use of its accelerator pedal. If the car is always initialized at rest and the penalty from pressing the pedal always dominates the speed reward in this state, then the optimizer may never learn to press the pedal. The behavior represents a local minima. There is a higher reward policy, but the optimizer can never explore the high reward states due to the pedal penalty. Fixing this issue requires re-tuning the reward function, choosing a better initialization for the car, or changing the reward structure. For problems with greater complexity, simply re-tuning the rewards may not be able to best solve this issue and sampling the full space of initial states may be intractable. Thus, this thesis explores how restructuring the reward functions affects RL efficiency such that pedal-use is only minimized once the car is near the desired speed.

Specifically, this thesis attempts to reformulate the standard dense reward function formulation inspired by the task-prioritization seen in [33, 12]. This work leverages the structure of a specific dense reward function: the squared exponential function $e^{-(\frac{x^2}{\sigma})}$ (assuming that x is some continuous system state). By tightening the squared exponential's σ value, this reward function can be used as a multiplicative switch (turning on and off whatever it is multiplied to) based on the value of x. Studies in this work explore how this σ switch can be used to introduce a prioritization between opposing dense reward terms (like desired speed and pedal use) to remove local minima during training. An example of how the reward landscape will change is presented in figure 1-2.

Experiments are completed to explore the effects of adding reward prioritization to the cartpole system's swing up and stabilize model-free training pipeline. First, a sweep of σ_{switch} coupling values is completed to identify an optimal candidate value. Then, a learning efficiency analysis is completed comparing the learned policy using

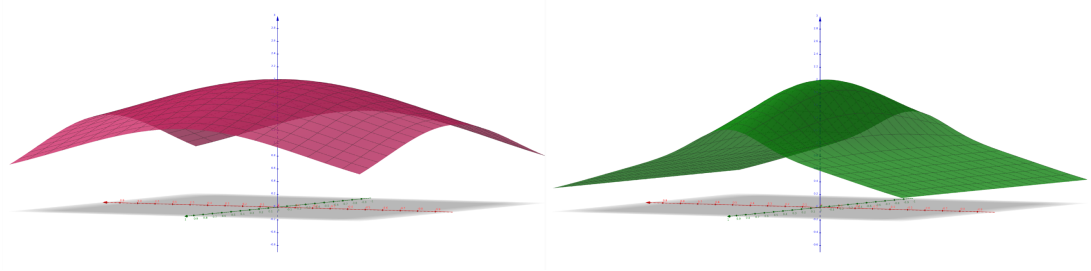

Figure 1-2: Two reward surfaces are shown with zero-centered squared exponential functions of x and y. The red graph shows the result of simply superimposing the sub-reward functions $e^{-(\frac{x^2}{\sigma})} + e^{-(\frac{y^2}{\sigma})}$. The green graph shows the result of implementing a task priority between rewarded states x and y as super and sub tasks. The sub task y is turned off until super task x is near zero. The new reward composition is $e^{-(\frac{x^2}{\sigma})} + \boldsymbol{e^{-(\frac{x^2}{\sigma_{switch}})}} e^{-(\frac{y^2}{\sigma})}$ where tuning σ_{switch} determines when the sub-reward function is activated.

the baseline reward function versus with coupled reward function. Finally, discussion is included about the results of these experiments.

Chapter 2

Systems Overview

This chapter details the implementation specifics of the reinforcement learning algorithm, the simulated learning environment, and the two systems on which the proposed ideas were deployed.

2.1 Reinforcement Learning Framework

2.1.1 Problem Formulation

This thesis uses the deep reinforcement learning algorithm PPO [47] to optimize the A3C architecture [41] containing two networks: the actor and critic. The actor network produces a mapping from the observation o_t of the current state s_t of an agent to an action output a_t. This stochastic mapping is represented as $\pi(o_t|\theta_t^a)$ where θ_t^a represents the tunable parameters of the actor neural network. The critic network estimates the expected value $\hat{V}(o_t|\theta_t^c)$ of being in some observed state according to the critic parameters θ_t^c.

PPO works in a loop of four repeated steps: deploying the actor network on many agents, collecting/processing deployment data, updating the critic network, and updating the actor network. During deployment, on-policy deployment data $\mathbf{D}$ are collected containing state-action-reward trajectories. Using this data, the actual value of observed states is calculated as an infinite-horizon discounted sum of rewards

such that $V(o_t) = \sum_t^{\mathbf{D}} \gamma^t r_t$ where γ^t is the discount factor and r_t represents the rewards collected by the environment. PPO uses supervised learning to optimize the critic network parameters to minimize the squared difference, $||\hat{V}(o_t|\theta_t^c) - V(o_t)||^2$. Finally, critic value estimates are used to update the actor network parameters θ_t^a in order to maximize expected value.

Beyond the more basic formulation of the deep reinforcement policy using the actor-critic architecture, the work [47] thoroughly describes all implementation details both for general PPO and [46] describes the specific formulation for legged systems. Key aspects of the problem formulation are also system dependent and are defined for both systems used in this thesis.

2.1.2 Learning Environment

Isaac Gym

In order to collect state-action-reward trajectory data, agent policies need to be deployed on real or simulated agents. This thesis opts to use a dynamics simulator called Isaac Gym to collect policy trajectory data. Developed by a collaboration between NVIDIA and the Robotics Science Lab at ETH Zurich, Isaac Gym 3 [38] is a simulation and learning environment built specifically for deep reinforcement learning. The software can simulate thousands of agents in parallel on GPU while simultaneously training networks on the same GPU. Since agents and networks are both contained in the same processing unit, Isaac Gym presents unparalleled speed in reinforcement learning training for physical systems. Table 2.1 details the PhysX environment parameters set for each environment explored within this work.

Hardware

All training for the cartpole system was completed using a NVIDIA 3090 GPU. Experiments for the Mini Cheetah system were completed on another system containing one NVIDIA A100 GPU. The larger A100 unit's storage was useful for simulating the increased complexity of the Mini Cheetah agent.

Table 2.1: The PhysX parameter values of the Isaac Gym physics simulating engine.

PhysX Parameter	Cartpole	Mini Cheetah
dt	0.0166	0.002
substeps	2	1
gravity $[\frac{m}{s^2}]$	$[0, 0, -9.81]$	$[0, 0, -9.81]$
num threads	4	10
solver type	1	1
num position iterations	4	4
num velocity iterations	0	0
contact offset $[m]$	0.01	0.01
rest offset $[m]$	0.001	0.0
bounce threshold velocity $[\frac{m}{s}]$	0.2	0.5
max depenetration velocity $[\frac{m}{s}]$	100.0	10.0
max gpu contact pairs	2^{20}	2^{23}
default buffer size multiplier	2.0	5
contact collection	0	2

Learning Algorithm and Architecture

The learning algorithm and architecture used for this thesis were primarily developed in [46] and open-sourced in the Legged Gym repository available on Github [9]. This repository implements a version of PPO which interacts directly with the Isaac Gym interface. Base versions of this repository boast the ability to learn deployable robotic policies in minutes due to parallel computing capabilities of GPUs combined with the distributed nature of the PPO algorithm. This thesis utilizes this parallel power to trains deep neural network control policies.

Leveraging this software stack and available hardware, $16,384$ (or 2^{14}) cartpole and Mini Cheetah agents can be simultaneously simulated on the 3090 and A100 systems respectively. These agents use the actor network to predict the best control action and the environment calculates rewards based on each agent's state. Each agent is

allowed to run for a specific amount of time unless its termination criteria is met. Deployment time and termination criteria are also system specific to be discussed in more detail.

Each system deployed in the Isaac Gym framework uses actor and critic networks which are randomly initialized at construction time. The networks deployed in this work take the form of feed-forward Multi Layer Perceptrons (MLPs). The number of hidden layers and units can be specified for the actor and critic independently. This work uses the Exponential Linear Unit (ELU) as a nonlinear activation function. Otherwise, the final hyper-parameter values used to train the cartpole and Mini Cheetah systems are available in table 2.2.

Table 2.2: The hyper-parameter values used for the PPO learning algorithm.

Hyper-Parameter	Cartpole	Mini Cheetah
seed	-1	-1
clip param	0.2	0.2
entropy coef	0.01	0.01
init noise std	1.0	1.0
value loss coef	1.0	1.0
use clipped value loss	True	True
num learning epochs	5	5
num mini batches	4	4
learning rate	$5e^{-4}$	$1e^{-3}$
schedule	adaptive	adaptive
gamma	0.99	0.99
lam	0.95	0.95
desired kl	0.01	0.01
max grad norm	1.0	1.0
num steps per env	12	24

Reinforcement Learning Specific Details

Training good policies is a function of both the networks themselves and the environments that agents are deployed into. There are a number of ways to improve training and decrease the sim-to-real gap that plagues simulated control systems when simulation trained policies are deployed on hardware. This thesis leverages the following methods.

For each system, domain randomization [DR] is used to craft more robust policies. Related work [50] has shown that DR helps craft policies which are more robust to model inaccuracies, can perform disturbance rejection, and can overcome unmodeled dynamics. In this work, observation noise is applied to each system at reasonable scales, initialization is randomized to increase exploration coverage, and instantaneous velocity disturbances are applied to the base of the Mini Cheetah system to simulate pushing. Specific domain randomization values are system specific and are available below.

In order to better monitor learning progress, experiment logging is completed using Weights and Biases [10]. This web-based experiment tracking software offers an interface to locally log user-specified reward, loss, and hardware data as well as other custom data. Typically, reinforcement learning is monitored by viewing reward trends to indicate how the agents are performing at the desired task. However, while reward graphs are certainly useful for debugging, they occasionally do not perfectly represent the behavior. Instead, the miscellaneous logging feature is used to record success rates for each agent directly tied to the desired behavior. The definition of a successful behavior and the thresholds used to record success are available in the specific system descriptions.

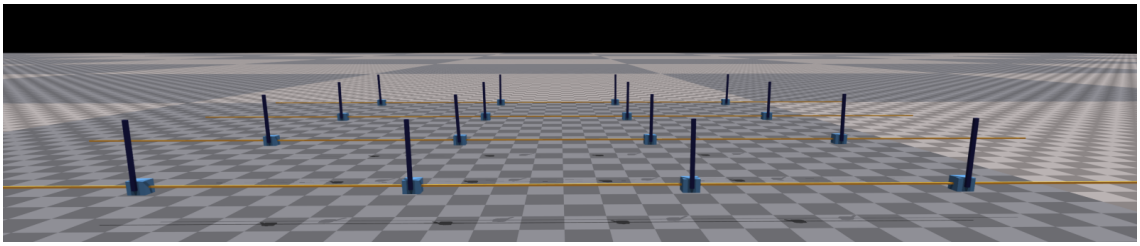

Figure 2-1: 16 cartpole agents simultaneously simulated in Isaac Gym and resting at their unstable vertical equilibria.

2.2 Control Platforms

2.2.1 Cartpole

General Description

The cartpole is a two degree of freedom [dof] system with a fixed base, prismatic cart actuator, and revolute pole joint shown in figure 2-1. This system was chosen as a test platform due to its relatively simple state and action spaces. Yet, the cartpole has nonlinear dynamics shown in equation 2.1 (with unit mass and length) according to [49]. The dynamics maps the pole angle θ, pole velocity $\dot{\theta}$, cart force u to the cart and pole accelerations $\ddot{x}$ and $\ddot{\theta}$ according to a scalar gravity g parameter.

$$\begin{bmatrix} cos(\theta) & 1 \\ 1 & cos(\theta) \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} -gsin(\theta) \\ u + \dot{\theta}^2 sin(\theta) \end{bmatrix} \tag{2.1}$$

Further, there are known model-based control laws for the cartpole available in [49]. Near its unstable vertical equilibrium, the cartpole can be stabilized using feedback gains tuned using Linear Quadratic Regulation (LQR). Outside of the vertical position, a tunable energy pumping controller with added PD cart regulation shown in equation 2.2 can swing the pole to vertical by regulating the energy of the system.

$$u = \frac{k_e}{2}\dot{\theta}^3 cos(\theta) + k_e \dot{\theta} cos^2(\theta) - k_e \dot{\theta} cos(\theta) - k_p x - k_d \dot{x} \tag{2.2}$$

Results for this paper demonstrate that this model-based understanding of the cartpole system is useful in improving policy training speed. The rest of the cartpole

section poses the baseline reinforcement learning problem such that an increment can be studied.

Swing Up and Stabilize

The goal for the cartpole is to apply forces at the cart's prismatic joint such that the pole swings up to its vertical position and is stabilized there at the center of the cart's range. The cart's actuation effort is limited to four Newtons to add realistic problem complexity. With constrained forces, it takes multiple pumps to swing the pole to vertical if the pole is initially at the bottom of its range.

Success for the swing up and stabilize behavior is recorded if the pole's position is within a threshold of its vertical position and the pole's velocity also within a small defined range when the agent's episode times out. The behavior fails if the cart's position exceeds three meters from the origin or if the success thresholds are not met at termination. An additional metric for success is centering the cart after the pole is swung to vertical. This 'safe' behavior is often implemented as an additive PD controller as seen in equation 2.2 to keep the cart away from the unsafe termination boundaries. Table 2.3 shows the ranges within which success is recorded for the cartpole system during operation.

Table 2.3: The success metrics, equations, and thresholds for measuring the swing up and stabilize behavior.

Success Metric	Equation	Threshold
Pole Vertical	$\lvert\theta\rvert < T_\theta$	$T_\theta = 0.2\pi[rad]$
Pole Slow	$\lvert\dot{\theta}\rvert < T_{\dot{\theta}}$	$T_{\dot{\theta}} = 0.3\pi[\frac{rad}{sec}]$
Pole Stable	Pole Vertical *and* Pole Slow	NA
Cart Center	Pole Stable *and* $\lvert x\rvert < T_x$	$T_x = 0.3[m]$

RL Problem Formulation

In order to train a policy to achieve this desired behavior, the problem must be well-specified for the reinforcement learning algorithm. In this case, the observations $\mathbf{o}_t$

for the cartpole system include five scalar values: x the cart's position, θ the pole's position, $\dot{x}$ the cart's velocity, $\dot{\theta}$ the pole's velocity, and a_{t-1} the cart's most recent action. These observations are stacked together into a matrix $\mathbf{O}_t$ of observation vectors $\mathbf{o}_t$ per agent shown in equation 2.3.

$$\mathbf{o}_t = [x \quad \theta \quad \dot{x} \quad \dot{\theta} \quad a_{t-1}] \tag{2.3}$$

It is important to normalize the observations to within the same order of magnitude of each other for better learning performance. Scaled values for cartpole observation are shown in table 2.4.

Table 2.4: The multiplicative scales used to normalize the observations for the cartpole system.

Observation	Symbol	Scale
Cart Position	x	0.33
Pole Position	θ	0.318
Cart Velocity	$\dot{x}$	0.224
Pole Velocity	$\dot{\theta}$	0.0652
Prior Action	a_{t-1}	1.0

The actor network must map observations to actions in order to maximize its value. Much like the observations are engineer-specified, the action space is as well. For the cartpole system, the actor network controls the forces applied at the cart's prismatic joint. As described in the desired behavior, this force is limited to $4Nm$. This force limit is also used as the action scaling applied to neural network control outputs. During training and deployment, physics is simulated at $120Hz$ and control forces are produced at $60Hz$.

RL also requires the specification of an objective function. For the cartpole system, The swing up and stabilize behavior is translated into a dense scalar reward function in order to communicate with the the reinforcement learning algorithm. The optimum value of the dense reward function should correspond with the desired behavior. Dense

reward terms for this system are weighted squared exponential functions centered around the origin of each degree of freedom. The linear combination of subterms is calculated in order to return a scalar reward value r_t for each agent at each timestep t shown in equation 2.4. Table 2.5 shows these sub rewards in more detail and the weights values set to achieve the behavior using baseline methods.

$$r_t = r_\theta + r_x + r_{\dot{\theta}} + r_a \tag{2.4}$$

Table 2.5: The reward terms, equations, and scales for incentivizing the swing up and stabilize behavior.

Reward Term	Equation	weighting	scaling
r_x	$w_x e^{-\frac{x^2}{\sigma_x}}$	$w_x = 3.0$	$\sigma_x = 3.0$
r_θ	$w_\theta e^{-\frac{\theta^2}{\sigma_\theta}}$	$w_\theta = 7.0$	$\sigma_\theta = \pi$
$r_{\dot{\theta}}$	$w_{\dot{\theta}} e^{-\frac{\dot{\theta}^2}{\sigma_{\dot{\theta}}}}$	$w_{\dot{\theta}} = 0.01$	$\sigma_{\dot{\theta}} = 14.0$
r_a	$w_{a_t} e^{-\frac{a_t^2}{\sigma_{a_t}}}$	$w_{a_t} = 1e-5$	$\sigma_{a_t} = 4.0$

Domain randomization is included in the cartpole system as well. For the cartpole system, this domain randomization takes the form of observation data randomization and random initialization. Observation randomization uses additive zero-mean uniform noise on the inputs to the actor and critic networks. Scales for this noise is shown in table 2.6. Further, random initialization is used for the cartpole system to boost exploration. Exploration helps the critic network find high value states. For this system, initial states are randomized within specific ranges of each degree of freedom which are shown in table 2.7.

2.2.2 Mini Cheetah

General Description

The Biomimetic Robotics Lab has designed the Mini Cheetah legged robotic platform [31] seen in figure 2-2 as an all-terrain quadruped legged platform with 12 actuated

Table 2.6: The scaling applied to the uniform noise $\in [-1, 1]$ added to the cartpole observations.

Observation	Noise Scale
Cart Position	0.001
Pole Position	0.001
Cart Velocity	0.01
Pole Velocity	0.01
Prior Action	0.0

Table 2.7: The ranges for randomly initializing each degree of freedom of the cartpole system.

Observation	Lower Bound	Upper Bound
Cart Position x	-2.5	2.5
Pole Position θ	$-\pi$	π
Cart Velocity $\dot{x}$	-0.1	0.1
Pole Velocity $\dot{\theta}$	-0.1	0.1

joints and a 6 degree of freedom floating base. The hardware, firmware, and software design for this system have enabled it to complete highly dynamic maneuvers in a breadth of environments. Using model-based control, the Mini Cheetah system is able to swiftly traverse flat ground with disturbances [13], consistently complete dynamic maneuvers and land safely [19, 29], and, by adding cameras, perform higher-level tasks in even more complex environments [32].

In order to understand the complexity that an end-to-end reinforcement learning model must approximate, it is useful to present one model-based architecture for the Mini Cheetah from [33]. This control architecture both informs model-free design decisions and offered part of the inspiration for the increments explored within this thesis.

Figure 2-2: The Mini Cheetah legged robotic platform.

A Model-Based Optimal Control Architecture

Locomotion controllers for the Mini Cheetah use a model-based control hierarchy. One example built for locomotion can be seen in the diagram in figure 2 in [33]. First, behavior commands (planar desired velocity, yaw rate, and gait frequency) are passed into a Model Predictive Control (MPC) formulation which is solved to produce feed-forward optimal trajectories at 40Hz. Trajectories produced by MPC include center of mass locations ${}^{W}\mathbf{p}^{B}[n]$, body orientations ${}^{W}\boldsymbol{\Theta}^{B}[n]$, footstep locations ${}^{B}\mathbf{r}_{i}^{foot}[n]$, and desired ground reaction forces ${}^{W}\mathbf{F}_{i}^{B}[n]$ for each discrete time step n over the next N time steps. In order to tractably craft optimal trajectories at this frequency, MPC uses an approximation of the Mini Cheetah called the single rigid body model (SRBM) and makes assumptions the body's tilt and angular velocity. The SRBM moves through space by making contact with the ground. The dynamics of the SRBM are shown in equations 2.5 and 2.6. Once the solver converges to an optimal trajectory, these control decisions are passed to the next control module in the hierarchy.

$$m\ddot{\mathbf{p}} = (\sum_{i}^{feet} {}^{W}\mathbf{F}_{i}^{B}) - \mathbf{g} \tag{2.5}$$

$$\frac{d}{dt}(\boldsymbol{I}\boldsymbol{\omega}) = \sum_{i}^{feet} {}^{B}\mathbf{r}_{i}^{foot} \times \mathbf{F}_{i} \tag{2.6}$$

Operating at 500Hz, the Mini Cheetah's Whole Body (Impulse) Controller (WBC) tracks the trajectory produced by the MPC module. Specifically, the WBC module outputs desired joint positions and velocities with feed-forward joint torques that regulate the system's joints to best match the MPC optimal trajectory. In order to complete the task, the WBC reasons about a higher-order model of the Mini Cheetah system including the joint angles $\mathbf{q}$ and velocities $\dot{\mathbf{q}}$.

WBC tracks desired body poses and ground reaction forces at the next time step by using the full-order dynamics of the Mini Cheetah. This full-order model uses the floating body manipulator equation seen in equation 2.7, the forward kinematics $\mathbf{FK}(\mathbf{q})$, and the contact jacobian $\mathbf{J}(\mathbf{q}, \dot{\mathbf{q}})$. However, including more degrees of freedom leads to null solution spaces. This means that the problem is under-constrained such that many (or, infinite) optimal solutions may exist. This problem is addressed in [33] by prioritizing tasks for the WBC to complete. Once WBC has settled on a solution, it passes its joint target information to joint-specific PD controllers. Joint regulating controllers operate at 40kHz to output desired currents to be regulated by even higher frequency current controllers.

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \boldsymbol{\tau}_{g}(\mathbf{q}) + \sum_{i}^{feet} \mathbf{J}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{F}_{i} \tag{2.7}$$

Similar to the cartpole system, proper understanding of the engineering process informs both the baseline and research increment of the model-free formulation of the Mini Cheetah system.

Jump to Specified Height

In this work, the desired behavior for the Mini Cheetah system is to repeatedly jump to a commanded height and land safely. Commanded heights are randomly selected in a range from $50cm$ and $90cm$. Due to the added degrees of freedom of the Mini Cheetah system, further complexity must be added to ensure safety when deploying

the jumping controller. While many control policies may successfully jump to the commanded height, fewer solutions do this in a human-desirable and 'safe' way. To accomplish better looking jumps, the Mini Cheetah should remain upright, have little linear or angular base velocity, and not flail its legs wildly in the air.

The Mini Cheetah system often fails to complete the behavior during training - termination for the Mini Cheetah occurs if any non-foot or shank link touches the ground. Unsafe (terminated) policies are considered unsuccessful even if the jump height is achieved.

RL Problem Formulation

The reinforcement learning problem formulation for this problem is similar to that of the cartpole system. The core RL problem requires specification of observations, the action space, and the objective function. The baseline observations $\mathbf{o}_t$ for the Mini Cheetah system include the base height of the robot z_t, the base linear and angular velocities $\mathbf{V}_t$ and $\mathbf{\Omega}_t$, the projected gravity vector $\mathbf{g}_t$ (used for orientation), the commanded jump height z_t^{des}, the joint positions and velocities $\mathbf{q}_t$ and $\dot{\mathbf{q}}_t$, the nominal joint positions $\mathbf{q}_t^{nom}$, a history of the last two control actions at each joint $\mathbf{a}_{t-1,t-2}$, the average actions $\mathbf{a}_t^{ave}$ which are used as a low pass filter, and the $sin(\phi)$ and $cos(\phi)$ of a phase variable ϕ. These observations about each parallel agents' states are stacked into an observation matrix $\mathbf{O}_t$ defined in equation 2.8.

$$\mathbf{O}_t = [z_t \quad \mathbf{V}_t \quad \mathbf{\Omega}_t \quad \mathbf{g}_t \quad z_t^{des} \quad \mathbf{q}_t \quad \dot{\mathbf{q}}_t \quad \mathbf{q}_t^{nom} \quad \mathbf{a}_{t-1} \quad \mathbf{a}_{t-2} \quad \mathbf{a}_t^{ave} \quad sin(\phi) \quad cos(\phi)] \tag{2.8}$$

Before being sent to the actor and critic networks, each observation has zero-mean uniform noise added to it and is normalized. The noise scales and normalization factors are also included in table 2.8.

The action space and control frequency for the Mini Cheetah must also be specified. Due to the complexity of simulating contact, the simulation environment steps the agents forward at $500Hz$. To avoid abuse of the physics environment, control

Table 2.8: The observation normalization scales and noise scales for the Mini Cheetah simulated agent.

Observation	Symbol	Dimension	Scale	Noise
base height	z_t	1	0.33	0.1
body linear velocity	$\mathbf{V}_t$	3	0.33	0.1
body angular velocity	$\mathbf{\Omega}_t$	3	0.058	0.2
projected gravity	$\mathbf{g}_t$	3	1.0	0.05
commanded jump height	z_t^{cmd}	1	1.0	0.0
joint positions	$\mathbf{q}_t$	12	0.318	0.001
joint velocities	$\dot{\mathbf{q}}_t$	12	0.01	0.01
default joint positions	$\mathbf{q}_t^{nom}$	12	0.318	0.0
prior actions history	$\mathbf{a}_{t-1,t-2}$	24	0.25	0.0
action average	$\mathbf{a}_t^{ave}$	12	0.25	0.0
phase	$sin(\phi)$, $cos(\phi)$	2	1.0	0.0

actions are decimated such that the control policy operates at $100Hz$. The action network does not output torques directly. Instead, the learned control policy outputs low pass filtered joint targets q_t^{des} (equivalently a_t) relative to the nominal joint positions q_t^{nom} for the Mini Cheetah system .

In order to achieve faster - yet, possibly sub-optimal - learning for the Mini Cheetah jump behavior, an additional feed-forward sketch of the desired jumping motion is used to set nominal joint positions. The nominal positions use a very simple two-state finite state machine. If any foot is in contact with the ground, the nominal positions follow through a sinusoidal jumping procedure tracked by incrementing a phase variable ϕ_{jump}. Otherwise, the joint position are set back to their default values and ϕ_{jump} is reset to zero. This state machine and the default position target are shown in figure 2-3. Note that the changing default reference positions are always available in the observations $\mathbf{o}_t$ for fair comparison.

Once the default positions are updated and the actor policy chooses desired joint targets $\mathbf{a}_t$, these targets are low pass filtered to avoid vibrating joints and non-physical

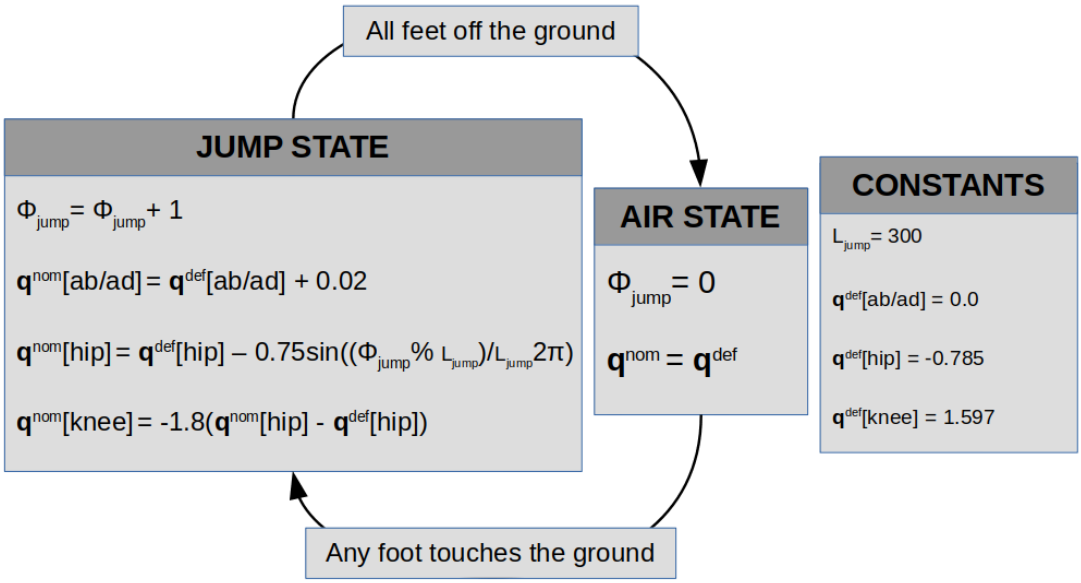

Figure 2-3: The finite state machine which serves nominal joint positions for the Mini Cheetah simulated plaform.

motions. [15] shows that low pass filtering actor outputs successfully achieves these effects. In this work, the average action $\mathbf{a}_{t-1}^{ave}$ for each joint is updated using the new action $\mathbf{a}_t$ via the exponential average decay function shown in equation 2.9 to produce $\mathbf{a}_t^{ave}$. The new resulting average action $\mathbf{a}_t^{ave}$ is passed to a joint regulating control law shown in figure 2.10 with tuned gains shown in table 2.9.

$$\mathbf{a}_t^{ave} = \alpha \mathbf{a}_t + (1 - \alpha)\mathbf{a}_{t-1}^{ave} \tag{2.9}$$

$$\tau^i = k_p^i(\sigma_a a_t^i + q_t^{nom,i} - q_t^i) - k_d^i \dot{q}_t^i \tag{2.10}$$

Posing the desired behavior as a formal objective function is the last required step for the reinforcement learning pipeline. Dense reward functions are also used for this system. Each sub-reward takes the form of a squared exponential $e^{-(\frac{x}{\sigma_x})^2}$ or square x^2 of some system state or action. The result of each reward function is scaled using a human-tuned weighting factor. Rewards are calculated during each control step and summed together according to equation 2.11. The full table of reward terms and functions are available in table 2.10.

Table 2.9: Proportional and Derivative gains used for model-based joint PD control for each type of Mini Cheetah actuated joint.

Joint Type	k_p	k_d
Ab/Ad	20	0.5
Hip	20	0.5
Knee	20	0.5
Action Parameters	**Symbol**	**Value**
action scale	σ_a	0.25
LPF	α	0.85

$$r_t = r_{term} + r_{linvel} + r_{angvel} + r_{\boldsymbol{\Theta}} + r_{\boldsymbol{\tau}} + r_{noslip} + r_a + r_{a^2} + r_z + r_q \tag{2.11}$$

Success rates are recorded during training to quantitatively validate the learned behavior. The added complexity of the system requires more work to validate the behavior than simply checking terminal states. At any point during a training episode, a successful jump switch is toggled on if the Mini Cheetah's base position is measured within a small threshold of the commanded jump height at the apex of its jump. The success threshold and formal definition of a successful jump are shown in table 2.11. Orientation and velocity measurements also require greater complexity to quantitatively measure success. For these, a history of the last 100 measured values is kept. At episode time-out, the average of these values is taken and compared to the success thresholds (also in table 2.11). If termination occurs before time-out, all success rates for an agent are automatically nullified.

Domain randomization is included into the Mini Cheetah RL pipeline as well. Here, DR takes the form of randomized ground friction coefficient per agent episode and push disturbances simulated by velocity impulses to the base of the robot. Further, the degrees of freedom and base height were randomly initialized. Each randomization technique is detailed in table 2.12.

Table 2.10: The reward terms, equations, and scales for incentivizing the accurate jump height behavior.

Reward Term	Equation	weighting	scaling
r_{term}	$^{(1)}w_{term}$	-1	NA
r_{linvel}	$w_{linvel} \sum_{i=0}^{1} e^{-(\frac{\mathbf{v}_i^2}{\sigma_{linvel}})}$	5.0	0.25
r_{angvel}	$w_{angvel} \sum_{i=0}^{2} e^{-(\frac{\mathbf{\Omega}_i^2}{\sigma_{angvel}})}$	5.0	0.25
$r_{\mathbf{\Theta}}$	$w_{\mathbf{\Theta}} \sum_{i=0}^{2} e^{-(\frac{\mathbf{\Theta}_i^2}{\sigma_{\mathbf{\Theta}}})}$	10.0	0.25
$r_{\boldsymbol{\tau}}$	$w_{\boldsymbol{\tau}} \sum_{i=0}^{11} \boldsymbol{\tau}_i^2$	-5.e-7	NA
r_{noslip}	$^{(2)}w_{noslip} \sum_{i=0}^{3} e^{-(\frac{\|\|r_i(t)-r_i(t-1)\|\|_2^2}{\sigma_{noslip}})}$	0.5	0.25
r_a	$w_a \sum_{i=0}^{11} (\frac{(\mathbf{a}_i(t)-\mathbf{a}_i(t-1))}{dt})^2$	-0.001	NA
r_{a^2}	$w_a \sum_{i=0}^{11} (\frac{(\mathbf{a}_i(t)-2\mathbf{a}_i(t-1)+2\mathbf{a}_i(t-2))}{dt^2})^2$	-0.001	NA
r_{jump}	$w_{jump} e^{-(\frac{(z-z^{cmd})^2}{\sigma_{jump}})}$	10.0	0.25
r_q	$w \sum_{i=0}^{11} e^{-(\frac{(q_i-q_i^{nom})^2}{\sigma_q})}$	2.0	0.318

Notes	
(1)	r_{term} assigned at agent termination
(2)	r_{slip} only active for feet on the ground

Table 2.11: The success metrics, equations, and thresholds for measuring the jump to specified height behavior.

Success Metric	Equation	Threshold
height reached	$\|z - z^{cmd}\| < T_z$	$T_z = 5.[cm]$
jump apex	$\|\mathbf{V}_z\| < T_{\mathbf{V}_z}$	$T_{\mathbf{V}_z} = 0.1[\frac{m}{s}]$
accurate jump	height reached *and* jump apex	NA
upright	$ave(\|\|g(t:t-100)\|\|_2^2) < T_g$	$T_g = 0.2$
linear velocity	$ave(\|\|\mathbf{V}(t:t-100)\|\|_2^2) < T_{\mathbf{V}}$	$T_{\boldsymbol{V}} = 0.25[\frac{m}{s}]$
angular velocity	$ave(\|\|\boldsymbol{\Omega}(t:t-100)\|\|_2^2) < T_{\boldsymbol{\Omega}}$	$T_{\boldsymbol{\Omega}} = 0.25\pi[\frac{rad}{sec}]$

Table 2.12: Domain Randomization ranges for the Mini Cheetah system during training.

Variable	Lower Bound	Upper Bound
Base Height $z[m]$	0.39	$z^{cmd} - T_z$
Base Velocity $\mathbf{V}_z[\frac{m}{s}]$	-0.05	0.05
Ab/Ad Positions $q[rad]$	$q^{def}_{Ab/Ad} - 0.05$	$q^{def}_{Ab/Ad} + 0.05$
Hip Positions $q[rad]$	$q^{def}_{Hip} - 0.85$	$q^{def}_{Hip} + 0.65$
Knee Positions $q[rad]$	$q^{def}_{Knee} - 1.45$	$q^{def}_{Knee} + 1.72$
Ground Friction μ	0.75	1.05
Base Mass $m[kg]$	2.0	2.0
Push Disturbance $\mathbf{V}^{disturb}_{x,y}$	once per $2[s]$	$1.0[\frac{m}{s}]$

Chapter 3

Augmentations

3.1 Goals and Hypothesis

The primary factor limiting model-free engineering feedback loops is learning speed. This work measures learning speed as the time it takes for a network to converge to a specific success rate for a behavior. This definition is used since any engineering iteration and tuning cannot be completed until the behavior stabilizes.

Another efficiency metric studied in this work is network capacity. Network capacity measures the number of tunable weights and biases in the actor network used to represent a control policy. The file size required to store the actor network can also measure network capacity. In a resource constrained environment, onboard microcontrollers may have limited storage, memory, or operation speed. Thus, using a smaller neural network may enable performance in more resource constrained environments.

The hypothesis in section 1.2 suggests that removing redundant work from neural networks can increase training efficiency. If model-free control is attempting to use data to recreate the same models as model-based control and these models are sufficient to craft working control policies, then redundant work is being completed every time a neural network is re-initialized. This work attempts to pre-process these repeated computations which are indicated by model-based control to be useful. These control and dynamics terms are then added to the observations of the neural network.

This way, the network may learn to leverage its observations rather than re-model physics or nonlinear controllers every time it is re-initialized.

3.2 Implementation Details

Augmenting a model-free pipeline requires a four step process: (1) targeting augmentations, (2) implementing the augmentations, (3) scaling the resultant values, and (4) incorporating them into the observations. Only the first step is robotic system dependent. Otherwise, the remaining steps use the same process regardless of the system being studied.

3.2.1 Cartpole

The selection of augmentations for the Cartpole system considers the dynamics of the system presented in equation 2.1 and known controllers for the system shown in equation 2.2. From these equations, nonlinear sub-terms are collected and organized. The extracted augmentations are shown in equations 3.1 and 3.2. During the learning process, system states s_t are received from the environment. Baseline RL adds uniform random noise N to these values in order to complete domain randomization during training. Augmentations for the neural networks are functions of these noisy values $s_t + N$ rather than the true system states to ensure that no privileged information is received by including augmentations. Next, the augmented values are normalized according to the values shown in table 3.1 and appended to the observation matrix $\mathbf{O}_t$.

$$A_{dynamics} = [cos(\theta) \quad sin(\theta) \quad \dot{\theta}^2] \tag{3.1}$$

$$A_{controls} = [cos(\theta) \quad cos^2(\theta) \quad \dot{\theta}^3] \tag{3.2}$$

Table 3.1: The multiplicative scales used to normalize the observations for the cartpole system.

Augmentation	Scale
$sin(\theta)$	1.0
$cos(\theta)$	1.0
$cos^2(\theta)$	1.0
$\dot{\theta}$	0.0045
$\dot{\theta}^2$	0.0002

3.2.2 Mini Cheetah Jump

Existing model-based research also inspires the choice of augmentations for the Mini Cheetah system. Related work [30] suggests that computing the end-effector positions ${}^{B}\mathbf{r}_t^i$ of each leg i was effective in learning control policies for the Mini Cheetah system. Leveraging this result, the nonlinear joint space terms $cos(\mathbf{q}_t)$ and $sin(\mathbf{q}_t)$ of the forward kinematic mapping $\mathbf{FK}(\mathbf{q}_t)$ and its a two step history of the resultant end-effector (feet) positions ${}^{B}\mathbf{r}_t^i$ and ${}^{B}\mathbf{r}_{t-1}^i$ are used as augmentations (see equation 3.3). The Single Rigid Body approximation shown in equations 2.5 and 2.6 and used for MPC [33] indicates that the Mini Cheetah may only influence its motion via contact with the ground. With this information, the contact jacobian values $\mathbf{J}(\mathbf{q}_t, \dot{\mathbf{q}}_t)$ and ground reaction forces $\mathbf{F}_t^i$ on each foot are also pre-processed to be used as augmentations as shown in equation 3.4. Like the cartpole, these values are constructed based on the noisy state values $\mathbf{s}_t + N_t$ collected from the environment, normalized, and added to the observations matrix for processing. The full list of augmentations and normalization scales used for training the Mini Cheetah system are shown in table 3.2.

$$A_{forward\_kinematics} = [sin(\mathbf{q}_t) \quad cos(\mathbf{q}_t) \quad {}^{B}\mathbf{r}_t^{feet} \quad {}^{B}\mathbf{r}_{t-1}^{feet}] \tag{3.3}$$

$$A_{forward\_kinematics} = [sin(\mathbf{q}_t) \quad cos(\mathbf{q}_t) \quad ||{}^{B}\mathbf{F}_t||_2^2 \quad \mathbf{J}(\mathbf{q}_t, \dot{\mathbf{q}}_t)] \tag{3.4}$$

Table 3.2: The pre-processed observation augmentations and their normalization scales for the Mini Cheetah system.

Augmentation	Dimension	Scale
$sin(\mathbf{q}_t)$	12	1.0
$cos(\mathbf{q}_t)$	12	1.0
${}^{B}\mathbf{r}_t^{feet}$	3×4	0.609
${}^{B}\mathbf{r}_{t-1}^{feet}$	3×4	0.609
$\|\|{}^{B}\mathbf{F}_t\|\|_2^2$	4	$1.e-3$
$\mathbf{J}(\mathbf{q}_t, \dot{\mathbf{q}}_t)$	$(3 \times 3) \times 4$	1.0

3.3 Results

3.3.1 Learning Speed

Cartpole

To explore the effects of augmentations on model-free learning speed, an A-B study was conducted. During this experiment, augmentations were either included or removed entirely. All other environment variables, reward scaling, and learning parameters were kept constant except for random seed selection enabling random network initialization. The results of this study are shown in figure 3-1. The average augmented policy exceeded the 95% success rate for the pole stable behavior after approximately 250 learning iterations. It took the naively trained policy about 1350 iterations to reach the same rate. These results correspond to an 81% reduction in required learning time.

Further, the spread between earliest and latest augmented policy to achieve 99% success was much smaller than the same defined spread for policies trained without augmentations. The earliest augmented policy to reach the success threshold did so at 100 iterations while the last augmented policy reached it in 350 learning iterations. For naively trained policies, the earliest to achieve 99% success did so in 800 itera-

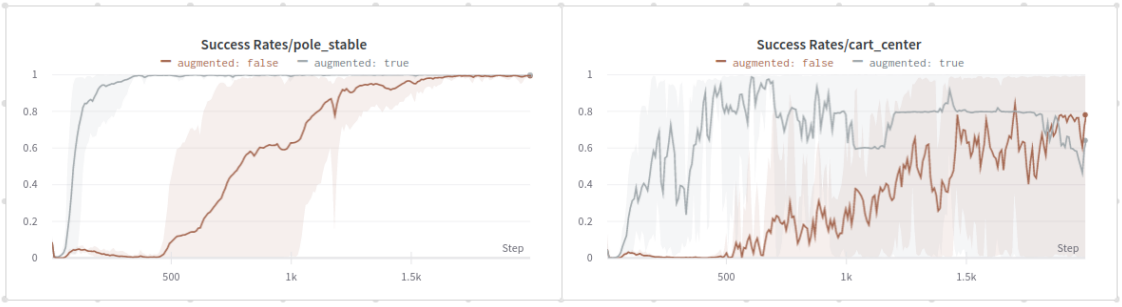

Figure 3-1: The results of training the cartpole both with (grey) and without (brown) dynamics and controls augmentations. There is a significant increase (81% faster) in learning speed when augmentations are introduced to the training policy.

tions while the last did at 1600. These results indicate that including augmentations decreases learning variance overall.

Mini Cheetah

The same study structure is used to determine the effect of augmenting the observations on learning time on the Mini Cheetah system. The results of ten training runs completed with and ten training runs without augmentations are shown in figure 3-2. The policies are attempting to learn the commanded jump height behavior as described in section 2.2.2.

Experiments show that policies trained with model-based augmentations converged at 3800 learning iterations on average. The steady-state jump height success is greater than 85%. Naively trained policies converge to a 45% success rate at around 4100 iterations. Longer training times do not significantly improve either set of performances. During this experiment - comparing learning speeds - the average augmented policy reaches the naive success rate (45%) at 2900 iterations - reducing learning time by 30%. The secondary success rates required for safe deployment are all satisfied at approximately the same rates by both the augmented and naively trained policies.

Unfortunately, since the naive policy could not reach the same performance as the augmented policies, time to convergence could not be compared. However, average policy performance is shown to be 1.75 times better by including augmentations into

Figure 3-2: The results of training the Mini Cheetah simulated agent. Two average trends are shown with a central average line per iteration and a shaded region indicating the best and worst performing learning runs. The dark green curve represents learning runs with augmentations and the light blue line represents naively trained policy performance. It is not possible to compare convergence rates between the graphs since final performance differed, but augmented policy performance at jumping accurately was 1.75 times better the naively trained policies.

the training pipeline. This result is consistent with the [30] paper which showed that policy performance is improved by estimating feet positions and ground reaction forces for the policy.

3.3.2 Augmentation Knockouts

Cartpole

In order to determine which augmentations caused the greatest increase in learning efficiency, the augmentations for the cartpole system were separated into two groups: dynamics and control. Dynamics augmentations included the nonlinear terms from the cartpole dynamics equations and controls augmentations were brought in from the Energy Pumping control law shown in equation 2.2. Two sets of policies were trained using either set of augmentations.

The results of these experiments are shown in figure 3-3. This experiment demon-

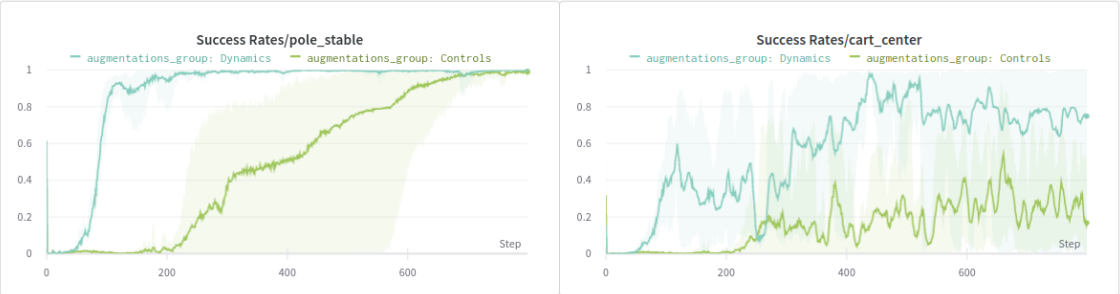

Figure 3-3: Cartpole experiment results training with two groups of augmentations: Dynamics (blue) and Controls (green). Here it is shown that dynamics augmentations increase learning speed by a full order of magnitude while controls augmentations only double the typical convergence rate.

strates that dynamics augmentations are the primary source of learning efficiency benefits for the cartpole system. The average run using dynamics augmentations converged in 209 learning iterations whereas controls augmentations required 669 iterations to converge. Both sets of augmentations make learning occur quicker than the 1350 average iterations required for naively augmented policies to converge.

Mini Cheetah

The grouping strategy is also used to determine which augmentations made the biggest impact on policy training for the Mini Cheetah jump behavior. For this system, there are two groups of augmentations to test: forward kinematics and ground reaction forces. The augmentations in the forward kinematics group are shown in equation 3.3 and the ground reaction forces augmentations are shown in equation 3.4. Similar to the cartpole, each augmentation group is individually tested to determine its effect on policy learning performance.

The results of augmentation knockouts on the Mini Cheetah system are shown in figure 3-4. Here, policy performance improves most when forward kinematics augmentations are included during policy training. Performance levels off at approximately 80% jump height success. Including only ground reaction force augmentations also improved policy performance reaching about 55% success on the task. Naively trained policies converge at 45% success on average indicating benefits by including either set

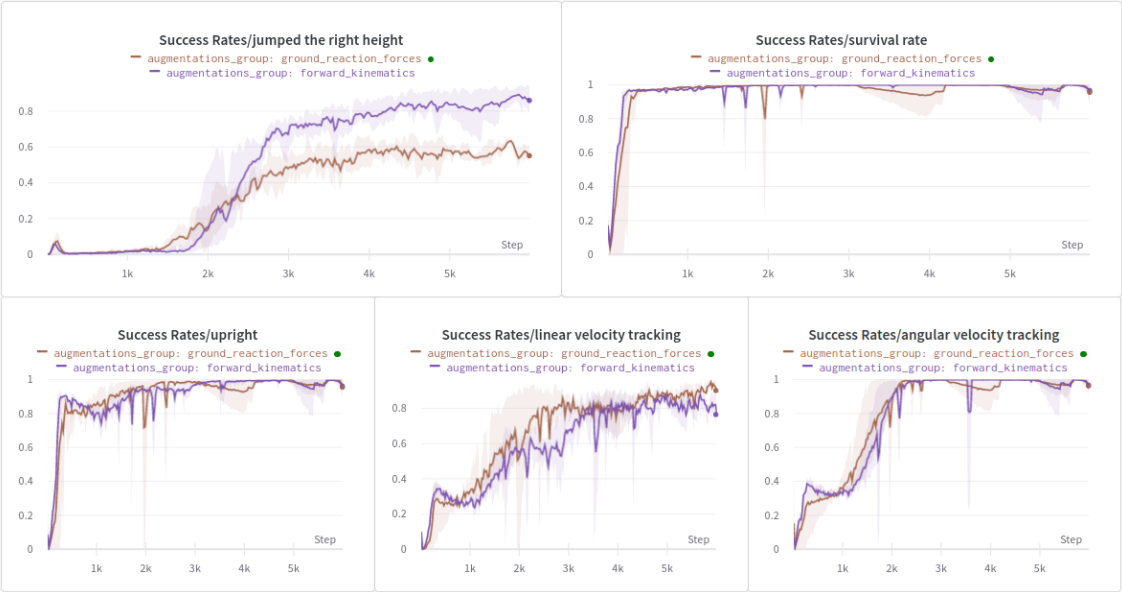

Figure 3-4: This Mini Cheetah experiment presents the results of training jumping policies with only forward kinematics (purple) and ground reaction forces (brown) augmentations. Naive policy performance levels off at 45%. Forward kinematics augmentations increase policy performance to 80% while ground reaction force augmentations increase performance to only 55%.

of augmentations in general.

3.3.3 Minimal Network Capacity

Cartpole

Because both naive and augmented policies were able to successfully swing up and stabilize the cartpole, it is possible to compare the network capacity reductions that come with extracting out work from the actor network. Note that only the actor network is typically deployed onboard a micro-controller, so there is less reason to reduce the size of the critic network. This experiment attempts to find the smallest network according to two metrics: storage capacity and total computations required to make a control inference. Storage capacity is measured by inspecting the file size of the '.pt' file which stores neural network parameters during training. The total number of network computations is computed according to equation 3.5 which accounts for weights, biases, and nonlinear activation functions. $C(n_i, n_u, n_l, n_o)$ is parameterized

by the size of the input observation dimension n_i, the number of hidden layers n_l, the number of hidden units per layer n_u, and the output action space dimension n_o. Successfully learning the swing up and stabilize behavior is defined as policy convergence above 95% pole stability success within 5000 learning iterations. Experimental results to determine the smallest network capable of learning the behavior are available in table 3.3.

$$C(n_i, n_u, n_l, n_o) = n_i * n_u + 2 * n_u + (n_l - 1)(n_u^2 + 2 * n_u) + n_u * n_o \tag{3.5}$$

Table 3.3: This table shows the experiment attmepting to find the smallest networks which could successfully swing up and stabilize the cartpole system within 5000 learning iterations. The smallest naive observations network required 3 hidden layers with 16 units per layer whereas the smallest augmented network only required 1 hidden layer with 14 units in that layer.

Status	Size	Success at X Iterations	network computations
Naive	3×128	1142	34304
Naive	3×32	1572	2432
Naive	$\mathbf{3 \times 16}$	**2284**	**704**
Naive	3×8	FAIL	224
Naive	2×64	3768	4736
Naive	2×32	4754	1344
Naive	2×16	FAIL	416
Augmented	3×128	120	34544
Augmented	2×32	497	1504
Augmented	1×32	756	416
Augmented	1×16	229	208
Augmented	1×8	FAIL	104
Augmented	1×12	FAIL	130
Augmented	$\mathbf{1 \times 14}$	**1266**	**156**

As shown in the table, the smallest successful augmented policy was trained using an actor network with 1 hidden layer and 14 network units. This network model is stored in a $427.8kB$ file and requires 182 network computations to evaluate. The smallest naively trained policy required 2 hidden layers with 32 units each to learn the behavior. Using $435.7kB$ of storage space, the network requires 1344 computations per network forward pass. The file sizes account for the additional network weights connected to augmentations in the input layer. Observation augmentations reduced storage requirements by only 1.8%, but total network computations were reduced by an order of magnitude.

Comparing learning speeds on the most resource restricted actor networks, the augmented policy converged to $> 95\%$ success in 1283 learning iterations whereas the smallest un-augmented policy required 4431 learning iterations to reach convergence. Compared to the resource-rich training environments, the full-sized naively trained average network shown in 3.3.1 required 1350 learning iterations to converge.

Chapter 4

Coupled Rewards

4.1 Goals and Hypothesis

The hypothesis in section 1.2 states that incorporating model-based optimal control can offer the best of both model-based and model-free worlds. Section 1.4.2 proposes that adding a task priority to the objective function of the learning pipeline can avoid local minima introduced by dense rewards while adding a physically-intuitive tuning parameter for the engineer to use. This section explore the effects of adding such a structure to the cartpole system's deep reinforcement learning pipeline.

Section 1.4.2 states that a hierarchy between of two reward terms can be constructed using multiplication. The dense reward functions in this thesis take the form of weighted squared exponential functions $r_x e^{-(\frac{x^2}{\sigma_x})}$ where r_x is the weighting of system variable x and σ_x is a human-tuned scaling factor to ensure that meaningful gradient information is available for the full domain of system variable x. When adding together unit weight and unit scale sub-reward squared exponential functions of system variables x and y, the resulting reward landscape is shown in the red surface in figure 1-2.

However, behaviors exist such that optimizing y does not matter unless x is within a particular range. In this case, the reward landscape should ignore y until x is satisfied to avoid local minima. This approach is related to [12, 33] which uses task prioritization specify a super and sub task. First, a super-task is solved before sub-tasks

are considered in the null-space the first solution. Back in model-free environments, turning off the reward signal is possible by simply using an if-statement. However, binary reward activation often leads to jerky solutions and unstable learning curves. It is best to use smooth reward functions like the squared exponential. Further, squared exponential can act like smooth if-statements using their σ parameter. Coupling a super-task x and sub-task y can be done using the following reward composition $e^{-(\frac{x^2}{\sigma_{switch}})}e^{-(\frac{Y^2}{\sigma_y})}$ where σ_{switch} is the activation region in which x is satisfied and y can be considered. The resulting reward landscape for the coupled reward is shown in the green surface of figure 1-2.

4.2 Implementation Details

For the cartpole system, the swing up and stabilize behavior can be posed as a super and sub task priority. Here, centering the cart only matters once the pole is stable at its vertical position. In fact, attempting to center the cart before the pole is at vertical actively impedes the swing up behavior causing a local minima. Relating to the proposed solution structure, if x represents the pole's position and y represents the cart's position, then coupling their squared exponential functions will only incentivize cart centering once the pole is within the vertical threshold.

The implementation of coupled rewards for the cartpole required two steps: multiplying the cart's reward function by the pole activation function and tuning the activation parameter. Table 4.1 shows the updated reward functions including task prioritization. Besides the rewards changes, all other baseline values were maintained during experimentation.

4.3 Results

4.3.1 Tuning the Pole-Cart Coupling Parameter

The first experiment with the coupled rewards idea studies the effects of tuning the sub-activation space σ_{switch} for the pole-cart coupling. The hypothesis in 4.1 states

Table 4.1: The cartpole reward terms, equations, and scales including the coupled reward increment.

Reward Term	Equation	weighting	scaling	activation scaling
$\mathbf{r_x^{coupled}}$	$w_x e^{-\frac{\theta^2}{\sigma_{switch}}} e^{-\frac{x^2}{\sigma_x}}$	$w_x = 3.0$	$\sigma_x = 3.0$	$\sigma_{switch} = 4\pi$
r_θ	$w_\theta e^{-\frac{\theta^2}{\sigma_\theta}}$	$w_\theta = 7.0$	$\sigma_\theta = \pi$	None
$r_{\dot{\theta}}$	$w_{\dot{\theta}} e^{-\frac{\dot{\theta}^2}{\sigma_{\dot{\theta}}}}$	$w_{\dot{\theta}} = 0.01$	$\sigma_{\dot{\theta}} = 14.0$	None
r_a	$w_{a_t} e^{-\frac{a_t^2}{\sigma_{a_t}}}$	$w_{a_t} = 1e-5$	$\sigma_{a_t} = 4.0$	None

that this parameter should be physically meaningful - turning off the cart reward signal when outside of the pole stabilizing region. However, empirical testing with this parameter behaved dramatically different. Results from the parameter tuning sweep shown in figure 4-1 demonstrate that physically intuitive values for σ_{switch} led to dramatic policy degradation by nearly 20% performance success at convergence. Further, time to convergence only barely decreased. Further tuning demonstrated that the best performing σ_{switch} for the reward coupling was 4π. This coupling value led to the earliest learning with least degradation in converged policy performance.

4.3.2 Sample Efficiency Experiments

Using the best available pole-cart activation coupling $\sigma_{switch} = 4\pi$, learning speed improvements did not occur as shown in figure 4-2. In fact, policy performance degraded by about 9% on average. However, coupled rewards did successfully avoid the local minima during training. For naively trained cartpole policies, an intermediate behavior is often learned in which the pole spins wildly while keeping the cart relatively centered. This local minima is demonstrated by the plateau in the gray learning curve of the same figure.

While the pole flailing local minima is indeed removed, the final reward trend explains the drop in pole stability performance. The reward termination graph returns zero reward when the learned policy never exceeds the termination criteria and -1 if the agent is reset before time-out. Possibly due to the increased complexity of the

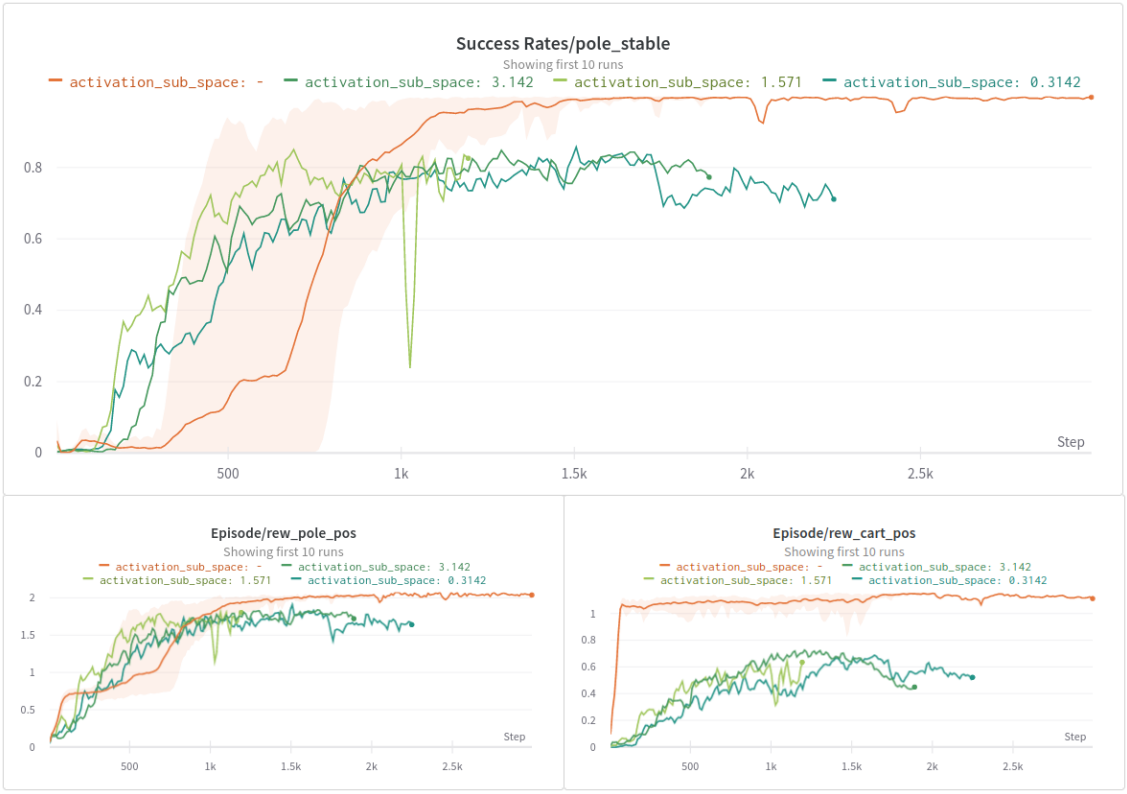

Figure 4-1: Cartpole experiment results training with coupled rewards and a physically intuitive value for σ_{switch}. Experiments show that the local minima between pole position and cart position is removed, but overall converged policy performance degrades.

reward landscape seen in figure 1-2, the coupled reward polices sometimes fail to stay within the legal cart range.

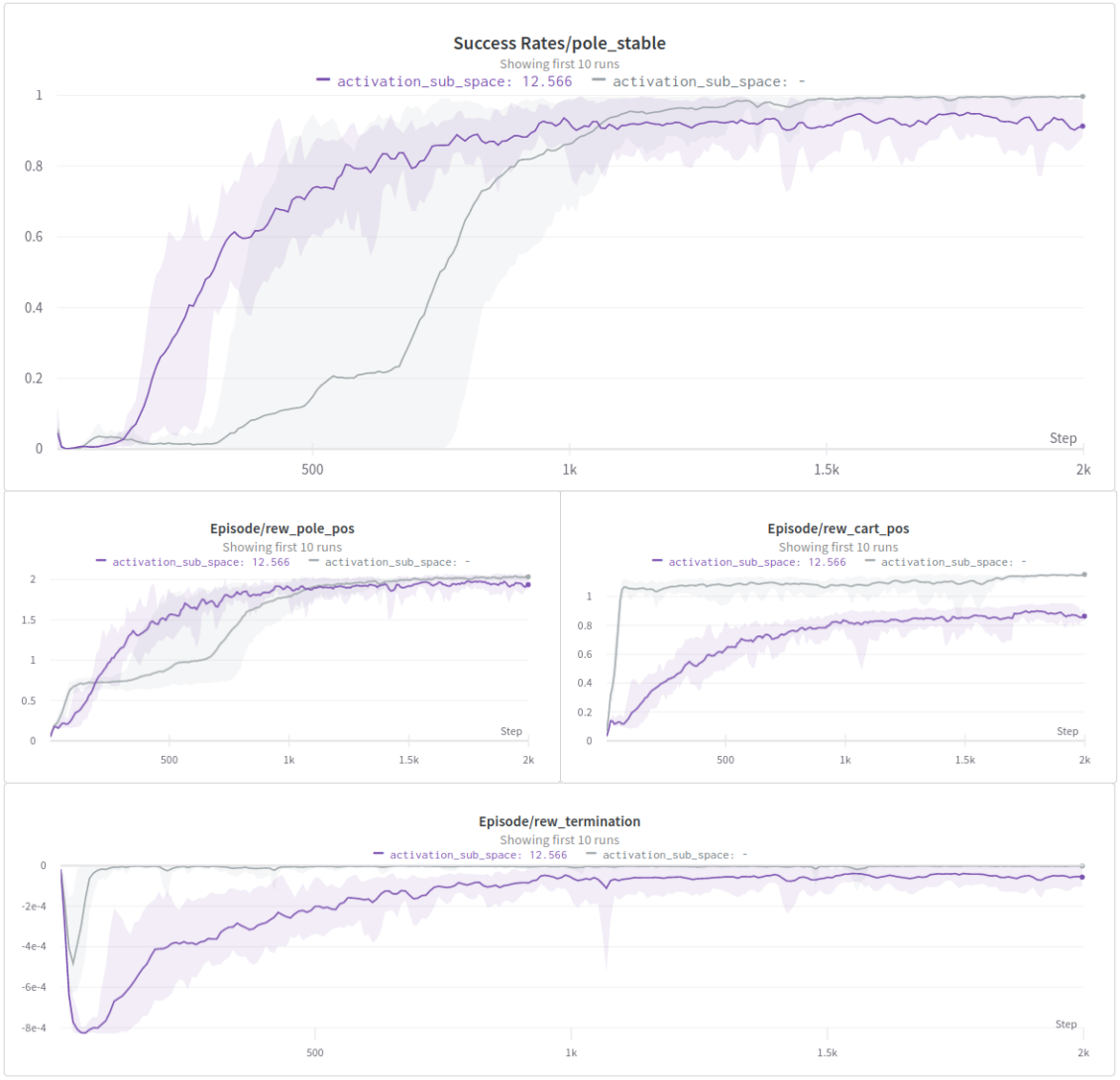

Figure 4-2: Using the best coupling activation scaling $\sigma_{switch} = 4\pi$, the coupled reward formulation continued to suffer policy degradation at convergence. The pole position reward graph shows that the naively trained cartpole (grey) enters an intermediate local minima between 300 and 800 learning iterations before learning the swing up and stabilize behavior. This local minima is avoided for the coupled reward formulation (purple); however, the reward termination graph indicates that coupled rewards never fully learn a perfectly non-terminating behavior.

Chapter 5

Conclusion

This work explored ways to increase deep reinforcement learning efficiency by leveraging understanding about the current state of the art in controlling legged robots. Robotic control is dominated by two general paradigms: model-based and model-free. Recall that model-based optimal controllers perform admirably and offer strong engineering control due to immense engineering infrastructure around these methods. They offer highly custom style control and can be quickly developed, debugged, and tuned. However, these controllers are subject to engineering approximation and assumptions which can fail to generalize to un-modeled disturbances or dynamics. The alternate route - model-free optimal control - has also successfully crafted control policies for legged robots. Methods have been developed which make these learned policies incredibly robust to the modeling errors and disturbances that can plague model-based controllers. However, the model-free methods also have incredibly long training times causing them to be frustrating to iterate and tune. This long training time issue is the core problem in model-free control pipelines.

This thesis supposes that it is possible to address this inefficiency by examining the differences between model-based and model-free control. Related work has shown incredible performance gains for control pipelines that leverage hybrid architectures between each paradigm. These works suggest that there is a clear reason why model-free learning takes so long to train and iterate: redundant work. Each time a neural network is re-initialized, all modeling progress is deleted and needs to be redone.

The ability to iterate without re-creating models enables fast iteration and tuning in model-based pipelines. This work attempts to make core model-based knowledge available to model-free methods via the observations and rewards. Using domain specific knowledge, model-free methods should not need to re-create everything from scratch on each training iteration. Instead, any nonlinear dynamics modeling should already be available and any clear state-based trade offs in the reward functions should already be made. In this realm, re-initializing a network should only need to make quick and small tuning adjustments similar to an engineer tuning a weight or scaling. Towards this goal, two ideas were developed: (1) incorporating known and successful model-based observation augmentations and (2) reformulating model-free objective functions into task hierarchies.

Augmenting the actor and critic neural networks with known dynamics, controls, and kinematics terms dramatically increase learning speed and policy performance for both the cartpole and Mini Cheetah simulated agents. On the cartpole system, the experiments clearly demonstrate that dynamics augmentations increase learning speed by a large margin. Control augmentations also increase learning speed, but to a lesser degree. Further, using augmentations allowed for smaller actor networks to be used to achieve the same learning performance. For the Mini Cheetah, steady-state policy performance increased when augmentations were introduced. Forward kinematics augmentations proved to nearly double policy performance over naive policies, but ground reaction forces did have a significant average effect as well. Based on these results, if model-based augmentations are available for an agent, they should be included into the model-free network observations. Faster learning speeds are imperative to better engineering iteration and tuning and augmentations are a valid route toward speeding up engineering feedback loops.

Reformulating model-free reward functions into a task priority did function as intended with a heavy performance caveat. Results for the cartpole system indicated that coupling the cart and pole reward functions did avoid the local minima that often impedes policy training under the traditional reward structure. However, these results were only shown with a non-physical value for the coupling between the cart and the

pole. Rather than adding engineering structure and intuition to the problem, coupling rewards in this manner simply added another knob to turn. Further, final policy performance degraded with the introduction of the more complex reward function. In a paradigm where local minima are impossible to escape, coupled rewards may be useful to avoid these more dramatic pitfalls. However, reward tuning is a simpler method to achieve the same goal with only limited learning rate decreases.

Finding and improving more ways to incorporate model-based expertise into model-free pipelines has made learning more efficient. As engineering feedback loops for these systems become faster and faster, legged robots' usefulness and capabilities will continue to grow. Boosting model-free methods will truly broaden the scope of what robotics can do when deployed in the real world. As robots leave the structure of factory floors and enter into the more natural and useful environments, they will need the robustness of excellent learning methods and iterability that comes with model-based control as well.

Bibliography

[1] Anybotics, Anymal, https://www.anybotics.com/anymal-autonomous-legged-robot/, [Online; accessed Jul. 2022].

[2] Boston Dynamics, Spot, 2021, https://www.bostondynamics.com/spot, [Online; accessed Jul. 2022].

[3] Boston Dynamics, Industry Solutions, https://www.bostondynamics.com/solutions, [Online; accessed Jul. 2022].

[4] Agility Robotics OSU, Cassie, https://robots.ieee.org/robots/cassie/, [Online; accessed Jul. 2022].

[5] CasADi, https://web.casadi.org, [Online; accessed Jul. 2022].

[6] Cheetah Software, https://github.com/mit-biomimetics/Cheetah-Software, [Online; accessed Jul. 2022].

[7] Drake, https://drake.mit.edu, [Online; accessed Jul. 2022].

[8] CVXOPT, https://cvxopt.org, [Online; accessed Jul. 2022].

[9] Legged Gym Repository, https://github.com/leggedrobotics/legged_gym, [Online; accessed Aug. 2022].

[10] Weights and Biases, https://wandb.ai/site, [Online; accessed Jul. 2022].

[11] A. Alaimo, V. Artale, C. Milazzo, and A. Ricciardello. Comparison between euler and quaternion parametrization in uav dynamics. *AIP Conference Proceedings*, 1558(1):1228–1231, 2013.

[12] P. Baerlocher and R. Boulic. Task-priority formulations for the kinematic control of highly redundant articulated structures. In *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)*, volume 1, pages 323–329 vol.1, 1998.

[13] Gerardo Bledt. *Regularized Predictive Control Framework for Robust Dynamic Legged Locomotion*. PhD thesis, 01 2020.

[14] Gerardo Bledt and Sangbae Kim. Implementing regularized predictive control for simultaneous real-time footstep and ground reaction force optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6316–6323, 2019.

[15] Steven Bohez, Saran Tunyasuvunakool, Philemon Brakel, Fereshteh Sadeghi, Leonard Hasenclever, Yuval Tassa, Emilio Parisotto, Jan Humplik, Tuomas Haarnoja, Roland Hafner, Markus Wulfmeier, Michael Neunert, Ben Moran, Noah Siegel, Andrea Huber, Francesco Romano, Nathan Batchelor, Federico Casarini, Josh Merel, Raia Hadsell, and Nicolas Heess. Imitate and repurpose: Learning reusable robot movement skills from human and animal behaviors, 2022.

[16] Russell Buchanan, Lorenz Wellhausen, Marko Bjelonic, Tirthankar Bandyopadhyay, Navinda Kottege, and Marco Hutter. Perceptive whole body planning for multi-legged robots in confined spaces. *Journal of Field Robotics*, 38, 05 2020.

[17] Richard H. Byrd, Jorge Nocedal, and Richard A. Waltz. Knitro: An integrated package for nonlinear optimization. In *Large Scale Nonlinear Optimization, 35–59, 2006*, pages 35–59. Springer Verlag, 2006.

[18] Matthew Chignoli, Savva Morozov, and Sangbae Kim. Rapid and reliable quadruped motion planning with omnidirectional jumping, 2022.

[19] Matthew Chignoli, Savva Morozov, and Sangbae Kim. Rapid and reliable quadruped motion planning with omnidirectional jumping, 2022.

[20] Jared Di Carlo, Patrick M. Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018.

[21] Yanran Ding, Charles Khazoom, Matthew Chignoli, and Sangbae Kim. Orientation-aware model predictive control with footstep adaptation for dynamic humanoid walking, 2022.

[22] Helei Duan, Jeremy Dao, Kevin Green, Taylor Apgar, Alan Fern, and Jonathan Hurst. Learning task space actions for bipedal locomotion. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1276–1282, 2021.

[23] Alejandro Escontrela, Xue Bin Peng, Wenhao Yu, Tingnan Zhang, Atil Iscen, Ken Goldberg, and Pieter Abbeel. Adversarial motion priors make good substitutes for complex reward functions, 2022.

[24] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2007.

[25] Michele Focchi, Romeo Orsolino, Marco Camurri, Victor Barasuol, Carlos Mastalli, Darwin G. Caldwell, and Claudio Semini. Heuristic planning for rough terrain locomotion in presence of external disturbances and variable perception quality. In *Springer Tracts in Advanced Robotics*, pages 165–209. Springer International Publishing, sep 2019.

[26] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[27] Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping, 2020.

[28] Linbin Huang, Jeremy Coulson, John Lygeros, and Florian Dorfler. Data-enabled predictive control for grid-connected power converters, 2019.

[29] Se Hwan Jeon, Sangbae Kim, and Donghyun Kim. Real-time optimal landing control of the mit mini cheetah, 2021.

[30] Gwanghyeon Ji, Juhyeok Mun, Hyeongjun Kim, and Jemin Hwangbo. Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion. *IEEE Robotics and Automation Letters*, 7(2):4630–4637, 2022.

[31] Benjamin Katz, Jared Di Carlo, and Sangbae Kim. Mini cheetah: A platform for pushing the limits of dynamic quadruped control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6295–6301, 2019.

[32] D. Kim, D. Carballo, J. Di Carlo, B. Katz, G. Bledt, B. Lim, and S. Kim. Vision aided dynamic exploration of unstructured terrain with a small-scale quadruped robot. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2464–2470, 2020.

[33] Donghyun Kim, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control, 2019.

[34] George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the fourier basis. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, page 380–385. AAAI Press, 2011.

[35] Scott Kuindersma, Frank Permenter, and Russ Tedrake. An efficiently solvable quadratic program for stabilizing dynamic locomotion. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2589–2594, 2014.

[36] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986, 2020.

[37] He Li, Robert J. Frei, and Patrick M. Wensing. Model hierarchy predictive control of robotic systems. *IEEE Robotics and Automation Letters*, 6(2):3373–3380, 2021.

[38] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning. *CoRR*, abs/2108.10470, 2021.

[39] Gabriel B. Margolis, Tao Chen, Kartik Paigwar, Xiang Fu, Donghyun Kim, Sangbae Kim, and Pulkit Agrawal. Learning to jump from pixels, 2021.

[40] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62), jan 2022.

[41] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[42] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[43] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287. Morgan Kaufmann, 1999.

[44] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control. *ACM Transactions on Graphics*, 40(4):1–20, aug 2021.

[45] Xue Bin Peng and Michiel van de Panne. Learning locomotion skills using DeepRL. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. ACM, jul 2017.

[46] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning, 2021.

[47] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[48] Jonah Siekmann, Kevin Green, John Warila, Alan Fern, and Jonathan Hurst. Blind bipedal stair traversal via sim-to-real reinforcement learning, 2021.

[49] Russ Tedrake. *Underactuated Robotics*. 2022.

[50] 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, 2017.

[51] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[52] Marco Tranzatto, Mihir Dharmadhikari, Lukas Bernreiter, Marco Camurri, Shehryar Khattak, Frank Mascarich, Patrick Pfreundschuh, David Wisth, Samuel Zimmermann, Mihir Kulkarni, Victor Reijgwart, Benoit Casseau, Timon Homberger, Paolo De Petris, Lionel Ott, Wayne Tubby, Gabriel Waibel, Huan Nguyen, Cesar Cadena, Russell Buchanan, Lorenz Wellhausen, Nikhil Khedekar, Olov Andersson, Lintong Zhang, Takahiro Miki, Tung Dang, Matias Mattamala, Markus Montenegro, Konrad Meyer, Xiangyu Wu, Adrien Briod, Mark Mueller, Maurice Fallon, Roland Siegwart, Marco Hutter, and Kostas Alexis. Team cerberus wins the darpa subterranean challenge: Technical overview and lessons learned, 2022.

[53] Albert Wang, Joao Ramos, John Mayo, Wyatt Ubellacker, Justin Cheung, and Sangbae Kim. The hermes humanoid system: A platform for full-body teleoperation with balance feedback. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 730–737, 2015.

[54] Zhaoming Xie. *Reinforcement learning for legged robot locomotion.* PhD thesis, University of British Columbia, 2021.

[55] Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonathan Hurst, and Michiel van de Panne. Iterative reinforcement learning based design of dynamic locomotion skills for cassie, 2019.