CS562: Advanced Robotics - Project

TA: Dhruv Metha Ramesh
dhruv.metha@rutgers.edu

Perfect score: 100.

Project Instructions:

Teams: Assignments should be completed by teams of up to three students. You can work on this assignment individually if you prefer. No additional credit will be given for students that complete an assignment individually. Make sure to write the name and RUID of every member of your group on your submitted report.

Submission Rules: Submit your files through Canvas (canvas.rutgers.edu). For programming questions, you need to also submit a compressed file via Canvas, which contains your code. Each team of students should submit only a single copy of their solutions and indicate all team members on their submission. Failure to follow these rules will result in lower grade for this assignment.

Program Demonstrations: You will need to demonstrate your program to the TA on a date after the deadline. The schedule will be coordinated by the TA. During the demonstration you have to use the file submitted on Canvas and execute it on your personal computer. You will also be asked to describe the architecture of your implementation and key algorithmic aspects of the project. You need to make sure that you are able to complete the demonstration and answer the TA's questions within the allotted 10 minutes of time for each team. If your program is not directly running on the computer you are using and you have to spend time to configure your computer, this counts against your allotted time.

Late Submissions: No late submission is allowed. 0 points for late assignments.

Collusion, Plagiarism, etc.: Each team must prepare its solutions independently from other teams, i.e., without using common notes, code or worksheets with other students or trying to solve problems in collaboration with other teams. You must indicate any external sources you have used in the preparation of your solution. Do not plagiarize online sources and in general make sure you do not violate any of the academic standards of the department or the university. Failure to follow these rules may result in failure in the course.

Introduction

Quadruped navigation and obstacle avoidance involve the critical tasks of guiding a robotic dog through its environment while ensuring it can safely navigate around obstacles. Quadruped navigation and obstacle avoidance are pivotal in robotics. In search and rescue, they reach survivors in inaccessible areas. In agriculture, they automate tasks and improve crop management. In logistics, they efficiently deliver goods in urban settings. In healthcare, they aid patients with mobility issues. These robots enhance safety and productivity across industries.

Figure 1: Real Unitree Go1 at the Rutgers Robotics lab

Figure 2: Simulated Unitree Go1 in Isaac Gym

The primary objective is to enable the robot dog to move from point A to point B within the environment while avoiding collisions with any obstacles that may be present in its path. The project will be developed on the Isaac Gym Simulator [1] using the Unitree Go1 robot as seen in Fig 2. Simulators allow robotic engineers to develop and experiment with robots in a digital environment before deploying them in the real world. This saves time, and money, and ensures safety.

Simulator Isaac Gym simulator is a physics-based simulator wherein, the physics simulation and the neural network policy training reside on the GPU. This leads to highly parallel and fast training of reinforcement learning policies.

Legged Robot The Unitree Go1 is an economical legged robot that has 12 revolute joints – each of the 4 legs comprises a hip joint, thigh joint, and calf joint. Since these are revolute joints, the joint position is just an angle and joint velocity is just the angular velocity. These joints are controlled by applying a torque on them, which means that this enables movement of the legs.

The project will involve setting up the environment in the simulator, building an interface (gym) to interact with it, and training neural networks using model-free reinforcement learning. The project will be done in 3 phases which will be explained more in the following sections.

1 Phase 1: Robot Locomotion (30 points)

Deadline: 18th October, 2023, 11:55pm.

1.1 Overview

In the first phase of the project, your goal will be to train a velocity-conditioned neural network policy $\pi_\theta(.)$ with θ as parameters, that take as input sensory data and velocity commands and gives as output joint position commands, which are then converted to joint torques by a PD controller. This will enable the robot to walk at the commanded longitudinal (v_x^{cmd}), lateral (v_y^{cmd}), and angular (ω_x^{cmd}) velocities. To simplify, the robot is given velocity commands, and it needs to walk with a stable gait at the commanded speeds.

The sensory data comprises of joint positions $q \in \mathbb{R}^{12}$, joint velocities $\dot{q} \in \mathbb{R}^{12}$ and g which denotes the orientation of the gravity vector in robot's body frame. Similarly, the output (action) $a_t \in \mathbb{R}^{12}$ of the policy is the joint position commands. The input to the policy is a history of length H_{loc} of the sensory data and the actions taken, denoted by $o_{t-H_{\text{loc}}:t}$, where $o_t = (q_t, \dot{q}_t, g_t, a_{t-1})$. Since the policy is velocity commanded, the complete input to the policy is denoted by $x_{t-H_{\text{loc}}:t}$, where $x_t = (o_t, v_t^{\text{cmd}})$

$$a_t = \pi_{\text{loc}}(x_{t-H_{\text{loc}}:t}) \tag{1}$$

Following the work done in [2, 3], you will use a teacher-student architecture to train the policy. For more details, please refer to the paper and try to get a deep understanding of it. Once we have the locomotion policy π_{loc}, we will be able to use it for robot navigation.

1.2 Setup

We will begin this phase by installing the Isaac Gym simulator either on your preferred device or on the ilab machines. You will find the simulator here. Please make sure to install and test it as per instructions provided in the documentation of the simulator. I highly recommend using the conda installation. If you are

installing this on ilab, use the conda installation only. Whenever you activate the environment follow it with these commands one by one:

```
export LD_LIBRARY_PATH=/common/home/NETID/.conda/envs/ENV_NAME/lib
```

```
export VK_ICD_FILENAMES=/usr/share/vulkan/icd.d/nvidia_icd.json
```

To see the graphical interface, do not use SSH. Either access the ilab machines through the web or use remote desktop connection.

Once you have your simulator working, clone the walk-these-ways repository. Run the file `scrips/play.py`. This should spawn a GUI and you should see a few robots walking around.

1.3 Implementation

Your objective in this phase is to train the policy such that the robot can only follow commanded velocities in the range $[-0.5\text{m/s}, +0.5\text{m/s}]$ in any direction (forward, backward, left, right, clockwise, or anti-clockwise). This will familiarize you with the simulator, the code base and the robot's functioning.

1.4 What you will submit/demonstrate

You will do the following:

1. Record a video (use the camera attached to the 0th environment in simulation) and present trajectories of the robot moving at different velocities in simulation. Using the `scripts/play.py` file, record in succession for 50 timesteps each the robot moving at 0.4m/s in the forward direction, 0.3m/s in the left direction, 0.3m/s in the backward direction, 0.5m/s in the right direction and left rotation at 0.25m/s. The entire video will comprise of 250 timesteps.

2. For the entire trajectory above, plot the velocity of the robot detected by the simulator (the variables `base_lin_vel` and `base_ang_vel`) and a black dotted line for the velocity it is commanded to be at. Use three different plots, one for forward-backward, one for left-right, and one for rotation.

Compile the above in a presentation that you will submit before the deadline along with the code (zipped) on Canvas. This is the presentation you will be going over during the 10 minute demonstration with TA Dhruv.

2 Phase 2: Robot Navigation within walls (50 points)

Deadline: 15th November, 2023, 11:55pm.

2.1 Overview

From Phase 1, the policy π_{loc} facilitates the robot's motion using stable gaits. This enables us to do downstream tasks, for example, navigation. You will train a policy $\pi_{\text{nav}}(.)$ using reinforcement learning for navigating a walled corridor from start position A to goal position B at distance of K from A within a time limit T_{max}, such that the robot does not collide with the wall. There are no obstacles in this phase.

The policy will provide these velocity commands to the locomotion policy and thus facilitate motion.

$$\hat{v}_t^{\text{cmd}} = \pi_{\text{nav}}(.) \tag{2}$$

where $\hat{v}_t^{\text{cmd}}$ is command velocity which can be used as a part of the input in x_t from Equation 1.

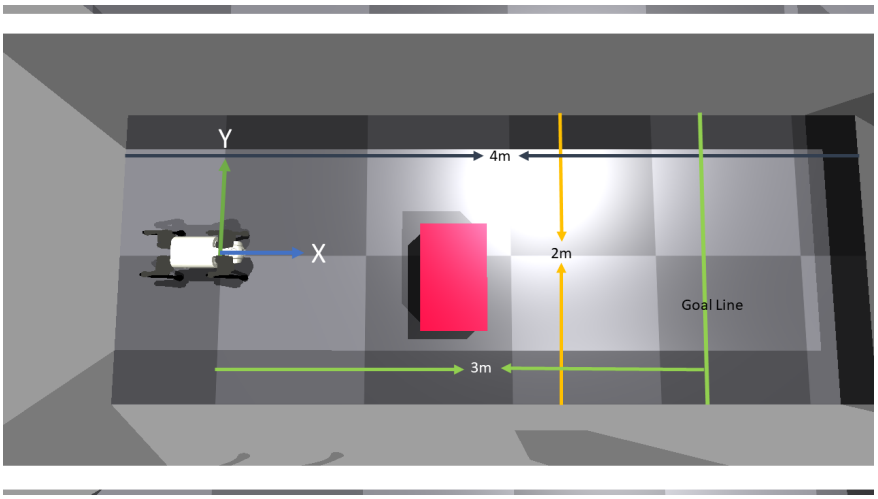

Figure 3: Environment Setup in simulation

2.2 Implementation

1. Implement the navigator class in a `navigator.py` file (with its own configuration file `navigator_config.py`, much like the locomotion code) that behaves like a gym for you similar to the `legged_robot.py`. This should contain the `step` function that commands the locomotion policy to move the robot. The distance between A and B is defined to be $K = 3$m. Use any gait you like. Use the code from the existing `scripts/play.py` file to command the robot.

2. In each environment, the robot is placed at point A which is the origin $(0, 0)$, and pointing towards the goal (0 rotation). Your task will be to

Robot	Ranges
Location along x-axis	[0.0m, 0.8m]
Location along y-axis	[-0.5m, 0.5m]
Orientation (y-axis)	[-pi, pi]

Table 1: Robot initial location

set up this environment to contain walls that are 4m long and are at a distance of -1m and $+1$m from the robot's origin in the y-direction (left and right). For more setup details please refer to the Figure 3. You can use the `box` asset for this. Make sure they behave like walls and do not move when a force is applied to them (look at the asset properties for this). You should do this by creating a new file called `world.py`.

3. Implement the Actor-Critic Proximal Policy algorithm (PPO) [4] (which exists in the locomotion policy codebase, take inspiration from it), define your input space, design your own reward function, and train the robot to navigate without colliding with the walls. Consider the max episode length $T_{\max} = 750$ timesteps of the navigation policy. You should train the model on many environments in parallel.

4. Now that you have a trained policy with a fixed initial position (0, 0) and orientation (0 rotation). Train a new policy such that the robot's initial position and orientation is randomized over the range as given in Table. 1. Once trained, this will yield a policy such that given any random location and orientation of the robot in the ranges provided, the robot should still be able to navigate to the goal.

2.3 What you will submit/demonstrate

You will do the following:

1. Record 1 trajectory of the robot navigating to the goal in simulation using policy trained on the fixed initial position.

2. Record 3 trajectories of the robot (random initial position) navigating to the goal in simulation using policy trained with the random initial position.

3. Plot a curve for the the total reward during training (should be an upward curve). If you use a combination of rewards functions, plot each of their values as well. If you use scaling on your reward functions, mention that as a table in your presentation. Do this for each of the two policies. These plots will be reward value v/s training iteration.

4. Since Isaac gym enables training in parallel, plot a curve showing the success rates during training. The idea here is to collect data points of finished episodes and record the number that succeeded overall. (Failures

will only occur if the episode is completed, and the robot has not reached the goal. Remember to count them). Do this for both policies.

Compile the above in a presentation that you will submit before the deadline along with the code (zipped) on Canvas. This is the presentation you will be going over during the 10 minute demonstration with TA Dhruv.

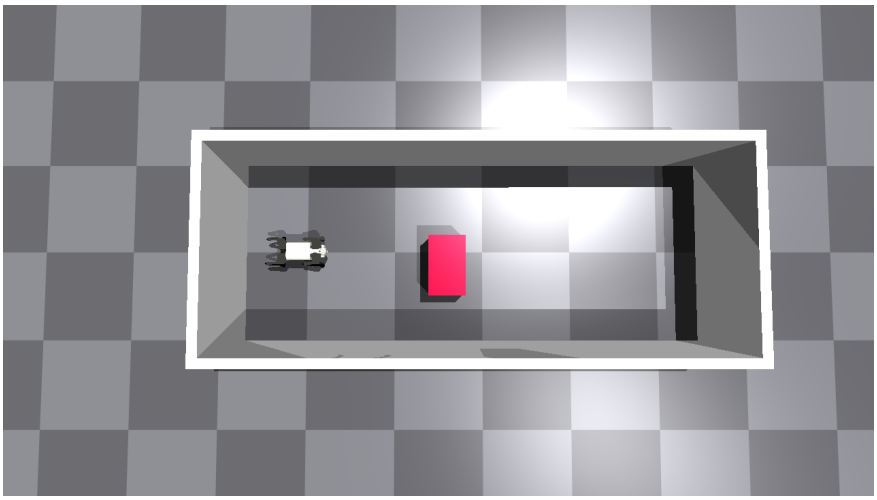

Figure 4: Example layout in simulation

3 Phase 3: Robot Locomotion with obstacle avoidance (20 points)

Deadline: 1st December, 2023, 11:55pm.

3.1 Overview

In this phase, you build a new navigation policy $\pi_{\text{nav}}^{\text{obs}}(.)$ with obstacle avoidance capabilities. This policy should include information about the obstacle's location and size in the entire environment which enables the policy to learn to avoid the obstacles present.

3.2 Implementation

1. Using the existing environment defined in Phase 2, add to this an obstacle that cannot be moved when a force is applied to it. Use the `world.py` file (Optional: Color the obstacle red). The obstacle's location and size can be varied based on Table 2.

Properties	Ranges (in m)
Location along x-axis	[0.8, 2.5]
Location along y-axis	This depends on the size, make sure the obstacle does not collide with the wall
Size - length (y-axis)	[0.3, 1.5]
Size - width(x-axis)	0.4

Table 2: Obstacle location and size ranges

2. Follow step 3 from Phase 2. In the input space also mention the location and size of the obstacle. Thereby enabling the policy to understand the location and size of the obstacle such that the policy can learn to avoid the obstacle. **NOTE: Use only the fixed initial location of the robot for this phase.**

3.3 What you will submit/present

You will do the following:

1. Record 3 trajectories of the robot (fixed initial position) with random obstacle locations navigating to the goal in simulation using the trained policy.

2. Plot a curve for the the total reward during training (should be an upward curve). If you use a combination of rewards functions, plot each of their values as well. If you use scaling on your reward functions, mention that as a table in your presentation. These plots will be reward value v/s training iteration.

3. Since Isaac gym enables training in parallel, plot a curve showing the success rates during training. The idea here is to collect data points of finished episodes and record the number that succeeded overall. (Failures will only occur if the episode is completed, and the robot has not reached the goal. Remember to count them).

Compile the above in a presentation that you will submit before the deadline along with the code (zipped) on Canvas. This is the presentation you will be going over during the 10 minute demonstration with TA Dhruv.

References

[1] 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, 2021.

[2] Gabriel B Margolis and Pulkit Agrawal. Walk these ways: Tuning robot control for generalization with multiplicity of behavior, 2022.

[3] Gabriel B Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning, 2022.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.