HABITAT-LAB QUADRUPED EMBODIED AI RESEARCH

An Undergraduate Research Scholars Thesis

by

SCOTT STEINHAUSER

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Dylan Shell

May 2022

Major: Computer Science

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Scott Steinhauser, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Habitat-Lab Quadruped Embodied AI Research

Scott Steinhauser
Department of Computer Science and Engineering
Texas A&M University

Research Faculty Advisor: Dr. Dylan Shell
Department of Computer Science and Engineering
Texas A&M University

With the rise of reinforcement learning, a number of physics engines and frameworks have been used to simulate virtual environments. Additionally, various benchmarks have been popularized in order to assess the ability of RL algorithms to learn from these environments and produce effective policies for agents to act on. In this project, we aim to replicate a popular reinforcement learning benchmark, Open AI's Ant-v2 environment, in the AI Habitat platform and apply Proximal Policy Optimization to train a quadruped robot to run on a platform. Doing so allows us to extend the standard benchmark by adding visual sensors to the robot and including more complex environments such as indoor spaces. This fits into a larger goal of solving point-navigation tasks with dynamic locomotion.

ACKNOWLEDGMENTS

Contributors

I would like to thank my faculty advisor, Dr. Dylan Shell, and my research mentor, Alexander Clegg, for their guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

Finally, thanks to my family for their encouragement throughout my education.

NOMENCLATURE

RL	Reinforcement Learning
PPO	Proximal Policy Optimization
PD	Proportional-Derivative
EAI	Embodied Artificial Intelligence
s	State
a	Action
t	Discrete time step
s_t	State at t
a_t	Action at t
r_t	Reward at t, dependent, like S_t, on A_{t-1} and S_{t-1}
π	Policy, decision-making rule

1. INTRODUCTION

A pop-culture vision for the future of technology often includes robotic agents which live in homes and assist with mundane and repetitive chores. These tasks can range anything from doing the dishes, making a bed, or folding laundry. Ideally, such robot's capabilities would be generalized, allowing them to navigate from room to room completing a wide range of complex tasks through the use of sensors and motors. Despite the ease with which humans complete these actions, it's proven incredibly difficult to create robots which can do the same. That being said, research towards accomplishing this vision is actively being done [1]. In order to make progress towards creating robots which can navigate and interact with the world, some researchers have focused efforts towards solving the sub-problems of agent navigation and dynamic-locomotion[2, 3]. These problems fit into the field of Embodied Artificial Intelligence and are often solved using reinforcement learning techniques.

1.1 Embodied Artificial Intelligence

Embodied Intelligence is a core aspect of all animals and can be understood in the context of humans - infants progressively learn about the world through their five senses and how to navigate within it using the motor-control of their body. With this in mind, the embodiment hypothesis proposes that "intelligence emerges in the interaction of an agent with an environment and as a result of sensorimotor activity." [4]. A human takes in information about their surrounding environment predominantly through sight, hearing, and touch, processes this information through their nervous system, and acts based on these processed inputs to achieve some set of objectives. Overtime, a person learns which actions in response to their environment best achieve their objectives in both the short and long term.

Building off this idea, Embodied Artificial Intelligence (EAI) is described as "the science and engineering of intelligent machines with a physical or virtual embodiment (typically, robots and egocentric personal assistants)" [5]. In order to automate complicated activities done by humans, machines must be constructed and programmed in a way as to achieve some level of the embodied intelligence that humans express, often with important differences. Examples may include a package delivery robot in a warehouse, a robotic arm on an assembly line, or a self-navigating robotic vacuum cleaner in a family home. Robots may be equipped with light, temperature, proximity, or accelerations sensors to gain information about the surrounding environment and its configuration. Any robotic agent which takes in information about an environment and acts on it requires some method of mapping observations to actions. EAI can be understood as a toolkit which provides these mapping methods. The sophistication of EAI methods and algorithms needed depends on the complexity of the robot's environment, the robot itself, and the robot's task. Within Embodied AI, point-navigation and dynamic locomotion are very important tasks which are seen as crucial steps towards developing more complex robots.

1.1.1 Point-Navigation

In a point navigation task, "an agent is spawned at a random starting position and orientation in an unseen environment and and asked to navigate to target coordinates specified relative to the agent's start location ('Go 5m north, 3m west relative to start'). No ground-truth map is available and the agent must only use its sensory input (an RGB-D camera) to navigate." [6]. Figure 1.1 shows an example of a point-navigation task.

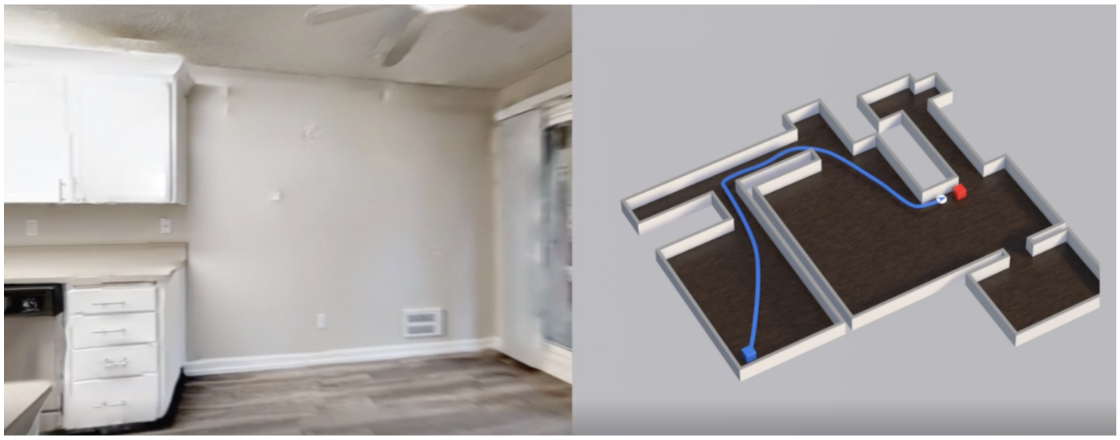


Figure 1.1: An agent is loaded in a virtual environment at a starting location and is tasked with navigating to a target point.

An agent observes its environment in the form of an RGB image, processes it in some way, and then takes an action so as to get closer to the target coordinates. In the depicted version of this problem, the potential action is discrete - the agent can either turn left, turn right, walk forward, or stop moving altogether. Despite the simplicity of the action, the indoor environment the agent exists in can be very complicated and the observations can be composed of hundreds of thousands of values which vary frame-to-frame. Recently, this version of the point-navigation problem has been solved by researchers using the Habitat simulator, a Proximal Policy Optimization (PPO) reinforcement learning algorithm, and a set of 3D Replica CAD indoor environments [2].

1.1.2 Dynamic Locomotion

While wheeled robots are easily constructed and controlled, their ability to traverse rough terrain, including a cluttered floor in a small apartment, is limited. In contrast, Legged robots have the "potential to perform any physical activity humans and animals are capable of" and have the ability to "one day rescue people in forests and mountains, climb stairs to carry payloads in construction sites, inspect unstructured underground tunnels, and explore other planets" [7]. Recently, Boston Dynamic's four-legged SPOT robot has been shown to traverse construction sites, paving the way for construction assistants [8]. While more difficult to build and program, legged robots such as quadrupeds provide greater versatility and navigation potential.

These robots typically contain a variety of hydraulic actuators or electric servomotors, which are carefully controlled to achieve a desired motion. The action space of aforementioned legged systems is continuous - each motor can be provided with a torque that can take on any value bounded by that motor's physical limitations. While a stable walking motion is commonly the desired movement, it is often necessary for legged robots to step over obstacles, jump across platforms, modulate velocity, and to adjust its walking motion in response to varying terrain. Robustness is a desired attribute of a dynamic locomotion controller, allowing it to be "reliable enough to maintain the robot's balance while executing a locomotion command, even in the presence of large disturbances." [9]. Despite people's innate ability to traverse complex terrain, it can be impractical for a human to entirely design a robust controller for a legged robot. Instead, researchers have turned to Reinforcement Learning (RL) for training legged robots. Recently, MIT's mini cheetah quadruped robot broke the quadruped running speed record using a model-free RL approach [10]. In some cases, a prior motion and RL are used together to generate natural motions which maintain robustness [11].

1.2 Reinforcement Learning

Due to the high-dimensional action spaces for robots and the large amount of variety in environments, RL is a popular tool for solving these problems. RL allows us to train agents to achieve specific goals without explicitly programming them to take certain actions, and has proven to be valuable in EAI problems such as agent navigation [12], object manipulation [3], and dynamic locomotion [10].

Every RL problem includes an agent who "have explicit goals, can sense aspects of their environments, and can choose actions to influence their environments." [13] The set of attributes about the environment the agent can sense is known as the state s and can include information about the agent's surroundings and configuration among other things. In response to the observed state S_t observed at discrete time step t, the agent chooses to take some action A_t, as shown in Figure 1.2, which often changes the agent's configuration in some way. Once the action A_t is taken, the agent receives a reward R_{t+1} which is based on S_t and A_t and is a measure of the progress made towards the agent's explicit goals. In this way, a high reward encourages actions which push the agent closer to its explicit goals.

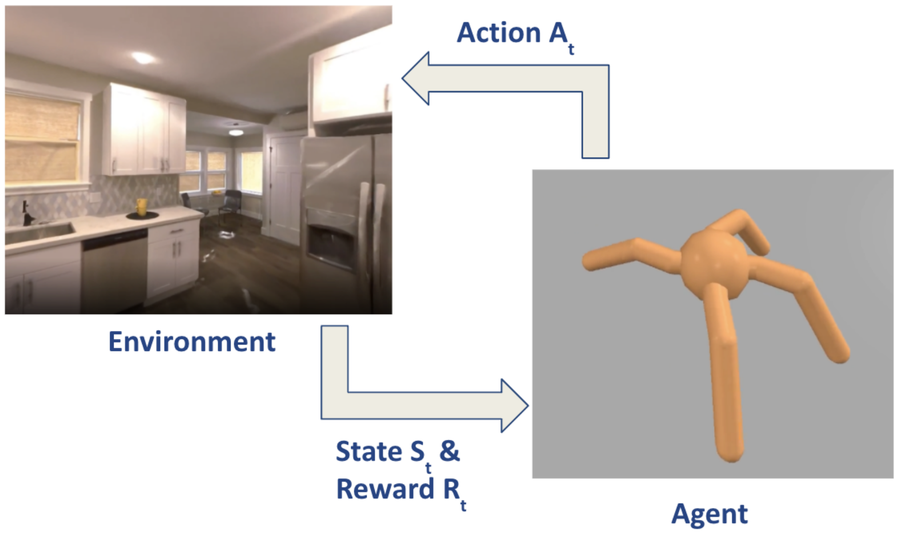


Figure 1.2: An example of a reinforcement learning setup.

Most of RL focuses on optimizing the function which maps the agent's state S_t to a resulting action A_t such that the reward received overtime is maximized. This function is known as the policy π, and the estimated value that an action will return in the long run is known as the cumulative discounted reward. The policy aims to take the action at each time step which maximizes value overtime.

1.2.1 Proximal Policy Optimization

Recently, Proximal Policy Optimization (PPO) has proven to be a highly efficient Reinforcement learning algorithm in embodied AI due to its balance between "sample complexity, simplicity, and wall-time" [14]. PPO is a It's proven to be valuable in solving point-navigation and dynamic locomotion tasks, while remarkably easier to implement and tune than other RL approaches.

1.2.2 Simulations

While the primary end goal of robots is usefulness in the real world, simulated environments "can run orders of magnitude faster than real-time, can be highly parallelized, and enable decades of agent experience to be collected in days" [15]. While this is very valuable, more research is necessary to determine the required level of realism and ideal approach for successfully transferring policies trained in simulation to physical robotic systems. Despite this, simulators often offer a high degree of flexibility when it comes to agent configurations, environments, and task creation and give researchers more opportunities to explore solutions to the wide variety of challenging real world problems they would like to solve. The Habitat Platform [2, 3], for example, is a software stack designed for applying reinforcement learning to interactive embodied tasks at scale with a highly performant rendering engine and physics simulator which can support dynamic environments.

A common approach to solving a complex, long horizon task is to first dissolve it into a simpler sub-task, solve that, and iteratively expand upon it until the complexity of the original problem is tractable [3]. With the proliferation of methods being developed, assessment and comparison of competing techniques is important; toward that end, many benchmarks have been proposed. Some of these include OpenAI's Gym [16] and Stanford's BEHAVIOUR household activity benchmarks [17]. Benchmarks range from training a lunar lander to land safely on the moon to training a quadruped ant how to run forward, all in simulated environments.

1.3 Proposal

In this project, I plan on implementing a quadruped robot in the Habitat Platform and training it to complete progressively more complicated tasks, with a focus on making progress towards solving point-navigation with dynamic locomotion. I'll start from a well-known quadruped RL benchmark, and explore how different environment configurations, action controllers, observation terms, and reward signals affect the RL training process. Upon successful results, I'll increase the complexity of the task and environment. My goal is to illustrate that by replicating and extending the functionality of a well-known RL robotics benchmark in a more flexible simulator, I can efficiently expand upon a classical reinforcement learning problem with novel twists.

2. INFRASTRUCTURE AND METHODOLOGY

The primary goal of this project is make progress towards solving point navigation tasks with quadruped robots. Our first step is to implement an existing Reinforcement Learning benchmark into a simulator and iteratively expand the task complexity as well as the agent's sensorimotor capabilities. We will replicate Open AI's Ant-V2[18] RL task into the Habitat Platform with some small differences. The Habitat Platform, developed by Meta Reality Labs, allows for the construction of 3D environments and agents, the defining of RL tasks, and the training of agents given those environments and tasks[2, 3]. We'll begin by defining the environmental observations out agent is capable of taking, the actions the agent can take in response to these observations, the goals and objectives we want the agent to achieve, and the reward measures used to signal successful progress towards these goals. Finally, we design a framework for tuning hyperparameters, reward signal weights, and any other configurations. As training results begin to show promising results, we will progressively increase task complexity and experiment with different reward structures.

2.1 The Simulator Setup

We start out by creating a quadruped robot modeled after OpenAI's Ant-V2 task, a wrapper for controlling and manipulating this agent, and an environment in which this agent is placed in the Habitat framework.

2.1.1 Open AI's existing Ant-v2 Benchmark

The Ant-V2 RL benchmark's aim is to train a quadruped robot to run forward such that it's speed is maximized [18]. The quadruped is spawned on a platform. At each timestep, the quadruped is able to obtain information about it's state including it's joint positions, orientation, linear velocity, angular velocity, and contact forces with the surrounding environment. Additionally, it's able to take an action by applying some torque to each of its 8 hinge joints; 4 hips and 4 knees. Each joint's rotation is limited by [-1, 1] radians. By applying different torque values to each joint at the right times, the goal is to maximize the agent's distance from the origin after some amount of time. In order to encourage actions which lead to desirable states, a positive reward signal is given when the quadruped advances in the x-direction, and a negative reward signal is given for large contact forces on the ground and high-torque actions. It's important to recognize that while minimizing contact forces and torque magnitudes is not the primary goal, penalizing these produces better training performance and more stable motions. The aggregate reward signal r, along with the state s and action t, are used to fine tune the agent's policy π through the use of a reinforcement learning algorithm.

We replicated this task, with some small changes, in the Habitat platform as a starting point for eventually solving more complex point-navigation tasks inside high-fidelity environments.

2.1.2 The Habitat Platform

This Habitat platform, affords us the flexibility for eventually using this quadruped agent to solve a point-navigation task in an indoor environment. In order to make progress towards this goal, we begin with a bare-bones environment and the aim of training the ant to walk. We then will increase the complexity of tasks. This could take the form of periodically placed obstacles for the ant to avoid, stairs for the ant to climb, and gaps in the floor for the ant to jump across. Additionally, Habitat affords us the flexibility to include RGB and depth sensors in the ant's known state-space, which may optimize the learning process and improve the ant's ability to react optimally to its surrounding environment. Using Habitat's open-source python library, we first created a controller to initialize and manipulate the ant's configuration.

2.1.3 The Ant Wrapper

A URDF file was created and modeled after the existing ant in schulman's [18] implementation. This file specifies the ant's body - including its links and the joints which connect the links. Each link includes a geometrical shape, color, and size specification. Each joint specifies the links being connected, their offsets from each other, and the joint's range of [-1, 1] radians. Once the URDF model was created, it was loaded into the Habitat simulator.

A controller was then written in python to be in charge of initializing the ant, manipulating the ant's joint positions, and retrieving information about the ant's state in the environment. As opposed to OpenAI's implementation which relies on providing torques The ant is initialized in a rest position hovering slightly above a large plane.

As it has 8 total hinge joints, it's joint space is 8 dimensional. At the lowest level, the controller applies torques to each joint in order to move them. The primary controller we use, however, works by giving the ant an 8 dimensional vector where each value represents a target joint configuration. The ant wrapper then uses a proportional-derivative (PD) controller which carefully applies torques on each of the ant's joint-motors in order to reach the target joint configuration specified. It does this using a combination of an impulse, velocity damping, and position-gain term. We use the PD controller provided by the Habitat framework's physics library. At a given timestep t, q_t^s represents the current positions of each joint, and q_t^{PD} represent the positions that the PD controllers are aiming to achieve.

Finally, various attributes specific to the ant can be retrieved, including the ant's global body position, orientation, directional velocity, and each joint's position and velocity. This information will later be retrieved and used in the observational data used to train the agent.

Figure 2.1, pictured below, shows the ant agent at initialization and it's configuration after various timesteps using randomly chosen joint states.

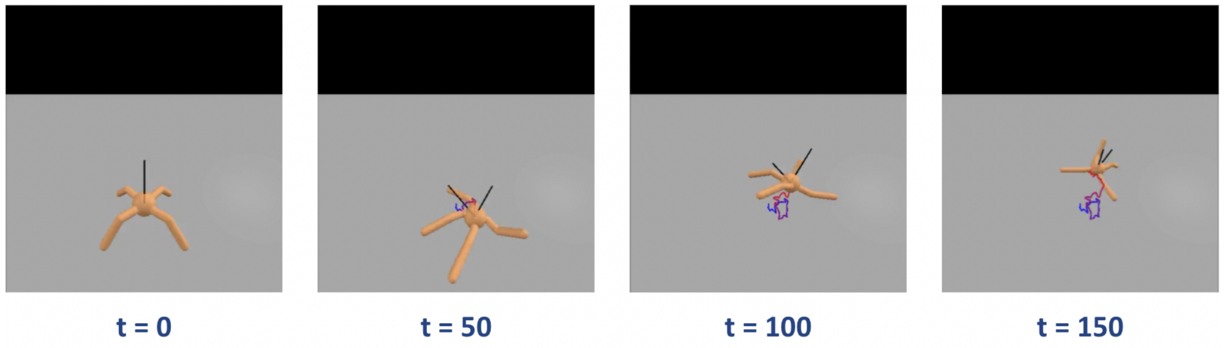


Figure 2.1: The ant quadruped loaded in the Habitat simulator

2.2 Observations

At each timesteps, a set of observations will be given to the agent to act on. This set is known as the state s. Depending on the task, some observations will be more important than others. Through the training process, the hope is that the policy will learn which observations hold more relevance to the desired action, and will be able to map from those observations to such action. Additionally, we can adjust which observations to include in the given state on a experiment-to-experiment basis.

There are four categories of observations, each with different potential uses and relevance.

2.2.1 Global Observations

Global observations include any basic information about the surrounding environment in world space. This includes the simulation episode time ($1D$) and the agent's body position ($3D$), orientation quaternion ($4D$), linear velocity ($3D$), and angular velocity ($3D$). Finally, whether or not the each of the agent's feet are in contact with the ground is included ($4D$).

2.2.2 Egocentric Observations

Egocentric Observations include information relative to the agent and/or in the agent's local space. Some of these include global observations which are multiplied by the inverse of agent's transform, M^{-1}:

$$\vec{v_e} = M^{-1} * \vec{v_g} \qquad \text{(Eq. 1)}$$

These include the agent's feet positions ($4 * 3D$), upwards vector ($3D$), target heading ($3D$), body velocity ($3D$), and body angular velocity ($3D$), all of which are relative to the agent's base position and orientation. Finally, a 128x128 RGB camera provides the agent with vision of its surroundings.

2.2.3 Joint Information

Joint information provides the agent with an understanding of its configuration at any given time. This includes the absolute joint position state, q_t^s ($8D$), which is the current position of each joint in radians, and target joint positions, q_t^{PD} ($8D$), which represent the joint positions the PD controllers are trying to achieve at that moment. Joint velocities, $\dot{q_t^s}$ ($8D$), are also included to give the ant more context about its current movement.

Additionally, it can be valuable to define some natural resting position or a more dynamic movement pattern we want the ant to approximate at each step. As a result, the current target pose, q_t^{tp} ($8D$) for timestep t, and the next target pose, q_{t+1}^{tp} ($8D$) for timestep $t+1$, are provided as observations.

2.2.4 Historical Observations

Finally, it can be valuable to provide information about past states and actions to give the ant context to its trajectory. Included in this is the previous n absolute joint positions, $\bigcup\limits_{i=t-n+1}^{t} q_i^s$ ($n * 8D$), and the previous m actions taken, $\bigcup\limits_{i=t-n+1}^{t} a_i$ ($m * 8D$).

2.3 Actions

We define three possible action paradigms; controlling relative joint positions, absolute joint positions, and an absolute deviation from some target pose, all represented by a_t. Each action is controlled by 8 dimensional vectors which provides the ant with a new target joint position, q_t^{PD}, which is obtained in the following timesteps by PD controllers.

2.3.1 Relative Joint Position

At each timesteps, this action provides a relative change in the agent's joint configuration. If the current configuration is [0.5, 0.3, 0.7, 0.1, -0.2, -0.3, -0.1, -0.5] and the action is [0.1, -0.1, 0.1, -0.2, 0.0, -0.1, 0.1, -0.2], the new target joint position would be [0.6, 0.2, 0.8, -0.1, -0.2, -0.4, 0.0, -0.7]. The magnitude of each relative change can also be controlled by a configuration parameter $DELTA\_POS\_LIMIT$. If $DELTA\_POS\_LIMIT$ is set to 0.1, the most each joint can be adjusted by in a single timesteps is 0.1 radians.

2.3.2 Absolute Joint Position

This action simply provides a new absolute joint position for the ant.

2.3.3 Target Pose Deviation

This action controller takes the current target pose, q_t^{tp}, adds a deviation to it, and sets the resulting value to the ant's new target joint position q_t^{PD}.

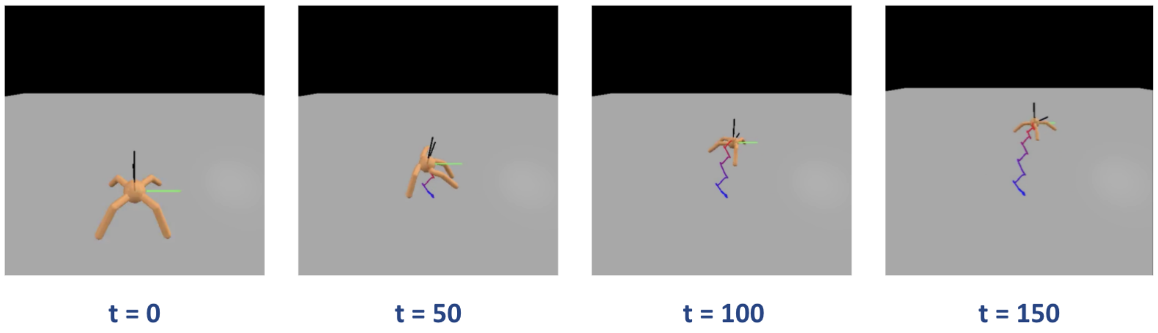


Figure 2.2: The ant quadruped loaded in the Habitat simulator

Figure 2.2 is a controller manually created.

2.4 Reward Terms

With the end goal of solving point navigation tasks, we first aim to solve various sub-tasks. These include walking forward, achieving target poses, orienting in specified directions, and utilizing vision, all while maintaining a upright orientation and minimizing action variability and instability. With these tasks in mind, we create reward signals that will be relevant to rewarding behavior in the agent which accomplishes these goals. In this way, we aim to estimate positive behavior with scalar values which are used to optimize the agent's final policy.

Rewards are broken down into three categories - kinematic rewards, joint configuration rewards, and action rewards. Some rewards are normalized, where they are bounded by ranges [-1, 1] or [0, 1], while others can take on any value.

2.4.1 Kinematic Rewards

- **X-Location**: This reward directly correlates to its absolute distance from its starting location along the x-axis.

- **Vector Root Delta**: This reward corresponds to the agent's change in in position along a target heading at a given moment.

- **Orientation Alignment**: This reward corresponds to the alignment of an agent's local vector, $\vec{l}$, with a global vector converted to the ant's local space, $M^{-1}\vec{g}$. This could include measuring the uprightness of the agent, or measuring the orientation alignment reward, r_t^o, of the agent's front-facing vector with some target direction.

  $$r_t^o = M^{-1}\vec{g} \cdot \vec{l} \qquad \text{(Eq. 1)}$$

  We can also square this measure to provide steeper rewards as the ant increases its alignment.

  $$r_t^{o_s} = sign(r_t^o) * ({r_t}^o)^2 \qquad \text{(Eq. 2)}$$

- **Velocity Alignment**: This reward corresponds to how closely the agent's current base velocity, $\vec{v_t^c}$, aligns with some target direction $\vec{v_t^t}$.

  $$r_t^v = \frac{\vec{v_t^c} \cdot \vec{v_t^t}}{|\vec{v_t^c}|} \qquad \text{(Eq. 3)}$$

  A specified target speed, s_t^t, can also be taken into account with the following measure:

  $$r_t^{vs} = \exp\left[-c_v * max(s_t^t - v_t^c \cdot \hat{v_t^t}, 0)^2\right] \qquad \text{(Eq. 4)}$$

  where c_v is some configurable constant.

- **Egocentric End-Effector Position Reward**: This reward corresponds to the egocentric location of the agent's feet, or end-effectors p_t^e, and their distance from some target egocentric position, $p_t^{e_t}$. This target can be constant, or part of some target motion.

$$r_t^e = \exp\left[-c_e * \sum_e ||p_t^{e_t} - p_t^e||^2\right] \qquad \text{(Eq. 5)}$$

where c_e is some configurable constant.

2.4.2 *Joint Configuration Rewards*

- **Joint State Error**: This measure corresponds to how far the agent's current configuration, q_t^s, is from some target pose, q_t^{tp} at timestep t. We will try the following set of formulas for computing this reward, each with different purposes:

$$r_t^{j_s} = ||q_t^s - q_t^{tp}||^2 \qquad \text{(Eq. 6)}$$

We also use a normalized product formulation:

$$r_t^{j_p} = \prod_i^{q_t^s} (q_{t\,i}^s - q_t^{tp}{}_i)/N_i \qquad \text{(Eq. 7)}$$

$$N_i = max[abs(q_t^{tp}{}_i - 1), abs(q_t^{tp}{}_i + 1)] \qquad \text{(Eq. 8)}$$

Finally, we implement a exponential pose reward:

$$r_t^{j_e} = \exp\left[-c_j * \sum_i ||q_{t\,i}^s - q_t^{tp}{}_i||^2\right] \qquad \text{(Eq. 9)}$$

where c_j is some configurable constant.

- **Joint Velocity Error**: This reward corresponds to the deviation between agent's joint velocity, $\dot{q}_{t\,i}^s$, and some target joint velocity, $\dot{q}_t^{tp}$.

$$r_t^{jv} = \exp\left[-c_{jv} * \sum_i ||\dot{q}_{t\,i}^s - \dot{q}_t^{tp}{}_i||^2\right] \qquad \text{(Eq. 10)}$$

where c_{jv} is some configurable constant.

2.4.3 Action Rewards

- **Action Cost**: This reward corresponds to the costliness of an action, and discourages unnecessarily extreme action trajectories.

$$r_t^{ac} = 1 - avg(|a_t|) \tag{Eq. 11}$$

- **Action Smoothness**: This reward corresponds to the likeness of successive actions and compares the previous n actions.

$$a_{avg} = \sum_{i=t-n+1}^{t} a_i/n \tag{Eq. 12}$$

$$r_t^{as} = \prod_{i=t-n+1}^{t} 1 - |a_i - a_{avg}|/2 \tag{Eq. 13}$$

After each action step, the agent is given a reward based on it's change in state and previously taken action. This reward could just be one of the measures above, or it could be a linearly weighted combination of them. Experiments were set up using different reward schemes.

2.5 Experimental Setup

We constructed a framework for rapidly iterating through different experiment designs. Our goal was to easily adjust episode-specific information including the state observations, the action controller, the composite reward terms and reward weights, as well as the PPO hyperparameters and training-specific parameters.

2.5.1 Experiment Specification Framework

Two configuration YAML files were created; one defines parameters for the simulator and task, and the other defines parameters for the RL training. An additional file for defining experiments with experiment-specific parameters, and running them was also created. In this way, multiple experiments could be defined and run at once on a compute cluster.

During a single episode, the simulator runs at 30 frames per second over the course of 150 or 300 total timesteps. While little environment variability is introduced, a straight corridor along with obstacles can be generated for the ant to navigate through, increasing the task's difficulty.

2.5.2 Training and Evaluation

In the training phase, millions of simulation steps are run. The state, action, and reward is recorded at each timestep and is used to compute a gradient update for the agent's policy. Our policy is represented by a neural network using the ResNet 18 architecture, which is trained by a PPO algorithm provided by the Habitat framework. Relevant PPO hyperparameters include the learning rate, clip parameter, reward discount gamma, action Gaussian noise, horizon size, number of training iterations, and the mini batch and PPO epoch sizes.

Once sufficient training has been completed, the results are evaluated. Reward measures are recorded at each training iteration and plotted on TensorBoard, allowing us to visualize training progress and compare multiple separate experiment configurations. Additionally, several videos are generated at each checkpoint which show the ant's movement patterns and allow us to better understand its progression. Several tools were developed to improve these videos, including visualizations of the ant's orientation vectors, path trajectory, and target pose at each moment.

3. RESULTS AND DISCUSSION

With the primary infrastructure for designing, running, and evaluating experiments setup, we set out to train the ant to solve increasingly challenging tasks. Due to the nature of RL problems, we recognized that out initial reward formulations, action controllers, and observations would need to be adjusted as experiments progressed. As results came back, it became clear that additional reward signals, visualizations, and hyperparameter tuning was needed.

3.1 Walking Forward

We began with the task which motivates OpenAI's Ant-v2 task; we aim to train the agent to move forwards as much as it can. With this in mind, we provided the agent with a majority of observations, the standard PPO hyperparameters defined in the Habitat framework, the relative position controller, and the X-location reward signal. The farther the agent is along the x-direction at any given step, the larger r_t would be. After several variations on this experiment, we observed the following behavior, shown in 3.1:

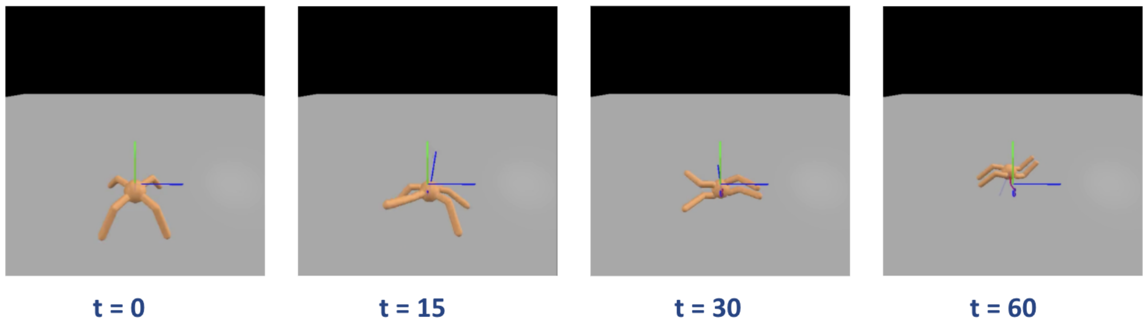


Figure 3.1: The agent learns that doing a somersault forward achieves a quick reward.

It appeared the agent learned that doing a forward somersault achieves a increase in x-location, consistently ending with a final x-location of around 2. While this is far from optimal, it did prove our methodology was working to some degree. This somersault behavior is best characterized as a local minimum. After some experimentation with hyperparameters, we were able to avoid this behavior by decreasing the PPO's clip parameter, as shown in Figures 3.2 and 3.3:

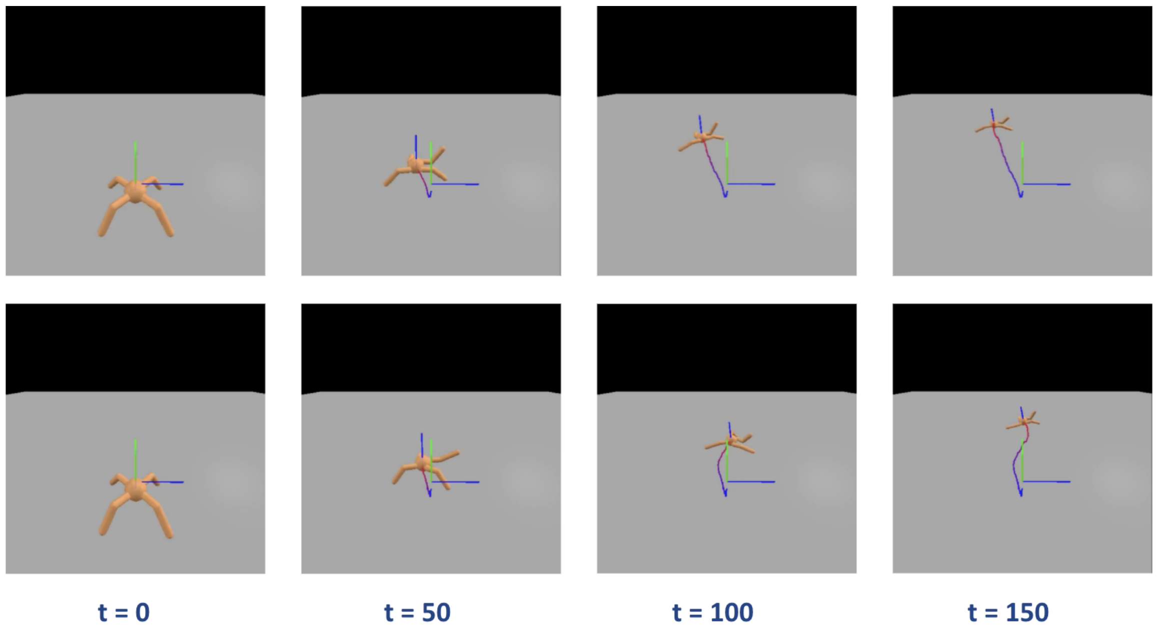


Figure 3.2: Examples of the agent's movements while walking forward.

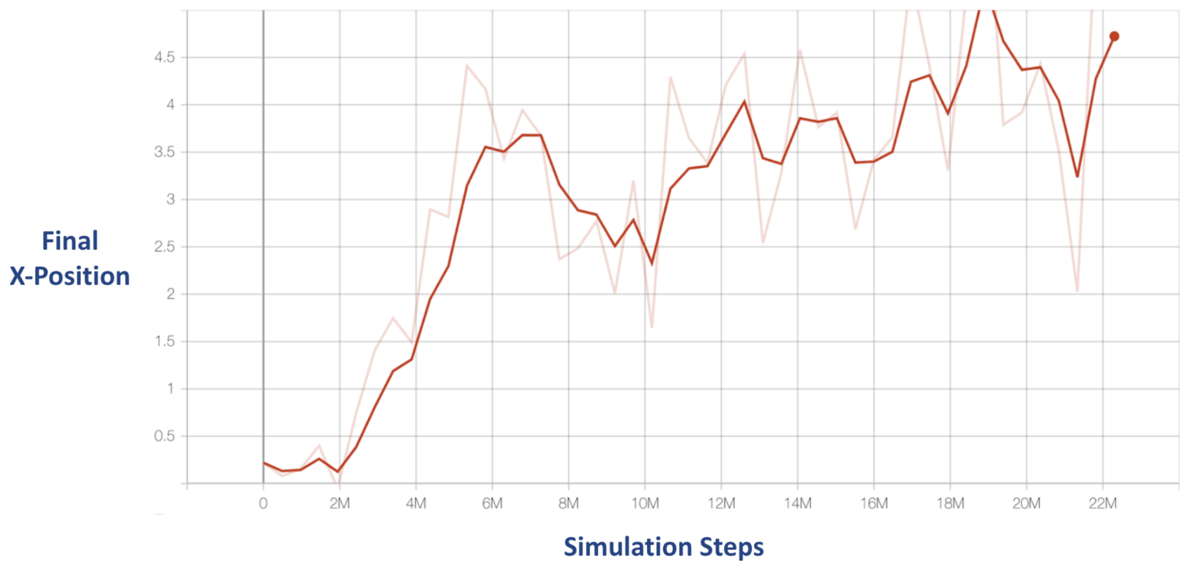


Figure 3.3: Distance traveled along the x-axis during training.

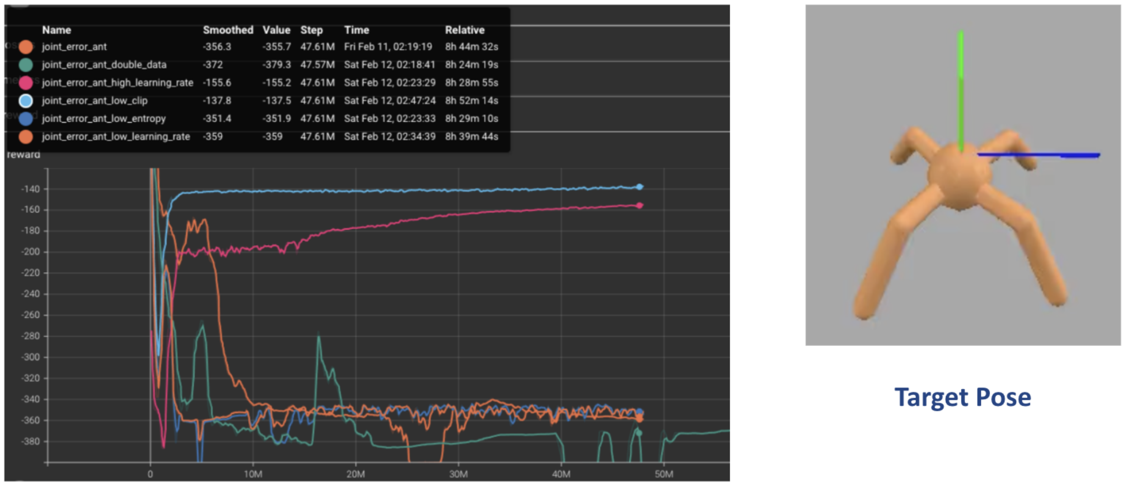
Improving upon the previous iteration, the ant was able to move to a final x position of about 4.5. However the observed motion appears unstable, inconsistent, and lacks the use of the ant's front-right leg. Rather than a natural walking motion, the ant learns to nudge its body forward with small, rapid movements of its back legs while raising its front-right leg in the air and using its front-left leg to orient itself forward. Despite the consistent movement in the x-direction, it seemed that this behaviour would not result in a robust walking motion resilient to perturbations and uneven terrain.

3.2 Joint State Regression

Based on the aforementioned results, it became apparent that designing a prior walking motion for the ant to learn and optimize was necessary. We turned our attention to a series of joint regression tasks which would allow us to better tune parameters and train the ant to achieve some target pose as well as a cyclical gait motion.

3.2.1 *Hyperparameter Tuning*

We used a simple task - training the ant to maintain its rest position - to validate our hyperparameters. We found that a lower clip parameter of 0.1 performed the best as shown in Figure 3.4.



	Name	Smoothed	Value	Step	Time	Relative
●	joint_error_ant	-356.3	-355.7	47.61M	Fri Feb 11, 02:19:19	8h 44m 32s
●	joint_error_ant_double_data	-372	-379.3	47.57M	Sat Feb 12, 02:18:41	8h 24m 19s
●	joint_error_ant_high_learning_rate	-155.6	-155.2	47.61M	Sat Feb 12, 02:23:29	8h 28m 55s
●	joint_error_ant_low_clip	-137.8	-137.5	47.61M	Sat Feb 12, 02:47:24	8h 52m 14s
●	joint_error_ant_low_entropy	-351.4	-351.9	47.61M	Sat Feb 12, 02:23:33	8h 29m 10s
●	joint_error_ant_low_learning_rate	-359	-359	47.61M	Sat Feb 12, 02:34:39	8h 39m 44s

Figure 3.4: Results from trying various hyperparameters for the Joint State Regression task. A low clip (light blue) and a high learning rate (pink) produced the best results.

3.2.2 *Flattened Pose*

We then turned out attention to training the ant to maintain a flattened pose with a low clip. We provided observations related only to the ant's joint configuration and history, and opted to use an absolute position controller based on previous testing. Reward Equations Eq. 6, Eq. 7, and Eq. 13 were all included in the composite reward measure and were equally weighted.

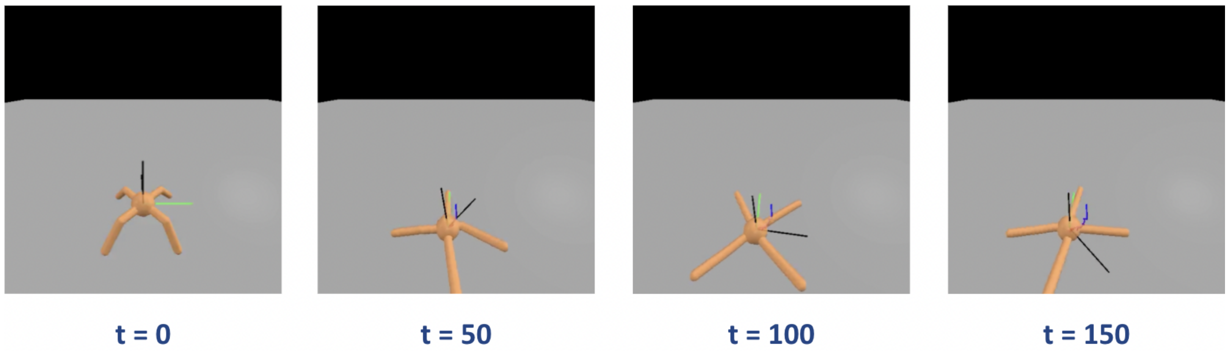
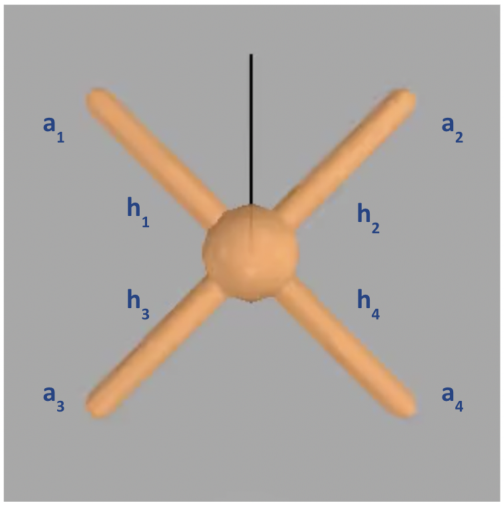


Figure 3.5: An episode after training the ant to maintain a flattened pose.

As shown in Figure 3.5, the ant was able to achieve a relatively stable flattened state and maintain it for the duration of the episode.

3.2.3 Natural Gait Imitation

We then moved on to designing a walking motion for the ant. DeepMimic, a paper published in 2018, showed that combining a motion clip with RL leads to natural motions which offer a "high degree of robustness without the need for human engineering"[11]. Inspired by observing the motion of other quadruped robots and animals, we designed a cyclical sinusoidal gait for the ant to follow.



$$h_1 = A_h * \sin(\pi + 2\pi t)$$
$$h_2 = A_h * \sin(\pi + 2\pi t)$$
$$h_3 = A_h * \sin(2\pi t)$$
$$h_4 = A_h * \sin(2\pi t)$$

$$a_1 = -(A_a * \sin(-\pi/2 + P\pi + 2\pi t) + O)$$
$$a_2 = -(A_a * \sin(\pi/2 + P\pi + 2\pi t) + O)$$
$$a_3 = A_a * \sin(-\pi/2 + P\pi + 2\pi t) + O$$
$$a_4 = A_a * \sin(\pi/2 + P\pi + 2\pi t) + O$$

Figure 3.6: A diagram showing the agent's 4 ankles, a_i, and its 4 hips, h_i, along with formulas which generate each joint's position at time t.

Figure 3.6 show a formula for each joint's position at a given moment. A_h, A_a, P, and O are all parameters which affect the ant's walking motion. After running a grid search on these 4 parameters, we found that $A_h = 0.775$, $A_a = 0.23$, $P = -0.26$, and $O = 0.77$ produce the best walking motion, with a final x-position of 5.2. This motion is pictured in Figure 3.7.

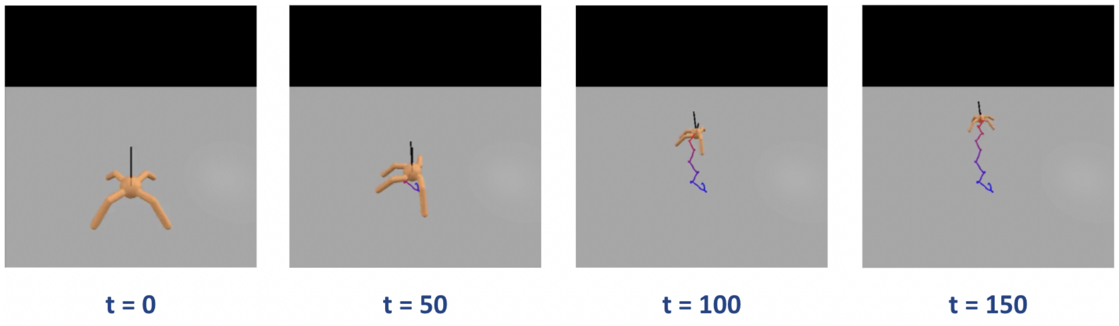


Figure 3.7: The motion of the ant's natural gait at different timesteps t.

While we could simply provide the cyclical gait motion as a direct control to the ant, it's important that the ant learns to generalize the motion to a wider range of states in order to develop a level of robustness to environmental noise. Thus, we used the same formulation as our flattened pose task, but set the target pose q_t^{tp} to be the natural gait we designed. We initially encountered poor training results with this formulation, and the ant seemed incapable of learning the the action well. After some investigation, we found that our PPO implementation was adding too much noise to the policy's output. While this noise is a neccessary feature of the PPO algorithm and accelerates the training process, we found that the parameter which controlled the maximum noise applied, max_std, was too high at a value of 1.0. After experimenting with different values for this parameter, we found that a value of 0.04 for max_std produced the best results, as shown below in Figure 3.8:

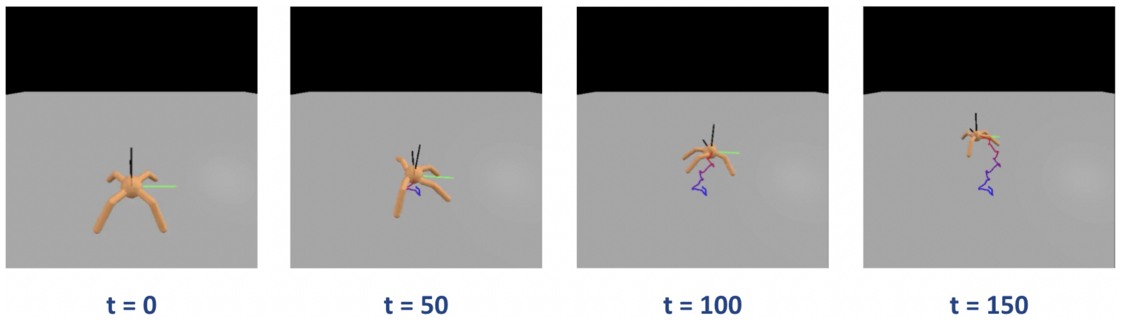


Figure 3.8: The motion of the ant's natural gait at different timesteps t using a max_std of 0.04.

The ant was successfully able to learn the walking motion with a reasonable level of consistency and stability.

3.3 Gait Imitation with Additional Tasks

With the ant properly learning the walking motion, we then experimented with increasing the complexity of the environment, task goals, and environment observations. We first added an orientation target in addition to training the ant to achieve the gait. We added a corridor for the ant to walk through, which would better emulate an indoor environment. We then ran several experiments toggling various features and providing additional tasks for the ant to complete while imitating the provided gait motion.

3.3.1 Orientation

In nearly every point-navigation situation, the agent is required to turn itself at various points. Thus, we aim to train the ant to orient itself with a specified direction while simultaneously replicating the gait. After several batches of experiments, we were unable to reproduce the gait in addition to achieving an orientation goal. Instead, we utilized the Target Pose Deviation action controller. Rather than providing the ant with a absolute position, we provide it with some amount to deviate from the natural walking motion. After several trials, we observed the following behavior, depicted in Figure 3.9:

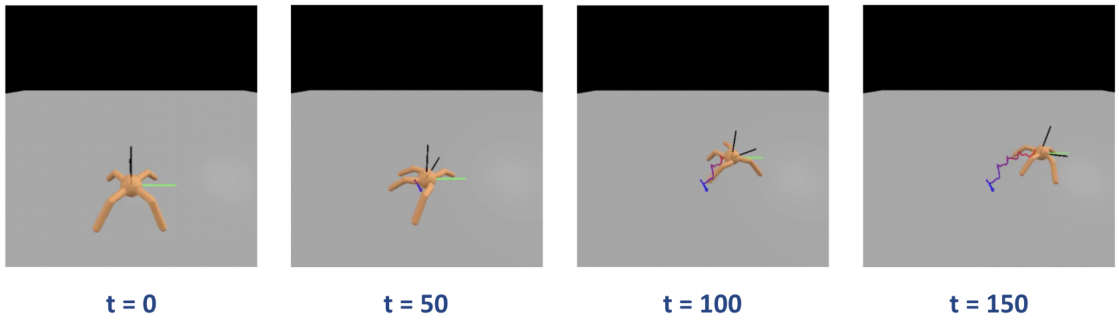


Figure 3.9: The ant successfully learns to orient itself to the right with the Target Pose Deviation action controller.

This experiment used the upright orientation (Eq. 1), the forward orientation (Eq. 1), the squared forward orientation (Eq. 2), the action cost (Eq. 11), and the action smoothness (Eq. 13) rewards, with reward weights of 1.0, 2.0, 4.0, 2.0, and 1.0 respectively. The ant appears to develop expected behavior for turning to the right - the motions of its left legs are more exaggerated while the motion of its right legs are minimized.

3.3.2 Visual Sensors

Visual sensors are also a critical component of point-navigation problems. Consequently, we implemented a first-person camera on the agent which gives the it significantly more information about its surroundings. In order to provide context to the ant, we generated a corridor and a green wall down at the end of the corridor. Finally, an extra blue ant which shows the target pose at the current timestep was created as an additional visualization. These are shown in Figure 3.10:

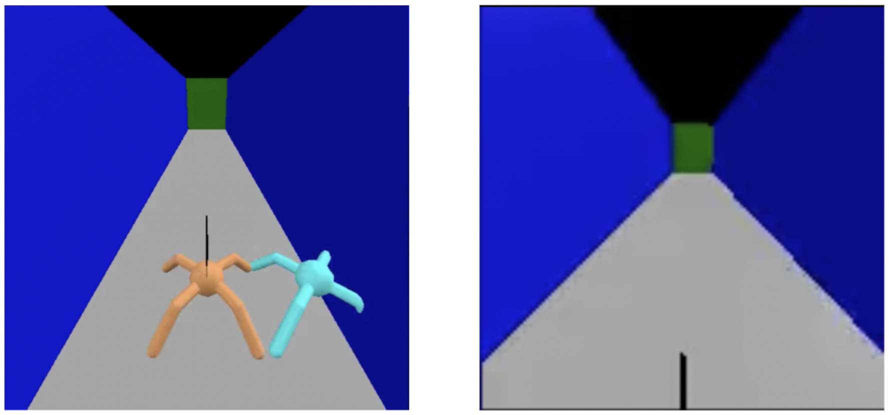


Figure 3.10: A corridor with blue and green walls was generated. A target pose visualizer is also depicted (left), along with the ant's first-person view of the corridor (right).

The introduction of vision drastically increases the complexity of the state s. While our current tasks don't necessarily require the ant to have vision, we include it as a proof of concept. If the agent can still learn with a 128x128 RGB image added to its state, it bodes well for future experiments which do require such information. We ran an experiment similar to the natural gait imitation task, but with an additional forward orientation reward to encourage the ant to maintain a more stable view. We observed the following behavior, as shown in Figure:

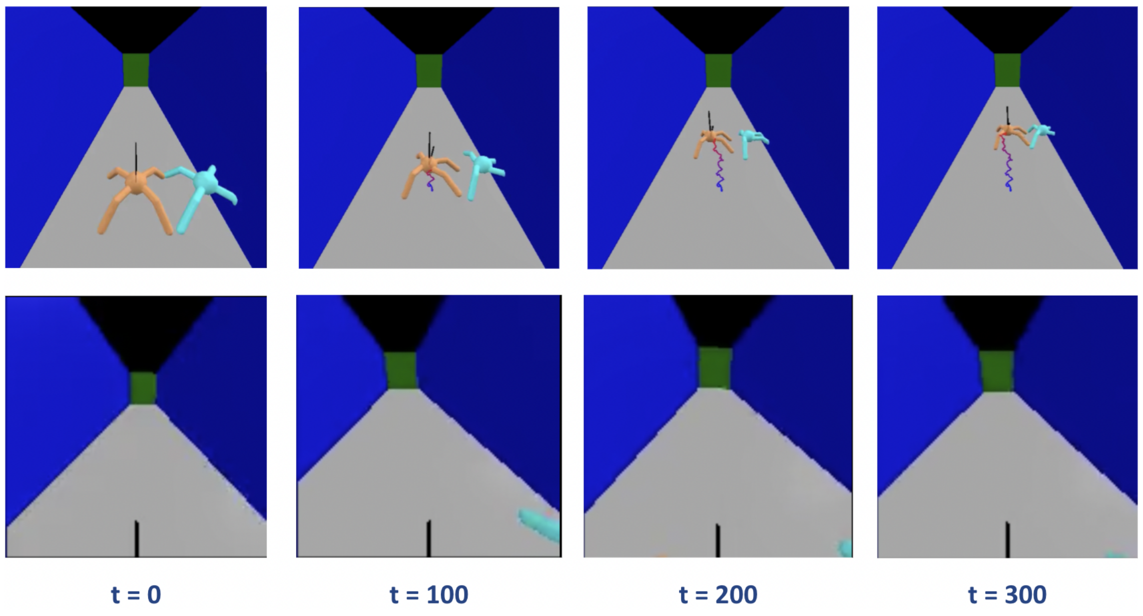


Figure 3.11: The ant navigates down the corridor while attempting to imitate the gait and maintain a forward orientation, all with vision of the corridor.

Using the absolute joint position controller and with vision introduced, the agent was successfully able to follow the natural gait while keeping its orientation stable. We can see that this stable orientation results in a consistent view of the corridor from the ant's perspective, a necessary feature of agents which reliably navigate via vision.

3.3.3 DeepMimic reward formulation

After several unsuccessful attempts at using the absolute position controller to imitate the natural gait while also achieving an additional task objective, we decided to reproduce some of the reward measures and methods used by DeepMimic[11]. Specifically, we replicated the target heading (Eq. 4), end-effector (Eq. 5), exponential pose (Eq. 9), and joint velocity (Eq. 10) rewards. Despite DeepMimic's successful results using these reward signals, we were unable to produce efficient controllers which imitate the natural gait. We suspect these will work with more hyperparameter turning, and are optimistic about generating more results in the future.

4. CONCLUSION

We were able to solve various tasks which serve as intermediary steps to our larger goal of solving point-navigation tasks with dynamic locomotion. We started out by training our agent to move forward. Recognizing the unstable controller this produced, we turned our attention to starting with a motion prior and using a combination of reward terms to emulate this motion with RL. After some hyperparameter tuning and multiple batches of experiments, the ant was able to imitate the gait we designed. We then added additional task objectives, including orienting the ant in a specific direction. Finally, vision was successfully added without any major performance impairments.

4.1 Future Work

As previously mentioned, work on this project will continue as part of Meta AI's EAI research. This includes mimicking the gait while obtaining additional task objectives such as orienting with a variable target heading and maintaining a specific speed. Additionally, with the introduction of vision, noise such as obstacles and uneven terrain will be added in an effort to emulate real-life environments. Finally, the ant will be loaded into a virtual indoor environment, such as a home, and given navigation tasks. The code written for this project and our findings will serve as a backbone for these research ventures.

4.2 Closing Remarks

Despite the complexity of the EAI problems that have yet to be solved, progress in this field has developed at a rapid pace. Our hope is that this project serves as another stepping stone which pushes the field of intelligent robotics further. Researchers are optimistic about creating robots which can solve the challenges of the 21^{st} century through the use of better EAI algorithms, increased compute power, and more efficient simulators.

REFERENCES

[1] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” 2016.

[2] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, “Habitat: A platform for embodied ai research,” 2019.

[3] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. M. Turner, N. D. Maestre, M. Mukadam, D. S. Chaplot, O. Maksymets, A. Gokaslan, V. Vondruš, S. Dharur, F. Meier, W. Galuba, A. X. Chang, Z. Kira, V. Koltun, J. Malik, M. Savva, and D. Batra, “Habitat 2.0: Training home assistants to rearrange their habitat,” in *Advances in Neural Information Processing Systems* (A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), 2021.

[4] L. Smith and M. Gasser, “The development of embodied cognition: Six lessons from babies,” *Artif. Life*, vol. 11, p. 13–30, jan 2005.

[5] “Ai habitat.” `https://aihabitat.org/`.

[6] “Habitat challenge 2020.” `https://aihabitat.org/challenge/2020`, Feb 2020.

[7] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, “Learning agile and dynamic motor skills for legged robots,” *CoRR*, vol. abs/1901.08652, 2019.

[8] K. Afsari, S. Halder, M. Ensafi, S. Devito, and J. Serdakowski, “Fundamentals and prospects of four-legged robot application in construction progress monitoring,” 2021.

[9] D. Kim, J. Carlo, B. Katz, G. Bledt, and S. Kim, “Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control,” 09 2019.

[10] R. Gordon, “3 questions: How the mit mini cheetah learns to run,” Mar 2022.

[11] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, “Deepmimic: Example-guided deep reinforcement learning of physics-based character skills,” *ACM Trans. Graph.*, vol. 37, pp. 143:1–143:14, July 2018.

[12] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra, "Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames," 2020.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[15] A. Kadian, J. Truong, A. Gokaslan, A. Clegg, E. Wijmans, S. Lee, M. Savva, S. Chernova, and D. Batra, "Sim2real predictivity: Does evaluation in simulation predict real-world performance?," *IEEE Robotics and Automation Letters*, vol. 5, p. 6670–6677, Oct 2020.

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[17] S. Srivastava, C. Li, M. Lingelbach, R. Martín-Martín, F. Xia, K. Vainio, Z. Lian, C. Gokmen, S. Buch, C. K. Liu, S. Savarese, H. Gweon, J. Wu, and L. Fei-Fei, "Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments," 2021.

[18] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2018.