Humanoid Control Design

Thang Phan Aarushi Shah Zachary Adler

November 22, 2019

1 Research Context and Problem Statement

As the world becomes more automated day by day, various problems in robotics are being addressed, in particular, the autonomous balancing and walking of a bi-pedal humanoid robot. Several machine learning methods, especially Reinforcement Learning (RL), are being explored by researchers to potentially be an efficient method to teach a bi-pedal humanoid robot to autonomously walk. Researchers initially train the robot in a simulator using RL algorithms, and then attempt to transfer their model to the real world. However, the "reality gap" between the simulation and the real world often causes inaccuracies while working with physical robots. If the simulation does not accurately represent metrics such as mass, height, friction, etc. the physical robot may fail in walking even though the simulation succeeds. Therefore, the simulation must be created as accurately as possible in order to reduce the reality gap between the simulation and physical robot.

Reinforcement Learning (RL) is a branch of machine learning containing algorithms that teach a machine how to make a certain decision. RL can be thought of as a game, where the agent, which is the decision making component, uses trial and error to test several different possible steps to reach a complex goal. The agent will be rewarded for performing successful steps, and the objective of the agent is to maximize their reward. The agent will start in the beginning of its journey knowing nothing about its environment, and through trial and error, will learn the optimal solution to its complex problem. The solution the agent comes up with is called the control function. The control function tells the robot the next action to take when given the current state of the machine.

For bi-pedal robot walking, a fundamental step is first using RL to teach the robot to stabilize itself and stand upright. In order for this to occur, the robot must use RL algorithms to obtain a control function that will output behaviors the robot will take to stabilize itself and prevent itself from falling over. These algorithms have been tested and proven in various simulation environments. However, the barrier lies between simulation and practical use of these algorithms. RL algorithms may be able to train robots in simulation, however in real life there are several other factors to consider. RL algorithms require millions of trials in order to train the model and obtain a control function that

will guarantee stability for the system. These trials can be harmful in practice, as they are inefficient to finding a correct control function and may lead to trials that damage the robot. For example, while learning to stay upright, the robot may go through trials causing it to lose balance and fall over, leading to damaged hardware. Conducting these trials also requires massive data collection on the robots, which can be costly. Consequently, RL algorithms have not been able to be used in practice just yet.

One approach to accommodate this need for vast data quantity is to design RL algorithms that are less data-intensive. These RL algorithms are model-based, meaning they apply control function learning on previously explored models of the environment. This requires less data collection, as most of the learning has already been done with the previous models. However, model-based reinforcement relies greatly on the accuracy of the previously learned model, and leads to limited results. If the previous models are not accurate, the model-based control function could result in failures and harmful behavior for the robot. [2]

A second approach to the data problem involves transferring the control function obtained in simulation to the physical robot. The idea is to first train using RL in simulation, where we can perform several trials and collect additional data as much as is necessary to find the control function. Once the control function is obtained, which allows for stability in the robot, that function is transferred over to the physical robot to achieve similar results. This will be possible only if the simulation environment created is very similar in nature to the environment of the physical robot. When important features such as friction, mass, and density are not accurately modeled, the gap in performance between simulated and real environments is large. As a result, a simulation environment that closely represents the physical robot and the environment surrounding it will need to be created, and the RL algorithms will be applied to the simulation to train the system and obtain a learned control function. Then, the next step is to create a transfer algorithm that will apply the control function from the simulation to the physical robot, and achieve similar results.

This work is similar to that of Turk C. et.al [3], however our work focuses on applying the simulation to real transfer algorithms to the problem of humanoid balancing versus the robotic arm problem.

To obtain this transfer algorithm, our group will first create a simulation environment that will closely represent a physical humanoid robot. After running RL algorithms on the simulation environment, we will explore the transfer of the learned function to the humanoid robot and test for similar behavior between the simulation and physical environments.

2 Proposed Solution

We aim to solve the problem of transferring a successful, control function from simulation to a physical robot and observing similar results. Our proposed so-

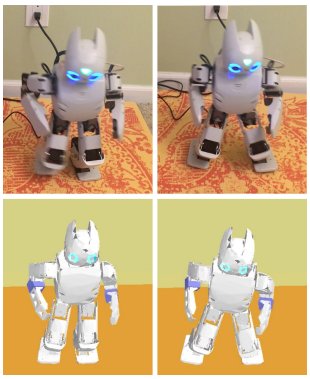

Figure 1: Sim-to-real transfer of biped locomotion[3]

lution is twofold.

To begin, we must train a bipedal robot walker in a simulation environment that is similar to the physical environment we will test the physical robot in. This includes accurately representing the weight of the robot, measurements of its parts, and accounting for motors and sensors. We will first create a simulation environment of a bipedal walker. To create a realistic simulation, we will represent the physical requirements of the bipedal walker including capacity of the motors, size of joints, and power supply. The structure of our simulation robot will be a simple rectangular body, with two legs, made up of two joints connecting the top and bottom halves of the legs. In addition, the physical bi-pedal robot will need sensors on its legs in order to collect information about its current joint angles and velocity. This information will be used as input to the control function, where the output will be the next action for the robot to take. We will account for these sensors in our simulation, and place one sensor on each of the two legs of the robot. In addition to our robot representation,the simulated environment the robot will be in will closely be modelled after the real world in terms of terrain, gravity, and physics.

We will create this environment using a physics engine called PyBullet. PyBullet is an easy-to-use Python module for physics simulation, robotics, and deep RL. PyBullet is open-source, which makes it easier to adapt, implement, and share algorithms. PyBullet accepts several different robot description file formats, however URDF is the simplest one, which we will be implementing. URDF is the standard for Robot Operating System and stands for Universal Robot Description Format. Through its XML format, we will specify the structure of the bipedal walker, including the exact size of its parts, the type of motors, position of sensors, etc.

In such a simulation environment, we will apply RL algorithms to train a bipedal robot and obtain a control function that will demonstrate success in controlling the simulated robot. Notably, we will apply domain randomization in our implementation of RL. Domain Randomization involves training over many

environments that vary slightly from each other in order to increase our odds of encompassing the true physical environment.[4] Parameters we change depending on the environment the robot is in include the initial angles and velocities of the legs. If our trials are successful, walking will subsequently be accomplished in the real world to a greater degree of success and with less dependence on the initial position of the walker.

Afterwards, the generated control function will be taken and implemented it in a physical robot. We will devise an online learning algorithm to train the physical robot as it runs its trials. We will measure the success of our transfer algorithm based on the number of attempts the robot takes before it can successfully walk. If the physical tests fail, we will reevaluate and retrain our algorithm in simulation. If the physical tests succeed, we will analyze our results and provide a confidence metric.

3 Evaluation and Implementation Plan

3.1 Evaluation Methods

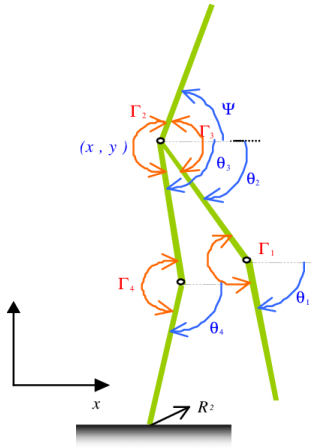

Figure 2: Representation of robot legs [1]

Success of a simulated environment is determined by the robot's performance difference between simulation environment and in the real environment. Given that the control function successfully walks the robot in simulation, we aim to produce similar outcomes in the real word that varies as little as possible from simulated performance. To measure this difference, there are two key pieces of data: angular displacement and angular velocity of leg motors (see Figure 2). Simulated and real motors in the hips and knees provide angular displacement and angular velocity as state information. To condense and evaluate this information, consider a list of simulated data points, S, and a corresponding list of real data points, R, where each element in S and R consists of a pair

of displacement and velocity. To minimize the "reality gap" is to minimize $G(R, S) = Sum(|R - S|)$, where each operation on the lists is element-wise. Over multiple simulation environments, we select the environment that provides performance most similar to the real-world. Growlithe @ Eviolite Level: 5 Ability: Intimidate EVs: 76 HP / 36 Atk / 156 Def / 196 SpD / 36 Spe Impish Nature - Will-O-Wisp - Morning Sun - Flare Blitz - Wild Charge

3.2 Timeline

Fall Quarter 2019

Weeks 7-10

1. Experiment with PyBullet Simulations
2. Read papers on simulation to real life transfer algorithms
3. Create basic robot structure on PyBullet

Winter Quarter 2020

Weeks 1-3

1. Experiment with PyBullet Simulations
2. Read papers on simulation to real life transfer algorithms
3. Create basic robot structure on PyBullet

Weeks 4-6

1. Finish first three rounds of simulation of robot
2. Apply RL Algorithms on simulation to obtain control functions

Weeks 7-10

1. Finish last three rounds of simulation of robot
2. Apply RL Algorithms on simulation to obtain control functions

Spring Quarter 2020

Weeks 1-3

1. Experiment with transfer algorithms
2. Decide with transfer algorithm to implement

Weeks 4-6

1. Program transfer algorithm using PyTorch
2. First round test of transfer algorithm on physical robot
3. Make necessary changes to simulation based on results

Weeks 7-10

1. Second round test of transfer algorithm on physical robot
2. Make necessary changes to simulation based on results
3. Work on poster to display final results

References

[1] Chevallereau Christine and Philippe Sardain. Design and actuation optimization of a 4-axes biped robot for walking and running. volume 4, pages 3365 – 3370 vol.4, 02 2000.

[2] John Schulman Yasuhiro Fujita Tamim Asfour Pieter Abbeel Ignasi Clavera, Jonas Rothfuss. Model-based reinforcement learning via meta-policy optimization. *arXiv arXiv:1809.05214*, 2019.

[3] Greg Turk C. Karen Liu Wenhao Yu, Visak CV Kumar. Sim-to-real transfer for biped locomotion. *arXiv arXiv:1903.01390*, 2019.

[4] Wojciech Zaremba Pieter Abbeel Xue Bin Peng, Marcin Andrychowicz. Sim-to-real transfer of robotic control with dynamics randomization. *arXiv arXiv:1710.06537v3*, 2017.