Push Recovery and Active Balancing for Inexpensive Humanoid Robots

by

Amirhossein Hosseinmemar

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
August 2019

© Copyright 2019 by Amirhossein Hosseinmemar

Thesis advisor
Jacky Baltes and John E. Anderson

Author
Amirhossein Hosseinmemar

Push Recovery and Active Balancing for Inexpensive Humanoid Robots

Abstract

Active balancing of a humanoid robot is a challenging task due to the complexity of combining a walking gait, dynamic balancing, vision and high-level behaviors. My Ph.D research focuses on the active balancing and push recovery problems that allow inexpensive humanoid robots to balance while standing and walking, and to compensate for external forces. In this research, I have proposed a push recovery mechanism that employs two machine learning techniques, Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) to learn recovery step trajectories during push recovery using a closed-loop feedback control. I have implemented a 3D model using the Robot Operating System (ROS) and Gazebo. To reduce wear and tear on the real robot, I used this model for learning the recovery steps for different impact strengths and directions. I evaluated my approach in both in the real world and in simulation. All the real world experiments are performed by Polaris, a teen-sized humanoid robot in the Autonomous Agent Laboratory (AALab), University of Manitoba. The design, implementation, and evaluation of hardware, software, and kinematic models are discussed in this document.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to begin by thanking my supervisors Dr. John Anderson and Dr. Jacky Baltes who have helped me on this journey without any hesitations. I have been very grateful to have you both as my knowledgeable supervisors in my doctoral degree. Each of you have mentored me in academia in a different way and made me the person who I am right now, whom I am proud of. A big thank you to both of you.

I would also like to thank my committee members, Dr. Bob McLeod, Dr. Yang Wang, and Dr. Ching Chang Wong, who ultimately had to spend many hours reviewing my thesis and gave me constructive feedback.

My next thank you goes to the members of the Autonomous Agents Laboratory (AALab) who have helped me in various ways. I would like to specially thank Dr. Meng Cheng Lau, Ziang Wang, Chi Fung Lun, Jamillo Santos, Joshua Jung, and Christopher James Iverach-Brereton.

A very special thanks goes to my father Saeid Hosseinmemar, my mother Mahboubeh Esfahani, and my sister Delaram Hosseinmemar, without whom getting to the point that I am right now was impossible. Thank you for supporting me mentally and financially for many years. I love you all.

I would like to thank my lovely wife Ashley Rogoski who I always told to wait for 5 more minutes outside of AALab and I went to her after 2 hours. You could of shown me an angry face but you did not. Thank you for understanding and believing in me. Also a thank you to my son Nicholas Saeid Hosseinmemar who did not let me sleep at nights and forced me to work on my thesis. I love you both.

Last but not least, I would like to thank my father in law Richard Leonard Rogoski,

my mother in law Kimberly Diane Joanne Rogoski, and grandmother in law Rita Bartmanovich who have helped us a lot in various ways and helped me to finish my doctoral degree.

This thesis is dedicated to my lovely family.

Glossary

3D three-dimensional.

A3C Asynchronous Advantage Actor Critic.

AI Artificial Intelligence.

ANN Artificial Neural Network.

CM Centimetre.

CMP Centroidal Moment Pivot.

COM Centre Of Mass.

COP Centre Of Pressure.

DDQN Double Deep Q-Learning.

D-H Denavit-Hartenberg convention.

DNN Deep Neural Network.

DOF Degrees Of Freedom.

DQN Deep Q-Learning.

DRL Deep Reinforcement Learning.

DRLPR Deep Reinforcement Learning Push Recovery.

FA Function Approximation.

g The standard gravity at the Earth's surface is 1g.

Gbps Gigabits per second.

GPU Graphical Processor Unit.

HTM Homogeneous Transformation Matrix.

IMU Inertial measurement unit.

Kg Kilogram.

LIPM Linear Inverted Pendulum Model.

M Meters.

ML Machine Learning.

PD Proportional Derivative.

PID Proportional Integral Derivative.

RL Reinforcement Learning.

RLPR Reinforcement Learning Push Recovery.

ROS Robot Operating System.

TTL Transistor-Transistor Logic.

USB Universal Serial Bus.

VCC Voltage Common Collector.

ZMP Zero Moment Point.

Chapter 1

Introduction

1.1 Introduction

This chapter discusses the motivation for my thesis. It also introduces push recovery for a humanoid robot that I developed at the AALab. This chapter also gives an overview of the methods that I used to conquer the problems of current approaches to push recovery of humanoid robots.

1.2 Motivation

Robots have been used for decades, but moving them from factory floors to the everyday lives of humans is a challenging task. Traditional wheeled robots are more stable and balanced than humanoid robots [Huang et al., 2001]. Also, they can avoid obstacles easily and generate new paths [Shojaeipour et al., 2010]. However, humanoid robots are closer to a human's body shape, and as a result, they can

potentially function better in a human environment. One of the main issues with humanoid robots is balancing: humanoid robots fall easily. So, for robots to walk like humans, a humanoid robot needs to be able to walk stably on various surfaces such as hard terrain, artificial turf, snow, stairs, up and down slopes of varying degrees, all while avoiding obstacles in its path. If a humanoid robot has a collision with an object, an external force will be applied to the robot, which typically causes the robot to fall. Falls are a common cause of injury in humans, and similarly can be very damaging to humanoid robots.

The majority of existing walking algorithms are designed to walk on flat surfaces at a predefined static angle to the ground and with feet parallel to it [Kim et al., 2007]. For this reason, with every external push to a humanoid robot during its walk, there is a very high chance that the robot will fall. To overcome this problem, many different strategies have been introduced. Among these strategies, there are three well known methods: 1) Centre Of Pressure (COP) [Assman et al., 2012], which uses the ankles of the robot for balancing by changing the Centre Of Mass (COM) [Graf and Röfer, 2012; Stephens, 2011]; 2) Centroidal Moment Pivot (CMP) [Stephens, 2007], which uses both the ankles and hips of the robot for balancing, allowing the body to change position and create an angular momentum by swinging the torso; and 3) the Capture step [Missura and Behnke, 2015] approach, which handles external forces by taking one or more extra steps in an appropriate direction.

Humans not only walk successfully over even and uneven terrain, but can recover from the interaction of external forces such as impacts with obstacles and active pushes. While push recovery has been demonstrated successfully in expensive robots

[Kuindersma et al., 2016; Feng et al., 2014; Stephens, 2011], it is more challenging with robots that are inexpensive, with limited power in actuators and less accurate sensing [Hosseinmemar et al., 2018]. The motivation of this work is to adopt affordable robotics devices and technologies to build a humanoid robot that is able to recover from external pushes similar to humans. Building a humanoid robot with less expensive equipment means that all elements of software, from vision to control, must be much more robust [Hosseinmemar et al., 2019].

In my research, I focused on push recovery and active balancing of an inexpensive (approximately USD $20,000) humanoid robot named Polaris, with a height of 95 CM, by taking recovery steps after the robot receives external pushes. Polaris is built from relatively inexpensive servo motors (when compared to modern industrial robots), making balancing and push recovery especially challenging. For the first stage of push recovery for Polaris, I implemented a hand-tuned closed-loop control that could recover from 100% of light pushes and 70% of strong pushes [Hosseinmemar et al., 2018]. The objective of this research is to design and implement a fully autonomous closed-loop push recovery control system that can mimic a human's recovery actions. In order to make the system fully autonomous, I used two machine learning techniques: Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) [Sutton and Barto, 1998] to find different walking trajectories of one or more steps (capture step) after an external push is applied to the robot. By using these techniques, the robot can decide the best course of action, based on exploration and exploitation, to control its balance from different external disturbances. I created a very accurate 3-Dimensional model of Polaris in a simulation to learn the actions.

1.3 Research Questions

The following are the research questions that my research will answer.

1. Is my closed-loop control going to be fast enough for inexpensive robots that are equipped with cheaper hardware for responding to an external push?

2. Are any of my reinforcement learning and deep reinforcement learning push recovery techniques going to be able to replace the parameters that a human operator provides (hand-tuned) for recovery based on his/her experiences?

3. Which of the proposed approaches (RL, DRL), results in better recovery if the surfaces, directions, and impacts are unknown to the robot (i.e. with no prior knowledge)?

1.4 Thesis Organization

The remaining chapters of my thesis are structured as follows:

- **Chapter 2: Background and Related Work**–Reviews literature related to this thesis, such as active balancing, push recovery in 3 dimensional-world, real world and other peripheral areas such as humanoid robot walking.

- **Chapter 3: Design and Implementation**–Describes the approaches for hardware, software, methods used in my research. In this chapter I also describe the definition of all the methods that are used and proposed in my research with their corresponding implementations from walking engine and different push recovery control systems to machine learning and simulation environment.

- **Chapter 4: Evaluation**– I discuss the evaluation of this work. This chapter describes experiments in simulation and in the real world, and presents the results of those experiments.

- **Chapter 5: Conclusion**–This chapter discusses the main contributions and possible future directions for research. It also answers the research questions based on the results of the experiments.

1.5 Summary

This chapter introduced active balancing and push recovery for humanoid robots. It also briefly discussed a few of the previous and current research areas relevant to this Ph.D. research.

Chapter 2

Background and Related work

2.1 Introduction

Humanoid robots have similar kinematics to humans and try to mimic human motions. Hence, in comparing this category (humanoid robots) with the other classes such as industrial robots or wheeled robots, the humanoid group is the most challenging and complicated one. Over decades, researchers have designed and implemented various techniques for making humanoid robots act more anthropomorphic and natural. For example, a typical humanoid robot has two arms, two legs, a head and a torso and it locomotes on its two legs. However, there are partially humanoid robots (these would not be humanoid robots because they only partly follow the humanoid form.) like Robovie [Ishiguro et al., 2001], PR2 [Cousins, 2010], and Pepper [Tanaka et al., 2015] that have two arms and camera but are not able to walk like a human, and instead use wheels to locomote.

Figure 2.1 shows Polaris, a humanoid robot with a height of 95 CM and a weight

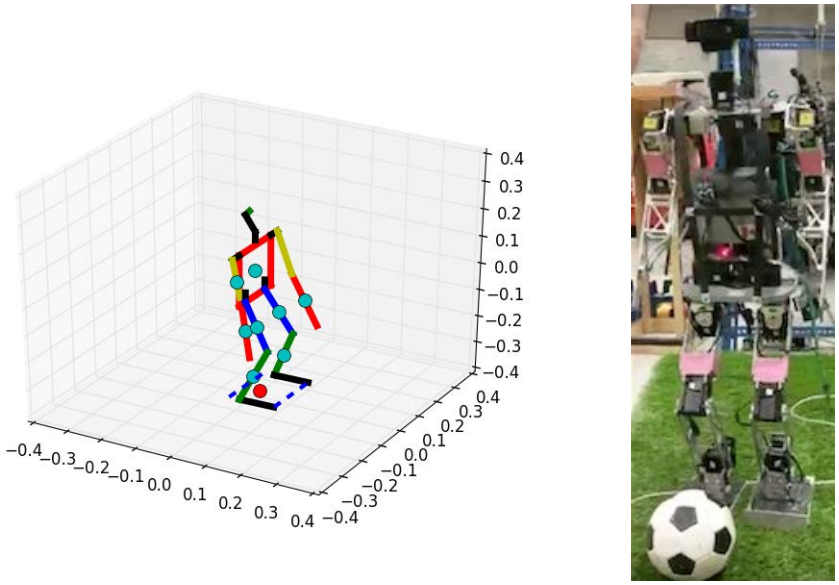


Figure 2.1: Polaris using a normal walking gait in a simulation (matplotlib environment) and the real world

of 6.95 Kg standing in its home posture in a simulation and in the real world. In this position, the robot is fully balanced, and the Center of Mass (COM) of the robot is divided equally between the robot's feet such that the robot can walk properly on a flat surface. Every single physical link on the robot's body has a mass, and I projected the COM of the whole body on the ground (red dot) between the right and left foot. As long as the COM of the robot is in the area of the support polygon (dashed area around the feet), the robot is balanced, and it is stable enough to do any particular action such as standing on one foot [Takenaka et al., 2009]. However, if the COM is outside of the support polygon the robot is not stable anymore, and this causes the robot to fall. Such a situation may happen for two main reasons. The first reason is that of applying direct external forces such as pushing the robot's chest. Such a push may come from any direction. The second reason would be stepping on uneven

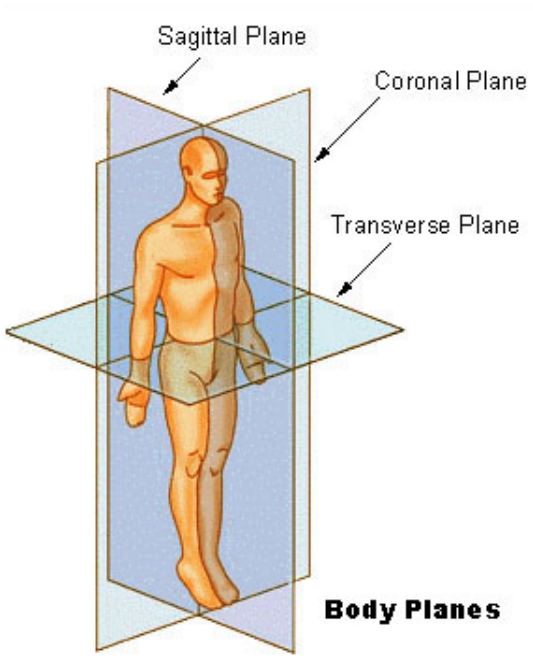


Figure 2.2: Three main anatomical planes of the human body. The three planes are: 1) the Coronal or Frontal plane that divides the body from front to back. 2) the sagittal plane that divides the body into left and right. 3) the transverse plane that divides the body from top to bottom [NIH, 2018]

terrain, including bumping into obstacles. To solve the stability issue of the robot, the COP [Collins and De Luca, 1993] needs to be in the support polygon region. One of the practical ways to address this problem is predicting the footstep from capturing walking motions [Schmitz et al., 2011] and using a gait generator such as capture steps [Missura and Behnke, 2013]. By adjusting the footsteps in the sagittal or coronal plane (Figure 2.2), or both, the area of support polygon can be made large enough to balance the robot.

2.2 Active Balancing and Push recovery

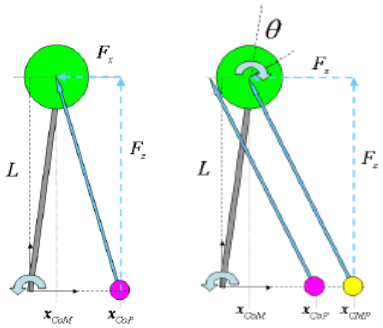
Push recovery refers to negotiating and recovering from an abnormal status to a normal situation when walking or standing [Stephens, 2007; Stephens and Atkeson, 2010]; this would happen when the robot is subjected to a large disturbance and external forces. These forces apply to the robot for a short period, and they destabilise the walking rhythm and cause the robot to fall. Generating a smooth and stable walking path over an uneven terrain that could be an unknown surface with obstacles is a complex task. The robot needs to adapt and generate footsteps based on the environment and external forces.

2.2.1 Balancing Strategies For A Humanoid Robot

Over decades many solutions have been introduced to solve the balancing problem of humanoid robots when they are subjected to considerable disturbance, external forces and walking on uneven terrain. Among them the three common strategies are [Stephens, 2007]: 1) Centre of Pressure, 2) Centroidal Moment Pivot, and 3) Stepping.

2.2.1.1 Center of pressure (COP)

This approach usually is based on controlling ankles on both feet, and it is also known as the *Ankle strategy* [Stephens, 2007]. This method is often used when there is a small disturbance and can be dealt with by shifting the centre of pressure to relocate the COM within the support polygon. Figure 2.3 (a) demonstrates COP. In this figure, L is the height, F_x the applied force to the body in the x direction, F_z is



(a) COP (b) CMP

Figure 2.3: Centre of pressure and Centroidal Moment Pivot [Stephens, 2007]

the ground force. The arrow next to the x_{CoM} is the adjustment that is applied to the ankle's degree for shifting the COM backward.

2.2.1.2 Centroidal Moment Pivot (CMP)

The Centroidal Moment Pivot approach, also known as the *hip strategy* [Stephens, 2007], is shown in Figure 2.3 (b). This can be used for small and medium disturbances, and is thus an improvement over COP. In this method, the hip servos play a significant role in recovering from the push. In Figure 2.3 (b), θ is the angle that the hip servo will change from the position before the force is applied. The change in the hip servo helps to absorb most of the external force and balance the COM. Since CMP can also modify the ankle position in the opposite direction of the external force, it subsumes COP.

Iverach-Brereton [2015] worked on active balancing of a kid-size humanoid robot for his master's thesis. His primary research focus was on balancing a kid size robot on a Bongo Board. He used three different control algorithms that were derived

Figure 2.4: DARwIn-OP humanoid robot, on a Bongo Board [Baltes et al., 2014].

from the cart-and-pole inverted pendulum problem: Proportional Integral Derivative (PID) control, fuzzy logic and Always-on Artificial Neural Networks. The first policy that he applied to the controllers was *Do the Shake*, which allows the robot to react to any external forces but didn't produce any new trajectories other than the recovery. The second approach was called *Let's Sway*, which allows the robot to create a new path to promote dynamic stability. The robot was able to recover its balance by adjusting the ankle and hip joint to relocate the COM in the frontal plane. Figure 2.4 shows a kid size robot (DARwin) on a bongo board.

Iverach-Brereton et al. [2017] used a very similar approach to balance a small alpine skiing humanoid robot (DARwin), actively on the snow (Figure 2.5) for the purposes of skiing. The approach was based on *Do the Shake*, which was previously discussed.

Figure 2.5: Jennifer, skiing straight down the hills [Iverach-Brereton et al., 2017]

2.2.1.3 Stepping

This approach is the last practical strategy, which can handle small, medium, and large disturbances in many cases [Stephens, 2007]. This involves taking another step to relocate the COM within the support polygon area. The process of push recovery from a significant disturbance is divided into three parts. The first is the *control interface* (that is, the high-level layer), and this uses the Omni-walk (omni-directional) [Behnke, 2006] to position the joints in a particular location on the XYZ axes. The omni-directional walk gives the ability to the robot to walk in any direction, and do rotations freely. In Equation 2.1, S denotes the velocity vector, and it is $\in \mathbb{R}^3$ in the three sagittal, lateral and rotational dimensions [Missura and Behnke, 2013]. The desired step parameters are denoted by S^*:

$$S^* = (S_x^*, S_y^*, S_z^*) \in \mathbb{R}^2 \times [-\pi, \pi] \tag{2.1}$$

S_x^* and S_y^* refer to the sagittal and lateral Cartesian coordinates of the footstep respectively, and S_z^* is the rotation of the feet.

The second component, below the control interface, is the *foot placement con-*

trol. The inputs of this stage are the desired step parameters Φ^* (the walking gait frequency) and S^*. However, apart from the desired step parameters, the forward kinematics and COM of the robot play a significant role here. The distances in the x and y planes are calculated, and a 4D COM is created, shown in Equation 2.2:

$$COM_{state} = (x, \dot{x}, y, \dot{y}) \tag{2.2}$$

Here, x and y denote the location of sagittal and lateral COM areas respectively and $\dot{x}$ and $\dot{y}$ denote the sagittal and lateral velocity of the COM.

The lowest level component is the *Motion Generator*, and in this stage, the trajectory for the next step will be generated. The step length S that was given to the second stage (foot placement control) has a direct impact on the swing amplitude. Additionally, the walking gait frequency Φ^* (discussed in the second phase) decides how fast the stride will be executed.

Figure 2.6 illustrates these three families, with the COM shown as a light blue circle, the COP shown as a red circle, and a brown arrow indicating the external force. The yellow arrow indicates one dimension of the support polygon.

There have been various implementations within each of these families. For example, Toyota's running robot (130 CM high, 50 Kg) generates a new trajectory after a push based on the position of its COM and the support foot, and successfully recovers from pushes against the chest during hopping and running [Tajima et al., 2009]. MABEL [Sreenath et al., 2011] also demonstrated the ability to stabilize the walk after a push by generating a new trajectory. However, the former is a highly expensive robot that would not compare to the comparatively low-torque servos in Polaris, and the latter is a planar bipedal robot mounted on a boom of radius 2.25

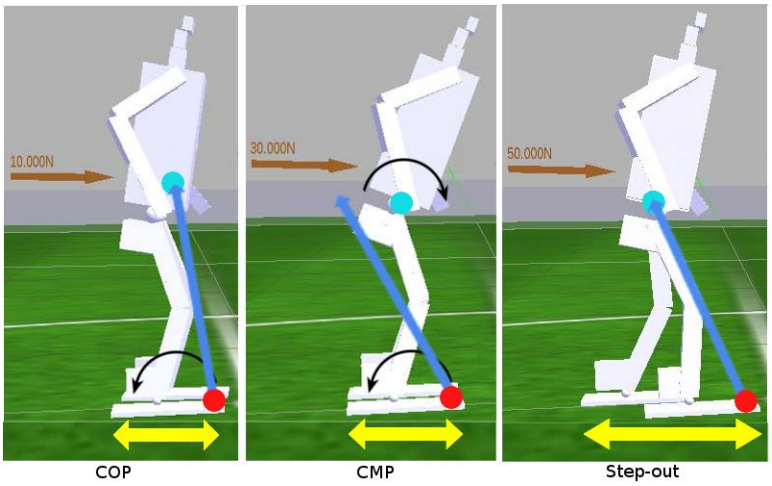


Figure 2.6: COP, CMP, and Step Out, using a model of Polaris (after [Stephens, 2007]).

Meters (M), and so can only walk around a fixed circle.

Yun and Goswami [2011] introduced a momentum-based stepping controller that tested an adult-sized humanoid robot in a simulation called Locomote, a software package based on Webots. In their solution, the simulator checks the maximum threshold of the angles as well as the torque for each joint of the two legs. If one of its joints passed their threshold, its step trigger function will be called and it will take a step for fall prevention.

Lee and Goswami [2010] presented another stepping approach that was specially for non-stationary and non-continuous grounds, also in a simulation. Their solution was very costly computationally, because it required calculating COP, COM, and the linear and angular momentum of the robot in real time. Since this approach is non-continuous, it cannot be applied to a normal walking gait.

Hofmann [2006] studied humanoid robot balance control. He argued that taking

a step for recovering from an external push is the solution for recovering from a large disturbances by moving the COP. Missura and Behnke [2013] also studied push recovery for a simulated humanoid robot that uses capture steps for recovering. In their approach, the simulated robot calculates a desired Zero Moment Point (ZMP) location for every step with respect to the COM. ZMP is a point of contact of the robot's foot with the ground in which the inertia is equal to zero. In other word, the contact point does not produce any moment in the horizontal direction (zero reaction forces) [Vukobratović and Borovac, 2004].

HRP-2 [Morisawa et al., 2010] used a hybrid theory of reactive stepping and disturbance suppression. This robot could recover its walking and balance from different pushes in various directions. HRP-2 uses *disturbance suppression* for small external forces by using a feedback controller. On the other hand, for large external forces, a feedback controller alone is not capable of controlling the balance. Hence, HRP-2 uses *reactive step* to obtain its balance by modifying the next steps in its walking trajectory. To generate a path, HRP-2 uses the COM and ZMP at all times, and the next foot step is based on the previous COM and ZMP calculation. However, the maintenance of this method is too long, and without having any external disturbance, it takes minimum 0.8 seconds to generate a trajectory. This is not fast enough to recover from pushes in the real world. According to Morisawa et al. [2010], recovering from a significant disturbance during a single support phase is very challenging, and HRP-2 could not recover from this type of interference. Also, it is necessary for HRP-2 to know about the environment and external force in advance to have an accurate recovery, and this method is not applicable where the environment is unknown.

Atlas [Kuindersma et al., 2016] is another leading humanoid robot that takes recovery steps for absorbing pushes. Atlas is hydraulically-actuated and is built by Boston Dynamics, one of the leading robotics companies. It has a height of 188 CM and a weight of 155 Kg. The robot has a high-voltage 3 phase power supply for the actuators. The reaction time of Atlas is fast, since it uses a fiber-optic line with a rate of 10 Gigabits per second (Gbps) for transmitting the instructions. As soon as a push is applied, its motion planner generates a footstep based on its current dynamics. According to the authors, in addition to walking and recovering from different degrees of pushes, Atlas is also able to run and climb stairs.

2.3 Reinforcement Learning and Deep Reinforcement Learning

Morimoto et al. [2004] employed reinforcement learning to teach a simulated biped robot how to walk with changing the frequency of a simulated robot's walk (timing of the steps) in a two-dimensional simulation environment. The authors tested their trained model on 1.0 degree angle and 4.0 degrees angle downward slopes. The robot could successfully walk for 50 steps on 1.0 degree angle and 4.0 degrees angle downward slopes. The authors set 50 steps as the successful trial. In this experiment the slop was not a dynamic parameter and from the beginning of the trial to the end of the trial it was set as a constant value. For example from the step one to the step 50 the slope was set as 1.0 degree angle.

Mnih et al. [2015] studied deep reinforcement learning and Q-learning using classic

Atari 2600 games. The input of their deep neural network was the image pixels and the score of the game in which it was training on. The output of the network was a particular discretized action. The action-states were discretized to 18 valid actions (e.g., up, right, left, and etc.).

Gu et al. [2017] investigated deep reinforcement learning and Q-learning in one particular setup in both simulation and the real world. They use robotic arms with seven motors (7 DOF) in a way that the origin of the arms were installed to fixed table. The authors assigned different tasks in simulation to the robot. These tasks were random-targets reaching (e.g., door handle), pushing and pulling a door, picking and placing an object. However, these tasks and its rules were altered in the real world because of a very low performance in the real world in comparison with simulation. Even though the tasks were altered in such a way that it is easier to achieve, the robots were not able to push the doors in the real world. The authors used multiple robotic arms in the real world to perform asynchronous updates to a deep neural network.

Kim et al. [2017] used the simulated NASA's Valkyrie humanoid robot and employed reinforcement learning to obtain recovery steps after being hit by an external force. Valkyrie has 135.9 KG weight and 1.83 meter height. It uses AC motors and the price of the real robot is more than USD $2000,000. The authors applied up to 520 Newton force that is equals to 53.03 KG for for 0.1 of a second to the robot's body in a 3D simulation. However, this approach was not tested in the real world. Also their approach's computational cost remains a challenge and cannot be adopted to low performance hardware robots.

Kumar et al. [2018] investigated a walking mechanism for a 2D simulated bipedal robot without the upper body with 6 DOF. The authors used reinforcement learning and deep neural network to achieve this task. This robot has a height of 61 CM with an approximate weight of 0.4 KG. All the environment external factors such as the slop, terrain were constant during the whole experiment. As the result of this work, the robot was able to walk for 10 meters without falling in the simulation.

Haarnoja et al. [2018] employed deep reinforcement learning to train Minitaur a quadrupedal legged robot for walking. The authors investigated their approach in both simulation and the real world. The legs were designed in such a way that the robot is always stable. The physical design of the legs were similar to a triangle shape. Minitaur was able to walk on different terrains, with or without obstacles. Also, it could walk up or down a slope without any problem.

Peng et al. [2018] used deep reinforcement learning in a simulation to train different 3D animation characters such as human, dinosaur, and etc. The authors used imitating reference motion capture to achieve highly dynamic motions. The animated characters were able to walk, run, cartwheel, dance, backflip, frontflip, and etc. This approach always needs a reference motion to guide exploration of reinforcement learning for each given task.

Gil et al. [2019] studied reinforcement learning and Q-learning for teaching a simulated NAO humanoid robot how to stand up and walk a short distance in a straight line. The ultimate goal of the authors was the simulated robot traverse from point A to point B in a short period of time without falling. Same as most of the previous related works, no changes were applied to the environment during the trials

and the environment was not dynamic. The authors used a provided simulated model of the NAO robot in the Webots simulator. The computational cost of the proposed approach was too high to run with the default hardware of the NAO robot. The default CPU uses Intel Core i5 with 8 Gigabyte of RAM. Using the default hardware took more than 2 seconds to send a new set of trajectory to the robot's joints. 2 seconds would cause the robot to fall and the control system is not fast enough to avoid falling. For that reason it is impossible to run it in a real robot in the real world as well as in a simulation. To overcome this issue in the simulation the authors used external Graphical Processor Unit (GPU) to do the computation. According to the authors their approach failed in many different scenarios because of a very specific walking setting that was defined for the robot, that is, walking in a straight line (no turns and no walking towards the y-axis) on a flat floor. Altering any of the mentioned setting would cause the robot fall.

2.3.1 Linear Inverted Pendulum Model

Kajita et al. [2001] models the human walk through the use of an inverted pendulum. In this approach, mass is denoted by m, representing a single point $p = [x, z_0]$, and z_0 is the constant hip height. The stance leg would apply a force $F = [F_x, F_z]^T$ and then the ankle position of the stance leg is given by $p_{ankle} = [x_{ankle,0}]^T$. Figure 2.7 demonstrates a pendulum (only the leg on the left side of the figure is showing the pendulum, and the right leg that is doing the swing is not).

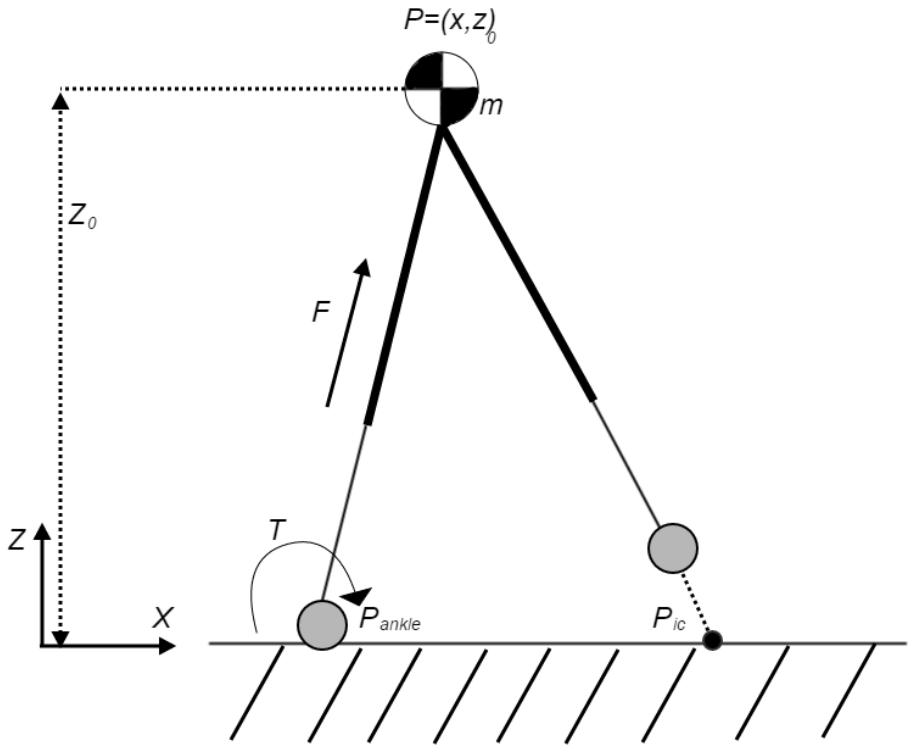


Figure 2.7: Simple Linear Inverted Pendulum Model (after [Kajita et al., 2010])

In the situation of dealing only with the sagittal plane, the linear inverted pendulum model (Equation 2.3) can be modelled as:

$$\ddot{x} = Cx \tag{2.3}$$

where C denotes the gravitational constant $6.67408 \times 10^{-11} m^3 kg^{-1} s^{-2}$. Given an initial state $(x_0, \dot{x})$ the set of equations (Equations 2.4–2.9):

$$x(t) = c_1 e^{\sqrt{C}t} + c_2 e^{-\sqrt{C}t} \tag{2.4}$$

$$\dot{x} = c_1\sqrt{C}e^{\sqrt{C}t} - c_2\sqrt{C}e^{-\sqrt{C}t} \tag{2.5}$$

$$t(x) = \frac{1}{\sqrt{C}}ln\left(\frac{x}{2c_1} \pm \sqrt{\frac{x^2}{4c_1^2} - \frac{c_2}{c1}}\right) \tag{2.6}$$

$$t(\dot{x}) = \frac{1}{\sqrt{C}}ln\left(\frac{\dot{x}}{2c_1\sqrt{C}} \pm \sqrt{\frac{\dot{x}^2}{4c_1^2C} + \frac{c_2}{c1}}\right) \tag{2.7}$$

$$c_2 = \frac{1}{2}(x_0 + \frac{\dot{x_0}}{\sqrt{C}}) \tag{2.8}$$

$$c_1 = \frac{1}{2}(x_0 + \frac{\dot{x_0}}{\sqrt{C}}) \tag{2.9}$$

can find the state of $x, \dot{x}$ at any time t or when COM reaches some particular position x or the velocity $\dot{x}$. For modelling the motions in the lateral (Equation 2.10) and sagittal (Equation 2.11) planes, the following two equations are used.

$$\ddot{x} = Cx, \tag{2.10}$$

$$\ddot{y} = Cy, \tag{2.11}$$

As was stated earlier, each axis has a one-dimensional Linear Inverted Pendulum Model (LIPM). One controls the x axis and the other controls the y axis. Figure 2.8 demonstrates the stick diagram of the inverted pendulum. The left picture is passing the mass over the swing pivot point, and it is during the walking gait with no external disturbance, and it is in the sagittal walking (walking forward). Furthermore,

the right picture demonstrates the inverted pendulum for the lateral direction (left and right). It is very crucial that the inverted pendulum does not pass the pivot point, and in cases that it is out of the support polygon, the robot will tip over and fall.

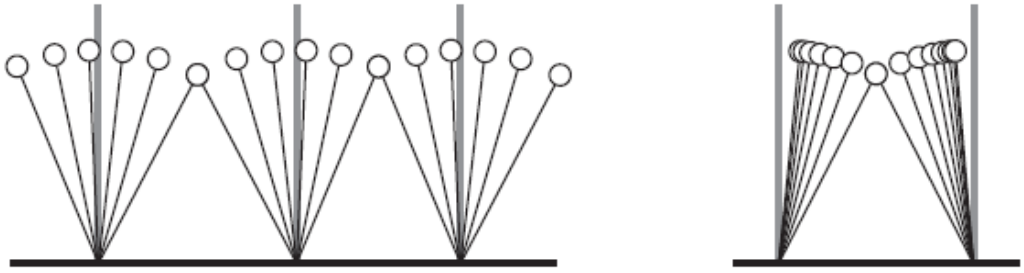


Figure 2.8: Stick diagram of sagittal motion (left picture) and lateral motion (right picture) [Missura and Behnke, 2013]

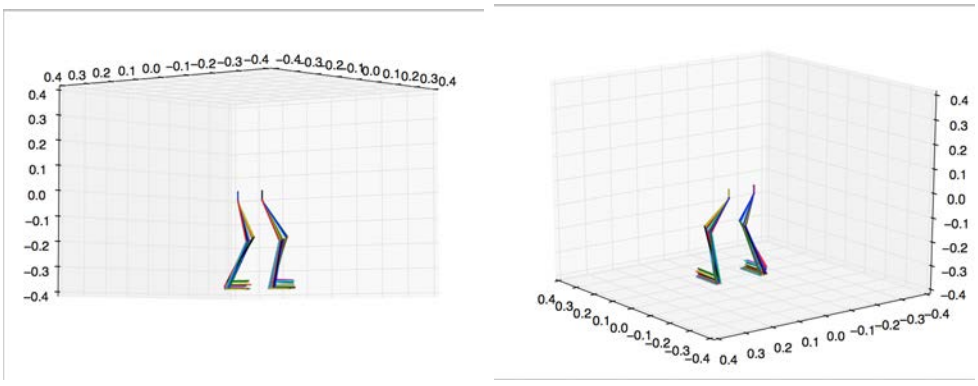


Figure 2.9: Stick diagram of Polaris for the sagittal plane

2.4 Lagrangian Dynamics

Lagrangian dynamics is a mathematical approach to model the dynamics of complex systems. It can be used to calculate stable biped locomotion. In [Vundavilli and Pratihar, 2011], the Lagrange-Euler equation was used to generate a dynamic balanced gait for a humanoid robot with 7 Degrees Of Freedom (DOF). The Lagrangian equation was used in [Kwek et al., 2003] for a biped robot with five links and two DOF on each leg. The authors used a Proportional Derivative (PD) controller to control the stability of the robot during walking in the sagittal plane. Similarly, this approach was used in [Rodriguez-Leal et al., 2011] and [Tzafestas et al., 1997] with biped robots with five and nine links respectively.

The Lagrangian function of movement for a humanoid robot [Tzafestas et al., 1997] can be obtained from Equation 2.12. In this formula T denotes the kinetic energy and U is the potential energy.

$$L = T - U \tag{2.12}$$

The kinetic energy for this problem is demonstrated in Equation 2.13, l in this formula denoted the length of the link.

$$T = \frac{1}{2}mv^2 \rightarrow \frac{m}{2}(l\dot{\theta})^2 \tag{2.13}$$

And the *potential energy* (Equation 2.14) is:

$$U = mgh \tag{2.14}$$

Now we can rewrite the Lagrangian formula (Equations 2.15, 2.16):

$$L = \frac{m}{2}(l\dot{\theta})^2 - mg(l - lcos\theta) \tag{2.15}$$

$$L = \frac{m}{2}(l\dot{\theta})^2 - mgl(1 - cos\theta) \tag{2.16}$$

Now we can put this in the Euler-Lagrange equation that is demonstrated in Equation 2.17.

$$\frac{\partial L}{\partial \theta} - \frac{d}{dt}(\frac{\partial L}{\partial \dot{\theta}}) = 0 \tag{2.17}$$

The formula mentioned above is a situation that we have one link and one mass and one length of the link. In cases that two or more links are connected to each other, we can write the Lagrangian for the compound pendulum (Equations 2.18–2.23).

$$L = T - U \rightarrow L = T_1 + T_2 - U_1 - U_2 \tag{2.18}$$

$$T_1 = \frac{m_1 l^2 \dot{\theta_1}^2}{2} \tag{2.19}$$

$$T_2 = \frac{m_2}{2}[{l_1}^2.{\dot{\theta_1}}^2 + {l_2}^2.{\dot{\theta_2}}^2 + 2l_1 l_2 \dot{\theta_1}\dot{\theta_2}\cos(\theta_1 - \theta_2)] \tag{2.20}$$

$$U_1 = m_1 g y_1 = -m_1 g l_1 \cos\theta_1 \tag{2.21}$$

$$U_2 = m_2 g y_2 = m_2 g(-l_1 \cos\theta_1 + l_2 \cos\theta_2) \tag{2.22}$$

$$L = \frac{m_1 l^2 \dot{\theta_1}^2}{2} + \frac{m_2}{2}[{l_1}^2.\dot{\theta_1}^2 + {l_2}^2.\dot{\theta_2}^2 + 2l_1 l_2 \dot{\theta_1}\dot{\theta_2}\cos(\theta_1 - \theta_2)]$$
$$-(m_1 g y_1 = -m_1 g l_1) - (m_2 g y_2 = m_2 g(-l_1 \cos\theta_1 + l_2 \cos\theta_2)) \tag{2.23}$$

2.5 Summary

Many of these and other related works are examined only in simulation, or in restricted settings such as walking in a circle while suspended from a pivoting boom. Those that are physically implemented tend to require highly expensive platforms. My approach is intended to function for small, medium and strong size pushes on an inexpensive platform, which can only rely on lower-torque servos, less precise body machining, and less accurate sensors.

In this chapter I have overviewed previous works, the next chapter details my own approach to push recovery for inexpensive humanoid robots.

Chapter 3

Design and Implementation

3.1 Introduction

The main contribution of my thesis is an implemented, fully autonomous push recovery framework that operates on varying terrain using RL and DRL. In order to implement such a robust push recovery framework, I needed to design and implement several individual modules. Finally I had to make sure that all the implemented modules are able to communicate with each other so that the final product, that is the push recovery, can prevent the robot from falling. The purpose of this chapter is to describe the approach and implementation of my framework.

Designing and implementing a push recovery technique for a relatively inexpensive humanoid robot is a very complex and challenging task. The three most important steps of all push recovery approaches are as follows: first, the robot must be able to walk freely in almost any direction (omni-directional) [Behnke, 2006] in real time. Furthermore, the robot should have contact with the outside world (environment)

and read the feedback with its sensors (e.g. accelerometer, vision). Finally, with the gathered information about the environment and the status of the robot, it will decide the best course of action to take for preventing a fall. As was discussed in Section 2.2.1.3, the robot needs to take one or more steps for recovering from strong pushes. For that reason walking is an important component of a good push recovery framework. The walking engines in my work are used for both normal walking as well as push recovery with stepping features. This is also true for humans, where the walking trajectories are used in both situations. Hence, I implemented a two parameter walking engine based on LIPM (discussed in Section 2.3.1). To make the robot walk, first I had to implement kinematic models (forward and inverse kinematics) of Polaris. To insure that the mathematical model embodies in the kinematic models was accurate without damaging the servo motors (to ensure motors go to the calculated positions for each link), I designed a three-dimensional (3D) model of Polaris in Python with the matplotlib module. Figure 2.1 shows this model. The use of this model was for the early stage of my implementation. Since matplotlib is a plotting software tool and it is not an extensive simulation environment, I decided to create a new model of Polaris that can be used in Gazebo [Gazebo, 2017]. This model is very similar to Polaris in the real world. Examining a 3D model of Polaris in Gazebo for push recovery experiments is more accurate and precise than using matplotlib. Also, the simulated model in Gazebo is designed to accurately mimic the counterpart's behaviour in the real world. Because of this the results in the real world and simulation are very close [Koenig and Howard, 2004]. For that reason I upgraded the simulation environment to Gazebo. Figure 3.16 shows the 3D model of Polaris doing a jump test

in Gazebo.

In this chapter I describe the design and implementation of all the modules in my framework. Section 3.2 describes the hardware and software that I used for this research. Section 3.3 describes the mathematical process for setting a robot joint to a particular position. In Section 3.4, I describe the walking engine, different feedback control loops, push recovery stages, simulation, and machine learning. For machine learning, I introduce and implement two novel push recovery methods called Reinforcement Learning Push Recovery (RLPR) and Deep Reinforcement Learning Push Recovery (DRLPR) that both use stepping (Section 2.2.1.3) to absorb an external push. These methods are used to replace a set of walking engine parameters that are based on a human's experience.

3.2 Building and Adapting a Robot

3.2.1 Hardware

Polaris [Ramezani et al., 2015] is 95 CM in height and 6.9 Kg in weight. The body links (mechanics) of Polaris were designed and machined at the Amirkabir University of Tehran (AUT), as our lab lacks fine metal machining capabilities. The links then were shipped to the AALab and I assembled the links along with other components of Polaris.

3.2.1.1 Servo Motors

The robot's kinematic structure has 20 DOF. This design uses 6 and 3 DOF for each leg and arm, respectively. The camera of the robot is mounted on two servos that enable Polaris to look up, down, left and right. I have used three types of Robotis servo motors in the architecture of Polaris: MX-106 motors in the legs, MX-64 motors in the arms and MX-28 motors in the neck. All the servos use the Transistor-Transistor Logic (TTL) serial communication protocol with three pins that share one line for sending and receiving data, one line for Voltage Common Collector (VCC) and one line for the ground.

Each hip can rotate in the sagittal, frontal and transversal planes. And the two arms support the sagittal and frontal planes. The neck servos support the sagittal and transversal planes, and are responsible for moving the camera. Figure 3.1 illustrates these three types of servo motors.

(a) MX-28 (b) MX-64 (c) MX-106

Figure 3.1: Three different types of servo motors that Polaris uses [Robotis, 2018a]

3.2.1.2 Computer (CPU)

When I started my research with Polaris, I used a Qute-PC *3000 Series* [Quanmax, 2017] as the computation unit for it. This computer was one of the cheapest mini-pcs

Processor type	Intel® Celeron® Dual Core
Processor speed	1.1 GHz * 2
Processor architecture	x86_64
Random Access Memory	2GB DDR3
Phone Jack for Line-Out IO	1 X
Phone Jack for MIC-In	1 X
USB2.0	4 X
eSATA/USB Combo connector	1 X
RJ-45, GbE port	1 X
VGA	1 X
HDMI	1 X
SATA HDD 64 GB	1 X
802.11b/g/n	1 X

Table 3.1: Specifications of the Qute-PC-3000 series

that was able to run my framework and the operating system. Figure 3.2 shows the computer, and Table 3.1 lists its hardware specifications.

Figure 3.2: Qute-PC-3000 series

After a few years of working with this computer, I upgraded Polaris's computer

to a better unit which has higher computational power. The current computer that Polaris uses is GB-BSi3H-6100, a Gigabyte Brix Mini-PC model. Figure 3.3 shows the upgraded computer, and Table 3.2 lists its hardware specifications.

Figure 3.3: GB-BSi3H-6100 [Gigabyte, 2018]

Processor type	Intel® Core i3-6100U
Processor speed	2.3 GHz
Processor architecture	x86_64
Random Access Memory	4GB DDR3
Phone Jack for Line-Out IO	1 X
Phone Jack for MIC-In	1 X
USB3.0	4 X
RJ-45, GbE port	1 X
VGA	1 X
HDMI plus mini display	1 X
SSD 120 GB	1 X
802.11b/g/n	1 X

Table 3.2: Specifications of the GB-BSi3H-6100 computer that Polaris uses.

3.2.1.3 Communication Converter

In order to overcome the problem of connecting servo motors to the computers, in the beginning of my research, I employed a USB2Dynamixel [Robotis, 2017] board from Robotis that uses a Universal Serial Bus (USB) port. All Polaris's servo motors use TTL (3 pins) network connection to communicate with the other servo motors or the computer. Figure 3.4 shows the USB2Dynamixel hardware and Figure 3.5 illustrates its connection diagram.

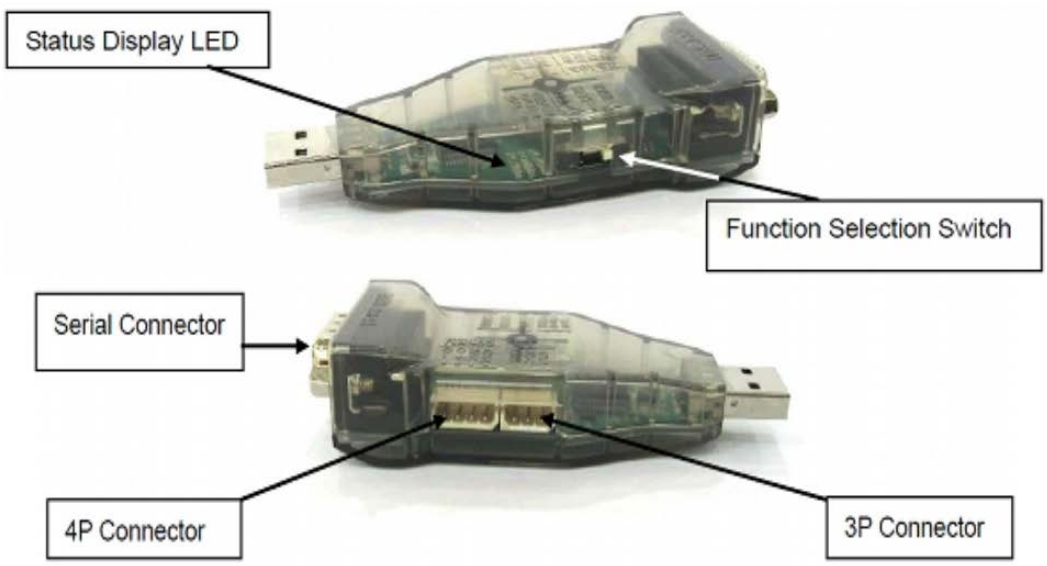


Figure 3.4: USB2Dynamixel connection diagram [Robotis, 2017]

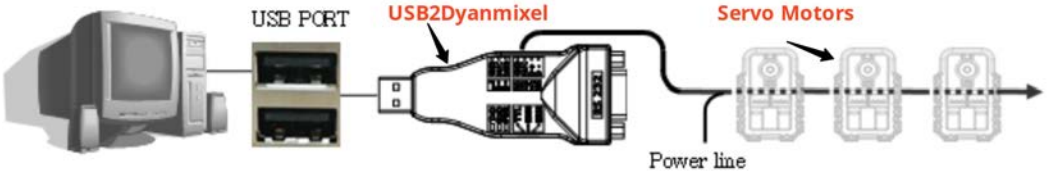


Figure 3.5: USB2Dynamixel connection diagram [Robotis, 2017]

It was very hard to stick the USB2Dynamixel to a part of Polaris's body during

its walking gait. This was because the body movement caused the USB2Dynamixel to disconnect and reconnect regularly. These disconnections are also very bad for the servo motors and for that reason I changed the USB2Dynamixel to U2D2, illustrated in Figure 3.6.

Figure 3.6: U2D2 connection diagram [Robotis, 2018b]

3.2.1.4 Dynamixel Hub Communication Board

I designed the servo connections on Polaris in a way that each part of its body has its own wiring. For example, each arm and leg has its own wiring connection and all the 20 servo motors are not connected with one long cable. Therefore, there will be four TTL wires from the arms and legs, and one from the neck. As was stated previously, the servo motors use a U2D2 to connect to the PC. Polaris uses a 6 Port AX/MX Power Hub that is shown in Figure 3.7 to connect all the TTL cables together. This hardware receives 5 TTL signals from servo motors and another TTL port is connected to the U2D2 hardware(Section 3.2.1.3) that is connected to the computer.

3.2.1.5 Inertial Measurement Unit

Polaris uses an Inertial Measurement Unit (IMU) as an input sensor for balancing, incorporating both a gyroscope and accelerometer. At the same time as the previously

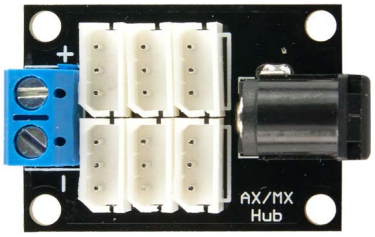


Figure 3.7: 6 Port AX/MX Power Hub [Trossenrobotics, 2018]

explained hardware components, I upgraded the IMU device too. For the first few years of my research, I used a CA$ 3 IMU that contains a single chip InvenSense MPU-6050 which contains an accelerometer and a gyroscope. The MPU-6050 chip provides 3-axis (x, y, z) for the accelerometer and 3-axis (x, y, z) for the gyroscope. This IMU module had to be connected to a micro-controller to be able to work. An Arduino Nano board micro-controller was used for connecting the MPU-6050 sensor to the Qute-PC for reading the sensor feedback. Figure 3.8 shows the IMU and the micro-controller.

One of the main issues with this IMU was the amount of noise that it generated along the feedback reading. Also its latency for delivering the feedback to the computer was high. So, I upgraded the IMU to PhidgetSpatial Precision 3/3/3 High Resolution IMU [Phidgets, 2018]. The new IMU consists of a 3-axis accelerometer, gyroscope and compass, and provides higher resolution readings compared to the earlier IMU model. Figure 3.9 illustrates the final upgraded IMU.

To show the consistency and accuracy of the IMU for use in detecting external forces, I recorded more than 2800 readings from both gyroscope and accelerometer.

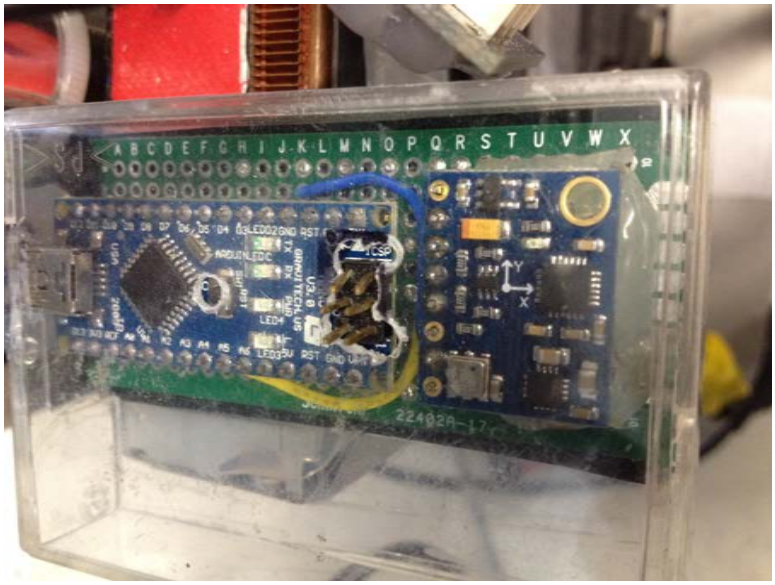


Figure 3.8: MPU-6050 IMU connected to the Arduino Nano board micro-controller

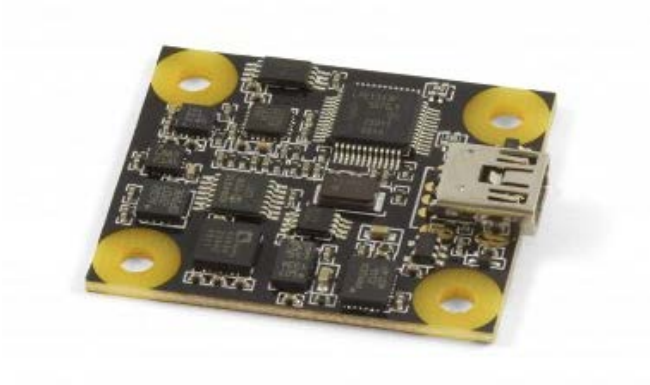
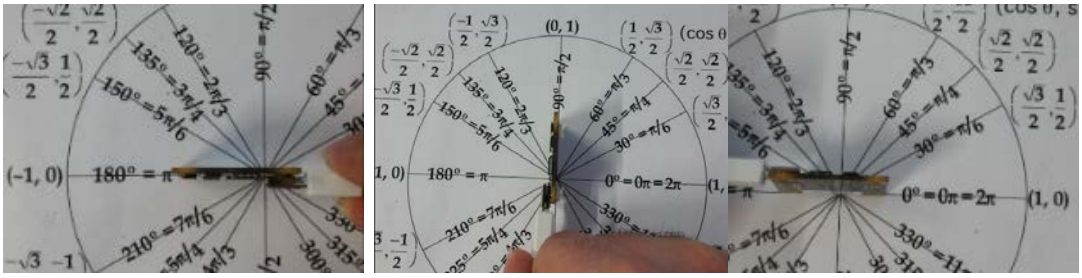


Figure 3.9: PhidgetSpatial Precision 3/3/3 High Resolution IMU [Phidgets, 2018]

I tested the rotations of the IMU for both sensors in the x, y, and z axes. I divided the test into two main parts: 1) when the IMU is almost stationary (I held the IMU and was trying to avoid applying any force to it). 2) when the IMU was moving (I applied acceleration and angular velocity). In the stationary tests, both sensors

(gyroscope and accelerometer) were tested in three different holding angles: π, $\pi/2$, and 2π. Figure 3.10 shows the scenario of holding the IMU when it is in the three stationary poses.



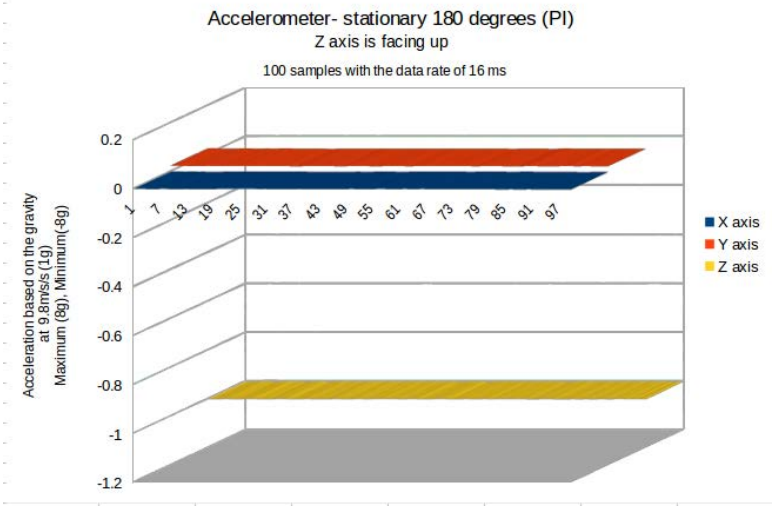
(a) Holding IMU stationary in 180 degrees (b) Holding IMU stationary in 90 degrees (c) Holding IMU stationary in 360 degrees

Figure 3.10: Holding the IMU at different angles while it is stationary

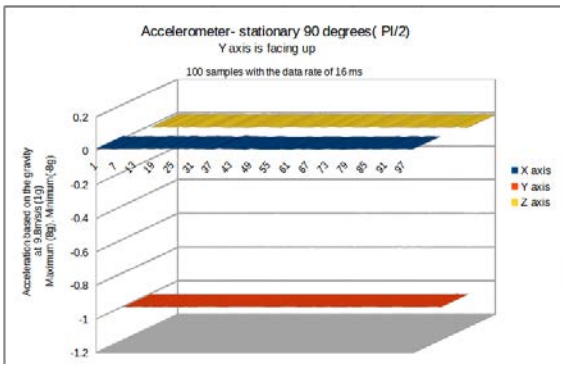
For each of these angles $\{\pi, \pi/2, 2\pi\}$, two readings were recorded one for the gyroscope and one for the accelerometer. Figure 3.11 shows the acceleration that is measured by the accelerometer. As is shown clearly in this figure, there are no significant changes in the acceleration in any of the axes. This IMU is very accurate, and it reports all kinds of small vibrations. I tested it by measuring acceleration based on gravity, which is 9.8 m/s^2 (The standard gravity at the Earth's surface is 1g (g)). This accelerometer can measure +-8g (that is approximately 78.453 m/s^2).

Figure 3.12 demonstrates a similar test for the gyroscope while it is almost stationary. This gyroscope can measure +-2000 degrees per second (angular velocity). Since I held the gyroscope and there is some shaking in a human hand, the gyroscope recorded the maximum angular velocity of +-2.5 for the three axes.

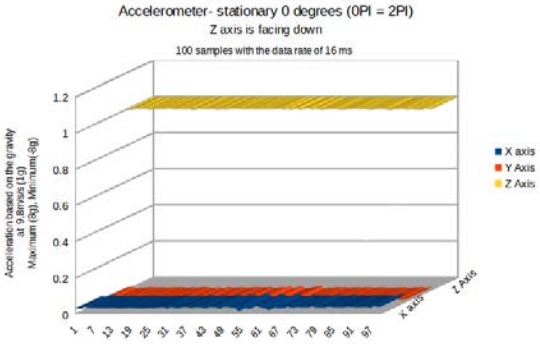
I wanted to make sure that both of my IMU sensors record intense forces from



(a) Holding IMU stationary at a 180 degree angle



(b) Holding IMU stationary at a 90 degree angle

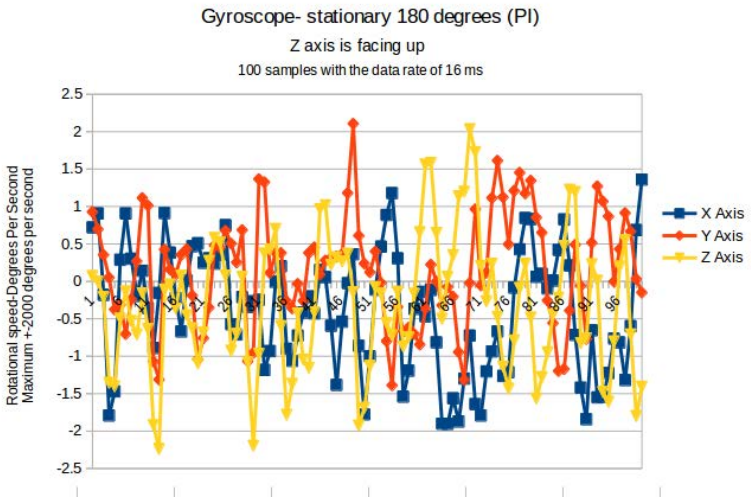


(c) Holding IMU stationary at a 360 degree angle

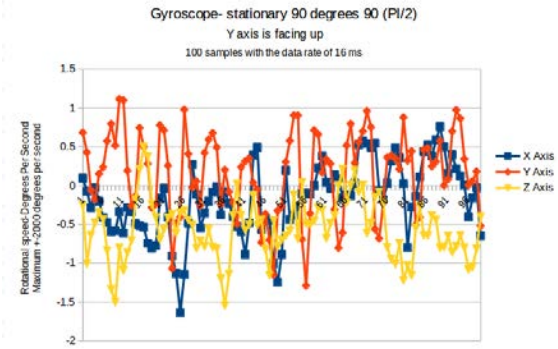
Figure 3.11: Holding the IMU at different angles while it is stationary

time to time. For that reason, I manually move the IMU in different axes with random speeds. Figure 3.13 illustrates different accelerations in different axes.

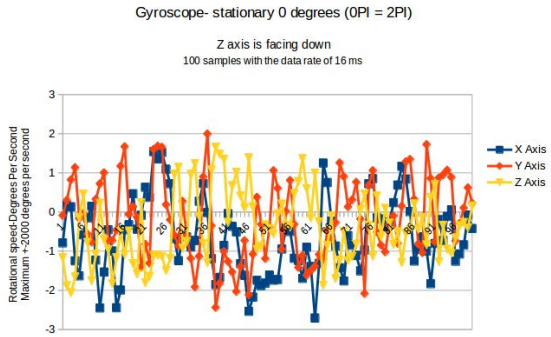
The test above shows consistency in the acceleration in different axes. I followed the exact same test for the gyroscope to check its consistency and accuracy for measuring the angular velocity. For this test, I rotated the IMU via the three {x, y, z}



(a) Holding IMU stationary at a 180 degree angle



(b) Holding IMU stationary at a 90 degree angle



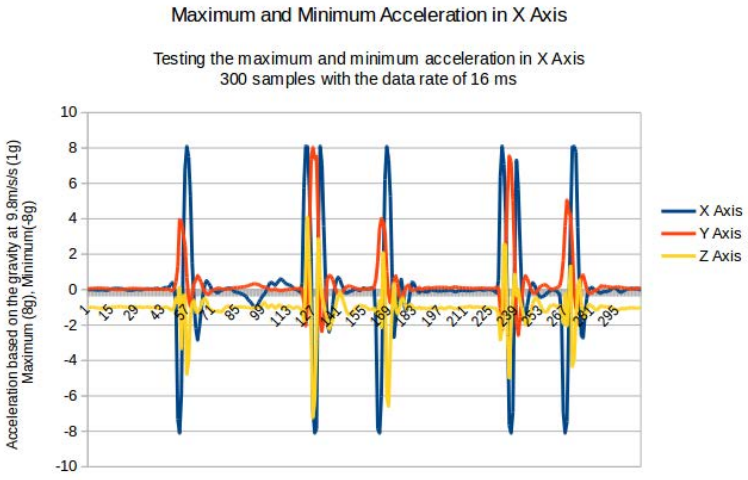
(c) Holding IMU stationary at a 360 degree angle

Figure 3.12: Holding the IMU at different angles while it is stationary

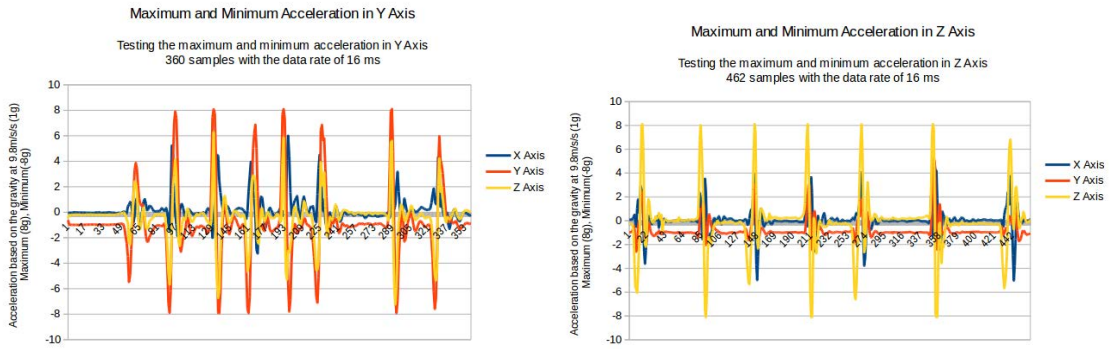
axes with random velocities. Figure 3.14 illustrates this test.

3.2.2 Software

In order to make the servo motors ready to work as a group, I had to program their firmware individually. I connected each servo to a USB2Dynamixel, and connected



(a) Trying to get the maximum acceleration in the X axis

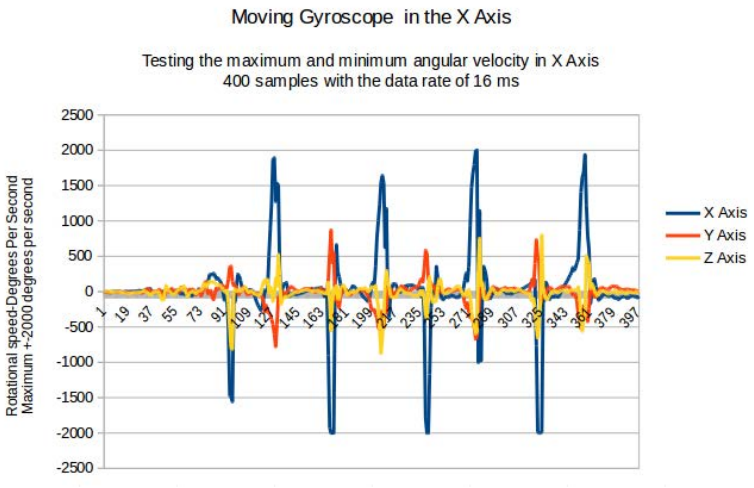


(b) Trying to get the maximum acceleration in the Y axis

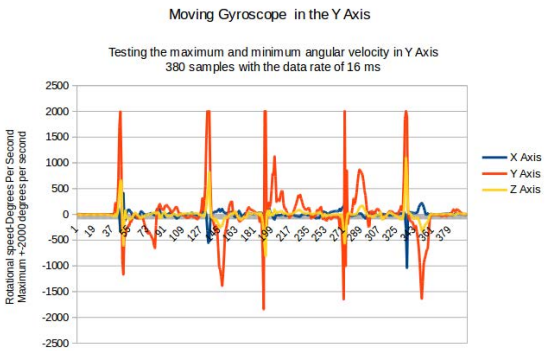
(c) Trying to get the maximum acceleration in the Z axis

Figure 3.13: Trying to get the maximum acceleration on all the axes

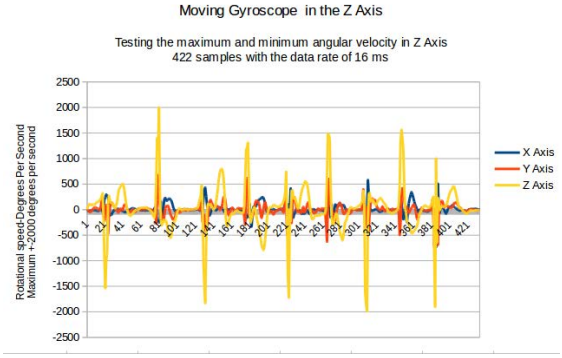
the USB2Dynamixel to a computer. The servos also needed power to function. So, I connected a 3 cell battery to the servos. The programming part of the firmware was: 1) Assigning a unique identity (ID) to each servo. 2) Changing their default baud rate (57142 bits per second (bps)) to $2 * 10^6$ bps. I changed the baud rate so that the servos can send and receive instructions faster. 3) Depending on the location of each servo on Polaris, I had to change the home position of that servo. 4) After I set the



(a) Moving the gyroscope with random angular velocity in the X axis



(b) Moving the gyroscope with random angular velocity in the Y axis



(c) Moving the gyroscope with random angular velocity in the Z axis

Figure 3.14: Moving the gyroscope with random angular velocity in all the axes

home (default) position, I place a horn set the way that the home position pin on the servo's gear is below the horn's home position. Figure 3.15 shows these components and their connections.

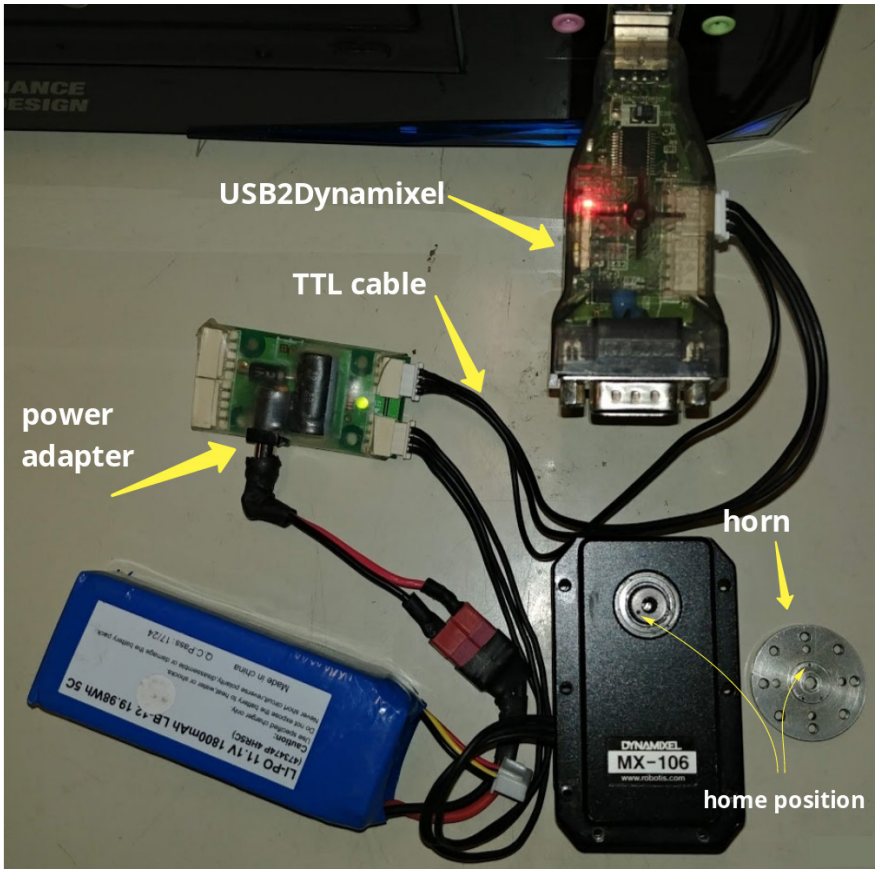


Figure 3.15: Servo firmware programming setup

3.3 Kinematic Models

3.3.1 Forward Kinematics

The forward kinematics problem is related to the relationship between each joint (servo motor) and the links and connections between them. There are a few different calculation techniques for the joints such as revolute or rotational joints (servo motor) and the link extension in case of prismatic or sliding joints [Tolani et al., 2000; Buss, 2004]. I have only concentrated on rotational joints, and because of that all my calculations and equations are related to these. Rotational joints provide the robot

a relative rotation along a single axis. In such a way, each joint has only one single DOF and to rotate in different directions more DOFs are needed.

The primary objective of forward kinematics is to determine the cumulative effect of all the joints' variables. A robot with n joints has $n+1$ links connected to it. I assume that the number of joints starts from 1 to 20 - that is, the total number of joints that Polaris uses. By the convention mentioned above, joint i connects the link $i-1$ to the link i and in all of my calculations link $i-1$ is the fixed link. At any point when the servo motor i rotates around an axis, only the link i would move and the link $i-1$ is not moving, and as a result, the location of the link i is fixed. The i^{th} joint is denoted by q_i, and it refers to the angle of rotation in that particular joint.

$$q_i = \theta_1 : \text{ joint } i \text{ revolute} \tag{3.1}$$

For doing a precise and proper kinematic chain analysis, I have assigned a coordinate frame to each link of Polaris' body structure.

$$\therefore o_i x_i y_i z_i \text{ is assigned to the link } i \tag{3.2}$$

For instance if the servo motor number 1 is moving, then the link 1 with the coordinate frame of $o_1x_1y_1z_1$ is moving, but the link $i-1$ (that is, link 0) with the frame coordinate of $o_0x_0y_0z_0$ is fixed and not moving. I used a Homogeneous Transformation Matrix (HTM) denoted by A and for the link i, is denoted by A_i to identify the position and the orientation of $o_ix_iy_iz_i$ in regards to $o_{i-1}x_{i-1}y_{i-1}z_{i-1}$. The equation above can also be demonstrated in a different way that acts as a function,

$$A_{i+1} = A_i(q_i) \tag{3.3}$$

The transformation matrix is denoted by T_j^i where j is the last number.

$$T_j^i = A_{i+1}A_{i+2} \ ... \ A_{j-1}A_j \Leftrightarrow i < j \tag{3.4}$$

While it is possible to use an arbitrary frame attached to each link and calculate the position of the end effector for this kinematic chain, a commonly used convention for referencing the robot's frames is the Denavit-Hartenberg convention (D-H) [Kelmar and Khosla, 1990]. For each HTM, A_i is a product of four basic transformations. D-H parameters define the motion of actuators connected by hard links. This is used for efficient calculation of the forward kinematics and inverse kinematics. The calculation starts with the base link or the previous link. For every rotational joint on a link, its z axis must point to the axis of rotation for that link. Moreover, each x axis will point away from the previous joint and after this, the y axis is now constrained to complete the right-handed coordinate frame. The important thing is that the origin is not at the centre of the physical servo motor and there are possibilities that the origin is in open space. So the D-H parameters are concerned about the motion links and not the physical placement of the components. The D-H convention uses four parameters to specify the joint-to-joint transformation. These are [Wang et al., 2014]:

- d = the depth along the previous joint's z axis from the origin to the common normal
- θ = the angle about the previous z axis to align its x with the new origin
- a = the distance along the rotated x axis. Also known as r, this stands for radius of rotation about previous z axis. In my thesis I refer to it as a
- α = rotation about the new x axis to put z in its desired orientation

In such a case there are two parallel z axes. Because we have an infinite number of common normals, it is possible to pick any d parameters, such as the centre of the link or the tip of the end-effector. The parameter a_i refers to the link length, α_i is the link twist, d_i is the link offset and θ_i is the joint angle. Equation 3.5 illustrates the homogeneous transformation matrix for A_i. A_i comprises a number of elementary transformations. It comprises a rotation around the Z axis that is denoted by R_z, a translation along the Z axis that is denoted by Trans_z, a translation along the X axis that is denoted by Trans_x, and a rotation around the X axis that is denoted by R_x.

$$A_i = R_{z,\theta_i}\text{Trans}_{z,di}\text{Trans}_{x,ai}R_{x,\alpha_i} \text{ [Watkins and Dayan, 1992]} \tag{3.5}$$

$$= \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3.2 Inverse Kinematics

After the robot receives a target location (x_i, y_i, z_i) of the end effector of a kinematic chain, such as a foot or hand of the robot, a mathematical model of that chain

calculates all the angles which relate to the end effector. After all the angles are calculated, each servo motor goes to its position, and that will result in the end effector being at the target location.

As was discussed in Section 3.2.1.1, each leg of Polaris consists of 6 servo motors and each of them represent one DOF. I designed and implemented (6*2) main variables in the mathematical model of the legs. Each variable represents a joint's angle on Polaris' legs. I rotated 180 degrees, all the lateral servo motors in the right leg in comparison with the left leg. All the other components are a mirror image.

Equations 3.6 and 3.7 calculate the projection of a given x coordinate on the ground for the left and right foot of the robot respectively. In these equations, x is the goal target in the x-axis and y is the goal target in the y-axis. Also, *dht* is desired hip transversal angle, and *thdy* is the distance between the origin of the robot (bottom center of the torso) to the hip joints in y-axis.

$$projected\_left\_foot_x = x * cos(dht) - (y - thdy) * sin(dht) \tag{3.6}$$

$$projected\_right\_foot_x = x * cos(dht) - (y + thdy) * sin(dht) \tag{3.7}$$

Equations 3.8 and 3.9 calculate the projection of a given y coordinate on the ground for the left and right foot of the robot respectively.

$$projected\_left\_foot_y = (x * sin(dht) + (y - thdy) * cos(dht)) + (thdy * 2) \tag{3.8}$$

$$projected\_right\_foot_y = (x * sin(dht) + (y + thdy) * cos(dht)) - (thdy * 2) \quad (3.9)$$

Equation 3.10 calculates the height of the robot for a given coordinate in the z-axis. *thdz* corresponds to the distance between the origin (on torso) and the hip, in the z-axis. *hthfz* corresponds to the distances between the hip transversal to hip frontal joints in the z-axis.

$$projected\_foot_z = z - (thdz + hthfz) \quad (3.10)$$

Equations 3.11 and 3.12 calculate the maximum possible length of the left and right legs based on their calculated $coordinates_{(x,y,z)}$, for the corresponding foot.

$$max\_left\_leg\_length = \sqrt{projected\_left\_foot_x{}^2 + projected\_left\_foot_y{}^2 + projected\_foot_z{}^2} \quad (3.11)$$

$$max\_right\_leg\_length = \sqrt{projected\_right\_foot_x{}^2 + projected\_right\_foot_y{}^2 + projected\_foot_z{}^2} \quad (3.12)$$

Equations 3.13 and 3.14 calculate the angles between the two links of the knee joints for left knee and right knee, respectively.

$$knee\_joint\_left = \frac{cos^{-1}(max\_left\_leg\_length^2 - hlk_z^2 - kal_z^2)}{(-2.0 * |hlk_z|) * |kal_z|} \quad (3.13)$$

$$knee\_joint\_right = \frac{cos^{-1}(max\_right\_leg\_length^2 - hlk_z^2 - kal_z^2)}{(-2.0 * |hlk_z|) * |kal_z|} \tag{3.14}$$

Equations 3.15 and 3.16 calculate the angles of the left hip frontal and right hip frontal, respectively.

$$hip\_joint\_frontal\_left = sin^{-1}(\frac{projected\_left\_foot_y}{\sqrt{projected\_left\_foot_y^2 + projected\_foot_z^2}}) \tag{3.15}$$

$$hip\_joint\_frontal\_right = sin^{-1}(\frac{projected\_right\_foot_y}{\sqrt{projected\_right\_foot_y^2 + projected\_foot_z^2}}) \tag{3.16}$$

Equations 3.17 and 3.18 calculate the angles of the left hip lateral and right hip lateral, respectively.

$$\begin{aligned} hip\_joint\_lateral\_left = sin^{-1}(\frac{projected\_left\_foot_x}{max\_left\_leg\_length}) \\ +sin^{-1}(\frac{|kal_z| * sin(knee\_joint\_left)}{max\_left\_leg\_length}) \end{aligned} \tag{3.17}$$

$$\begin{aligned} hip\_joint\_lateral\_right = sin^{-1}(\frac{projected\_right\_foot_x}{max\_right\_leg\_length}) \\ +sin^{-1}(\frac{|kal_z| * sin(knee\_joint\_right)}{max\_right\_leg\_length}) \end{aligned} \tag{3.18}$$

Equations 3.19 and 3.20 calculate the angles of the left ankle lateral and right

ankle lateral, respectively.

$$ankle\_joint\_lateral\_left = \pi - knee\_joint\_left - hip\_joint\_lateral\_left \tag{3.19}$$

$$ankle\_joint\_lateral\_right = \pi - knee\_joint\_right - hip\_joint\_lateral\_right \tag{3.20}$$

Equations 3.21 and 3.22 calculate the angles of the left ankle frontal and right ankle frontal, respectively.

$$ankle\_joint\_frontal\_left = hip\_joint\_frontal\_left \tag{3.21}$$

$$ankle\_joint\_frontal\_right = hip\_joint\_frontal\_right \tag{3.22}$$

3.4 Push Recovery

Push recovery for a humanoid robot is not an independent module that functions alone. To design and implement this module, I had to first design and implement other dependent modules such as parameterized walking engines and loop-control feedbacks (open-loop and closed-loop). For testing the forward and inverse kinematics and the walking engines on Polaris, I had to first test them in simulation to avoid damaging the servos. To verify the accuracy of my first walking engine (open-loop control), I

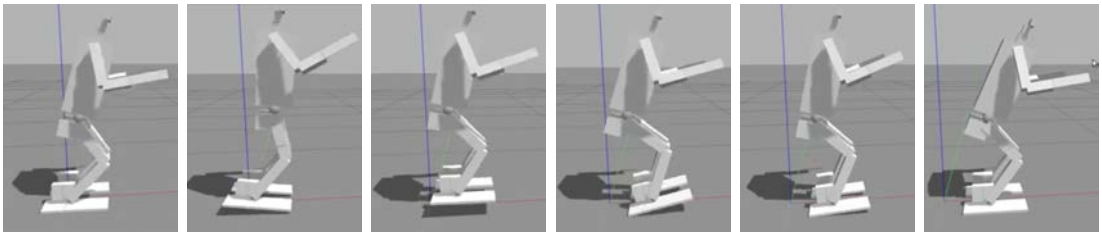


Figure 3.16: Polaris doing a jumping test in Gazebo

used Python with the matplotlib library. The next step was to implement a closed-loop feedback control. In order to test the accuracy of this control system, I used a new set of tools: Robot Operating System (ROS) 2016 and Gazebo [2017]. ROS is a set of software libraries and tools that helps to develop robot software. Gazebo is a physics enabled, high quality graphics simulation that is compatible with ROS.

Figure 3.16 demonstrates a model of Polaris in a Gazebo simulation during a jumping test. The next step was the implementation of the gyroscope and accelerometer to make them compatible with ROS and Gazebo.

The operating system that runs on the robot is Ubuntu Linux 16.04. I implemented two different walking engines in C++ and adapted the walk to ROS. All the modules use ROS to communicate with each other in real-time. *Kinetic Kame* is the 10^{th} official ROS release, which is currently running on Polaris and is compatible with my implementation.

3.4.1 Parametrized Walking Engine: Open-loop Control

An open-loop control is a control that does not use any sensor readings to modify or generate a trajectory. This situation is very close to a human whose eyes and ears are closed and who does not know if he/she is standing on the floor or lying on a bed.

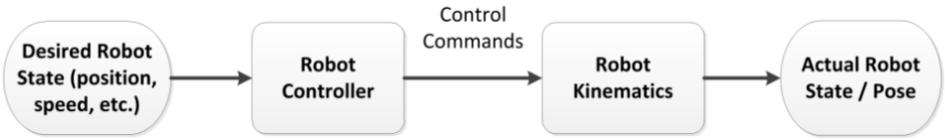


Figure 3.17: Open-loop control diagram [Lau, 2014]

In this situation, walking or recovering from a push is very difficult and everything is based on assumptions (e.g. the robot assumes that it is walking and did not fall). Figure 3.17 demonstrates an example of an open-loop control system for robots.

We can assume that the robot's ankles are perpendicular to the ground at all times. The robot then uses some predefined parameters to generate its walking or recovery path. This approach would work only for very small external disturbances during a walking gait. Any sudden change in the angle of the ground to the feet, or an impact, can cause a fall, since the robot does not have any feedback. One of the main goals of every robotic application is to make it fully autonomous [Hosseinmemar et al., 2016]. For a better performance in push recovery, other closed-loop control systems are discussed in Sections 3.4.2 and 3.4.3.

When using open-loop feedback control, the walk is almost the same all the time, and it does not matter if the robot walks on an uneven terrain or on a flat surface. For this research I designed and implemented two parameterized walking engines to test different balancing approaches. These approaches were discussed in Section 2.2.1.

The first engine that I implemented is based on taking a minimum of two steps (one complete cycle) at a time. That is, one left and one right step, or vice versa. Figure 3.18 and Equation 3.23 demonstrate a test for this engine with an open-loop control. I sent five cycles to this engine. The duration of one full cycle time for this

test was $650000 * 10^{-6}$ of a second.

$$\begin{aligned}(650000 * 10^{-6}) * 5 &= 3250000 * 10^{-6} \\ &= \frac{3250000 * 10^{-6}}{2} = 1.62 seconds\end{aligned} \tag{3.23}$$

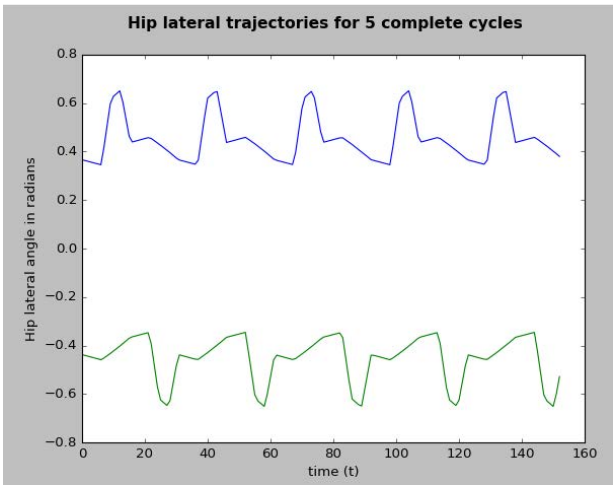


Figure 3.18: An open-loop control of five complete cycles on Polaris for the hip laterals

It will take 1.62 seconds for each foot to complete the given cycles. The blue trajectory in Figure 3.18 is the right hip lateral joint, and the green trajectory is describing the left hip lateral joint. As shown in Figure 3.18, all the waves in blue look the same and all the green waves look similar. The reason is that it is an open-loop control and any significant disturbance does not have any impact on generating trajectory.

The walking engine of Polaris is a parametrized walking engine. Five main parameters are used in the walking engine for adjusting and tuning its walk. These parameters are: *torso tilt angle*, *hip height*, *step length*, *step height* and *walking fre-*

quency. The walking frequency is calculated by $f = \frac{1}{T}$. Table 3.3 provides the details of each of the walking parameters respectively. The parameter T is one full cycle time that consists of two steps (one right step and one left step). It is measured in microseconds. In the example below (Equation 3.24), if the robot wants to take two steps (one left, one right) with the duration of $1000000 * 10^{-6}$ microseconds (one second), the walking frequency would be:

$$f = \frac{1}{1000000 * 10^{-6}} \Rightarrow f = \frac{1}{1.0} \Rightarrow f = 1 \tag{3.24}$$

Parameters	**Action**
Torso tilt angle	Adjust the robot's torso to the front or back based on angle
Hip height	Adjust the height of the robot based on knees degrees
Step length	Specify the length of each step (how long is 1 step)
Step height	Specify the height of each step
Walking frequency	Specify the number of steps per second

Table 3.3: General walking parameters

In Figure 3.19, a trajectory of the joints left hip lateral and right hip lateral with the frequency of one is demonstrated.

The second walking engine that I designed and implemented is based on single step cycle intervals. Every step can have different walking parameters, which helps the robot to be more balanced and agile. I mainly implemented this new walking engine for push recovery, or if Polaris needs to adapt its walk to a specific environment. However, it can be used as the normal walking engine too.

The output of the walking engine are the positions of the two feet. The current walking engine uses simple linear interpolation, which means the joints move in a straight line from one key frame to another. Since the robot does not receive any real-time sensory feedback in this control loop, it is impossible to make any corrections for

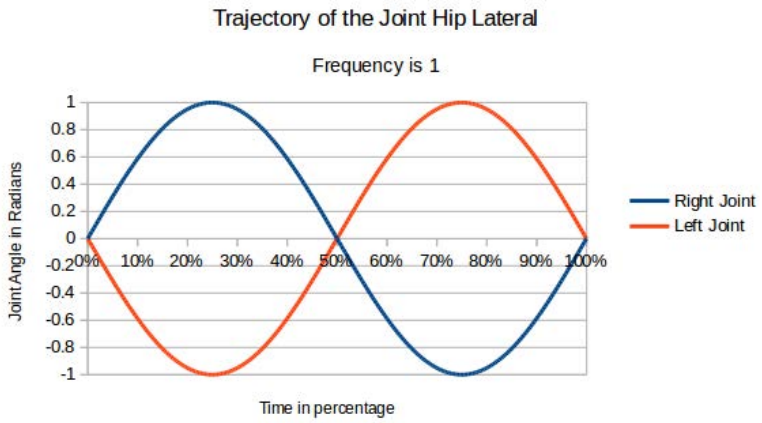


Figure 3.19: Trajectory of the left and right hip lateral for one complete cycle. The frequency is one.

the next step if that is necessary. Therefore, I designed a closed-loop control method, discussed in the following section.

3.4.2 Closed-loop Control: Using CMP

A closed-loop control system is a control system that employs one or more feedback loops. A humanoid robot using a closed-loop control system can receive feedback from its sensors (vision, inertia) describing changes to the environment (pushes, uneven terrain) and correct its trajectory to adapt to these changes. A closed-loop control system is more computationally expensive compared to an open-loop control system, due to the enormous amount of sensory data being processed by the controller. However, it is a much more robust control system because of this sensory feedback.

Polaris employs a walking engine based on the linear inverted pendulum model [Pratt et al., 2006; Lee and Goswami, 2007], which generates appropriate robot mo-

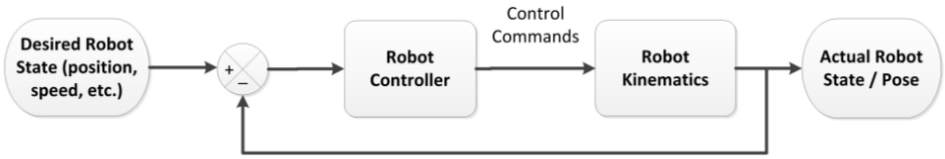


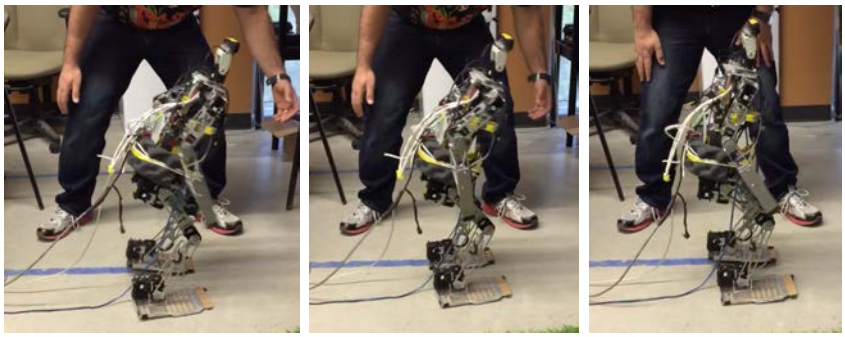
Figure 3.20: Closed-loop control diagram [Lau, 2014]

tions based on a description provided by the inverse kinematics of the robot (i.e. sets motion vectors for all servos over time). My closed-loop control mechanism sets inputs to the walking engine, allowing it in turn to adjust the robot's COM, dynamically altering this as the environment changes. As stated previously in Section 3.2.1.5, Polaris uses the sensory feedback from the IMU. Both the gyroscope and the accelerometer in the IMU are used as the inputs to measure the control output. The walking engine will adapt the robot's trajectory (control output) as necessary to deal with changes in the environment, from varying terrain to external forces.

Figure 3.20 demonstrates an example of an closed-loop control system for robots.

I describe my control approach through three phases illustrated in Figure 3.21, showing a sample push recovery using this approach. This control approach was accepted for publication at the IEA-AIE 2018 International conference [Hosseinmemar et al., 2018] and was declared as the second best award winning paper among 146 accepted papers . In Stage 1 of the figure, the robot is pushed by hand at a point in its walking gait on a concrete floor, and must recognize that it is in a falling state. Stage 2 represents a brief window in which the robot can calculate a reaction to the push by devising control changes based on the angular velocity and the linear velocity of its torso. Stage 3 illustrates recovery, where these control changes alter parameters in the robot's walking engine, and these in turn adjust servos accordingly to prevent

the robot from falling.



(a) Stage 1 (b) Stage 2 (c) Stage 3

Figure 3.21: Push, Reaction, and Recovery (using CMP).

In Stage 1, the closed-loop controller takes as input values from the gyroscope and/or accelerometer, and from these must detect a falling state. In practice this can be computationally expensive because of the range of potential values. To allow a fast response, I discretize values. The control methodology categorizes angular velocity in 50 degrees per second intervals, allowing a definition of constant values for light, medium and strong pushes. This interval was chosen based on three, 5-minute robot walk tests to find the maximum angular velocity that Polaris encountered naturally while using different walking parameters for each test. I similarly experimented with linear velocity thresholds by hitting the robot with a 2 Kg weight (more than a quarter of its body weight), and defined light, medium, and strong pushes as linear velocities of 0.9 m/s, 1.2 m/s and 1.4 m/s, respectively. Figure 3.22 shows these discretized values.

My closed-loop control collects 1000 gyroscope and accelerometer readings per second from the IMU, and uses these to continually check if the robot is in a falling

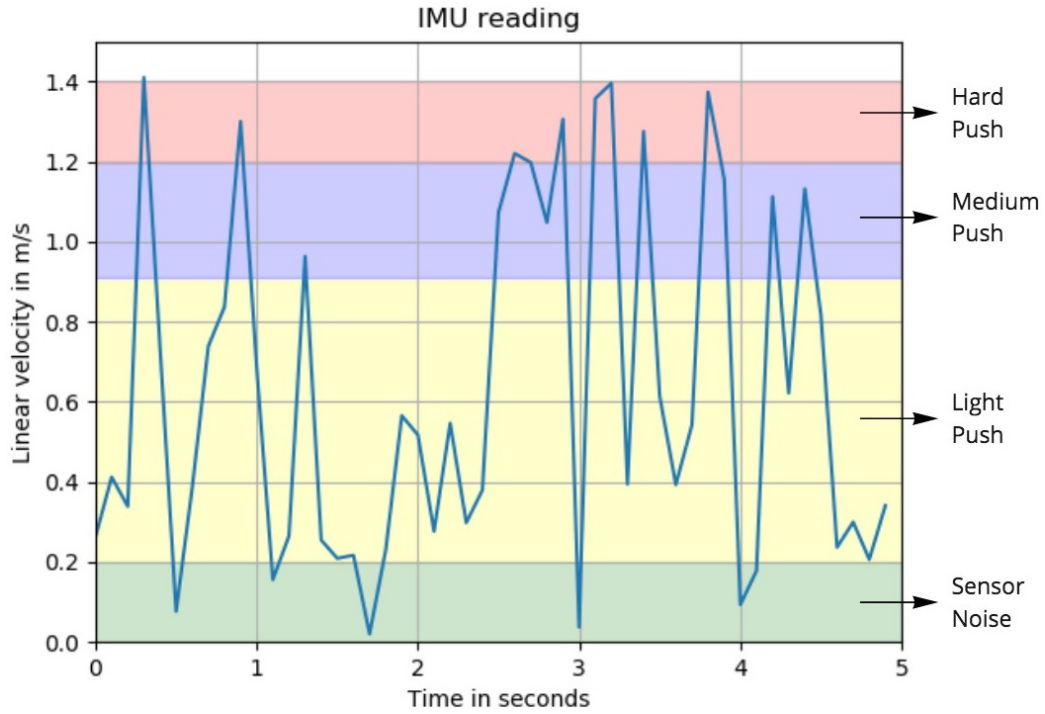


Figure 3.22: IMU linear velocity discretization

state. Experiments showed that the frequency of the applied pushes were less than 1kHz. Therefore, the IMU could sense all the applied pushes during the tests and it shows that data rate of 1000 samples per second would be more than enough for any type of in-range push.

Based on the three thresholds for linear and angular velocities, the robot can very quickly determine whether it is in a stable state ($0\ m/s >$ linear velocity $\leq 0.2\ m/s$, with this range necessary to deal with sensor noise), or when it is in a falling state by exceeding angular velocities or linear velocity thresholds for light, medium, or strong pushes.

In the brief window between when a falling state is recognized and when the fall would be irreversible (Stage 2), the robot must make an appropriate response.

Knowing which threshold (light, medium, or strong push) has been exceeded allows for a quick response look-up. My control approach implements CMP active balancing (Section 2), which incorporates COM if only the ankles are moved, and similarly discretizes potential responses to support a fast reaction.

I discretized the robot's responses into 9 *discrete action* spaces. Based on these discrete actions, my closed-loop controller produces nine outputs that are used to alter parameters in the robot's walking engine through modifying hip and/or ankle positions. These are illustrated in Table 3.4. *Step-x* is the step length on the x-axis of the robot frame (forward-backward). *Step-y* is the step length on the y-axis of the robot frame (left-right). *Step-height* is the foot height from the ground of each step. The *x-offset* is the offset distance of the feet from the origin/centre of the robot (centre point of the torso) on the x-axis. *Y-offset* is the offset distance on the y-axis from the centre of the torso to each foot. *Z-offset* is related to the height of the robot, i.e. standing fully vs. in a crouch. *Step-pace* is the robot's speed in terms of the time it takes to take a single step (not a full walking cycle). The final two parameters are *hip-pitch* and *ankle-pitch* for the lateral motion at the hip and ankles respectively.

Each of these walking engine parameters has minimum and maximum values, also indicated in Table. 3.4. Any setting acts as an offset value for the current robot pose. For example, if the robot's torso is leaning too much to the front, the robot will fall over. To encounter this problem, *hip-pitch* can be tuned to adjust the torso's lateral motion in order to prevent the robot from falling.

All nine of these responses have particular values based on the strength and direction of the push that has been recognized. Values can be zero, indicating no change.

Walking Engine Parameters	**Threshold**	
	Minimum	**Maximum**
Step-x	-5 cm back	+5 cm front
Step-y	-5 cm right	+5 cm left
Step-height	1 cm	8 cm
x-offset	-5 cm back	+5 cm front
y-offset	+4 cm	+8 cm
z-offset	34 cm	44 cm
Step-pace	125,000 μs	500,000 μs
Hip-pitch	-15° tilt back	+15° tilt front
Ankle-pitch	-10° tilt back	+10° tilt front

Table 3.4: Walking engine parameters and value ranges.

For example, a light push on the right side results in detecting a fall to the left, and modifies the *step-y* value by 2 CM and the *step-pace* value by 0.5 per second, leaving all other values unchanged. These values and their mapping to discretized angular and linear velocities (fall states) have been tuned over several years of robotics competitions as well as testing the robot specifically under push recovery conditions. These have proven themselves in the field to be a very fast method of adapting control to changing conditions (e.g. carpet vs. hard surfaces) in addition to push recovery.

Once the appropriate response has been mapped, the parameters to the walking engine are changed (Stage 3), and servos are collectively altered in the time window that remains to correct for the disturbance. Central to all of this is making Stage 2 as short as possible, leaving time for the servos to be adjusted to recover from the fall. Figure 3.23 illustrates a complete transition of the three stages on a concrete surface in the Autonomous Agents Laboratory. For pushing the robot for this test, I used my hand and based on my experience, I tried to apply a medium level push to the robot.

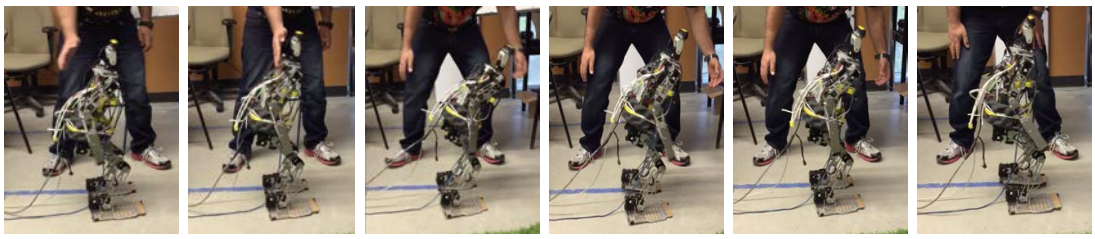


Figure 3.23: Recovering from a push (sequence starts from left)

To test my closed-loop control approach more accurately so that the amount of force is known and not an approximate value, I tested this approach on an artificial turf that counts as a different surface. Recovery on turf with a soft surface is a harder task for the robot compared to a concrete surface. I also used a solid 2 Kg weight suspended from a rope, for hitting the robot from different distances by releasing the weight, to provide a reproducible push strength. Figure 3.24 demonstrates Polaris employing CMP (discussed in Section 3.4.2) for push recovery with the 2 Kg weight released from a 40 CM distance.

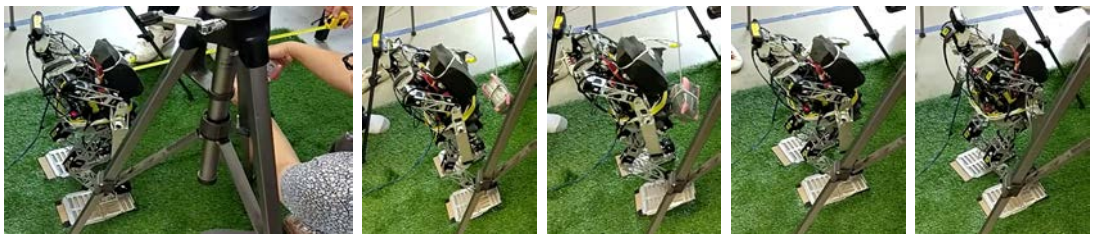


Figure 3.24: Releasing a 2 Kg solid weight from 40 CM distance to hit Polaris's torso standing on the turf (recovering sequence starts from left, using CMP)

Based on the strength and direction of the push, I discretized the states of the robot into 14 states that can be mapped with a look-up table. Figure 3.25 shows all the 14 states. For every direction {front, right, left, back} there are four distinct

discrete states plus two common discrete states. The complete set of states are: 1) Stable state: the robot is stable in this state, and this is one of the common discrete states. 2) Light push state: ω, which represents angular velocity, is $\leq \theta_l$, where θ_l denotes the maximum ω that a 2 Kg object suspended from a 1 meter rope causes after being released from a 30 CM distance. 3) Medium push state: ω is $\leq \theta_m$, where θ_m denotes the maximum ω that a 2 Kg object causes after being released from a 40 CM distance. 4) Strong push state: ω, is $\leq \theta_s$, where θ_s denotes the maximum ω that a 2 Kg object causes after being released from a 50 CM distance. 5) Fall state: the robot fell and could not recover successfully. This last state is the other common discrete state.

3.4.3 Closed-Loop Control: Using Step-out

Even though Polaris was able to recover from many different light and medium pushes from different directions, and on different surfaces (e.g. concrete and carpet), Polaris was not able to recover from a strong push using the techniques described in the previous section. To give the robot the ability to recover from even stronger external forces, I designed and added a stepping functionality to the robot's push recovery module. By taking extra steps, the robot can handle the stronger external pushes more easily, and less pressure will be applied to the servo motors during the recovery process. Therefore, the walking engine was modified to take an individual step (half of a complete cycle) for faster reactions to the external disturbances. This means the robot can take only one step with either the right or left foot, not a full walking cycle. I implemented the stepping in such a way that every step has its

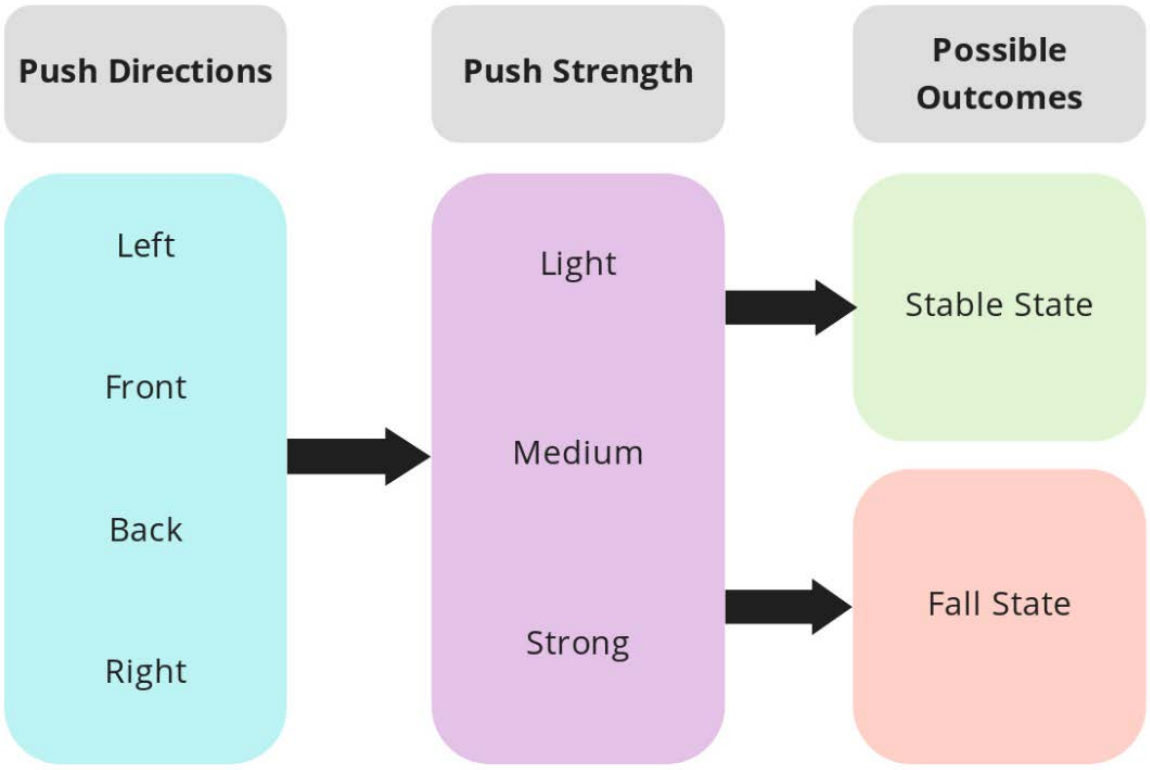


Figure 3.25: Discretization of the robot states (4 push directions * 3 push strength = Total number of 12 discretized states)

own walking engine parameters, as shown in Table. 3.4. Based on different walking situations, these parameters can be altered. My stepping approach is an extension of the CMP model [Hosseinmemar et al., 2018] that was discussed in Section 3.4.2. Similar to my CMP model, that has three stages (Stage1, Stage2, and Stage3) as shown in Figure. 3.21.

Figure 3.26 demonstrates a successful push recovery by taking extra steps. The 2 Kg weight was released from a 50 CM distance, causing a strong push when hitting the robot. For this experiment, to examine the robustness of my closed-loop control, I tested the robot on a different floor. Artificial turf is one of the most difficult surfaces

for the robot in terms of recovery. The turf's surface is very soft and usually its height is between 2-3 CM, based on our competition experiences. Walking on such a surface alone makes the robot unstable. Applying pushes from different directions to the robot with an external object can easily cause a fall.

In Figure 3.26, during Stage 1 (Push), Polaris reads the value of falling state from its sensory feedback value from gyroscope and accelerometer. Then in Stage 2 (Reaction), Polaris maps the sensory feedback values to the discretized table, and obtains the strength of the force. Finally, in Stage 3 (Recovery), Polaris updates its walking engine parameters to counter the applied push, and recovers successfully.

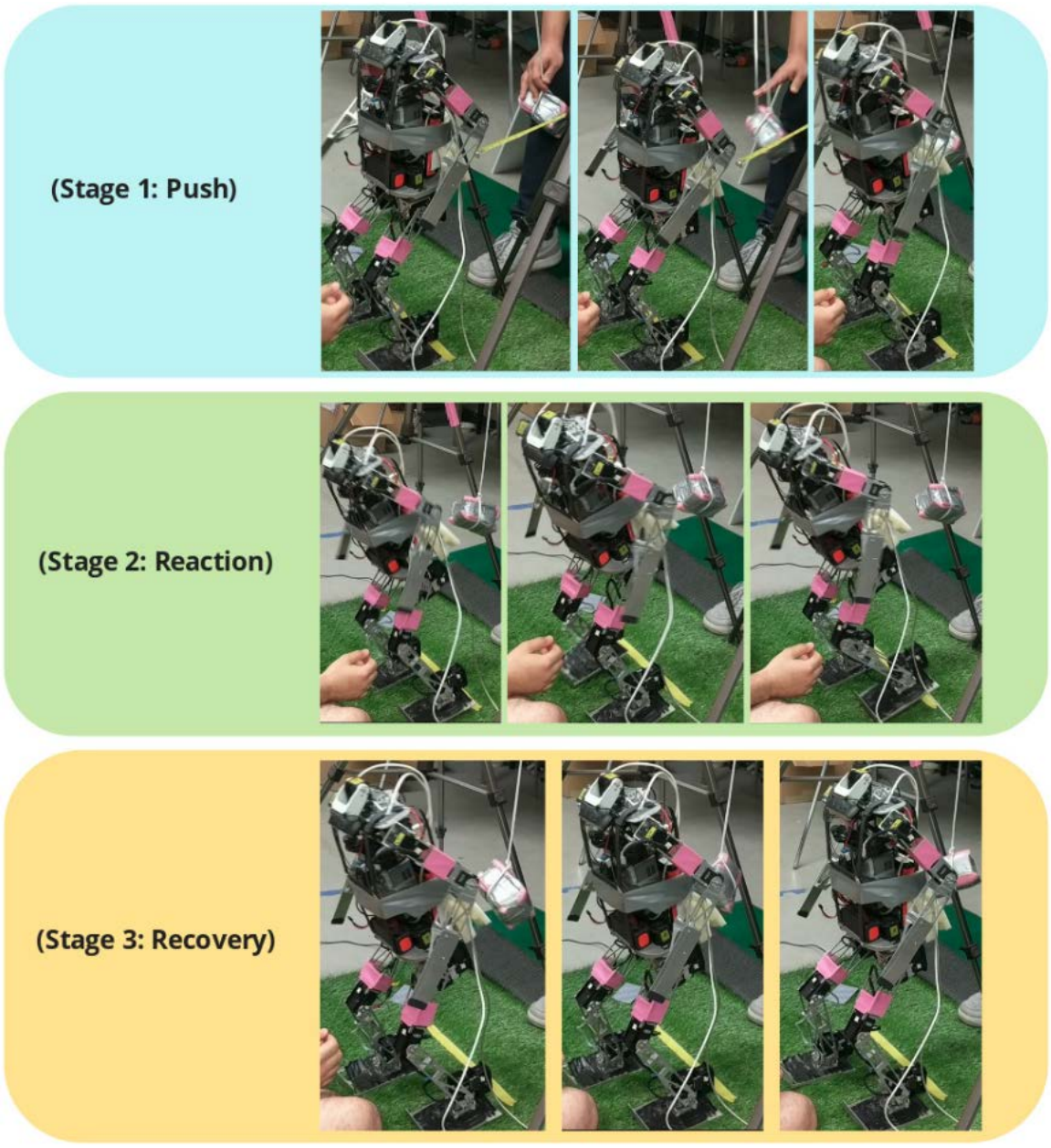


Figure 3.26: Push, Reaction, and Recovery (using stepping strategy, sequence starts from left).

3.4.3.1 IMU Simulation

In Section 3.4.2, I explained how the IMU was discretized into different states in order to detect the push. Also, I used both angular and linear velocities to find the direction of a push as well as its force. I used the stride based walking engine which is based on one right and one left steps. The results of this approach were promising if the robot knew which surface was standing on (for example, turf, carpet or concrete) while Polaris was using the stride based walk. After I examined the previous implementation of the IMU with the single step walking engine and unknown surfaces, I found out that this IMU implementation would not be sufficient for my final task. For that reason I designed and implemented a new approach for calculating the force and its direction.

In this approach I just use the accelerometer of the IMU module. The main reason that I did not use the gyroscope for this approach was that gyroscope was not useful if the surface was not known. This meant I had to measure all the angular and linear velocities on 3 different surfaces individually. Also, I previously had to let Polaris know that it is standing on a particular surface (for example, artificial turf), then it read the correct discretization from the gyroscope. If the robot is not informed what surface it is standing or walking on, the problem of active balancing becomes much harder. In my new approach, I did not let Polaris know about the surfaces at all times (surfaces are partly unknown). This way we have a more universal solution to the push recovery problem. I used the acceleration in the x-axis (front and back) and y-axis (right and left) to detect pushes, measuring the magnitude and direction

of applied force. The calculation of the magnitude of the force is given by

$$effective\ acceleration = \sqrt{a_x + a_y} \tag{3.25}$$

where a_x is the acceleration in the x axis and a_y is the acceleration in the y axis. I also calculated the angle at which the torso of Polaris was hit using

$$angle = \frac{tan2^{-1}(a_x, a_y)}{\pi * 180} \tag{3.26}$$

To examine the accuracy of the accelerations and the angles that were calculated based on Equations 3.25 and 3.26 in practice before running my experiments, I assigned 150 pushes for each discretized state - this amounted to 1800 (12 states * 150 pushes) pushes in total in the simulation. Figure 3.27 shows the top view of the applied pushes in four different directions. The robot is at the center of this circle, facing at 180° for all the trials. The maximum linear acceleration that the accelerometer recorded for a push while Polaris was walking was $0.72m/s^2$. In total 450 pushes were applied to 0° (back), 450 pushes were applied to 90° (right), 450 pushes were applied to 180° (front), and 450 pushes were applied to 270° (left). In Figure 3.27, the further the colorful circles are to the center of the white circle, the stronger the pushes are, with a bigger circle area. This figure is based on the recorded angles and their corresponding acceleration at the impact times. Since Polaris was walking during every push, the recorded angles are not exactly $0°, 90°, 180°, 270°$.

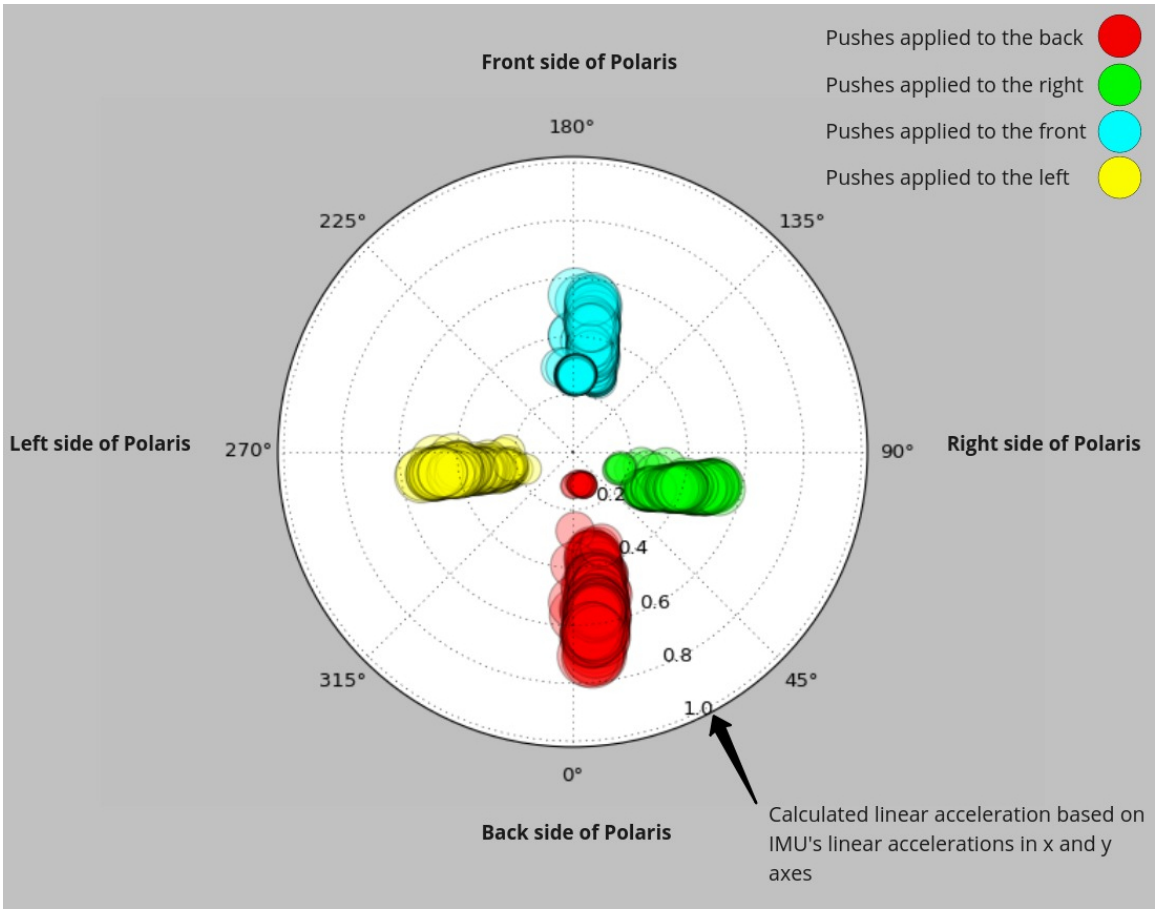


Figure 3.27: Top view of the 1800 (12 states * 150 pushes) trial applied pushes to four sides (0° = back, 90° = right, 180° = front, 270° = left) of Polaris's torso during its walk. Bigger circles signify stronger pushes.

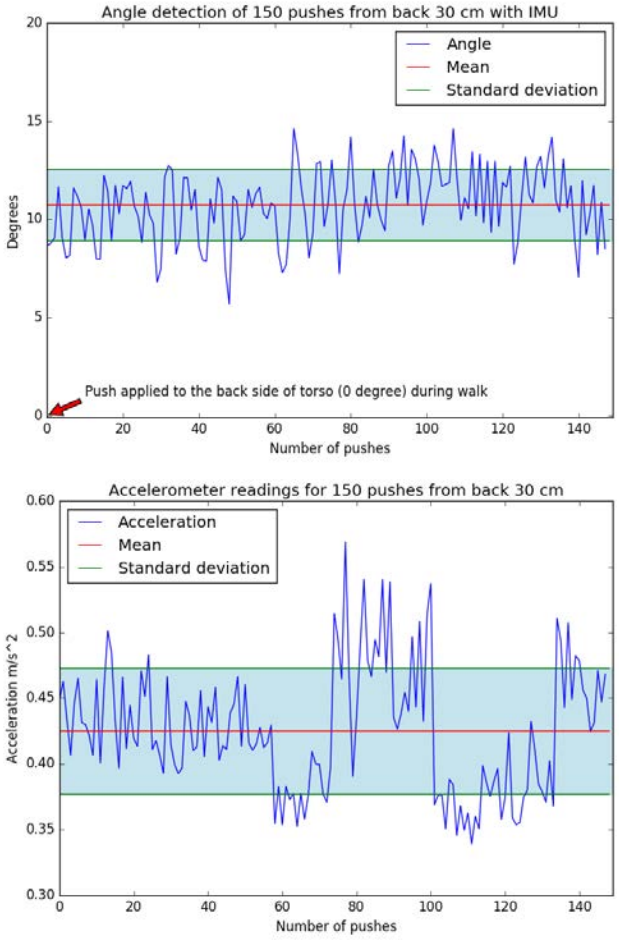


Figure 3.28: 150 pushes applied to Polaris's back from 30 CM. Top image shows the detected angles of the pushes. Bottom image shows the corresponding accelerations of the pushes.

Figure 3.28 shows the readings of 150 pushes that were applied to Polaris's back (0 degree in polar space in the simulation) with an approximate force equal to releasing the bottle from a 30 CM distance. The applied force is calculated based on equation 3.27. This test was during a normal walking gait (two complete strides).

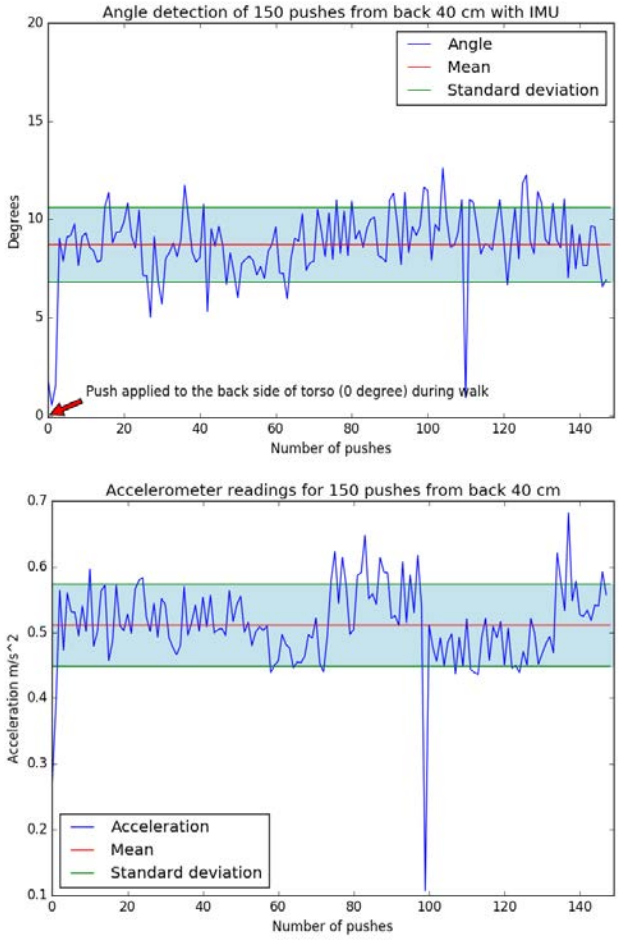


Figure 3.29: 150 pushes applied to Polaris's back from 40 CM. Top image shows the detected angles of the pushes. Bottom image shows the corresponding accelerations of the pushes.

Figure 3.29 shows the 150 pushes that were applied to Polaris's back (0 degree in polar space in the simulation) with an approximate force equal to releasing the bottle from a 40 CM distance. The applied force is calculated based on Equation 3.27. This test was during a normal walking gait (two complete strides).

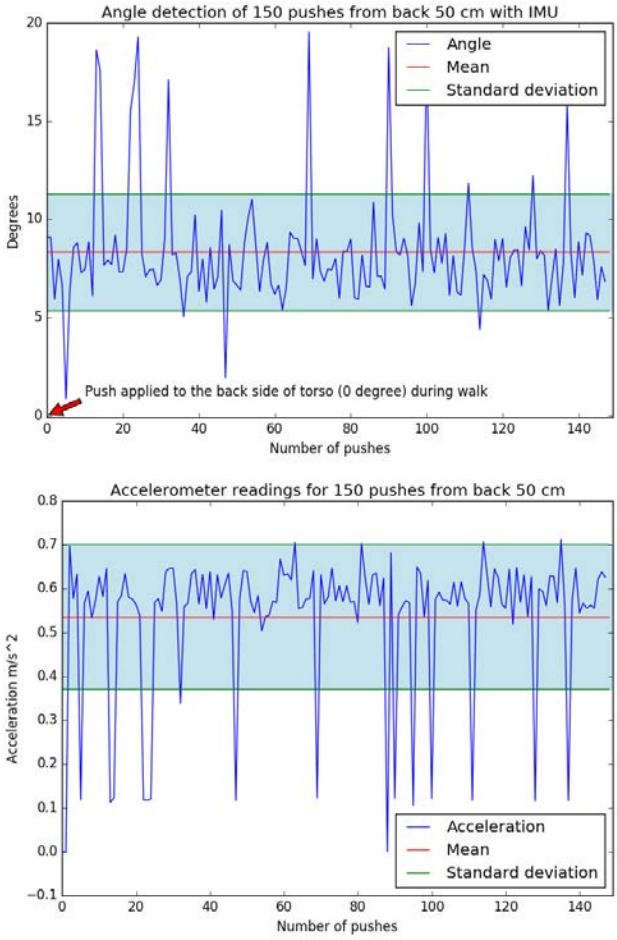


Figure 3.30: 150 pushes applied to Polaris's back from 50 CM. Top image shows the detected angles of the pushes. Bottom image shows the corresponding accelerations of the pushes.

Figure 3.30 shows the 150 pushes that were applied to Polaris's back (0 degree in polar space in the simulation) with an approximate force equal to releasing the bottle from a 50 CM distance. The applied force is calculated based on Equation 3.27. This test was during a normal walking gait (two complete strides).

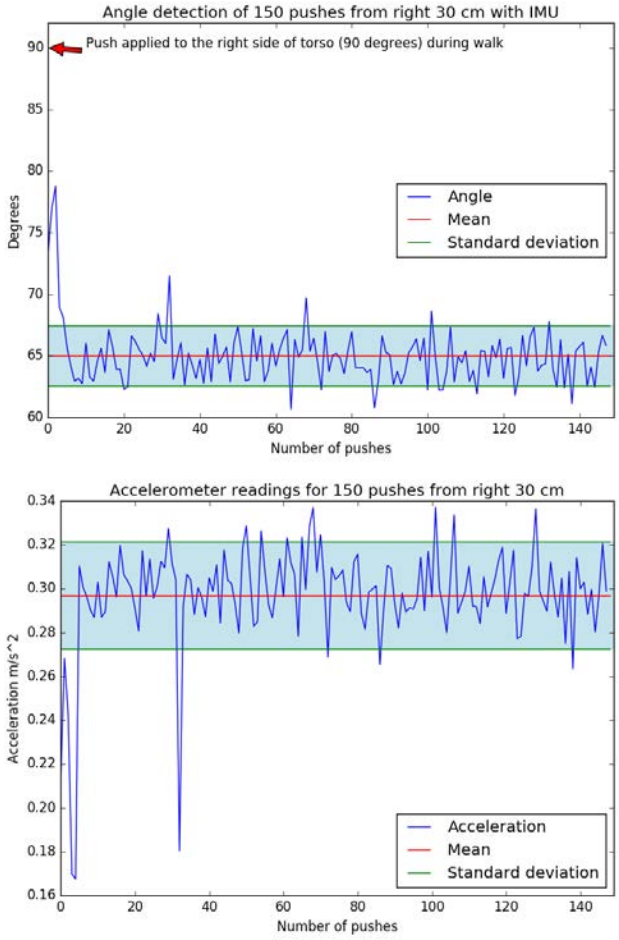


Figure 3.31: 150 pushes applied to Polaris's right side from 30 CM. Top image shows the detected angles of the pushes. Bottom image shows the corresponding accelerations of the pushes.

Figure 3.31 shows the 150 pushes that were applied to Polaris's right side (90 degrees in polar space in the simulation) with an approximate force equal to releasing the bottle from a 30 CM distance. The applied force is calculated based on Equation 3.27. This test was during a normal walking gait (two complete strides).

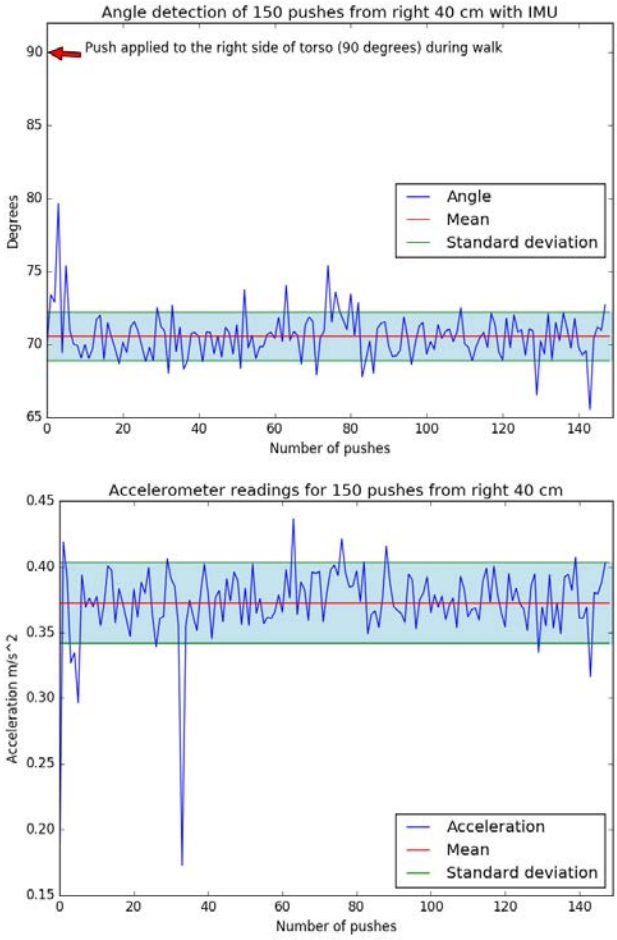


Figure 3.32: 150 pushes applied to Polaris's right side from 40 CM. Top image shows the detected angles of the pushes. Bottom image shows the corresponding accelerations of the pushes.

Figure 3.32 shows the 150 pushes that were applied to Polaris's right side (90 degrees in polar space in the simulation) with an approximate force equal to releasing the bottle from a 40 CM distance. The applied force is calculated based on Equation 3.27. This test was during a normal walking gait (two complete strides).

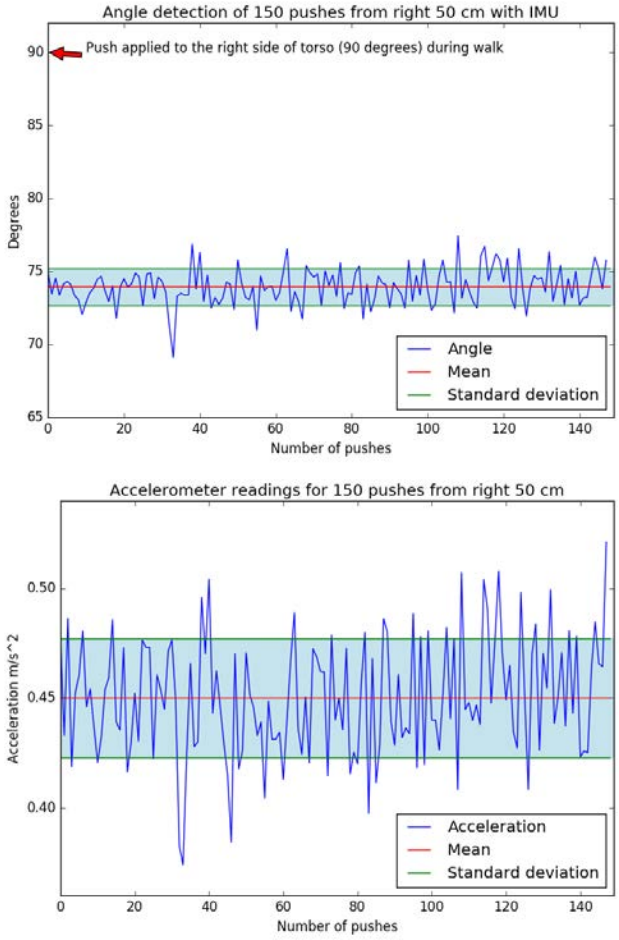


Figure 3.33: 150 pushes applied to Polaris's right side from 50 CM. Top image shows the detected angles of the pushes. Bottom image shows the corresponding accelerations of the pushes.

Figure 3.33 shows the 150 pushes that were applied to Polaris's right side (90 degrees in polar space in the simulation) with an approximate force equal to releasing the bottle from 50 a CM distance. The applied force is calculated based on Equation 3.27. This test was during a normal walking gait (two complete strides).

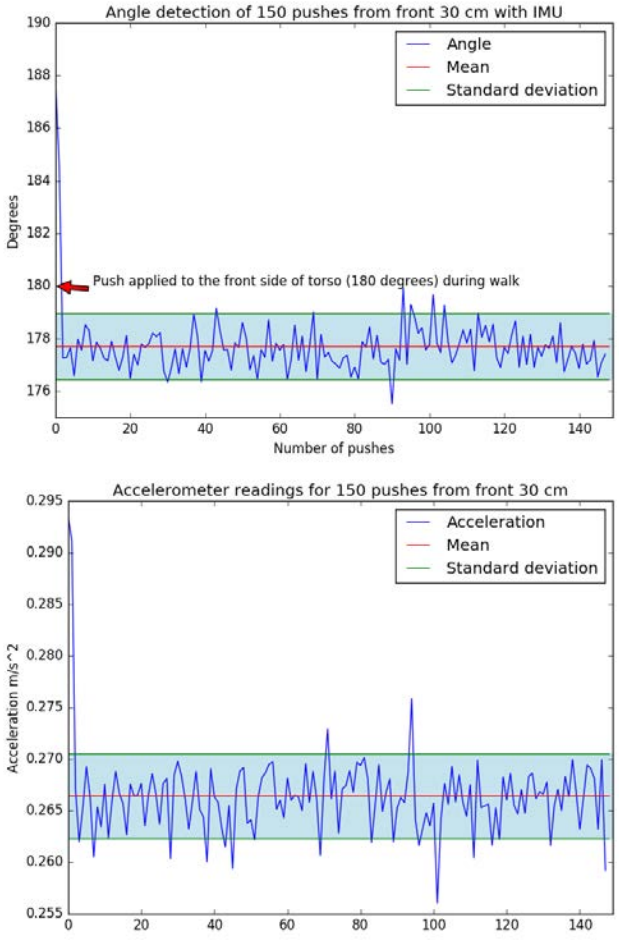


Figure 3.34: 150 pushes applied to Polaris's front side from 30 CM. Top image shows the detected angles of the pushes. Bottom image shows the corresponding accelerations of the pushes.

Figure 3.34 shows the 150 pushes that were applied to Polaris's front side (180 degrees in polar space in the simulation) with an approximate force equal to releasing the bottle from a 30 CM distance. The applied force is calculated based on Equation 3.27. This test was during a normal walking gait (two complete strides).

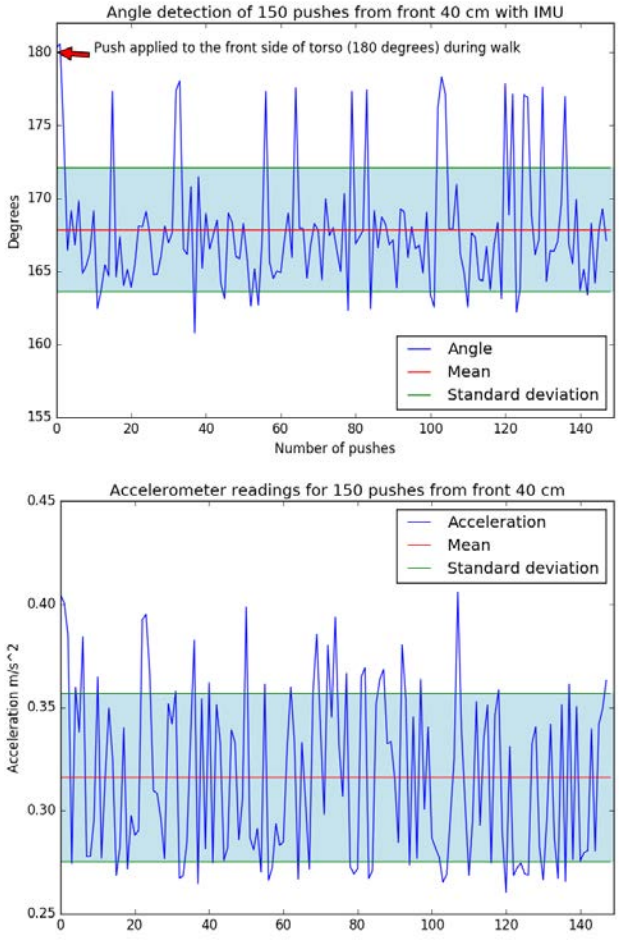


Figure 3.35: 150 pushes applied to Polaris's front side from 40 CM. Top image shows the detected angles of the pushes. Bottom image shows the corresponding accelerations of the pushes.

Figure 3.35 shows the 150 pushes that were applied to Polaris's front side (180 degrees in polar space in the simulation) with an approximate force equal to releasing the bottle from a 40 CM distance. The applied force is calculated based on Equation 3.27. This test was during a normal walking gait (two complete strides).

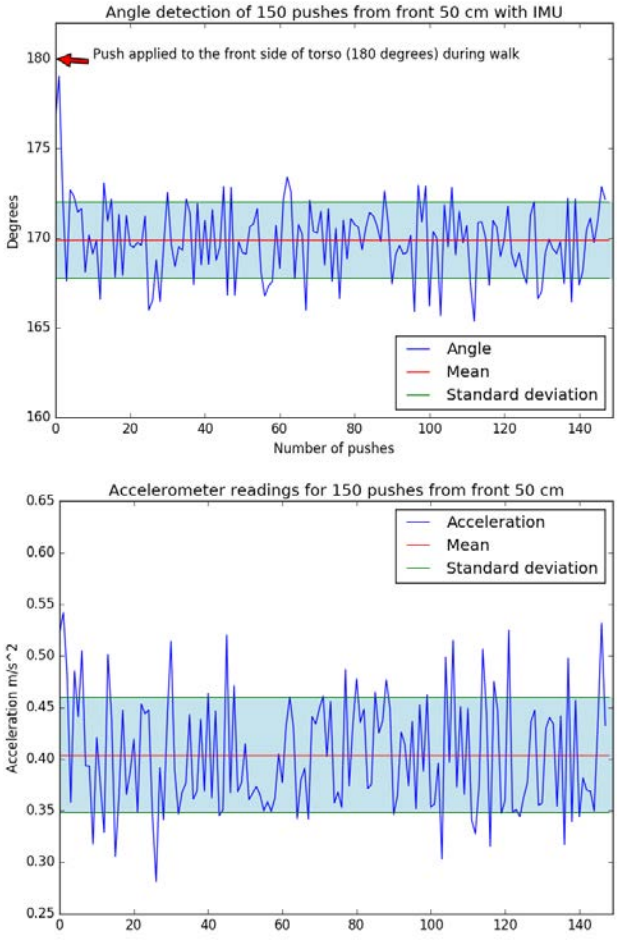


Figure 3.36: 150 pushes applied to Polaris's front side from 50 CM. Top image shows the detected angles of the pushes. Bottom image shows the corresponding accelerations of the pushes.

Figure 3.36 shows the 150 pushes that were applied to Polaris's front side (180 degrees in polar space in the simulation) with an approximate force equal to releasing the bottle from a 50 CM distance. The applied force is calculated based on Equation 3.27. This test was during a normal walking gait (two complete strides).

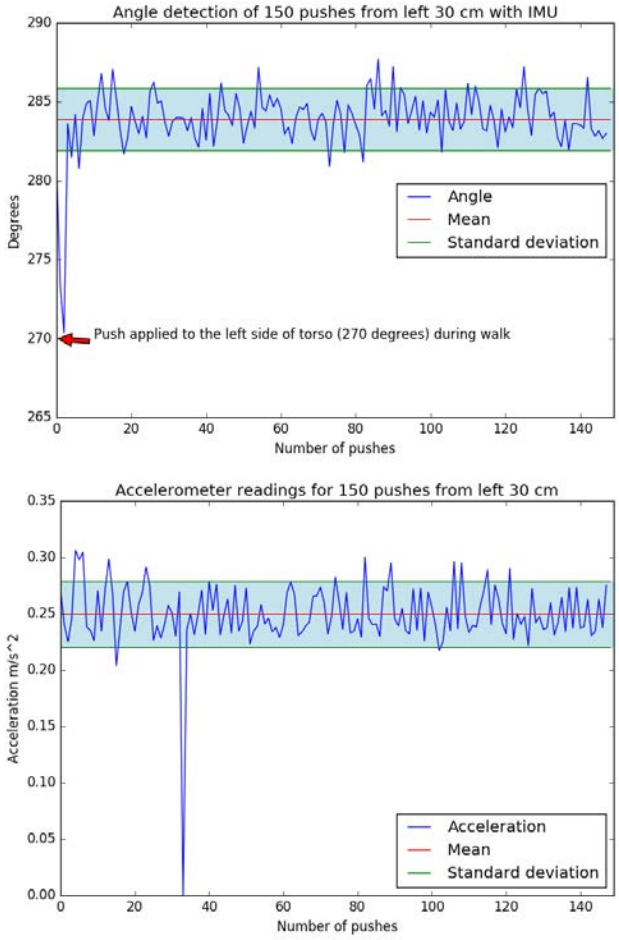


Figure 3.37: 150 pushes applied to Polaris's left side from 30 CM. Top image shows the detected angles of the pushes. Bottom image shows the corresponding accelerations of the pushes.

Figure 3.37 shows the 150 pushes that were applied to Polaris's left side (270 degrees in polar space in the simulation) with an approximate force equal to releasing the bottle from a 30 CM distance. The applied force is calculated based on Equation 3.27. This test was during a normal walking gait (two complete strides).

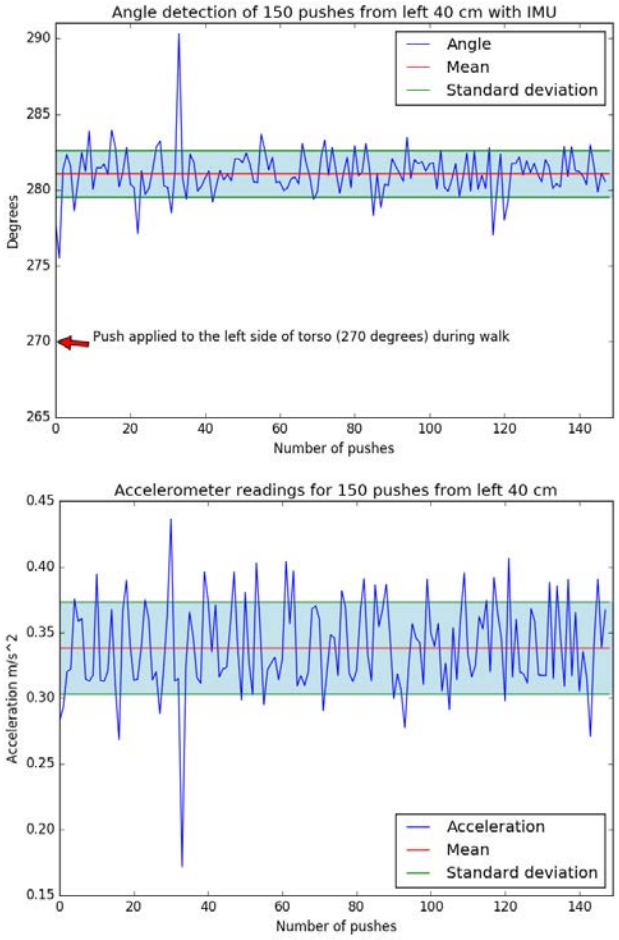


Figure 3.38: 150 pushes applied to Polaris's left side from 40 CM. Top image shows the detected angles of the pushes. Bottom image shows the corresponding accelerations of the pushes.

Figure 3.38 shows the 150 pushes that were applied to Polaris's left side (270 degrees in polar space in the simulation) with an approximate force equal to releasing the bottle from a 40 CM distance. The applied force is calculated based on Equation 3.27. This test was during a normal walking gait (two complete strides).

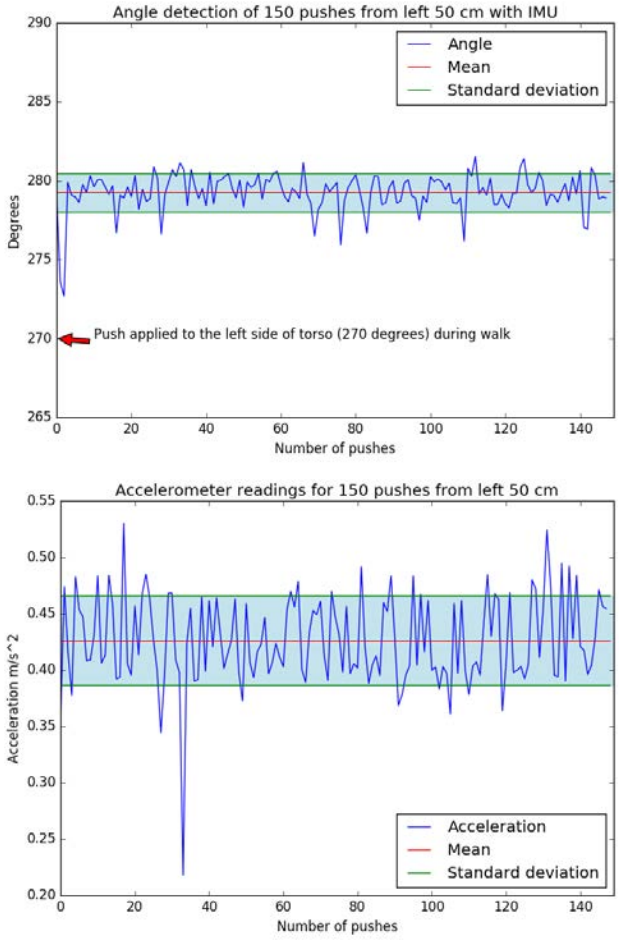


Figure 3.39: 150 pushes applied to Polaris's left side from 50 CM. Top image shows the detected angles of the pushes. Bottom image shows the corresponding accelerations of the pushes.

Figure 3.39 shows the 150 pushes that were applied to Polaris's left side (270 degrees in polar space in the simulation) with an approximate force equal to releasing the bottle from a 50 CM distance. The applied force is calculated based on Equation 3.27. This test was during a normal walking gait (two complete strides).

3.4.4 Approximating the real world force in simulation

At the beginning of every push, the robot was positioned in the environment, standing in a stable state. According to RoboCup's rule for push recovery as a technical challenge, the robot should be in a walking mode while it is being hit by the external object. For that reason, every time before the robot was hit, Polaris took two steps (one right and left). Then some fixed measured external forces that are equal to releasing the 2 Kg weight from different distances was applied to the COM of Polaris (the bottom of its torso) in a particular direction. In order to calculate these forces, I used the following forumulas [Pook, 2011]:

$$F = m \times a \tag{3.27}$$

Equation 3.27 is for calculating the force. In this equation, F denotes the force, m denotes the mass of the object being released (2 Kg), and a denotes the acceleration of the released object. Both F and a are not known. By finding the velocity of the object, we can then calculate the force.

$$V = \sqrt{2gl(1 - cos\theta)} \tag{3.28}$$

Equation 3.28 calculates the velocity of the point mass, m. In this equation, g is acceleration due to gravity, l is the length of the pendulum, and θ is the angle of the pendulum. This equation is derived from the potential energy, W, and the kinetic energy, K. Equations 3.29 and 3.30 show these two energy equations respectively.

$$W = mgl(1 - \cos\theta) \tag{3.29}$$

Distance	**30 cm**	**40 cm**	**50 cm**
Applied Force	16 N	21 N	27 N

Table 3.5: The three calculated forces based on Equation 3.27 in which were applied to the bottom of Polaris's torso in the simulation.

$$K = \frac{1}{2}mV^2 \tag{3.30}$$

To calculate θ in Equation 3.28, I used the formula $\sin^{-1}(\frac{d}{L})$, where d is the ground projected distance ($\{30, 40, 50\}_{cm}$) between the robot's torso and the bottle, and l is the length of the rope (140 CM).

The acceleration of an object can be expressed as:

$$a = \frac{\Delta v}{\Delta t} \tag{3.31}$$

where Δt is an approximate time that the velocity reaches from its maximum to 0. After few trials I recorded 0.1 second as Δt. Hence, we can rewrite Equation 3.27 as

$$F = m \times \frac{\Delta v}{t} \tag{3.32}$$

Table 3.5 shows the calculated forces that I applied to the bottom of the torso of Polaris in every push.

Following impact, the robot detected the push with its IMU, and took a set of actions (one step).

3.4.5 Machine Learning

3.4.5.1 Software

For the machine learning part of my research I used the TensorFlow (version 1.9) [Google, 2017] framework with Python (version 2.7). I used Python as the programming language to implement the learning and testing parts of my research. In the following sections I am going to explain the two learning approaches that I employed and implemented in my research.

3.4.5.2 Reinforcement Learning

Reinforcement Learning (RL) is learning what to do - how to map situations to appropriate actions - so as to maximize a numerical reward signal [Sutton and Barto, 1998]. The learner is not told what set of actions to take in different situations, but must instead discover which actions yield the best reward over time. These actions are usually selected randomly in the beginning when the learner does not have enough information about the environment, in order to gain feedback and make better choices as time goes on [Cassandra, 1998] .

In addition to the set of actions a available for the robot to perform, RL generally has four basic main components [Ng et al., 1999]: a *policy*, *reward function*, *value function* and a *model of the environment*. At the very beginning, the robot has some given knowledge of the world, but does not know anything about the policy to be learned. The robot continually alters the environment by taking actions, and then perceiving the results as a new state s_{t+1} (where time t falls into discrete steps t = 0, 1, 2, 3 ...). The environment is uncertain - executing an action may not lead to

a single successor state, but rather results in one of several possible successor states where the probability of reaching a given successor state is governed by an unknown probability distribution. In addition to perception, the robot also receives a numeric reward based on its actions.

In order to learn, the robot constructs a mapping from the available states to the available actions, and calculates the probabilities of given outcomes from selecting each possible action for that particular state. This whole mapping is called the *policy* and is denoted by π. When referring to the *policy* at a particular time point t during learning, it is denoted π_t and for the particular π_t there will be a particular state and action, denoted as $\pi_t(s, a)$, which produces the probability that the action a should be chosen in state s [Watkins and Dayan, 1992]. The other components of RL serve to improve the *policy*.

By receiving some representation of the environment state s^e, where e denotes the environment, for any particular time t we have $\Rightarrow s_t^e \mid\in S$ and $\forall s_t \in S \exists a_t \in A$ where a_t denotes an action for that s_t and A denotes the set of available actions [Watkins and Dayan, 1992]. For the next step s_{t+1} may be rewarded by a real number $r_{t+1} \in \aleph$ for the action that it took previously. $\aleph$ denotes real numbers.

This type of RL is called Q-learning [Watkins and Dayan, 1992]. There are also different mathematical models of RL such as *Sarsa* [Cassandra, 1998]; however, I used the Q-learning technique as the RL mechanism in my work. Equation 3.33 shows the components of the Q-learning formula [Watkins and Dayan, 1992] .

$$\begin{aligned} Q(state, action) = R(state, action) + gamma \\ *Max[Q(next\_state, all\_possible\_actions)] \end{aligned} \tag{3.33}$$

In this equation, gamma (γ) ranges between 0 and 1. If γ is closer to 0, we are more interested in immediate rewards and if this number is closer to 1 we are more interested in delayed rewards. We can then rewrite equation 3.33 as:

$$Q(s,a) = R(s,a) + \gamma max_{a'}(Q(s',a')) \text{ [Watkins and Dayan, 1992]} \tag{3.34}$$

Where *R(s,a)* denotes the reward function, s' denotes the next state after the action a was taken, a denotes the action in state s and a' denotes action in state s'. In general the reward function can be represented in two different ways: 1) In situations where the task is *episodic* (discrete state and discrete action); 2) In situations in which we are dealing with continuous state and continuous actions. Equations 3.35 and 3.36 show these two situations respectively.

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_{t+n} \text{ [Watkins and Dayan, 1992]} \tag{3.35}$$

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma r_{t+3}^2 + \gamma r_{t+4}^3 \ldots + r_{t+n}^{n-1} = \sum_{k=0}^{n} \gamma^k r_{t+k+1} \text{[Watkins and Dayan, 1992]} \tag{3.36}$$

Polaris was rewarded +100 if it was pushed and took a proper set of actions that did not let it fall. Otherwise, -100 was given, as the punishment. Before choosing these two arbitary numbers, I tested the learning process with +100 and -10000. However, I did not find this set of numbers to be a good combination. Since the robot's learning process was in real-time (no fast forwarding in the simulation), the procedure of every push took 10 seconds from reseting to the fall or stable state in the simulation environment. Because of this constraint, I set the initial number of random tries to 500 for every direction and distance. For example when Polaris

started learning the actions for pushes from front direction and the distance of 30 CM, it was allowed to take only 500 random sets of actions (in the given threshold: Table. 3.4) for the front direction with the distance of 30 CM. After it explored randomly for 500 times, it started to exploit from its previous exploration. Based on my previous experiences, I chose the learning rate of 0.05 for the next step - that is, taking actions based on the robot's previous experience. All the discretized domains had a chance of having 500 random actions at the very beginning of learning. However, the total number of successful and unsuccessful actions that the robot made for each direction and each distance was not necessarily the same.

After an action was taken by the robot, the look-up table was updated to reflect the newly-earned reward. If the action was successful, 100 was added to the previous reward, otherwise 100 was deducted from the previous reward. The goal is to maximize the cumulative rewards in the long run. As the look-up table is improved over time, we must also decide when further training would not be helpful. I used a fixed value of 10,000 iterations. To choose this number, I tested some arbitrary numbers of iterations such as 500, 1000, 5000, and 10000, on one category of push (front 40 CM). The results for 10000 iterations was better than others. I chose the front direction to test, because based on my preliminary tests, recovery from the front side is harder than other sides. Also, I chose the 40 CM distance, as an average external force.

As was discussed earlier (Section 3.4.2), the robot's environment is discretized into 14 discrete states (Figure. 3.25). Each of these discrete states was explained in Section 3.4.2. By employing RL, Polaris learned how to recover from various pushes $\{30, 40, 50\}_{cm}$ from different directions {left, right, front, back}. The final output

of each model is a set of actions (walking engine parameters). Each learned action corresponds to a discrete state, and Polaris selected the corresponding action when it found itself in that discrete state.

In my approach to recover from every push, the robot takes one right and one left step. The first step must be able to absorb the majority of the push, since it is an initial step. If a push comes from the front or back sides, the first step will be the right foot. Also, if the push is applied from the right side, the robot takes the right foot as the first step. The only time that the robot takes the left step as the first step is if it is pushed from the left side. After the first step is learned, the learning process for the second step will begin.

The first step is considered as learned step, only if by taking it, the robot will be in the stable state in the simulation environment. However, a learned step might not be as good in the real world environment for various reasons. In the simulation environment, many limitations are not considered, such as over torquing servo motors or voltage oscillation. For example the robot might take one right step after the push is applied and it recovers from the impact. However, there might be lots of weight (pressure) on a single servo (that is, the right ankle servo). Even though this might not affect the robot in the simulation, it will over torque the right ankle of Polaris in the real world, which will cause Polaris to fall. To overcome this possible issue, the second step will be used to reduce the pressure on the servos. Learning of the second step is started upon completion of the learning of the first step.

In the learning process, for every attempt (push), only one parameter will be altered at a time. This is similar to the hand-tuned approach to push recovery. Many

times (e.g. eight out of nine times), walking engine parameters have generally good values, and only one of them is not good. This might lead the robot to a falling state. Let's assume a push has been applied to the back of the robot and the bottle is released from a 30 CM distance. The generated set of action for this push is: {step-x: 2, step-y: 0, step-height: 3, x-offset: -1, y-offset: 4, z-offset: 38, step-pace: 300000, hip-pitch: 14, ankle-pitch: 0}. All the generated parameters except the hip-pitch, are considered good values. When using a hip-pitch of 14, the robot leans too far forward, and as the result it will fall (facing the ground). Since the applied force will push the robot to the front, one of the reasonable strategies is to shift the body mass of the robot to the back (lean backward). In order to do this, the hip-pitch value must be reduced (for example, to 4). Changing one value at a time and comparing the performance based on the new generated value, would eventually lead to a successful recovery.

I implemented each model (for example, direction: front, distance: 40 CM) as a look-up table and constructed it as a multi-dimensional numpy array [Walt et al., 2011]. In the look-up table, all the rows refer to the discrete states and all the columns refer to the discrete actions . After this step, I implemented Q-learning as a function which receives the look-up table as its parameter and updates the table's value(s) in every iteration. In every iteration, the Q-learning function selects an action from the 9 action categories, with a value between the minimum and maximum of that action category (Table. 3.4). Depending on the learning rate, the strategy (behaviour) of the robot will be different. If the learning rate is high (maximum 0.99), the robot would be interested more to the current reward as opposed to the long term future

reward (cumulative reward). Generally, if the rate is high the outcome is not as good as having the rate at a lower value.

Polaris learned different combinations of walking parameters that it used in push recovery to replace the hand-tuned entries. The entries were the possible actions of the robot (discussed in Section 3.4.2). Figure 3.40 shows the system diagram of my adaptive closed-loop control using the RL for one step of a stride. In this diagram a disturbance represents an external push that can be added to the closed-loop control at anytime (before or after a set of instructions send to the motors) during a walking step. The IMU measures a potential disturbance signal along with some added ambience noise and send it to the controller module. After that the controller measures the strength and the direction of the disturbance. Then using the look-up table that was generated by RL module it matches a category that is appropriate for the push and send an updated set of walking parameters to the motors.

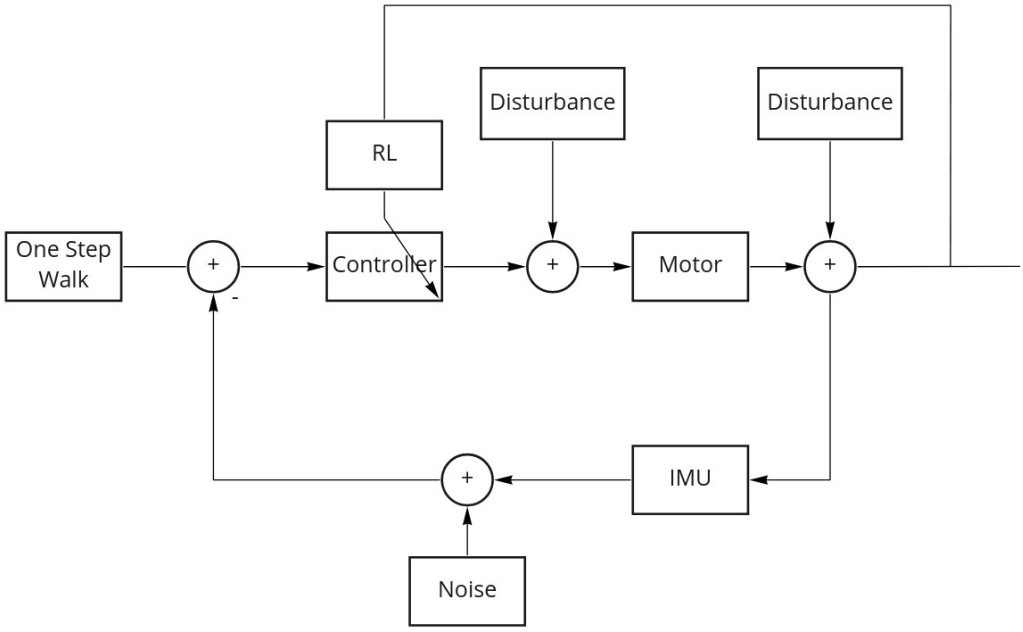


Figure 3.40: System Diagram of my adaptive closed-loop control using the RL.

3.4.5.3 Deep Reinforcement Learning (DRL)

In Section 3.4.5.2, I explained the transition of my hand-tuned approach [Hosseinmemar et al., 2018] to a fully autonomous approach that used RL to generate walking step trajectories. However, there is room for further improvement. The choice of reward function in my CM implementation was based on the simulations' assessment. The main issue with the reward function for this approach is that Polaris can only distinguish between the best and worst actions. Best actions lead the robot to the stable state. Worst actions are actions that lead the robot to a fall state. However, some action choices for which the robot is penalized do actually have the potential to lead the robot to a stable state, but a fall results in negative rewards and the robot then avoids attempting those actions in the future. As a result, many possible

step trajectories will not likely be learned. If instead, I employ Deep Reinforcement Learning (DRL), the robot should have the ability to learn the reward value in a different way, which may be an improvement over both human design and the RL approach based on the simulation.

Deep learning is a machine learning technique that enables computers to learn features or tasks from a set of data (dataset). Since the gained knowledge is based on the computer's experience, there is no need for a human operator to specify all the required knowledge needed by the computer [Goodfellow et al., 2016]. DRL is a combination of deep learning and reinforcement learning. In DRL, an approach called Function Approximation (FA) can be used to avoid manual discretization of a continuous state space [Boyan and Moore, 1995]. FA is a Machine Learning (ML) approach in which the process of approximation is the process of learning. This is well suited for estimating a value in an unknown continuous state space. If $D = \{d_1, d_2, \ldots, d_n\}$, where D denotes a dataset, and $d_1 \in D$, there will be a $d_{n+1} \notin D$ [Šalát, 1980; Funahashi, 1989]. This means a function in a continuous space can have some properties which cannot be found in the dataset and these must be counted as unknown properties or unknown states (there is always a number which is not in the dataset and it is unknown for the dataset). FA helps to overcome this type of problem by trying to approximate functions in unknown states. In other words, FA finds the underlying relationship between a given set of inputs and outputs (dataset). Depending on the type of the problem, a continuous function can be approximated using different approaches. FA methods are commonly categorized into two [Hardle and Mammen, 1993]: Parametric Regression and Non-Parametric Regression. Linear

Models with Ordinary Least Squares [Dismuke and Lindrooth, 2006], and Online Approximation with Recursive Least Squares [Leung et al., 1996] are examples of Parametric Regression. On the other hand, Interpolation and Extrapolation [Chow and Lin, 1971], Gaussian Process Regression [Quiñonero-Candela and Rasmussen, 2005], and Artificial Neural Networks [Ferrari and Stengel, 2005] are examples of Nonparametric Regression. Usually, it is not possible to map a continuous function's exact parameters to the sample dataset $\mathbb{A}$. The main reason for this is if $x \in \mathbb{R}$, where x is a parameter in a continuous state space and belongs to a continuous function, and $\mathbb{R}$ represents real numbers, there should be an infinite number of members in $\mathbb{A}$, which means $\mathbb{R} \subseteq \mathbb{A}$, which is impossible. The main goal of FA is to minimize the error that the model produces when it is predicting the output (discovering an unknown state). There are many different approaches to measure this error, but the common ones are: Mean Absolute Error, Mean Squared Error, and Root Mean Squared Error [Chai and Draxler, 2014].

The most common function approximator in Artificial Intelligence (AI) is an Artificial Neural Network (ANN) [Ferrari and Stengel, 2005]. There are many types of ANN to choose from in terms of their structure and usage: feed-forward [Bebis and Georgiopoulos, 1994], recurrent [Williams and Zipser, 1989], modular [Kimoto et al., 1990], as well as radial basis functions [Park and Sandberg, 1991]. I employed a feed-forward ANN in my deep learning mechanism. In feed-forward ANNs, there are no loops and the information on the network moves directly from the input layer through a hidden layer to the output layer.

To apply this to deep learning, we introduce more than one hidden layer, producing

a Deep Neural Network (DNN) [Hinton et al., 2012]. I designed my DNN structure with one input layer, 20 hidden layers, and one output layer. The input layer takes the 9 robot walking engine parameters (Section 3.4.2), and learns the reward based on the actions taken by the robot.

Training and testing the DNN involves using both the RL and DRL components running in tandem. I reserved the final output of the previously-trained RL component (i.e. the finished policy using only RL), and then reset the RL to start fresh. The RL component begins training as previously. However, instead of using the simulation to assess an action, it uses the currently learned reward function from the DNN. I have divided the whole DNN process into three steps.

First, I need a rich dataset for training the DNN, comprised of an even mix of good and bad action choices. The logical place to start with this would be the policy that results from the previous RL training, i.e. the resulting look-up table.

Second, I need to use this dataset to train the DNN. For this I used supervised learning techniques [Cunningham et al., 2008]. The network needs to learn a model based on the sample data (my sample data consisted of 24000 data points). Every data point in the dataset will have a label, which is the reward for that particular action. For example, if $d_1 \in D$ is the first data point, it will have a corresponding label that is $l_1 \in L$. As was discussed in Section 3.4.5.2, a reward can be either +100 or -100. I used this information to generate the corresponding labels. Based on this, the labels 1 or 0 were assigned to each data point, where 1 represents 100 and 0 represents -100. The DNN predicts if an action will lead the robot to a fall state or the stable state. I used a sigmoid activation function (Equation 3.37) to keep the

output of the DNN between 0 and 1.

$$S(x) = \frac{1}{1 + e^{-x}} \qquad \text{[Han and Moraga, 1995]} \tag{3.37}$$

In Equation 3.37, x denotes the output of the DNN and e is Euler's number ≈ 2.718. The activation function will be placed after the output layer to apply the expected adjustment to the output. Using this formula, the DNN's output is a probability, a real number between 0 and 1. I formulated my DNN as a binary classification problem [LeCun et al., 2015], and for that reason I needed to measure the cost with a classification criterion for the loss function. I used the binary-cross-entropy loss function [Buja et al., 2005] to measure the loss during training. Equation 3.38 shows the binary-cross-entropy function. In this equation, l_i corresponds to the label for d_i in the dataset with the value of 0 or 1; p denotes the probability that the network predicted and it is a real number between 0 and 1.

$$-(l_i * log(p) + (1 - l_i) * log(1 - p)) \quad \text{[Buja et al., 2005]} \tag{3.38}$$

The last step is to test my DNN with unseen inputs. To test this, I used the DNN to generate reward values, and show that the RL component can be trained with this reward function and possibly perform better in push recovery. To do this, I reset the RL component to an untrained state, and the training process will ensue in the same way as before, except that the chosen action will be sent to the DNN to generate a reward value rather than using the prior reward function (using simulation). If the output p (the probability of the action being successful) of the DNN (the output of the network) is $0 \geq \text{p} < 0.9$, the action will be classified as a fall and a -100 reward will be generated. However, if the output is $0.9 \geq \text{p} \leq 1.0$, the action will be classified

as stable and +100 will be returned as the reward value.

The resulting RL component should allow better responses to push recovery, since the DNN is exploring a larger space than was possible with the original reward function. This also means that we should be able to repeat this process, and use the newly-trained RL component to generate a larger dataset for training the DNN, and possibly improve performance further. However, this is a very time consuming process and I did not have enough time to examine it. It would be an interesting avenue of future work.

Figure 3.41 shows the structure of my DNN showing 5 of the 20 hidden layers. In this figure, the green layer is the input layer of the network and the parameters {Input #1,..., Input #9} refer to the robot actions that were listed in Table 3.4. I implemented the network with 20 hidden layers with 18 neurons in each layer. The number of the neurons in the hidden layers are usually related to the quantity of inputs to the network and are often double the number of inputs. However, this is not a universal rule, and requires some tests and modifications to parameters such as the number of neurons in the hidden layer. For choosing the number of hidden layers, I examined the performance of the network with varying numbers of hidden layers {2, 9, 15, 20} and neurons on each layer {9, 18}. For example, I checked the performance of the network with 15 hidden layers, once with 9 neurons on each layer and with 18 neurons on each hidden layer. The last layer is the output layer, which produces a floating point number between [0,1] which is a learned reward for an action, representing the probability that the action will result in a stable state.

Selecting a correct set of parameters for building, training and testing a DNN is a time consuming process. The network needs to be tested and evaluated with different sets of parameters to check the accuracy and performance of the network. For example, there are parameters such as *learning rate* (how fast the network should learn), *batch size* (the number of training samples in which will be given to the network during training), *epoch* (one forward pass and one backward pass for all the training sample data), and *number of hidden layers*, which play an important role in every DNN. After testing the performance of the network with many different combinations, I chose the learning rate of 0.1, batch size of 20000, with the epoch of 10000 and 20 hidden layers.

Figure 3.42 shows the system diagram of my adaptive closed-loop control using the RL for one step of a stride. In this diagram a disturbance represents an external push that can be added to the closed-loop control at anytime (before or after a set of instructions send to the motors) during a walking step. The IMU measures a potential disturbance signal along with some added ambience noise and send it to the controller module. After that the controller measures the strength and the direction of the disturbance. Then using the look-up table that was generated by RL module it matches a category that is appropriate for the push and send an updated set of walking parameters to the motors.

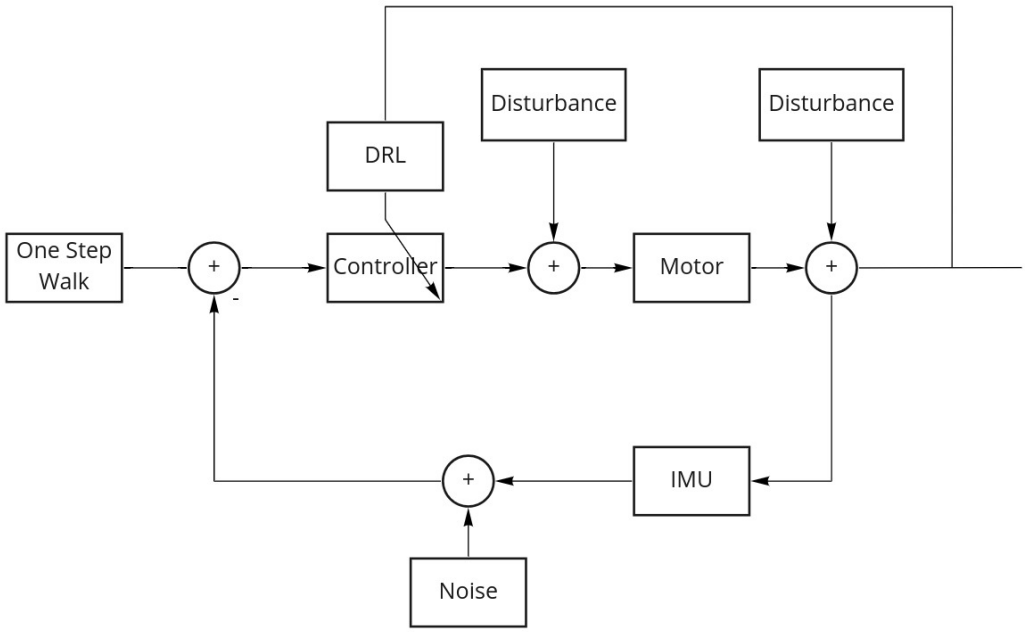


Figure 3.42: System Diagram of my adaptive closed-loop control using the DRL.

3.5 Summary

In this chapter I described my novel approach to humanoid robot push recovery using RL and DRL. In Section 3.4.5.2 I explained how RL module uses the simulation software for rewarding robot's actions, and stores the actions and the rewards in a look-up table. In Section 3.4.5.3 I discussed the architecture of my DNN and how this network replaces the simulation software for rewarding the robot's actions. Similar to Section 3.4.5.2 I used a look-up table to store the actions and rewards. Also, this chapter has described my push recovery implementation in order to support an evaluation of my approach. The following chapter describes the experiments I performed in both real world and simulation, to evaluate my approach against baseline

conditions.

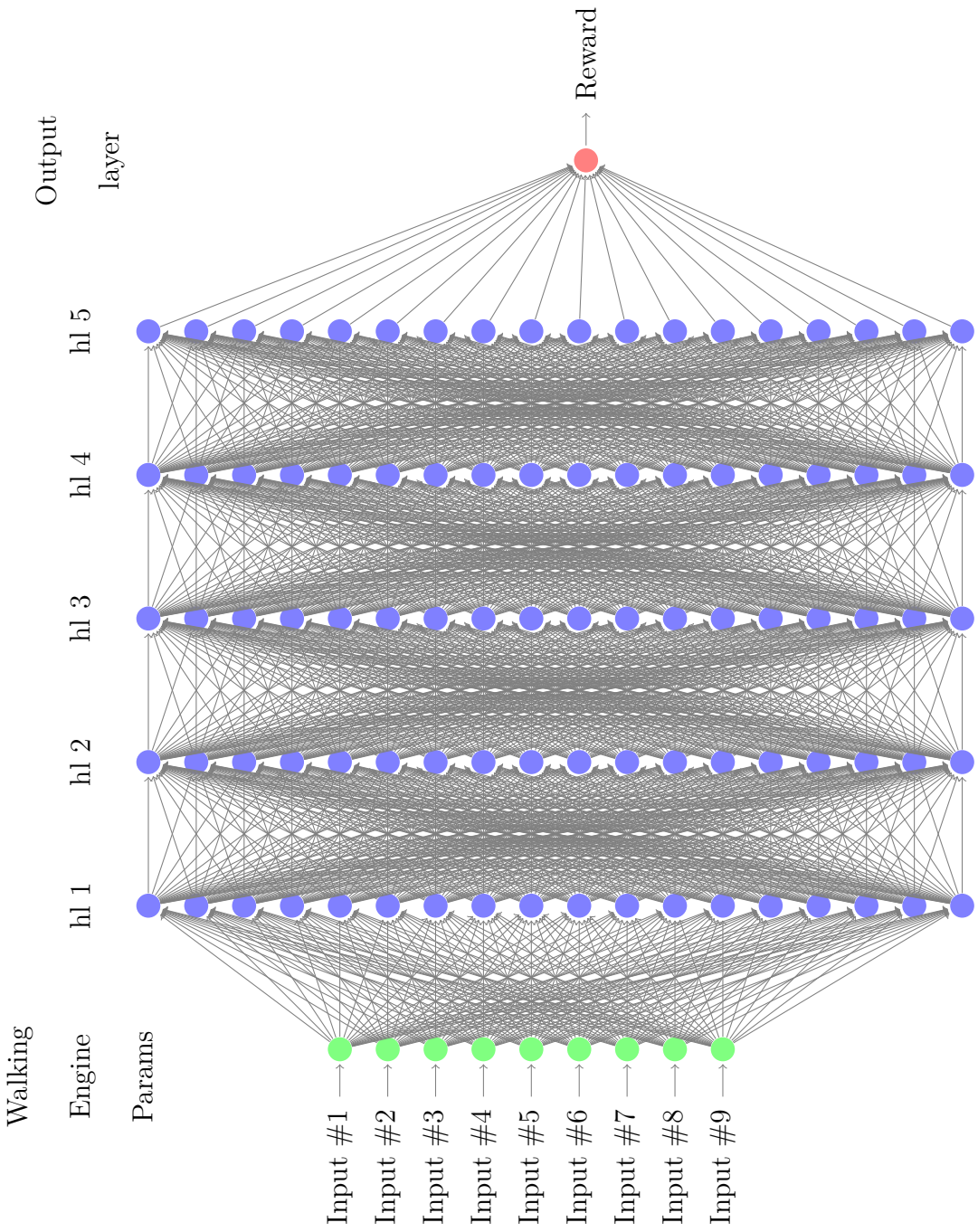


Figure 3.41: Partial view (instead of 20 hidden layers, only 5 are shown) of the architecture of the Deep Neural Network

Chapter 4

Evaluation

4.1 Introduction

In order to examine the effectiveness of my approach (Chapter 3), I ran various experiments in simulation and in the real world. I compared the performance of my approach against two baseline cases. In the first base case, the robot was not able to use any sensor feedback (IMU) during the experiment (open-loop control feedback). In the second base case, the robot took random actions within the given thresholds of walking engine parameters and was able to use the IMU (closed-loop control feedback). During the experiments, the robot did not know anything about the environment (different surfaces). The robot was examined on concrete, carpet, and artificial turf in the real world. In the simulation, it was examined on simulated concrete and turf. This chapter begins with the review of my research questions (Section 1.3). I chose to not to modify any parameters during the experiments. This included the type of the ground that the robot was examined on during the

experiments. The type of surface, and the directions and strength of the pushes were all chosen randomly.

4.2 Review of Research Questions

Recall my research questions from Section 1.3:

1. Is my closed-loop control going to be fast enough for inexpensive robots that are equipped with cheaper hardware for responding to an external push?

2. Are any of my reinforcement learning and deep reinforcement learning push recovery techniques going to be able to replace the parameters that a human operator provides (hand-tuned) for recovery based on his/her experiences?

3. Which of the proposed approaches (RL, DRL), results in better recovery if the surfaces, directions, and impacts are unknown to the robot (i.e. with no prior knowledge)?

4.3 Evaluation Criteria

I recorded one value throughout each trial: whether the robot recovered or did not recover. A push recovery is considered successful if the robot did not fall after it was hit (while walking) and took its desired recovery action.

I invalidated a trial in two conditions:

1. If the IMU stopped working before the impact time. This happened a few times and the cause was the USB cable.

2. If a servo motor stopped working (e.g. gear worn out)

If any of the above conditions were met, I fixed the issue and repeated the trial.

4.4 Experimental Environment

In this section I describe the experimental environments in two worlds. The first world is in simulation and second is the real world. Every world has its own environments and the robot was examined in those environments.

To evaluate my approach quantitatively, I set up various experimental environments to control the push force applied to the robot, based on the international robotics competition, RoboCup 2018 push recovery technical challenge [Robocup, 2018]. This scenario is illustrated abstractly in Figure 4.1, showing the robot in the double support phase of a walking gait (i.e., both feet on the ground). The bottle represents a 2 Kg mass (more than a quarter of Polaris' body weight) that is connected to a 1.5 M rope . The bottle is pulled back to a given distance and released naturally, resulting in the mass hitting the robot at torso height, introducing a fixed external force that can be varied depending on the distance at which the mass is released.

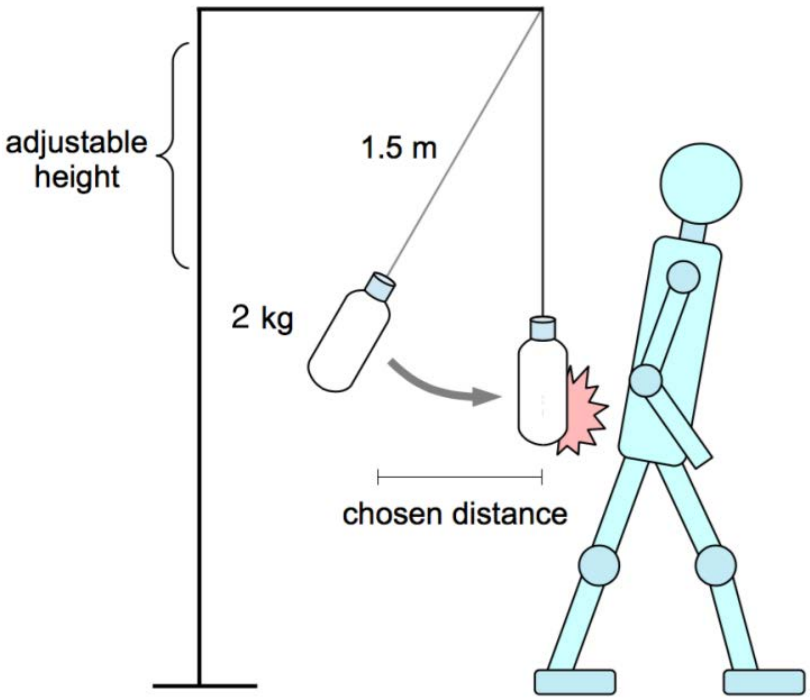


Figure 4.1: Set up for push recovery challenge [Robocup, 2018]

In all of the trials that took place in real world, I followed the exact same setup. For all the trials in simulation, I calculated the final force based on different distances and applied it to the robot.

4.4.1 Simulation

In the simulation I generated two environments. The only differences between the two environments are the surface the robot was walking on and the strength and direction of the pushes applied. In the first environment, the surface is simulated concrete, and the second environment is a simulated soccer field with artificial turf.

Every experiment consisted of 360 trials. For all 4 sides (back, right, front, left) and 3 distances $(30,\ 40,\ 50)_{cm}$, 30 pushes were applied. For every environment, I

examined my RL and DRL approaches. In total, 2880 pushes were applied to the robot in the simulation for all the experiments. Figure 4.2 shows the experimental design in the simulated world.

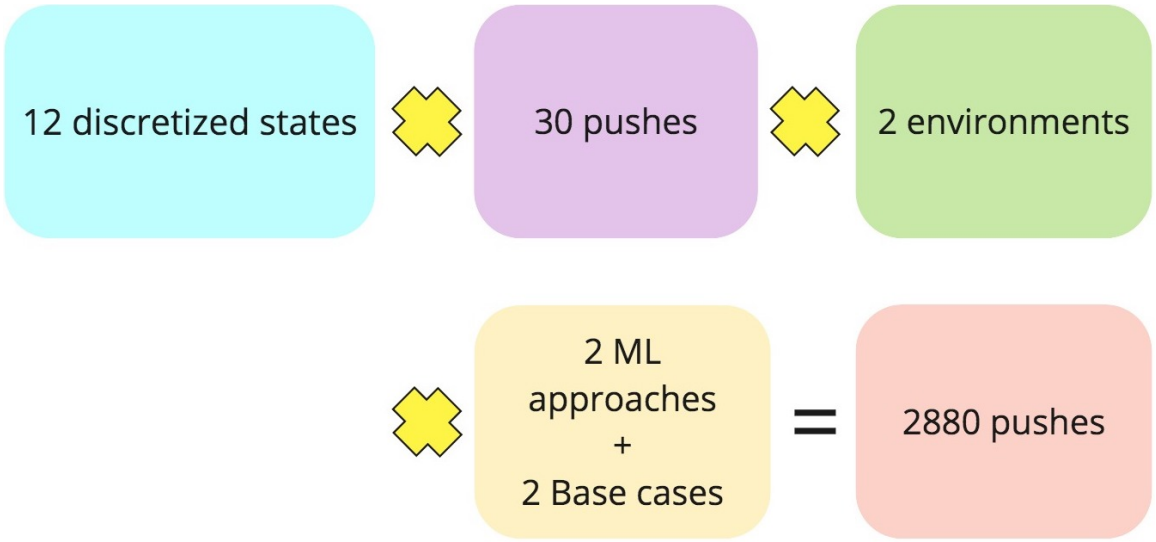


Figure 4.2: Experimental design in simulation

4.4.2 Real World

In the real world I designed three environments. As in the simulation, the only differences between the environments are the surface on which the robot was walking and the strength and direction of the pushes applied. In the first environment, the surface is concrete, while the second is a low carpet, and the last is an artificial soccer turf. Figure 4.3 shows these three surfaces opon which the robot walked during the experiments in the real world.

Figure 4.3: Three different surfaces for three different real world environments. From left to right: concrete, carpet, and artificial soccer turf.

Polaris was tested on the concrete floor in the experiments in Sections 4.7.1.1, 4.7.1.2, and 4.7.1.3; It was tested on the carpet in the experiments in Sections 4.7.2.1 and 4.7.2.2, and 4.7.2.3. Finally, Polaris was tested on the artificial turf in the experiments in Sections 4.7.3.1 and 4.7.3.2, and 4.7.2.3.

Figure 4.4 shows the setup that I employed during the three experiments to apply pushes to the robot. This particular instance is set up to test on the concrete floor. I used red tape to mark Polaris's feet on the floor to make sure that for every trial, Polaris is standing on the same location.

Figure 4.4: Experimental setup in the real world on the concrete surface for both RL and DRL.

Figure 4.5 shows the same setup, with the carpet floor in place. Tape is similarly used to ensure the robot is placed consistently.

Figure 4.5: Experimental setup in the real world on the carpet surface for both RL and DRL.

Figure 4.6 shows the same setup, with the artificial turf floor in place. Tape is once again used to ensure the robot is placed consistently.

Figure 4.6: Experimental setup in the real world on the artificial turf surface for both RL and DRL.

I taped two filled, one-litre water bottles together, forming the 2 Kg weight mass used to strike the robot and produce a push. Then I connected the massless rod (I used ethernet cable) to the midpoint of the two water bottles. The other end of the rod was connected to a frictionless pivot. Figure 4.7 shows the pendulum setup that I used for all of my experiments.

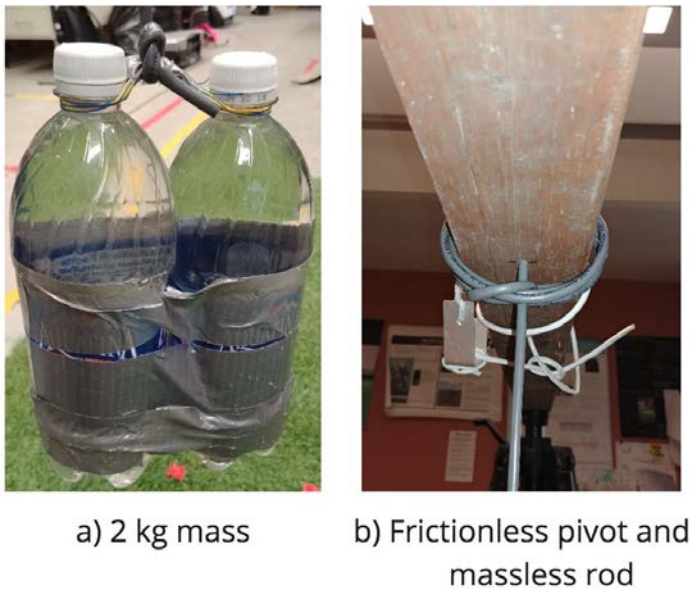


Figure 4.7: The 2 Kg mass connected to a massless rod.

Every experiment contained 36 trials. For each of the 4 sides (back, right, front, left) and 3 distances $(30, 40, 50)_{cm}$, 3 pushes were applied. For every environment, I examined my RL and DRL approaches and compared these to a base case of the open-loop feedback control. I did not want to risk damaging any of the servo motors by taking a random action. In total, 324 pushes were applied to the real robot in the real world for all the three environments. Figure 4.8 shows the experimental design in the real world.

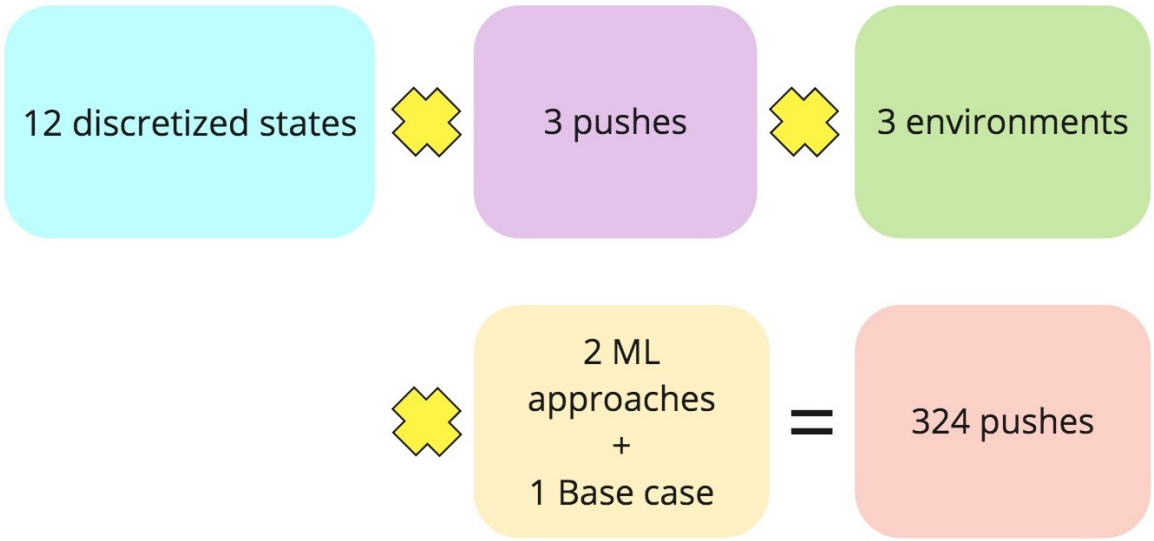


Figure 4.8: Experimental design in the real world

4.5 Choosing Environments

In the real world, I designed a total of three environments, Figures 4.4, 4.5, and 4.6, identical except for the type of walking surface. I examined my push recovery with two machine learning (RL, DRL) approaches and one base case that uses the open-loop feedback control. This means each of the following: RL, DRL, and the base case, had to be examined 36 times for all the directions and distances (Figure 4.8).

I implemented a function that randomly generated two values for me: 1) which approach to take (e.g. RL, DRL, or base case)? 2) a list of surfaces based on priority (e.g concrete, artificial turf, carpet). The main reason that I managed the environments this way was to avoid having results affected by factors such as servo gear wear, which would bias environments always chosen first against those always

chosen last. For example, if this function returns {DRL,(concrete, artificial turf, carpet)} , it means, the approach to be examined is DRL, and this approach should be examined first on the concrete surface, second on artificial turf, and last on the carpet. Also, instead of finishing the 36 pushes in one round in one environment, I tested 18 pushes in one environment and moved to the next one. After all the environments were tested 18 times, I went back to the first environment and examined the 18 pushes that were left for all the consecutive environments. For example, if the approach is DRL, as discussed above, 18 pushes were applied on concrete, then 18 pushes on turf, and 18 pushes on carpet. Then, again, 18 pushes on concrete, 18 pushes on turf and finally, 18 pushes on carpet. So in two rounds the total number of 36 pushes were applied to the robot in each environment.

4.6 Choosing Push Directions and Distances

Similar to Section 4.5, I generated all the push directions and distances randomly. This is again to avoid the bias that would result if one environment was always chosen first and another chosen last. As shown in Figure 4.8, I examined every direction and distance three times, resulting in a total of 36 pushes. I implemented another function that assigned all of these 36 pushes randomly. For example, there are 3 pushes assigned to the left direction with the distance of 30 CM. Since the pushes are randomly assigned, there is a possibility of having the first one appears to be the 5^{th} push, and the second one 6^{th} push, and the third left 30 CM push to be the 28^{th} push out of 36 pushes.

4.7 Experimental Results

This section describes the experimental results of my approach in both the real world and in simulation. The results are grouped based the surface and environment in which the robot was examined during the experiments.

4.7.1 Environment: Real World - Concrete Surface

This section describes the results I obtained from my experiments on the concrete surface. This involves the comparison of the results between reinforcement learning, deep reinforcement learning, and the base case. In the following subsections, first, the results of RL are discussed. Second, I discuss the results obtained from testing my deep reinforcement learning approach. Third, I discuss the results of the base case. Finally, I describe the performance of my approach versus the base case.

Figure 4.9 shows a successful recovery of a strong push (50 CM) from the left side of Polaris on the concrete surface. This force lifted half of Polaris's body. However, Polaris could recover without falling.

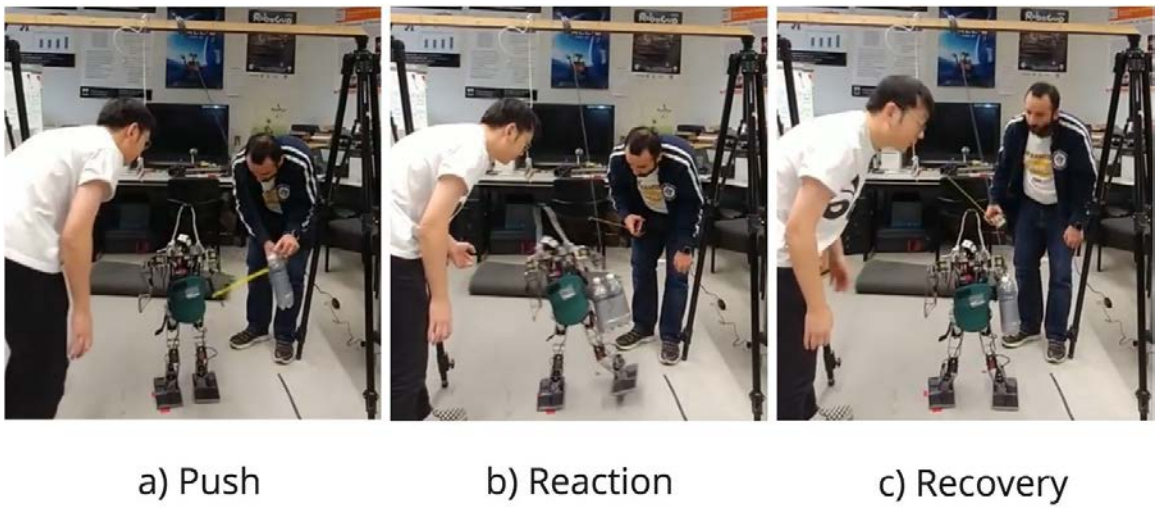


Figure 4.9: Push, reaction, and recovery on a concrete surface

4.7.1.1 Approach: Reinforcement Learning

In this section I discuss the results of 36 pushes (12 * 3) on the concrete surface using my reinforcement leaning approach. All the actions were the results of the same reward function used in the simulator.

The results of all trials related to this experiment are shown in Table 4.1. For each case in which the robot successfully recovered from the impact, a score of 1 was given, otherwise no score was granted. Figure 4.10 shows the summarized results of my approach using RL. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The blue, orange, green, and purple columns represent 30 CM pushes, 40 CM pushes, 50 CM pushes, and total success rate by each direction in percentage, respectively.

Concrete surface: RL		Successful attempts			Total number of successful attempts by directions
Distances in centimeter		30	40	50	Out of 9
Directions	Back	2	3	2	7
	Right	3	3	3	9
	Front	3	2	0	5
	Left	3	3	2	8

Table 4.1: Results of trials for the experiment in the real world on concrete using RL.

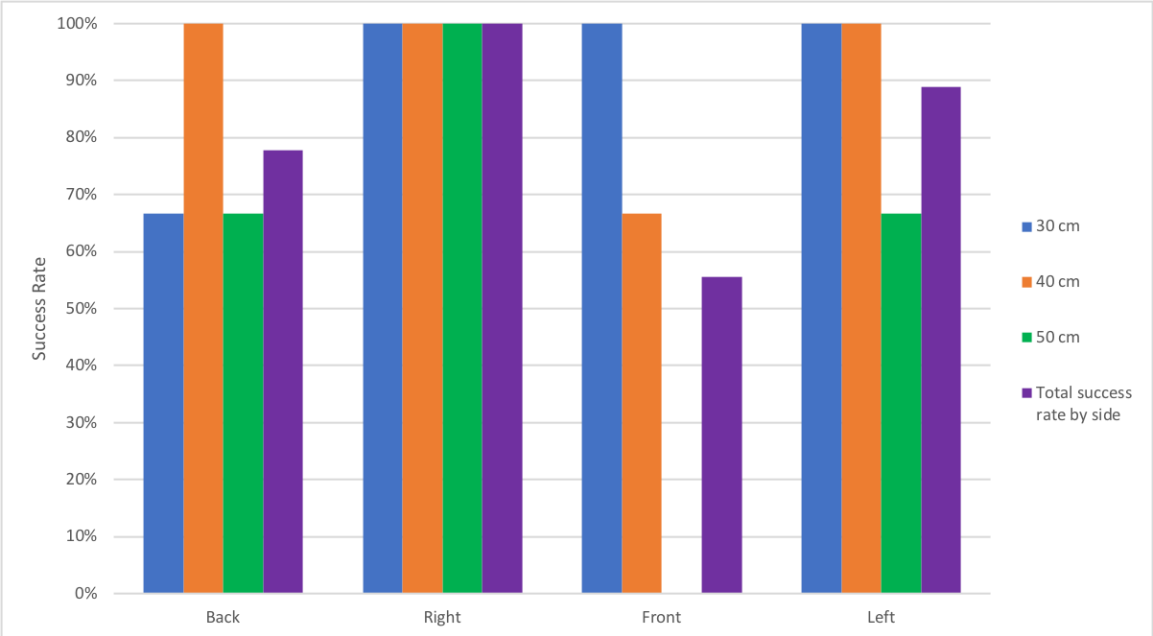


Figure 4.10: Experimental design in the real world

My closed-loop control mechanism using the RL approach was able to recover from a strong majority of pushes, with an average success rate of 92% for all of the directions at the low impact level (30 CM swing). Polaris was able to recover from 100% of pushes that were received from the Right, Front, and Left directions. Also, it could recover from 67% of the pushes from the Back.

At the medium impact level (40 CM swing), the recovery average results were similar to 30 CM (92%). All the pushes from back, right, and left were recovered successfully. 67% of the pushes that were applied to the front were recovered successfully.

At the strong impact level (50 CM swing), there is a more distinct difference between the four directions. The average success rate of all directions is 58%, with 100% of the pushes from the right side, and 67% of the pushes from Back and Left sides recovered successfully. Polaris was not able to recover from any of the pushes that were applied to the Front side. Based on my observation, apart from a possibility of having weaker motors on one side there is one likely reason that Polaris could not recover as well from the pushes that were applied to the Front side. Polaris uses two batteries to operate: a 3-cell battery to run the computer and a 4-cell battery for the motors. Unfortunately, the design of Polaris did not include a place assigned for mounting the batteries. For that reason, I mounted both batteries in the back side of its torso. I tried mounting them in different locations on the body frame, however, the best place was the back. The weight of the two batteries together is approximately 700 grams. This weight is focused (not evenly distributed) in the back and if the applied force to the front is very strong, Polaris does not have enough time to react before it falls.

The average success rate of recovery, based on the impact level, using the RL approach is shown in Figure 4.11.

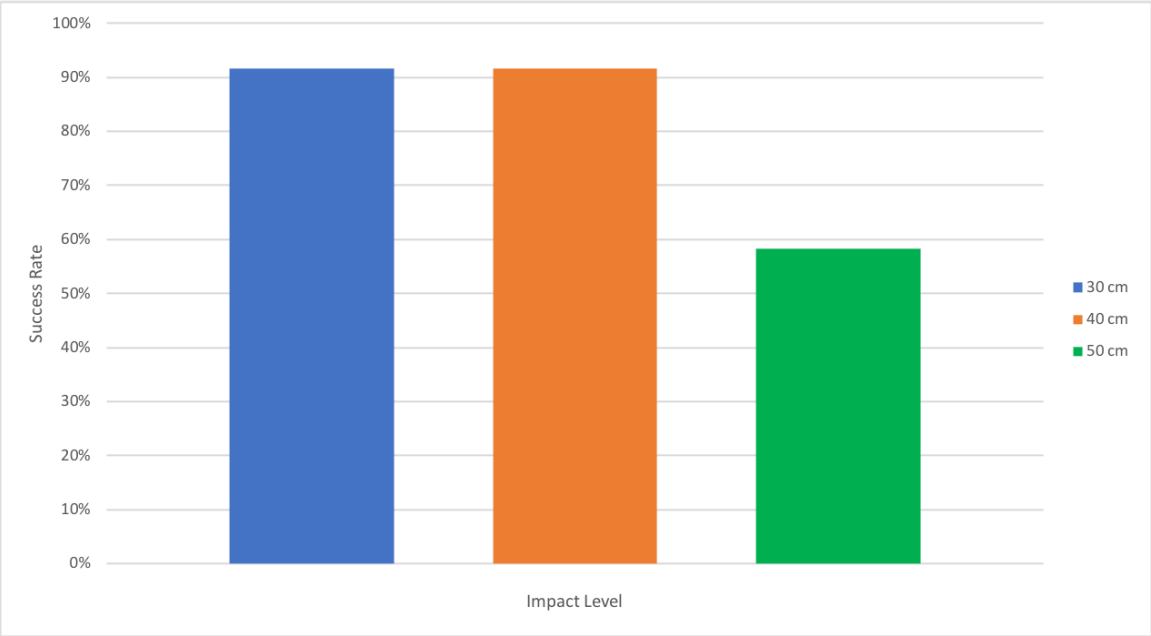


Figure 4.11: Average success rate of recovery based on the three impact levels using RL on the concrete floor.

4.7.1.2 Approach: Deep Reinforcement Learning

In this section, I discuss the results of 36 pushes (12 * 3) on the concrete surface using my deep learning approach. All the actions were the results of using my deep neural network as the reward function.

The results of all trials related to this experiment are shown in Table 4.2. For each case in which the robot successfully recovered from the impact, a score of 1 was given, otherwise no score was granted . Figure 4.12 shows the summarized results of my approach using DRL. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The blue, orange, green, and purple columns represent 30 CM pushes, 40 CM pushes, 50

CM pushes, and total success rate by each direction in percentage form, respectively.

Concrete surface: DRL		Successful attempts			Total number of successful attempts by directions
Distances in centimeter		30	40	50	Out of 9
Directions	Back	3	3	2	8
	Right	3	3	3	9
	Front	3	3	1	7
	Left	2	3	2	7

Table 4.2: Results of trials for the experiment in the real world on concrete using DRL.

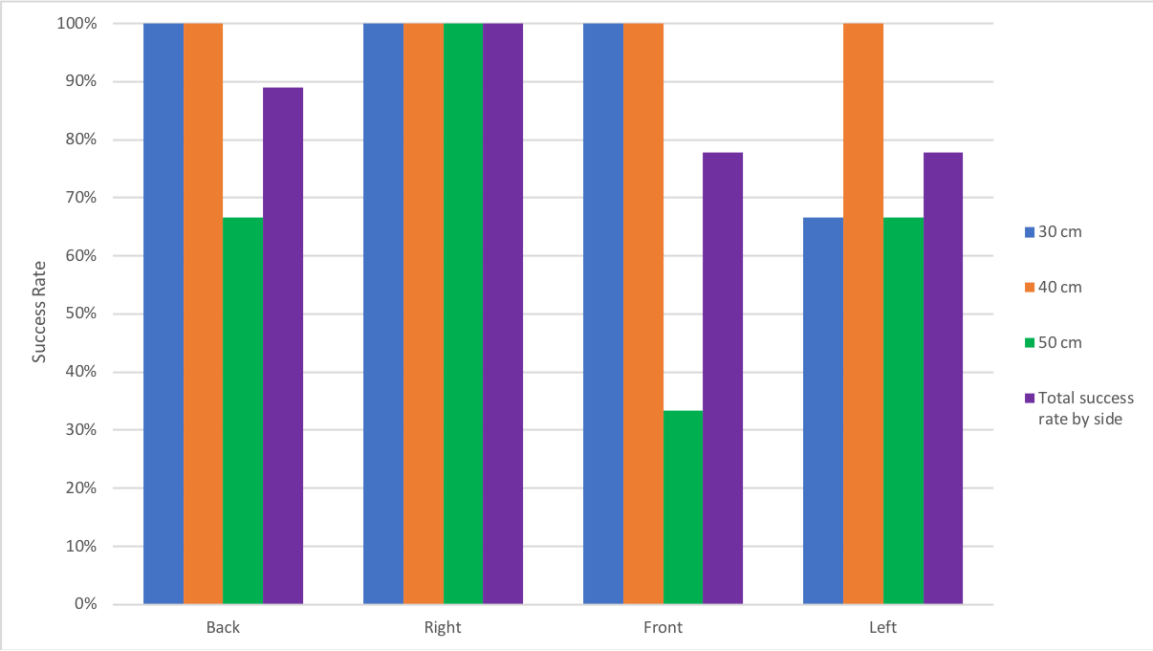


Figure 4.12: Experimental design in the real world using deep reinforcement approach on concrete

Similar to the RL (Section 4.7.1.1), my closed-loop control mechanism using the DRL approach was able to recover from a strong majority of pushes, with an average

success rate of 92% for all of the directions at the low impact level (30 CM swing). Polaris was able to recover 100% of pushes that were received from the Back, Right, and Front directions. Also, it could recover from 67% of the pushes from the Left.

At the medium impact level (40 CM swing), the recovery average results were outstanding, with 100% recovery for all the pushes from all the directions.

At the strong impact level (50 CM swing), the average success rate of all directions is 67%: 100% of the pushes from the Right side, and 67% of the pushes from the Back and Left sides were recovered successfully. Polaris was also able to recover from 33% of the pushes that were applied to the front side. As was discussed in Section 4.7.1.1, since the weight of batteries (approximately 700 grams) is concentrated in the Back side, recovering from the pushes that are applied to the Front side is harder than the other sides. Also, there is a possibility of having weaker motors on one leg (the right leg) of the robot. Since the cost of each motor is more than C $650 and each leg uses 6 motors, it was not possible for me to change 12 servos at once. Because of this there is a possibility of having weaker motors in a leg that led the robot having a weaker performance on a side or two sides.

The average success rate of recovery, based on the impact level, using the DRL approach is shown in Figure 4.13

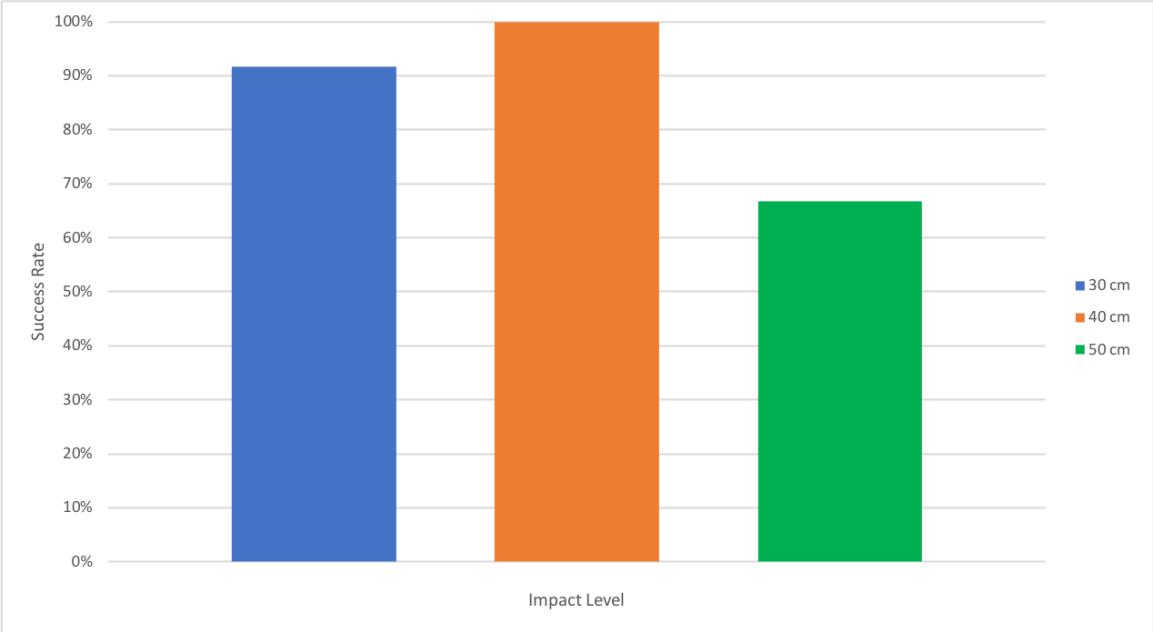


Figure 4.13: Average success rate of recovery based on the three impact levels using DRL on the concrete surface.

4.7.1.3 Approach: Base Case

In this section I discuss the results of 36 pushes (12 * 3) on the concrete surface using the base case. As was stated earlier, as a base case Polaris used its open-loop control feedback approach, which does not use any sensors. As a result, no specific actions were taken apart from a constant walk which was not related to any of the pushes.

The results of all trials related to this experiment are shown in Table 4.3. For each case in which the robot successfully recovered from the impact, a score of 1 was given, otherwise no score was granted.

Figure 4.14 shows the summarized results of the base case approach using the

open-loop control feedback. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The blue, orange, green, and purple columns demonstrate 30 CM pushes, 40 CM pushes, 50 CM pushes, and total success rate by each direction in percentage form, respectively.

Concrete surface: Base case		Successful attempts			Total number of successful attempts by directions
Distances in centimeter		30	40	50	Out of 9
Directions	Back	3	1	0	4
	Right	3	3	3	9
	Front	2	1	0	3
	Left	3	0	0	3

Table 4.3: Results of trials for the experiment in the real world on concrete using open-loop control as a base case.

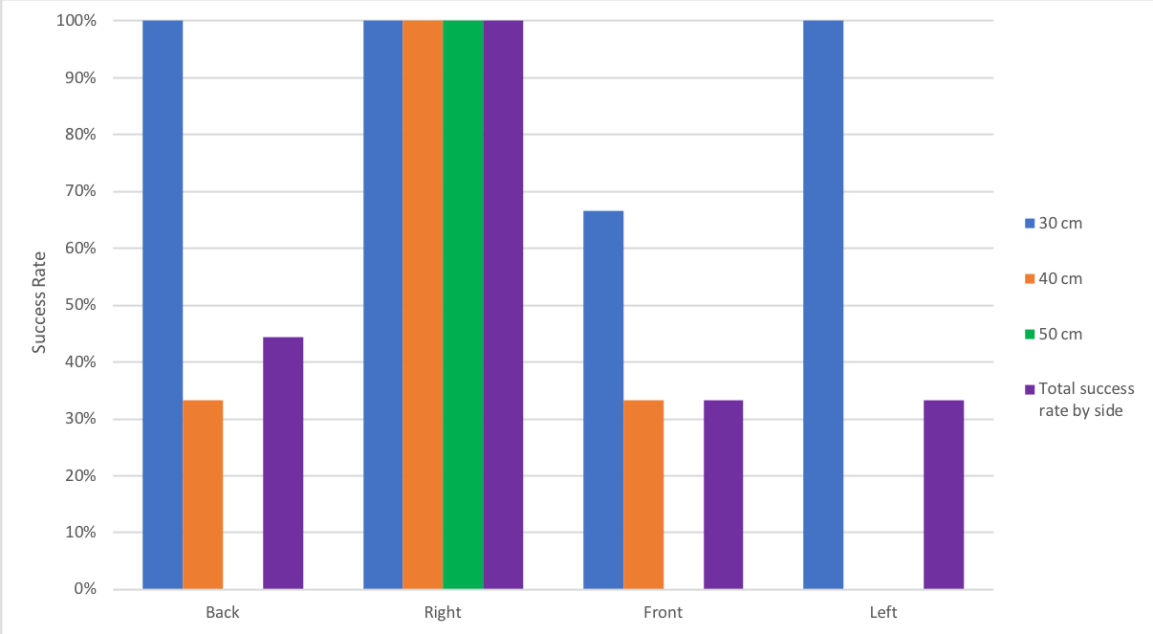


Figure 4.14: Experimental design in the real world using open-loop control feedback (IMU was disabled)

Similar to the RL and DRL (Sections 4.7.1.1, and 4.7.1.2) approaches, the base case was able to recover from the majority of pushes with the average success rate of 92% for all of the directions at the low impact level (30 CM swing). Polaris was able to recover 100% of pushes that was received from Back, Right, and Left directions. Also, it could recover from 67% of the pushes from front.

At the medium impact level (40 CM swing), the recovery average results decreased, with 42% recovery for all the pushes from all the directions. Polaris was only able to recover from back, right, and front with 33%, 100%, and 33%, respectively. Polaris was not able to recover from any of the pushes that were applied to the left side.

At the strong impact level (50 CM swing), the average success rate of all directions is 25%. 100% of the pushes from the right side, and 0% of the pushes from all the other sides were recovered successfully.

The average success rate of recovery, based on the impact level, using the base case approach is shown in Figure 4.15

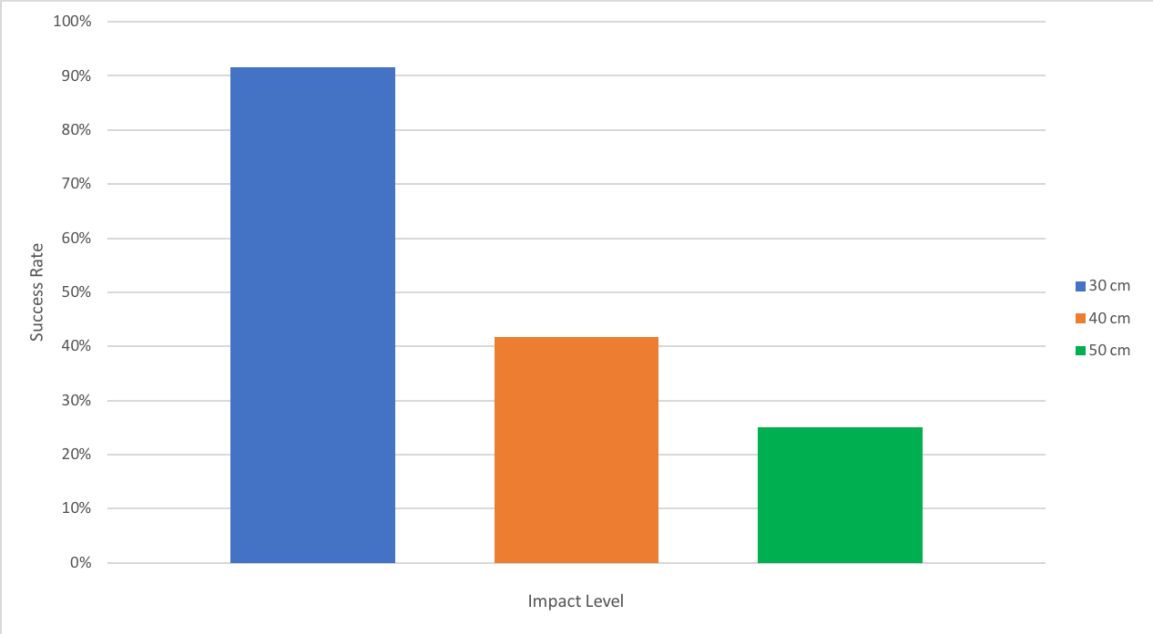


Figure 4.15: Average success rate of recovery based on the three impact levels using the base case on the concrete surface.

4.7.1.4 Summary

Figures 4.16 and 4.17 show the difference between the performances of my two approaches RL (Section 4.7.1.1) , DRL (Section 4.7.1.2), and the base case (Section 4.7.1.3). Final results show that Polaris was able to recover from all the pushes from the right side, using all three approaches. However, there are significant differences between my approaches and the base case in other directions. The base case performed worst in comparison with RL and DRL.

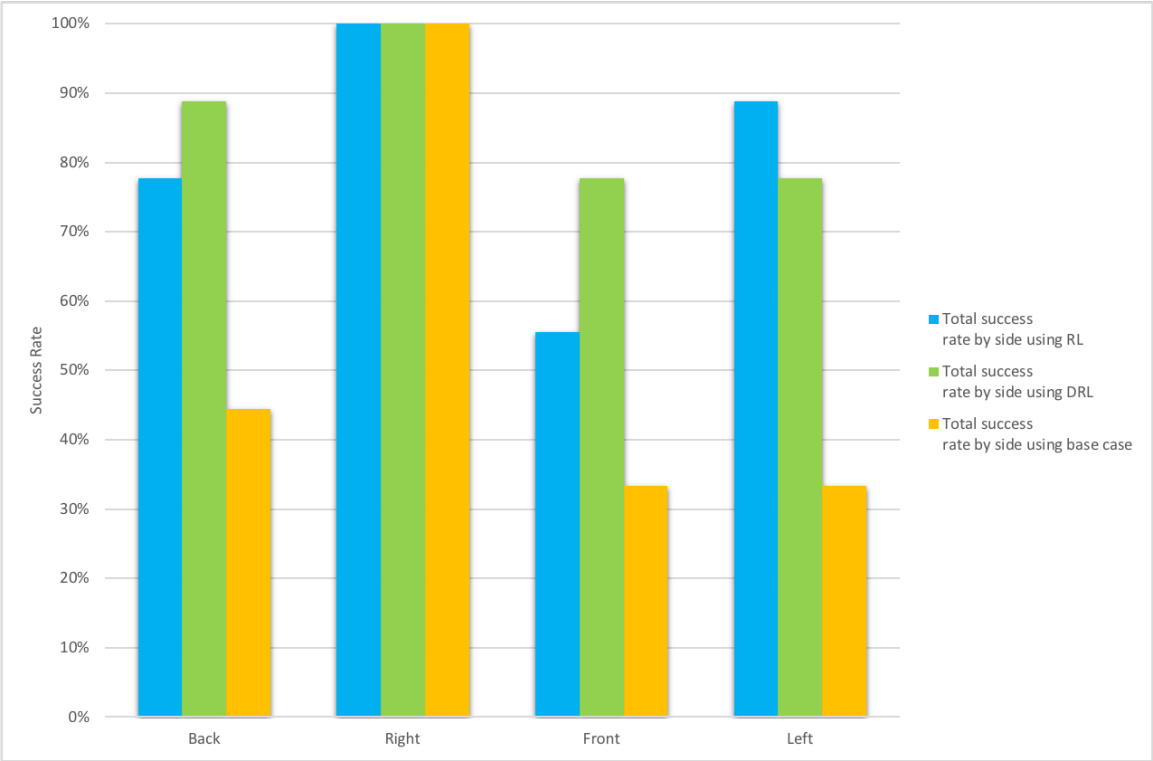


Figure 4.16: Comparison of my approaches vs the base case

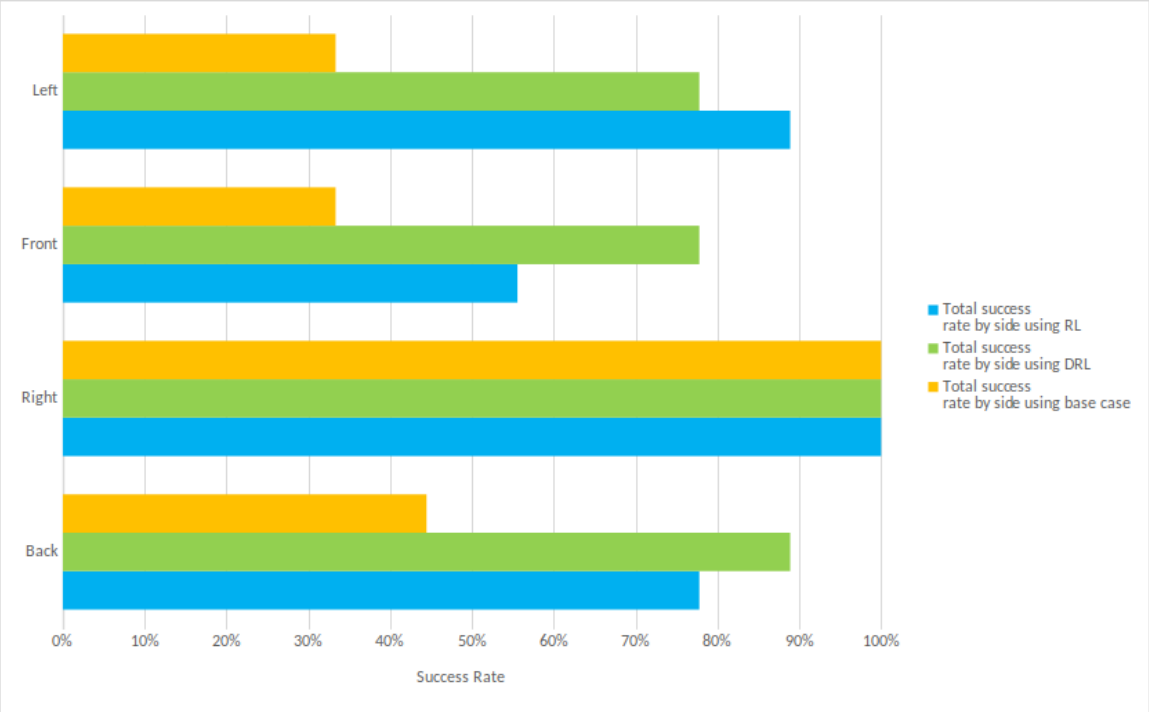


Figure 4.17: Comparison of my approaches vs the base case on the concrete floor

The average recovery rate of all directions and impacts for my DRL approach is 86%, followed by my RL approach with 81%. Finally the base case, with 53% of success recovery. Figure 4.18 shows this comparison.

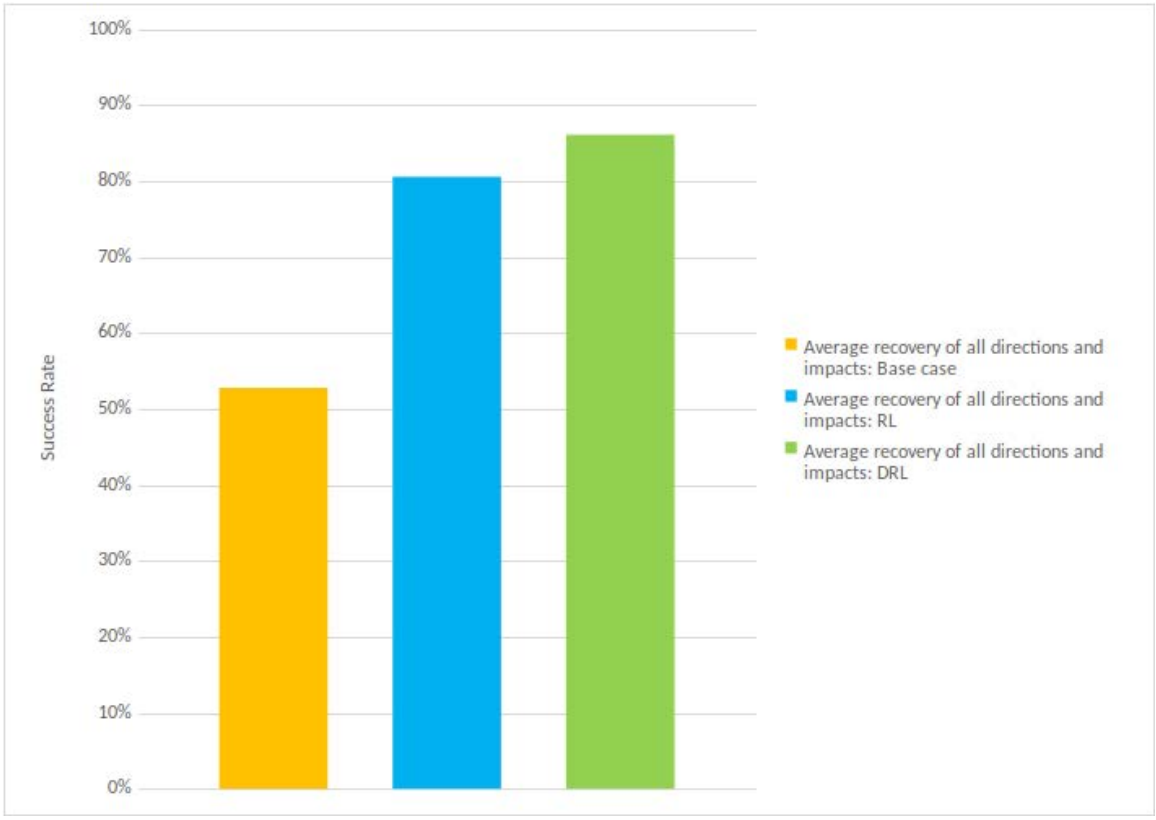


Figure 4.18: Comparison of my approaches vs the base case

4.7.2 Environment: Real World - Carpet Surface

This section describes the results I obtained from my experiments in the real world on the carpet surface. This involves the comparison of the results between RL, DRL, and the base case. In the following subsections, first, the results using RL are discussed. Second, I discuss the results obtained from testing my DRL. Third, I discuss the results of the base case. Finally, I discuss the performance of my approach versus the base case.

Figure 4.19 shows a successful recovery of a strong push (50 CM) on the carpet surface. The force was applied to the back of Polaris.

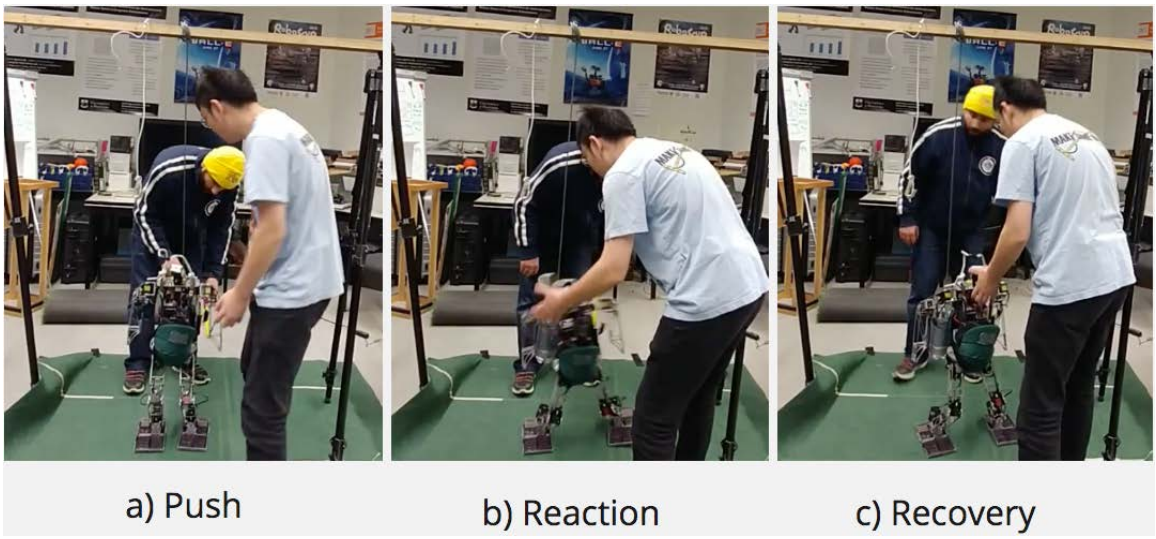


Figure 4.19: Push, reaction, recovery on the carpet

4.7.2.1 Approach: Reinforcement Learning

In this section I discuss the results of 36 pushes (12 * 3) on the carpet surface using my RL approach. All the actions were the results of the same reward function used in the simulator.

The results of all trials related to this experiment are shown in Table 4.4. For each case in which the robot successfully recovered from the impact, a score of 1 was given, otherwise no score was granted . Figure 4.20 shows the summarized results of my approach using RL. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The blue, orange, green, and purple columns represent 30 CM pushes, 40 CM pushes, 50 CM pushes, and total success rate by each direction in percentage form, respectively.

Carpet surface: RL		Successful attempts			Total number of successful attempts by directions
Distances in centimeter		30	40	50	Out of 9
Directions	Back	3	2	2	7
	Right	3	1	3	7
	Front	3	1	0	4
	Left	2	1	0	3

Table 4.4: Results of trials for the experiment in the real world on carpet using RL.

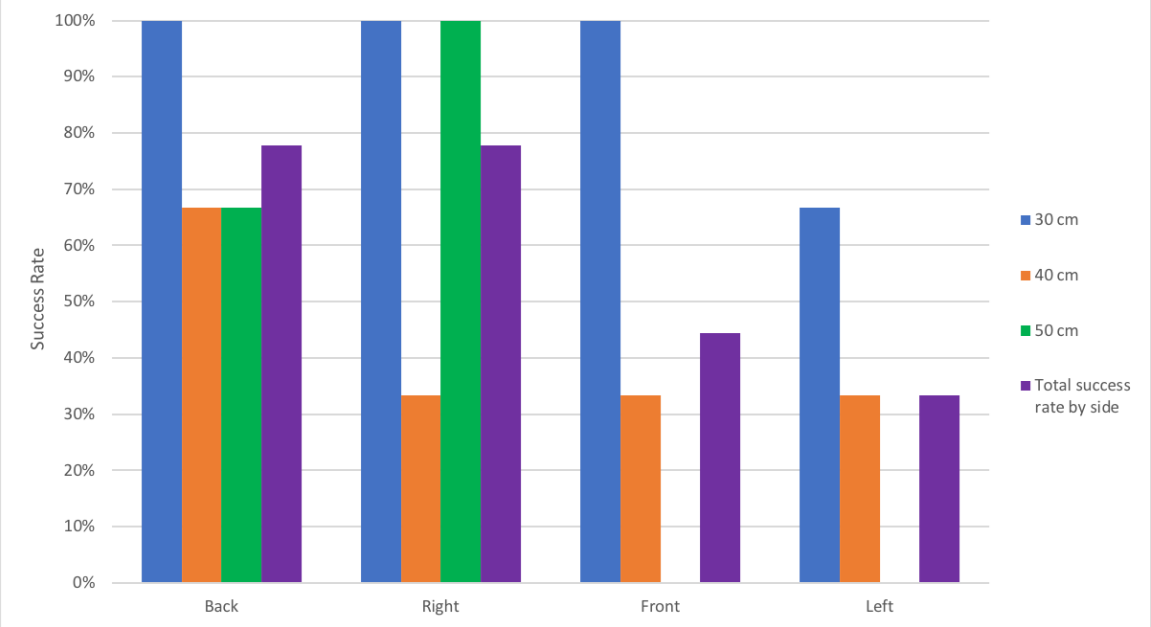


Figure 4.20: Experimental design in the real world using reinforcement learning on carpet

My closed-loop control mechanism using the RL approach was able to recover from a majority of pushes with an average success rate of 92% for all of the directions at the low impact level (30 CM swing). Polaris was able to recover from 100% of pushes that were received from Back, Right, and Front directions. Also, it could recover from 67% of pushes from the Left.

At the medium impact level (40 CM swing), the recovery average result was 42%, with: 67% recovery from back, and 33% recovery from right, front, and left.

At the strong impact level (50 CM swing), the average success rate of all directions is 42%. 100% of the pushes from the right side, and 67% of the pushes from back were recovered successfully. Polaris was not able to recover from any of the pushes that were applied to the front and left sides.

The average success rate of recovery, based on the impact level, using the RL approach is shown in Figure 4.21.

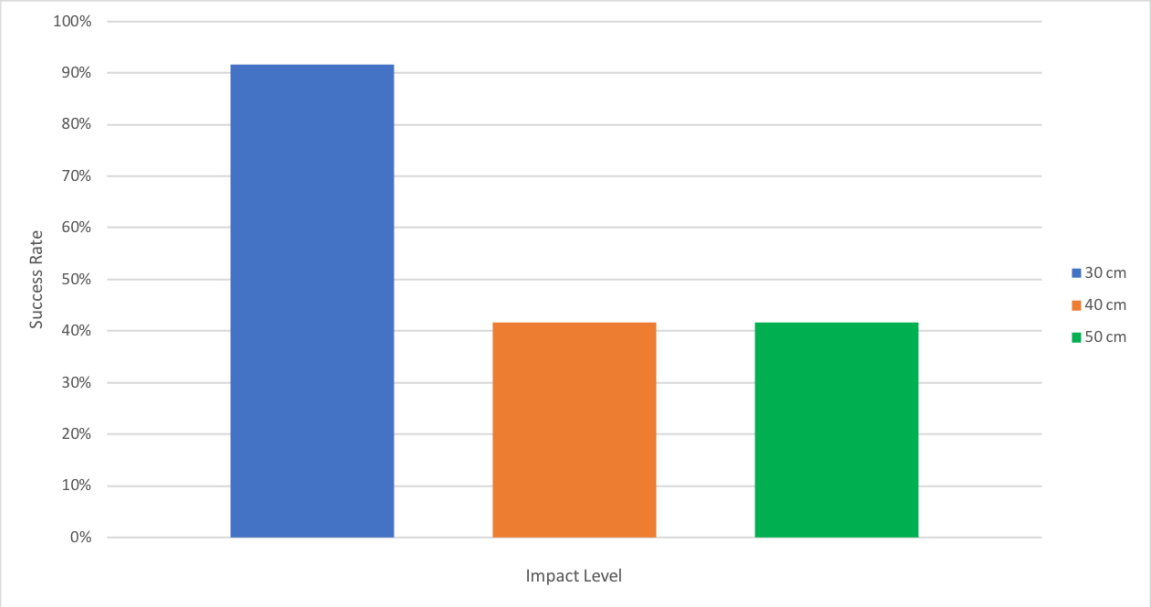


Figure 4.21: Average success rate of recovery based on the three impact levels using RL on the carpet floor.

4.7.2.2 Approach: Deep Reinforcement Learning

In this section I discuss the results of 36 pushes (12 * 3) in the real world on the carpet surface using my deep learning approach. All the actions were the results of

using my deep neural network as the reward function.

The results of all trials related to this experiment are shown in Table 4.5. Each case in which the robot successfully recovered from the impact resulted in 1 as a score, otherwise no score was granted. Figure 4.22 shows the summarized results of my approach using DRL. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The blue, orange, green, and purple columns represent 30 CM pushes, 40 CM pushes, 50 CM pushes, and total success rate by each direction in percentage form, respectively.

Carpet surface: DRL		Successful attempts			Total number of successful attempts by directions
Distances in centimeter		30	40	50	Out of 9
Directions	Back	3	3	1	7
	Right	3	2	3	8
	Front	2	1	0	3
	Left	2	2	0	4

Table 4.5: Results of trials for the experiment in the real world on carpet using DRL.

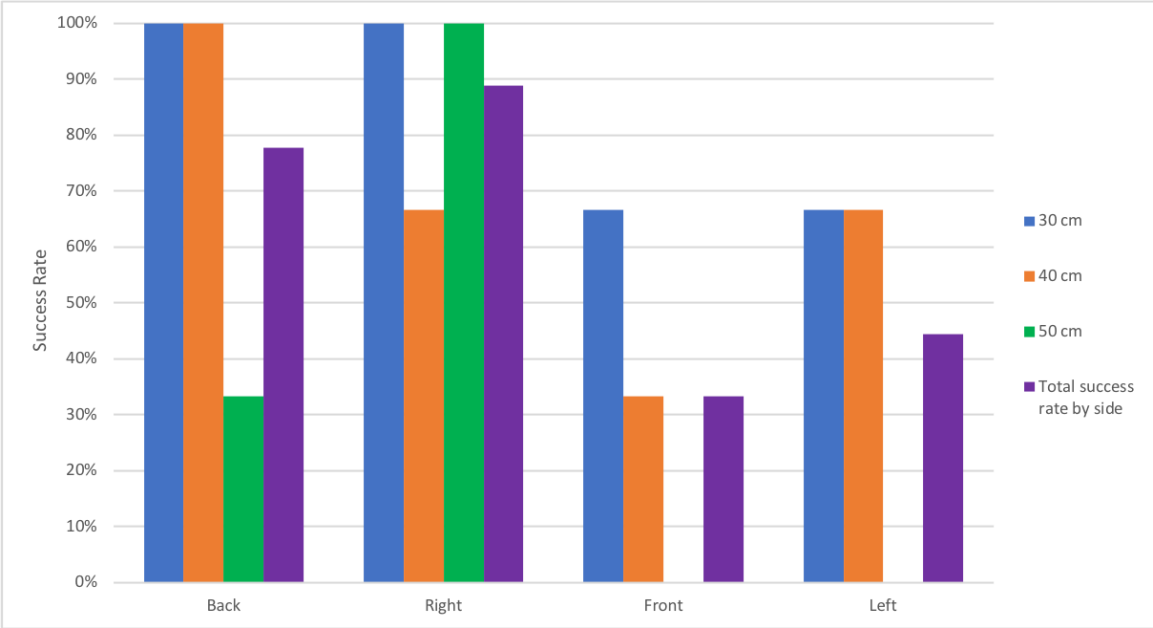


Figure 4.22: Experimental design in the real world using deep reinforcement learning on carpet

My closed-loop control mechanism using the DRL approach was able to recover from the average success rate of 83% for all of the directions at the low impact level (30 CM swing). Polaris was able to recover 100% of pushes that were received from Back and Right, and 67% from all of the pushes from Front and Left.

At the medium impact level (40 CM swing), the recovery average results was 67%, with 100% recovery from back, 67% from Right and Left, and 33% from Front.

At the strong impact level (50 CM swing), the average success rate of all directions is 33%. 100% of the pushes from Right, and 33% from Back were recovered successfully. Polaris was not able to recover from any of the strong pushes that were applied to the front and left sides.

The average success rate of recovery, based on the impact level, using the DRL

approach is shown in Figure 4.23.

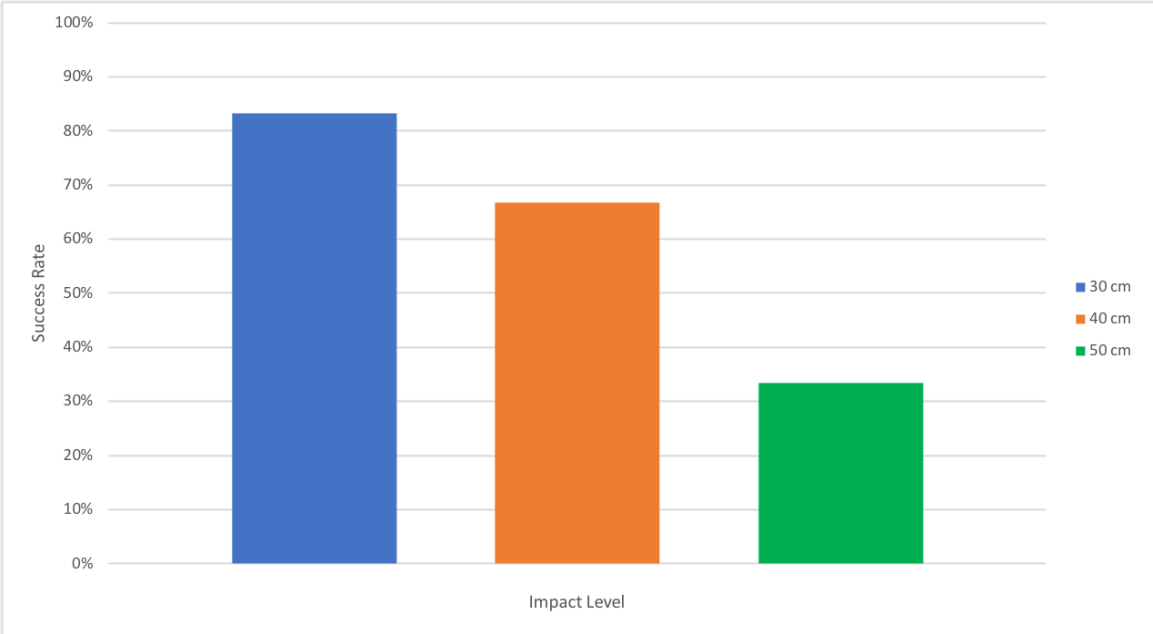


Figure 4.23: Average success rate of recovery based on the three impact levels using DRL on carpet.

4.7.2.3 Approach: Base Case

In this section I discuss the results of 36 pushes (12 * 3) in the real world on the carpet surface using the base case.

The results of all trials related to this experiment is shown in Table 4.6. Each case in which the robot successfully recovered from the impact resulted in 1 as a score, otherwise no score was granted.

Figure 4.24 shows the summarized results of the base case approach using the open-loop control feedback. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The

blue, orange, green, and purple columns represent 30 CM pushes, 40 CM pushes, 50 CM pushes, and total success rate by each direction in percentage form, respectively.

Carpet surface: Base case		Successful attempts			Total number of successful attempts by directions
Distances in centimeter		30	40	50	Out of 9
Directions	Back	2	0	0	2
	Right	3	3	1	7
	Front	2	0	0	2
	Left	3	0	0	3

Table 4.6: Results of trials for the experiment in the real world on carpet using the base case.

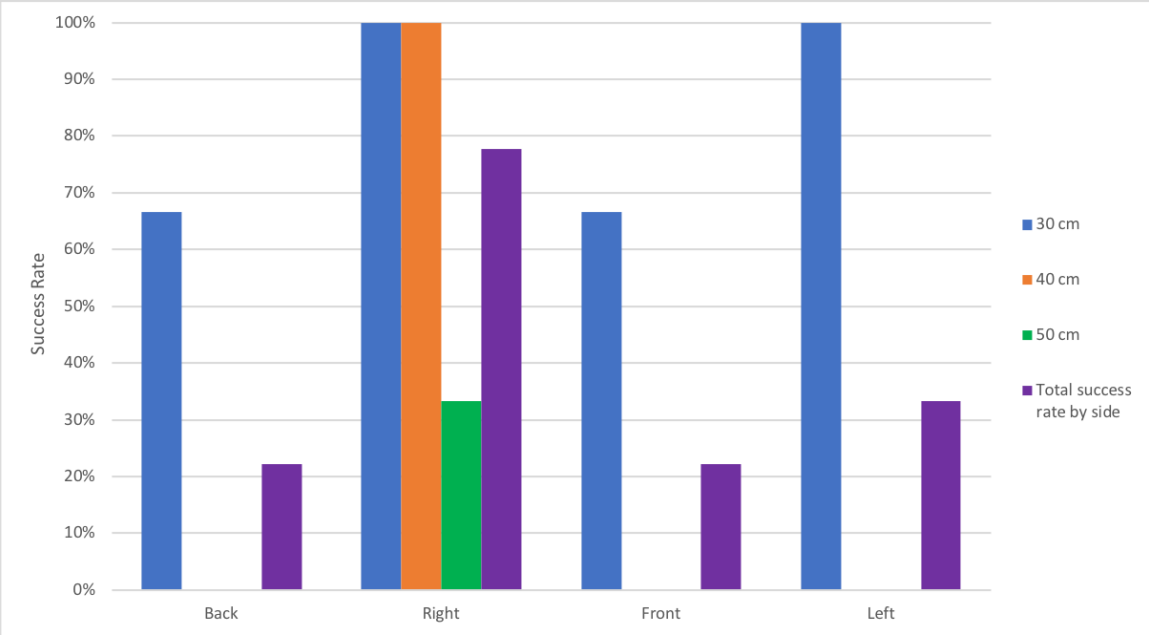


Figure 4.24: Experimental design in the real world using open-loop control feedback (IMU disabled)

The average recovery success rate using the base case is 83% for all of the directions at the low impact level (30_{cm} swing). Polaris was able to recover 100% of pushes that

was received from Right and Left. For the other two directions, Back and Front, it could recover from 67% of the pushes.

At the medium impact level (40_{cm} swing), the recovery average results were reduced, with 25% recovery for all the pushes from all the directions. Polaris was only able to recover from the Right side, but it recovered from all of the pushes in that direction. However, the recovery rate for the Back, Front, and Left was 0%.

At the strong impact level (50_{cm} swing), the average success rate of all directions is 8%: with 33% of the pushes from the right side, and 0% of the pushes from all the other sides were recovered successfully.

The average success rate of recovery, based on the impact level, using the base case approach is shown in Figure 4.25.

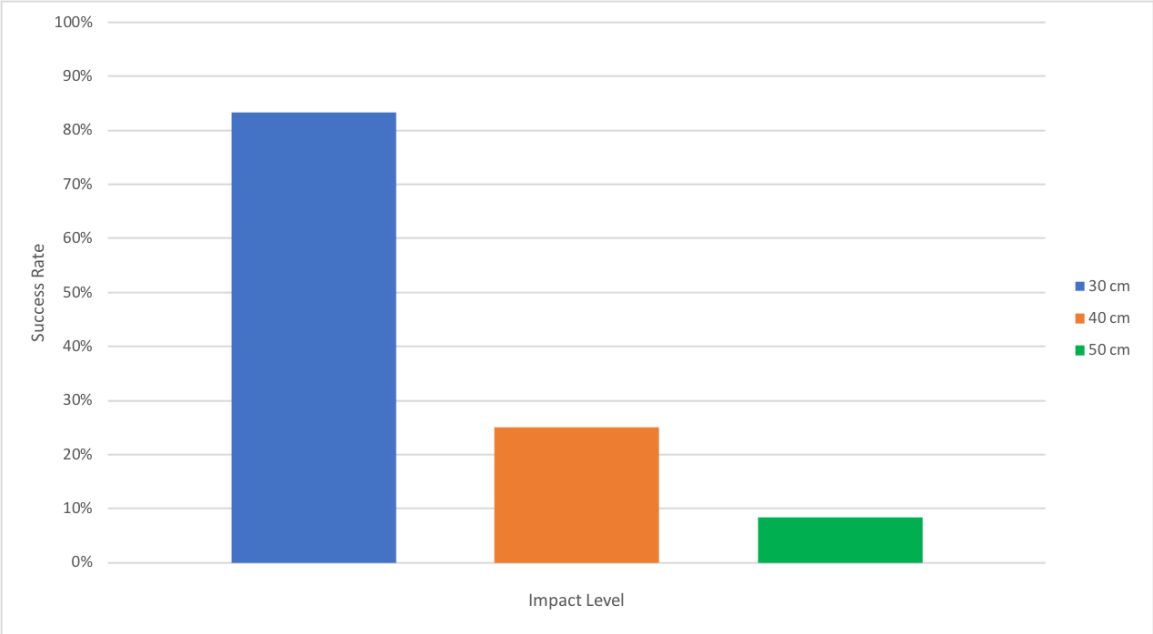


Figure 4.25: Average success rate of recovery based on the three impact impact levels using the base case on the carpet surface.

4.7.2.4 Summary

Figures 4.26 and 4.27 show the difference between the performances of my two approaches: RL (Section 4.7.3.1) , DRL (Section 4.7.3.2), as well as the base case (Section 4.7.3.3). Final results show that Polaris was able to recover from most of the pushes from the right side, using all three approaches. However, there are significant differences between my approaches and the base case in other directions. Both of my approaches RL and DRL performed better in comparison to the base case .

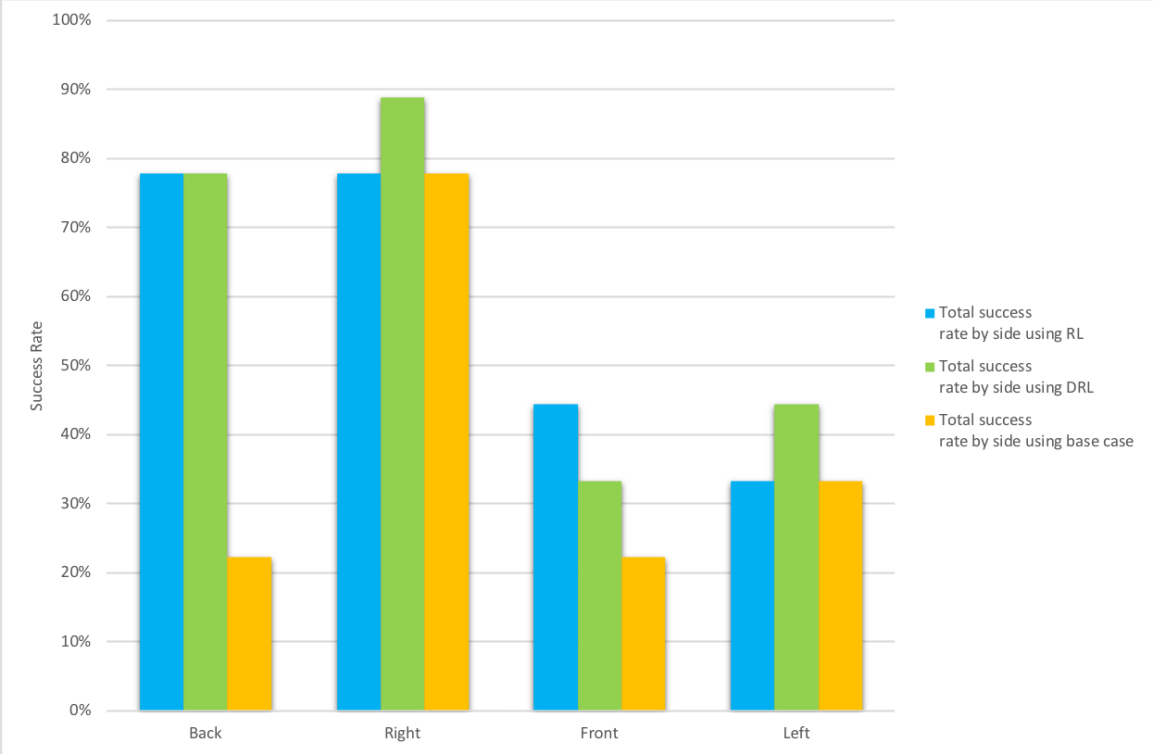


Figure 4.26: Comparison of my approaches vs the base case

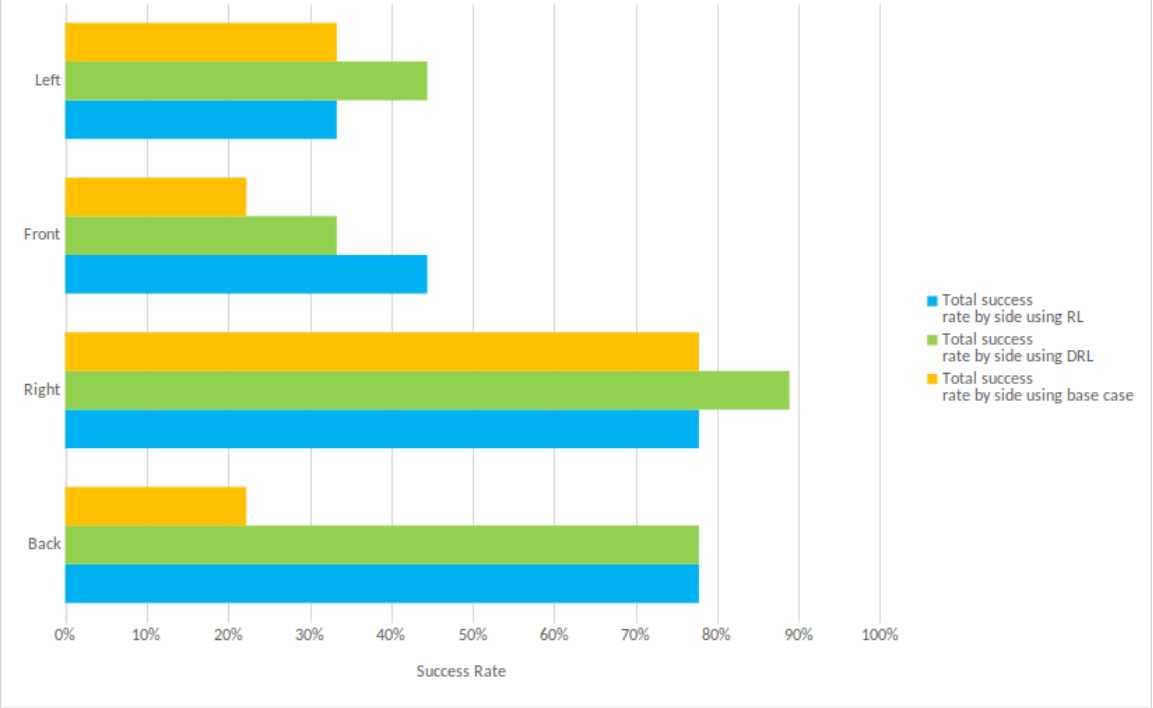


Figure 4.27: Comparison of my approaches vs the base case

The average recovery rate of all directions and impacts for my DRL approach is 61%, followed by my RL approach with 58%. Finally the base case, with 39% of success recovery. Figure 4.28 shows this comparison.

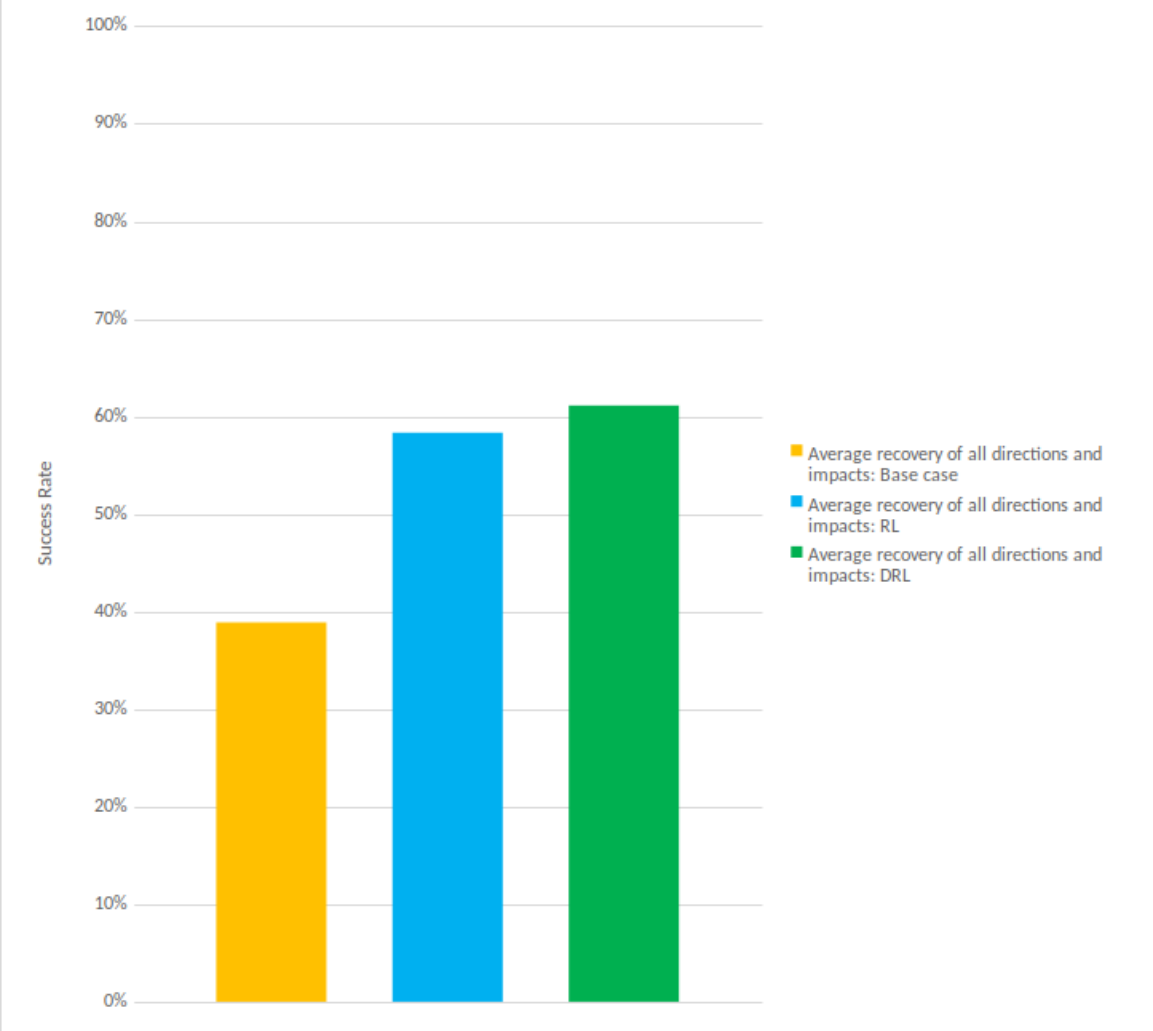


Figure 4.28: Comparison of my approaches vs the base case

4.7.3 Environment: Real World - Turf Surface

This section describes the results I obtained from my experiments in the real world on the artificial turf surface. This involves the comparison of the results between RL, DRL, and the base case. In the following subsections, first, the results using RL are discussed. Second, I discuss the results obtained from testing my DRL approach.

Third, I discuss the results of the base case. Finally, I describe the performance of my approaches versus the base case, similar to the previous sections.

Figure 4.29 shows a successful recovery of a strong push (50_{cm}). The force was applied to the back of Polaris.

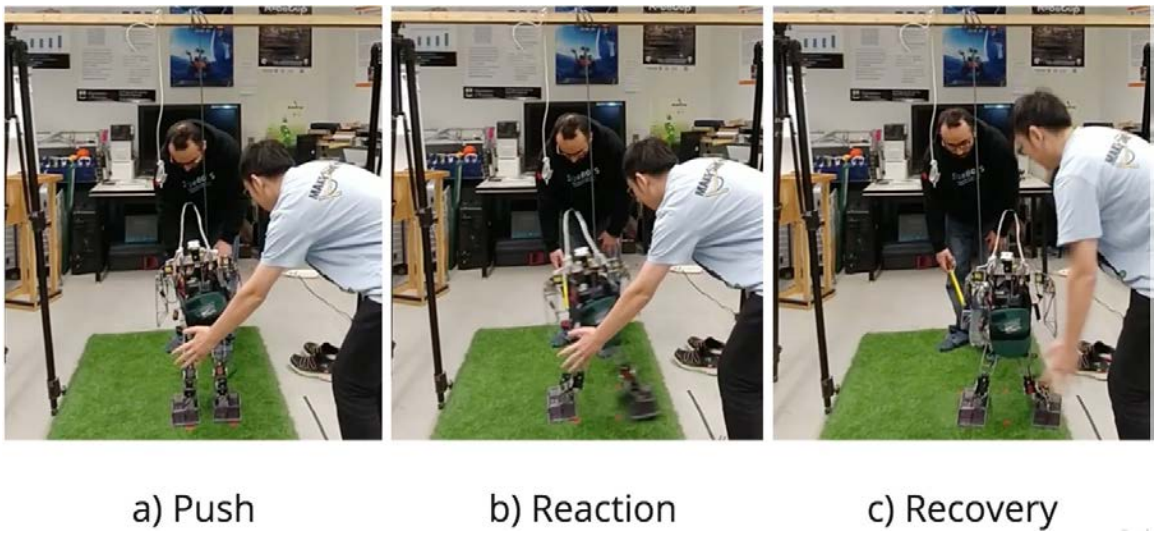


Figure 4.29: Push, reaction, recovery on artificial turf

4.7.3.1 Approach: Reinforcement Learning

In this section I discuss the results of 36 pushes (12 * 3) on the turf surface using my RL approach. All the actions were the results of the same reward function used in the simulator.

The results of all trials related to this experiment are shown in Table 4.7. Each case in which the robot successfully recovered from the impact resulted in a score of 1, otherwise no score was granted . Figure 4.30 shows the summarized results of my approach using RL. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The

blue, orange, green, and purple columns represent 30 CM pushes, 40 CM pushes, 50 CM pushes, and total success rate by each direction in percentages, respectively.

Turf surface: RL		Successful attempts			Total number of successful attempts by directions
Distances in centimeter		30	40	50	Out of 9
Directions	Back	3	1	0	4
	Right	3	3	2	8
	Front	1	0	0	1
	Left	2	1	1	4

Table 4.7: Results of trials for the experiment in the real world on turf using RL.

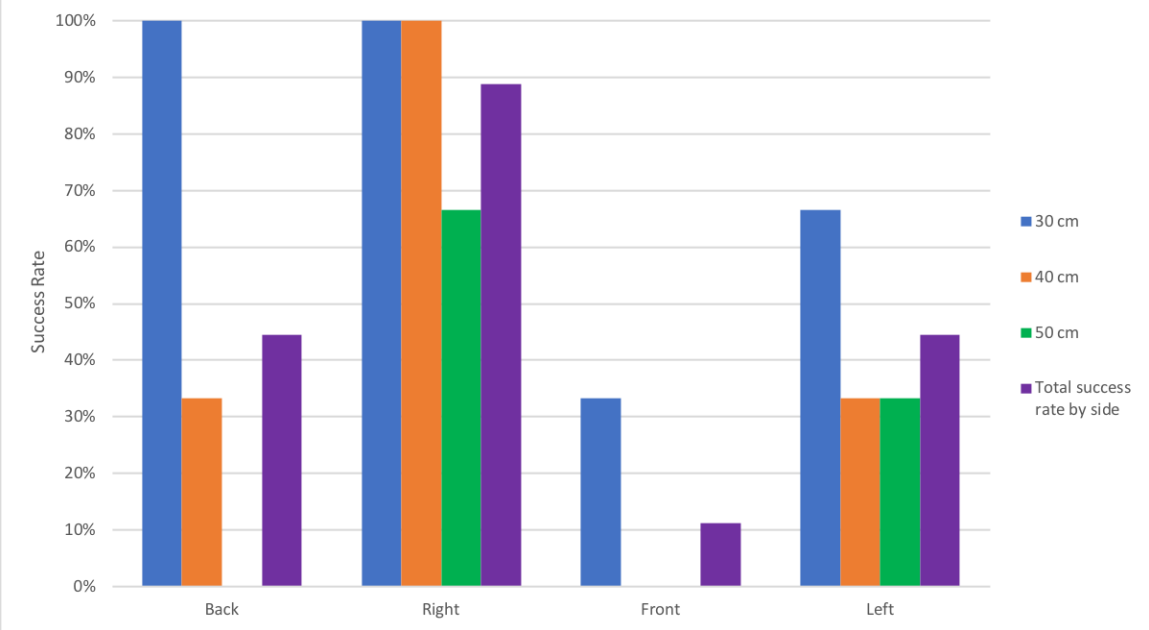


Figure 4.30: Experimental design in the real world using reinforcement learning on artificial turf

My closed-loop control mechanism using the RL approach was able to recover with an average success rate of 75% for all of the directions at the low impact level (30_{cm} swing). Polaris was able to recover from 100% of pushes that were received

from the Back and Right directions. Also, it could recover from 67% of the pushes from the Left and 37% from the Front.

At the medium impact level (40_{cm} swing), the recovery average result was 42%, with: 100% recovery from right, and 33% recovery from back and left. Polaris was not able to recover from any of the pushes that were applied to the front side.

At the strong impact level (50_{cm} swing), the average success rate of all directions was 25%, with 67% of the pushes from the Right side, and 33% of the pushes from Left recovered successfully. Polaris was not able to recover from any of the pushes that were applied to the Front and Back sides.

The average success rate of recovery, based on the impact level, using the RL approach is shown in Figure 4.31.

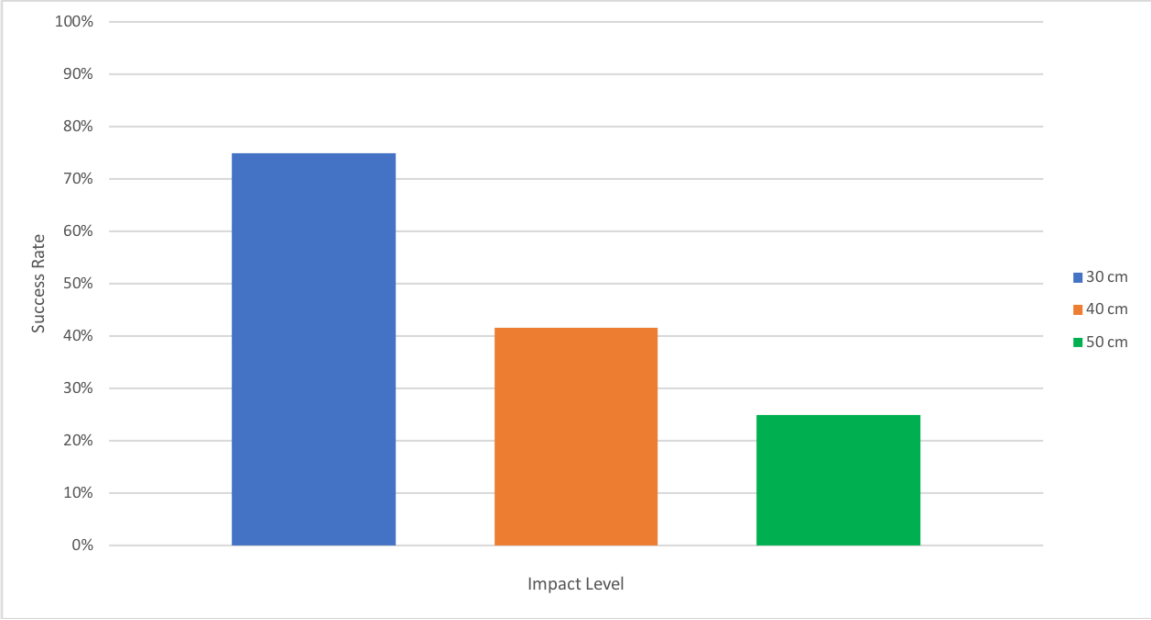


Figure 4.31: Average success rate of recovery based on the three impact levels using RL on artificial turf.

4.7.3.2 Approach: Deep Reinforcement Learning

In this section I discuss the results of 36 pushes (12 * 3) on the artificial turf surface using my deep learning approach. All the actions were the results of using my deep neural network as the reward function.

The results of all trials related to this experiment are shown in Table 4.8. Each case in which the robot successfully recovered from the impact resulted in a score of 1, otherwise no score was granted . Figure 4.32 shows the summarized results of my approach using DRL. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The blue, orange, green, and purple columns represent 30_{cm} pushes, 40_{cm} pushes, 50_{cm} pushes, and total success rate by each direction in percentage, respectively.

Turf surface: DRL		Successful attempts			Total number of successful attempts by directions
Distances in centimeter		30	40	50	Out of 9
Directions	Back	2	0	2	4
	Right	3	3	2	8
	Front	0	0	0	0
	Left	1	2	0	3

Table 4.8: Results of trials for the experiment in the real world on artificial turf using DRL.

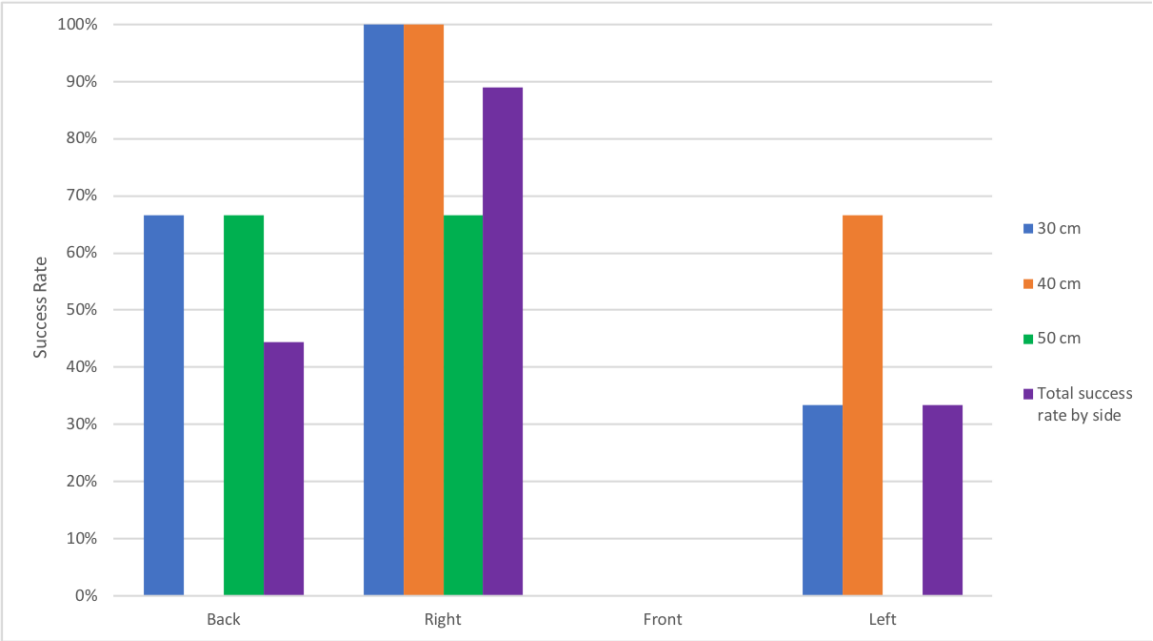


Figure 4.32: Experimental design in the real world using deep reinforcement learning on artificial turf

My closed-loop control mechanism using the DRL approach was able to recover from an average success rate of 50% for all of the directions at the low impact level (30_{cm} swing). Polaris was able to recover from 100% of pushes that were received from the Right, 67% from all of the pushes from the Back, and 33% from the Left side. None of the pushes from the Front side were recovered by Polaris.

At the medium impact level (40_{cm} swing), the recovery average result was 42%, with 100% recovery from the Right, and 67% from the Left. Polaris was not able to recover from any of the pushes from the Back and Front.

At the strong impact level (50_{cm} swing), the average success rate of all directions was 33%, with 67% recovery from the pushes from the Back and Right. Polaris was not able to recover from any of the pushes that were applied to the Front and Left

sides.

The average success rate of recovery, based on the impact level, using the DRL approach is shown in Figure 4.33

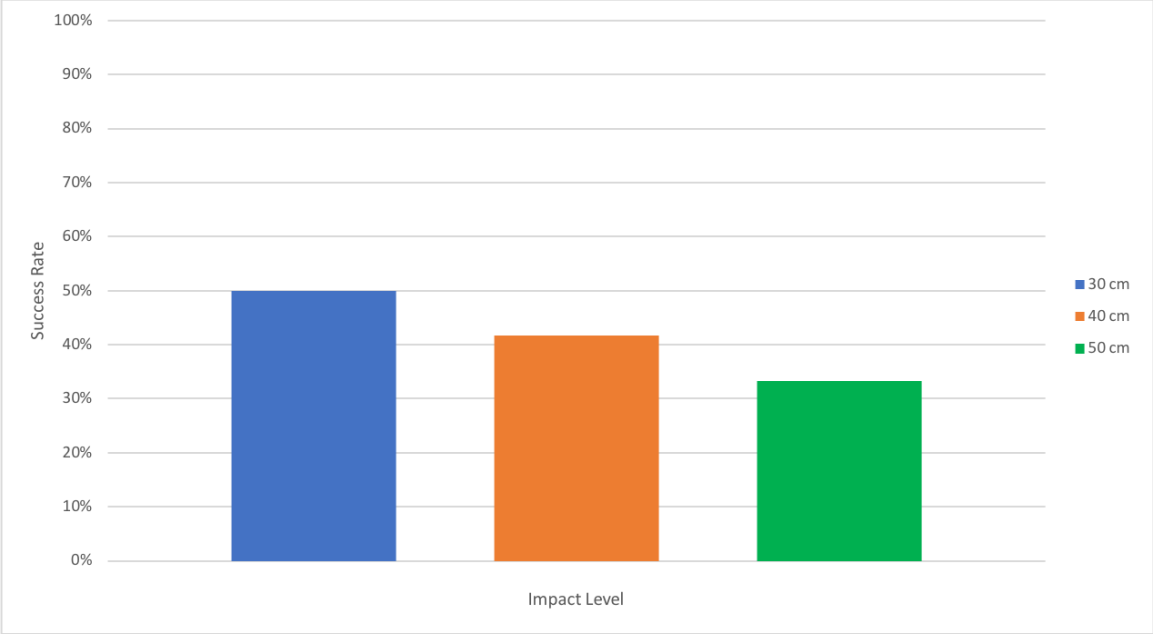


Figure 4.33: Average success rate of recovery based on the three impact levels using DRL on the artificial turf surface.

4.7.3.3 Approach: Base Case

In this section I discuss the results of 36 pushes (12 * 3) in the real world on the artificial turf surface using the base case.

The results of all trials related to this experiment are shown in Table 4.9. Each case in which the robot successfully recovered from the impact resulted in a score of 1 being given, otherwise no score was granted.

Figure 4.34 shows the summarized results of the base case approach using the

open-loop feedback control. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The blue, orange, green, and purple columns represent 30_{cm} pushes, 40_{cm} pushes, 50_{cm} pushes, and total success rate by each direction in percentage form, respectively.

Turf surface: Base case		Successful attempts			Total number of successful attempts by directions
Distances in centimeter		30	40	50	Out of 9
Directions	Back	1	0	0	1
	Right	3	1	0	4
	Front	0	0	0	0
	Left	2	0	0	2

Table 4.9: Results of trials for the experiment in the real world on artificial turf using the base case.

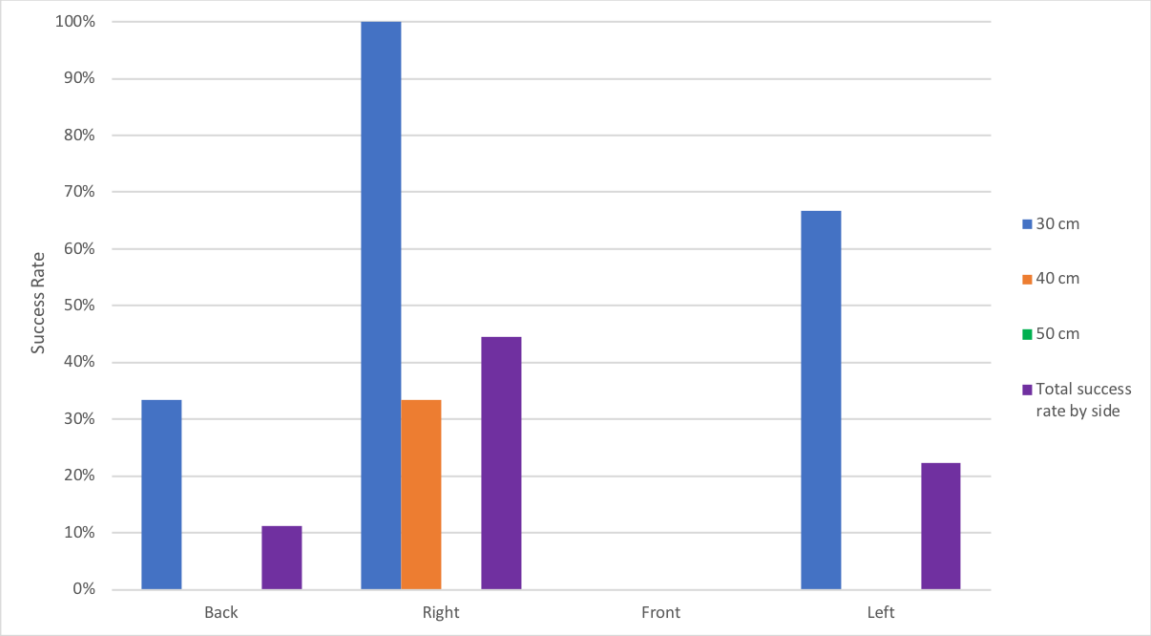


Figure 4.34: Experimental design in the real world using open-loop feedback control (IMU disabled)

The average recovery success rate using the base case is 50% for all of the directions at the low impact level (30 CM swing). Polaris was able to recover 100% of pushes that were received from the Right, 67% from the Left, and 33% from the Back. None of the pushes from the Front were recovered by Polaris.

At the medium impact level (40 CM swing), the recovery average results was reduced to 8%, with only 33% recovery from the Right side. The recovery rate for the Back, Front, and Left was 0%.

At the strong impact level (50 CM swing), the average success rate of all directions is 0%: Polaris was not able to recover from any of the applied pushes in any direction.

The average success rate of recovery, based on the impact level, using the base case approach is shown in Figure 4.35.

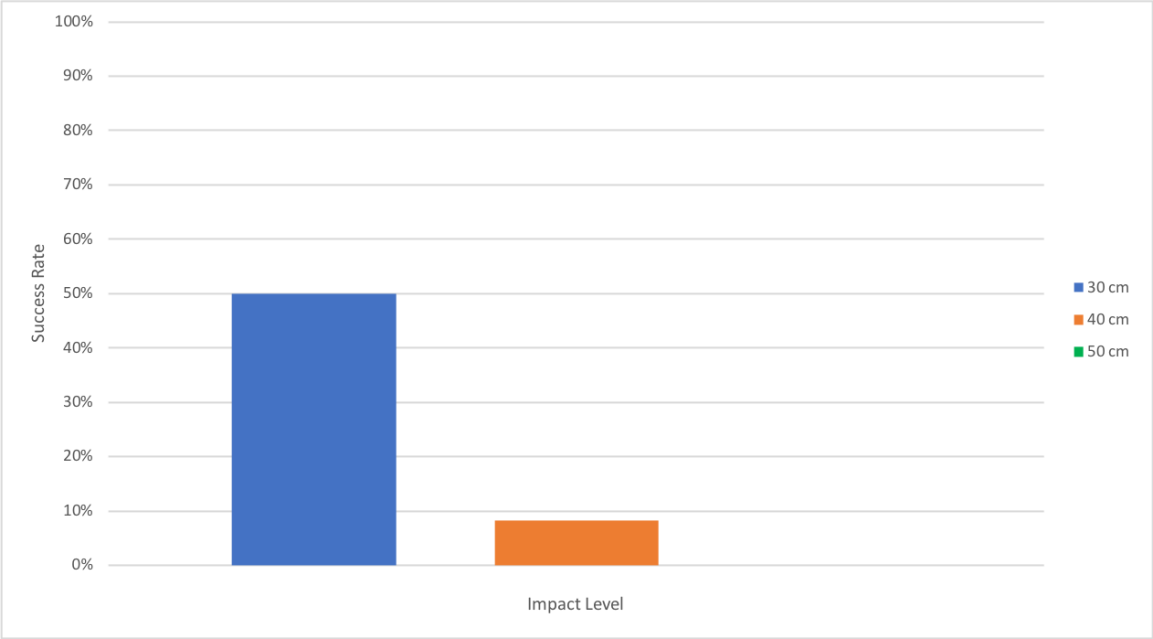


Figure 4.35: Average success rate of recovery based on the three impact levels using the base case on artificial turf.

4.7.3.4 Summary

Figures 4.36 and 4.37 show the difference between the performances of my two approaches RL (Section 4.7.3.1) , DRL (Section 4.7.3.2), and the base case (Section 4.7.3.3).

Final results show that Polaris was able to recover from 89% of the pushes from the Right side using the RL and DRL approaches. However, the recovery rate for the base case from the same side was 44%.

On the Back side, both RL and DRL performed equally with the total success rate of 44%. The recovery for the base case was 11%.

When using the RL approach, Polaris was able to recover from 11% of the pushes from the Front. Using both DRL and the base case, the success rate was 0%.

Finally, from the Left side, the RL approach led to 44%, DRL 33%, and base case 22% successful recovery rate.

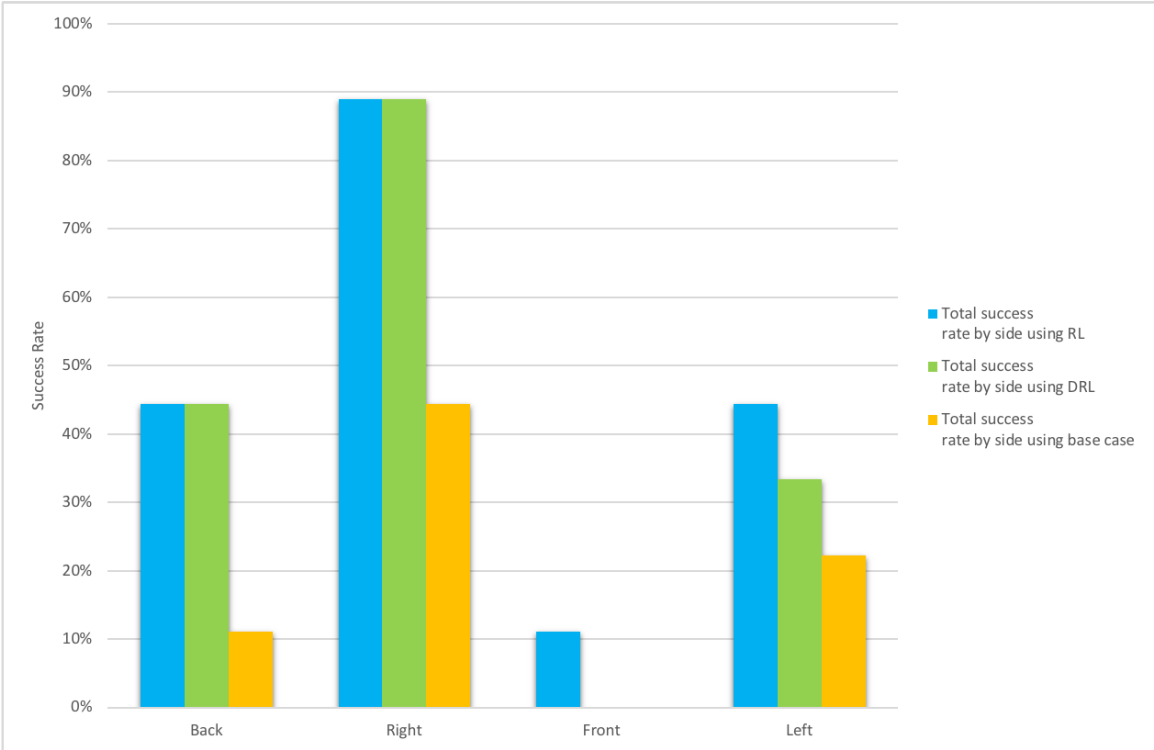


Figure 4.36: Comparison of my approaches vs the base case

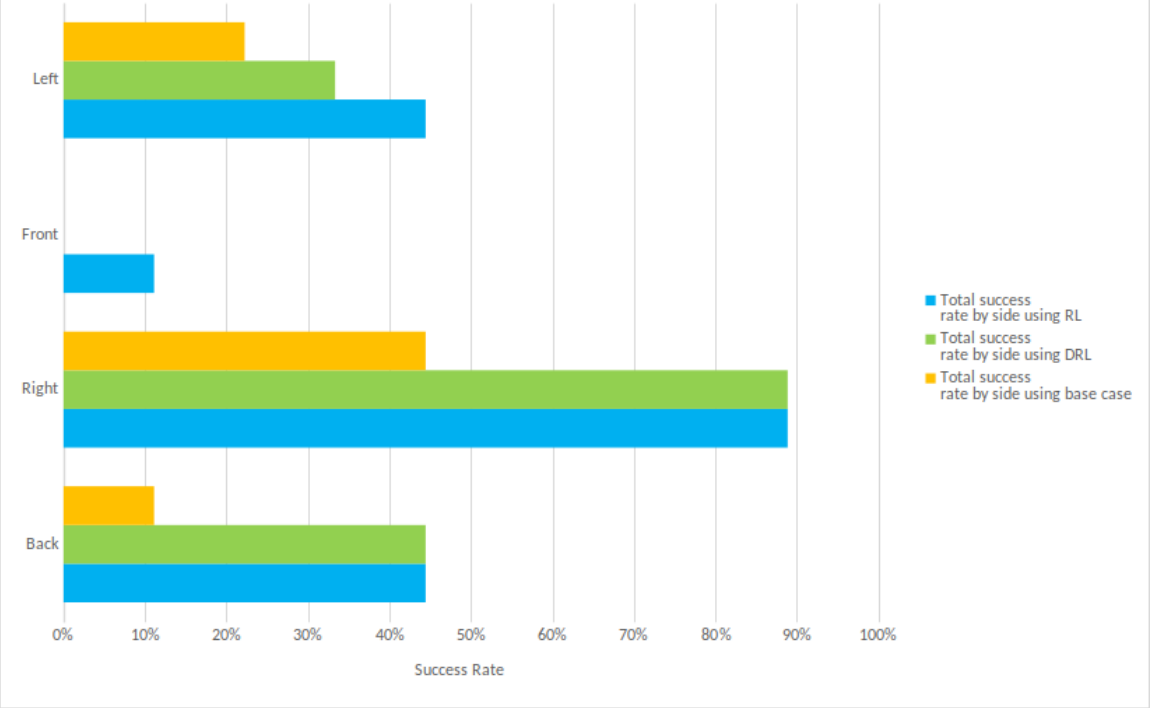


Figure 4.37: Comparison of my approaches vs the base case

The average recovery rate of all directions and impacts for the base case is 19%, improved by my DRL approach with 42%. The best results were from RL, with 47% successful recovery. Figure 4.38 shows this comparison.

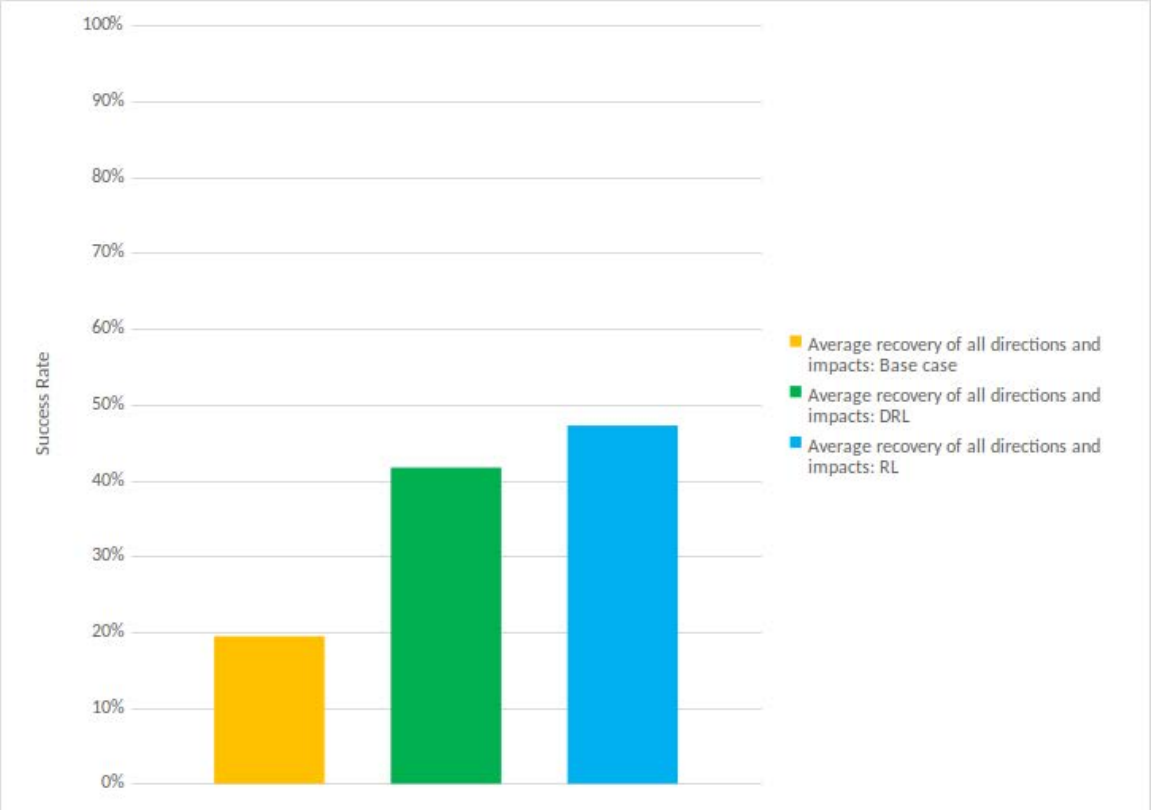


Figure 4.38: Comparison of my approaches vs the base case

4.7.4 Environment: Simulation - Concrete Surface

This section describes the results I obtained from my experiments in the simulation on the default surface (concrete). This involves the comparison of the results between reinforcement learning, deep reinforcement learning, and the base case. In the following subsections, first, the results of the RL are discussed. Second, I discuss the results obtained from testing my deep reinforcement learning approach. Third, I discuss the results of the base case. Finally, I describe the performance of my approach versus the base case.

Figure 4.39 shows a successful recovery from a strong push (50 CM). The force

was applied to the front side of Polaris. Figures 4.40 - 4.42 show the entire simulation environment during one of the trials.

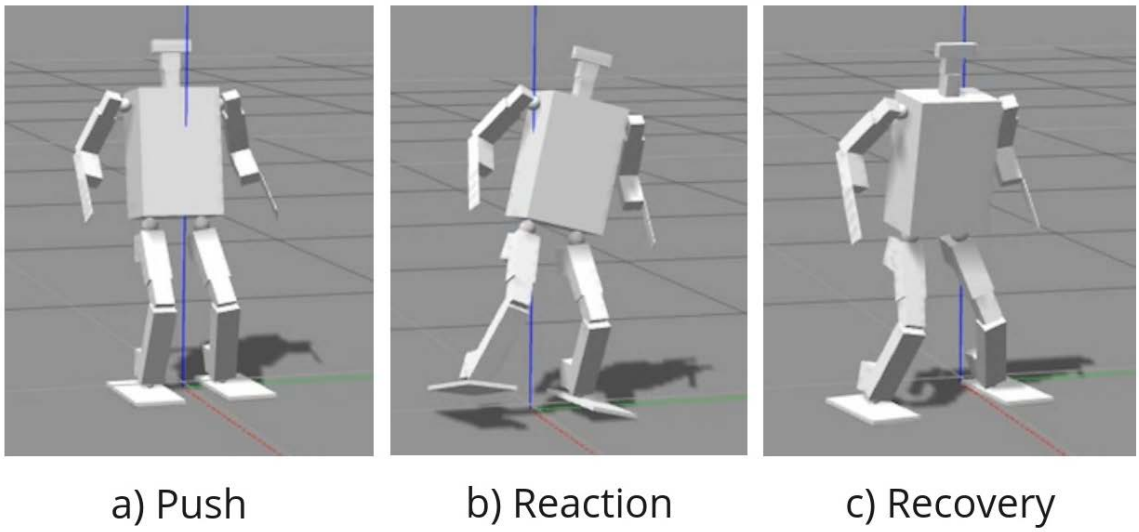


Figure 4.39: Push, Reaction, and Recovery sequence in the simulation environment on concrete

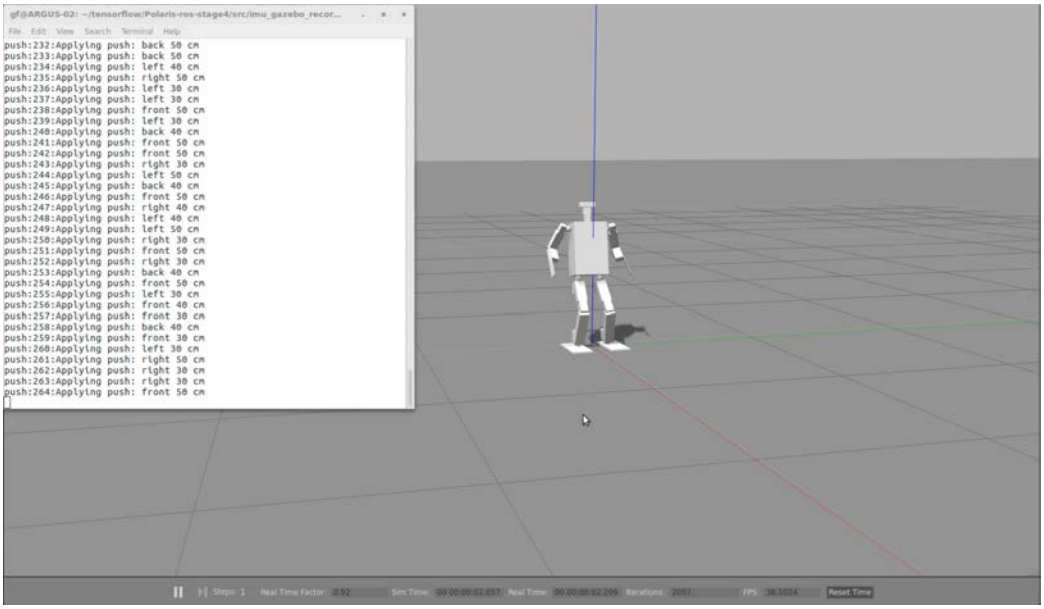


Figure 4.40: **Push**, Reaction, and Recovery in the simulation environment on concrete.

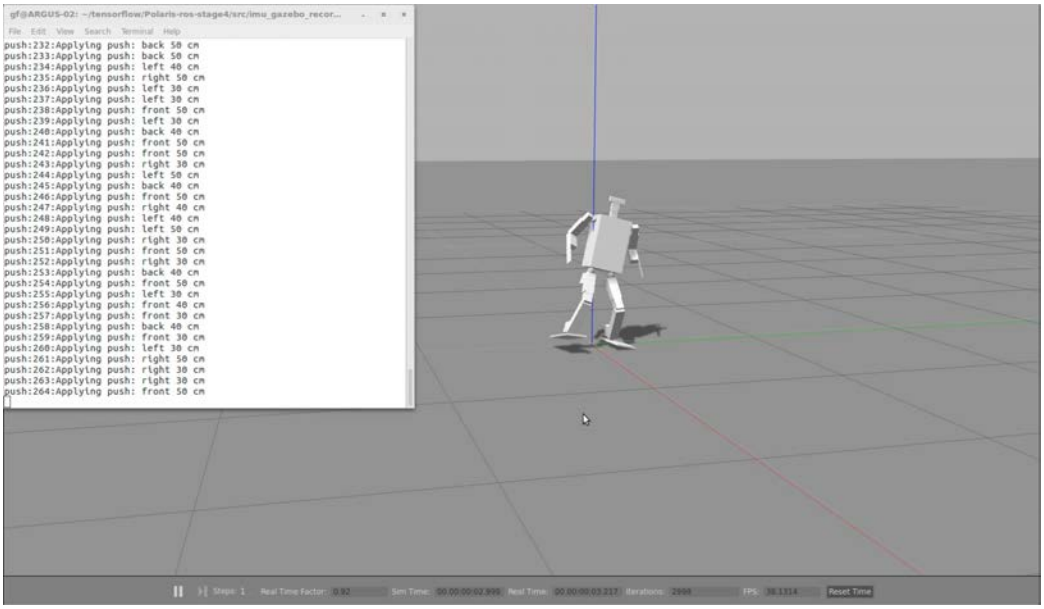


Figure 4.41: Push, **Reaction**, and Recovery in the simulation environment on concrete.

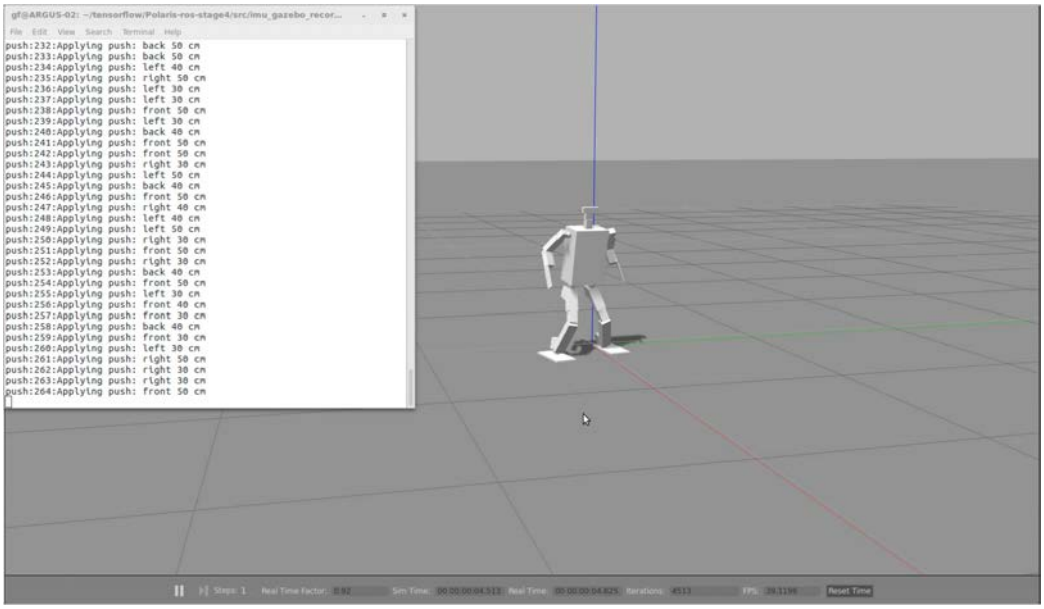


Figure 4.42: Push, Reaction, and **Recovery** in the simulation environment on concrete.

4.7.4.1 Approach: Reinforcement Learning

In this section I discuss the results of 360 pushes (12 * 30) on the simulated concrete surface using my reinforcement leaning approach.

The results of all trials related to this experiment are shown in Table 4.10. For each case in which the robot successfully recovered from the impact, a score of 1 was given, otherwise no score was granted . Figure 4.43 shows the summarized results of my approach using RL. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The blue, orange, green, and purple columns demonstrate 30 CM pushes, 40 CM pushes, 50 CM pushes, and the total success rate by each direction in percentages, respectively.

Concrete surface: RL		Successful attempts			Total number of successful attempts by directions
Distance in centimeters		30	40	50	Out of 90
Directions	Back	25	29	29	83
	Right	22	25	27	74
	Front	27	23	24	74
	Left	22	22	27	71

Table 4.10: Results of trials for the experiment in simulation on concrete using RL.

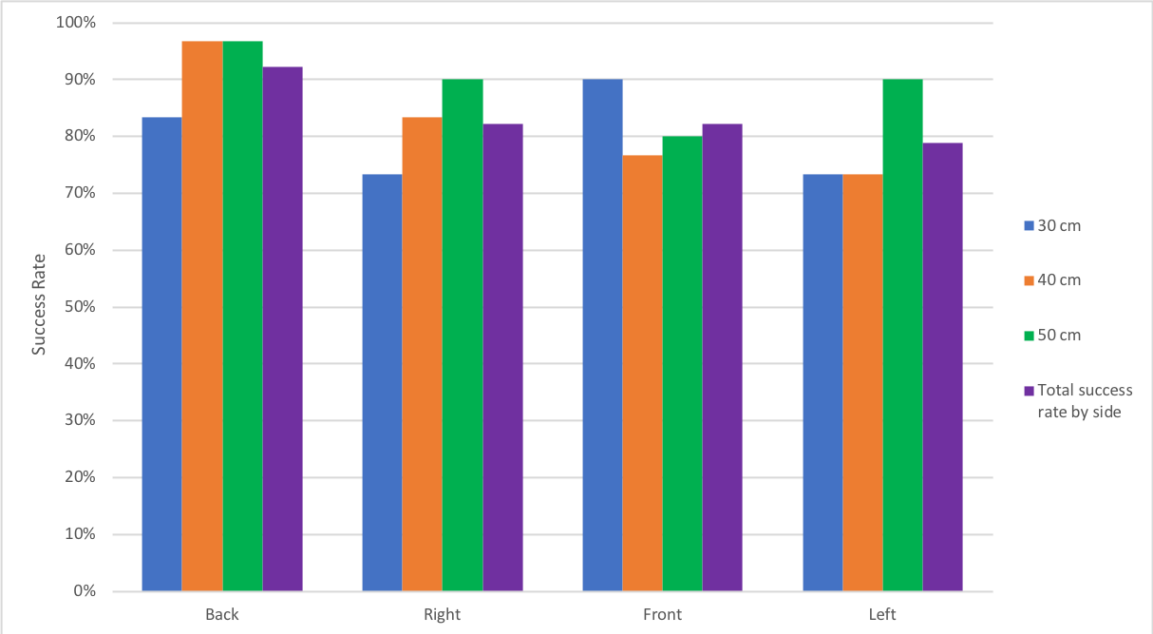


Figure 4.43: Experimental result in simulation on simulated concrete

My closed-loop control mechanism using the RL approach was able to recover from a large majority of pushes, with the average success rate of 80% for all of the directions at the low impact level (30 CM swing). Polaris was able to recover 90% of the pushes that was received from the Front, 83% from the Back, and 73% from the Left and Right.

At the medium impact level (40 CM swing), the recovery average result was 83%

overall, with 97% from the Back, 83% from the Right, 77% from the Front, and 73% from the Left were recovered successfully.

At the strong impact level (50 CM swing), the average success rate of all directions is 89%, with 97% of the pushes from the Back side, 90% of the pushes from the Right and Left sides, and 80% from the Front side were recovered successfully.

The average success rate of recovery, based on the impact level, using the RL approach is shown in Figure 4.44.

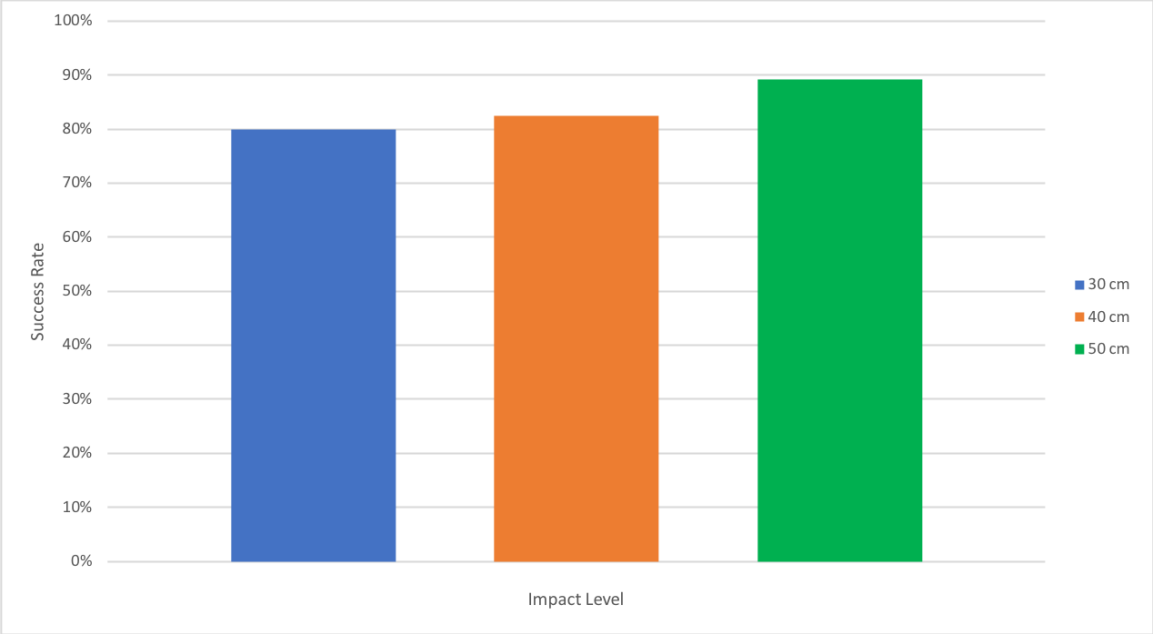


Figure 4.44: Average success rate of recovery based on the three impact levels using RL on the simulated concrete floor.

4.7.4.2 Approach: Deep Reinforcement Learning

In this section I discuss the results of 360 pushes (12 * 30) in simulation on the concrete surface using my deep reinforcement learning approach. All the actions were

the results of using my deep neural network as the reward function.

The results of all trials related to this experiment are shown in Table 4.11. For each case in which the robot successfully recovered from the impact, a score of 1 was given, otherwise no score was granted. Figure 4.45 shows the summarized results of my approach using DRL. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The blue, orange, green, and purple columns represent 30 CM pushes, 40 CM pushes, 50 CM pushes, and total success rate by each direction in percentage, respectively.

Concrete surface: DRL		Successful attempts			Total number of successful attempts by directions
Distances in centimeter		30	40	50	Out of 90
Directions	Back	27	28	28	83
	Right	29	28	29	86
	Front	24	29	28	81
	Left	27	28	26	81

Table 4.11: Results of trials for the experiment in simulation on concrete using DRL.

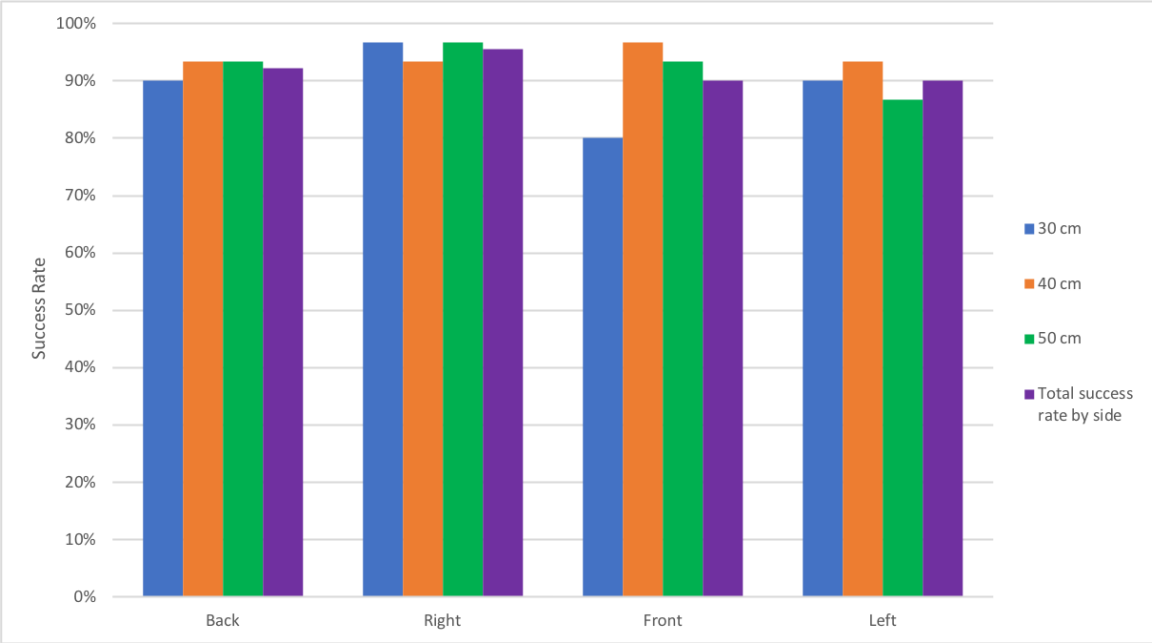


Figure 4.45: Experimental results in simulation on concrete

Similar to the RL approach (Section 4.7.4.1), my closed-loop control mechanism using the DRL approach was able to recover from a strong majority of pushes, with an average success rate of 89% for all of the directions at the low impact level (30 CM swing). Polaris was able to recover 97% of pushes that were received from the Right, 90% from the Back and Left, and 80% from the Front.

At the medium impact level (40 CM swing), the recovery average result was 94% recovery for all the pushes from all the directions. It recovered from 97% of the pushes from the Front, and 93% from all the other directions.

At the strong impact level (50 CM swing), the average success rate of all directions was 93%, with 97% of the pushes from the Right side, and 93% of the pushes from the Back and Front sides recovered successfully. Polaris was able to recover from 87% of the pushes that were applied to the Left side.

The average success rate of recovery, based on the impact level, using the DRL approach is shown in Figure 4.46.

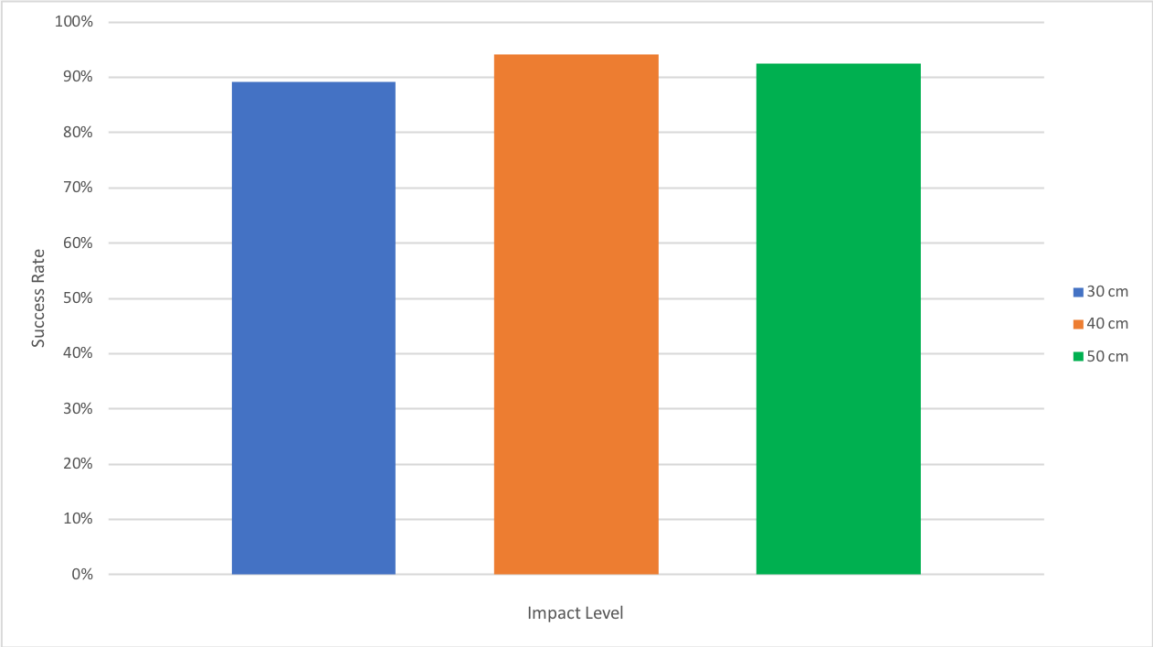


Figure 4.46: Average success rate of recovery based on the three impact levels using DRL on concrete.

4.7.4.3 Approach: Base Case

In this section I discuss the results of 360 pushes (12 * 30) on the simulated concrete surface using the base case.

The results of all trials related to this experiment are shown in Table 4.12. For each case in which the robot successfully recovered from the impact, a score of 1 was given, otherwise no score was granted.

Figure 4.47 shows the summarized results of the base case approach using closed-loop feedback control and taking random actions in the walking engine's parameter

thresholds (between minimum and maximum values, see Table 3.4). In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The blue, orange, green, and purple columns represent 30 CM pushes, 40 CM pushes, 50 CM pushes, and total success rate by each direction in percentage form, respectively.

Concrete surface: Base case		Successful attempts			Total number of successful attempts by directions
Distances in centimeter		30	40	50	Out of 90
Directions	Back	1	2	1	4
	Right	2	1	6	9
	Front	4	5	7	16
	Left	5	1	4	10

Table 4.12: Results of trials for the experiment in simulation on concrete using the base case.

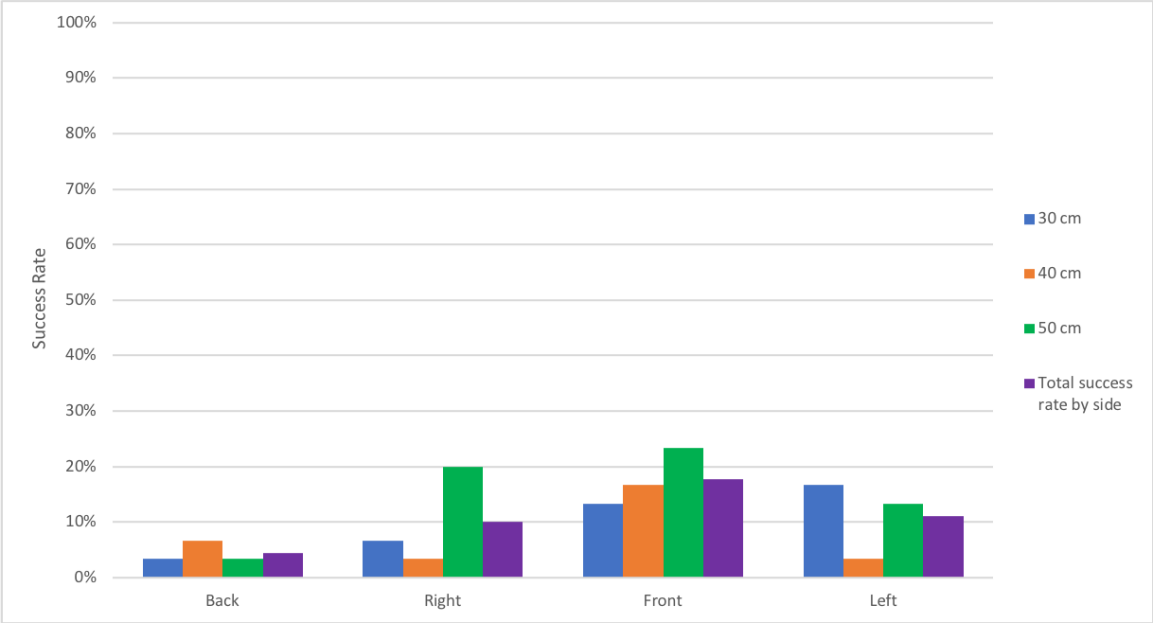


Figure 4.47: Experimental results in the simulation using random actions between the threshold limits, on concrete.

The average recovery success rate using the base case is 10% for all of the directions at the low impact level (30 CM swing). Polaris was able to recover from the Back, Right, Front, and Left pushes, with 3%, 7%,13%, and 17% respectively.

At the medium impact level (40 CM swing), the recovery average results was 8%, with 7% from the Back, 3% from the Right and left, and 17% from the front.

At the strong impact level (50 CM swing), the average success rate of all directions was 15% with 3% recovery from the Back, 20% from the Right, 23% from the Front, and 13% from the Left.

The average success rate of recovery, based on the impact level, using the base case approach is shown in Figure 4.48.

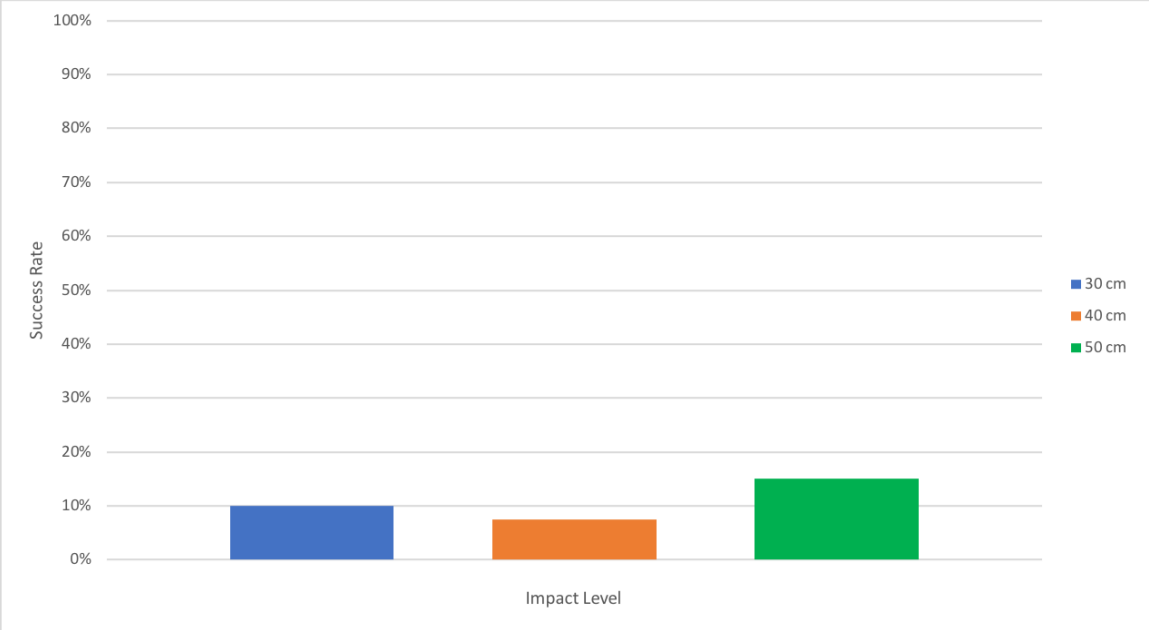


Figure 4.48: Average success rate of recovery based on the three impact levels using the base case on the simulated concrete surface.

4.7.4.4 Summary

Figures 4.49 and 4.50 show the difference between the performances of my two approaches RL (Section 4.7.4.1) , DRL (Section 4.7.4.2), and the base case (Section 4.7.4.3).

Final results show that Polaris was able to recover from 92% of the pushes from the Back side, using the RL and DRL approaches. However, the recovery rate for the base case from the same side was 4%.

On the Right side, by using RL, it could recover from 82% of the pushes. DRL performed very well with a total success rate of 96%. The recovery for the base case was 10%.

When using the RL approach, Polaris was able to recover from 82% of the pushes from the Front. Using the DRL approach makes the recovery results better at 90%. For the base case, the success rate was 18%.

Finally, from the Left side, the RL approach led to 79% successful recovery, the DRL 90%, and the base case 11% total success recovery rate.

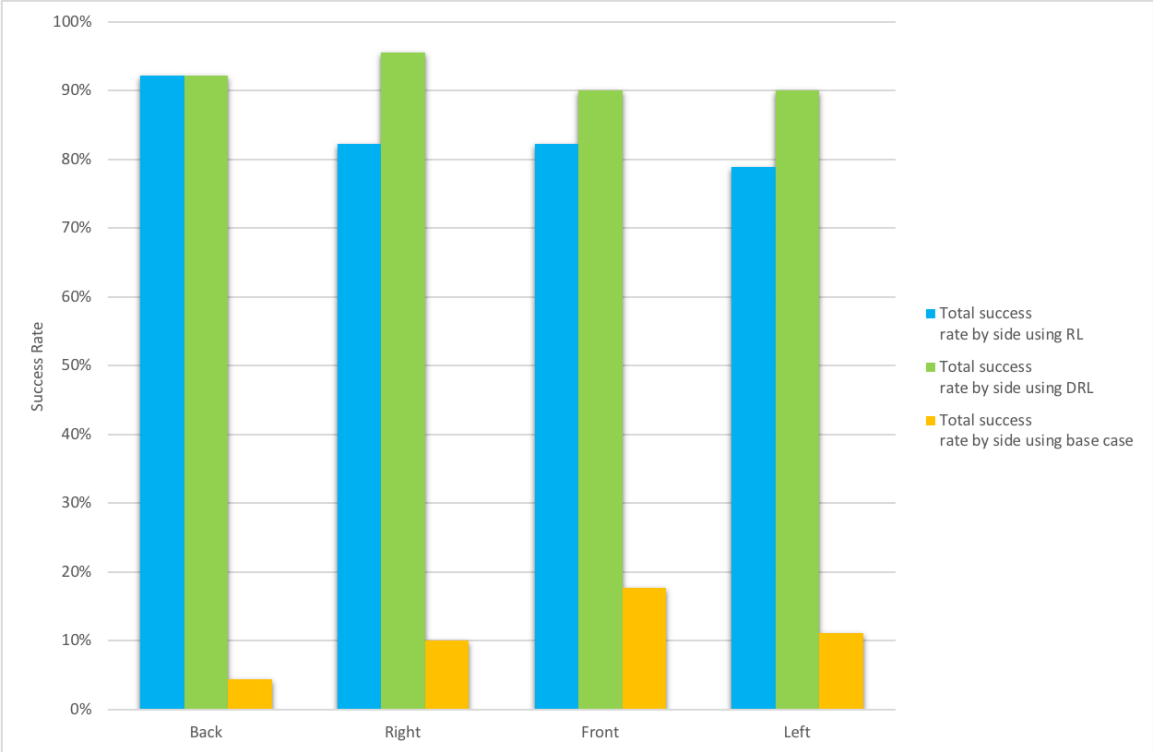


Figure 4.49: Comparison of my approaches vs the base case

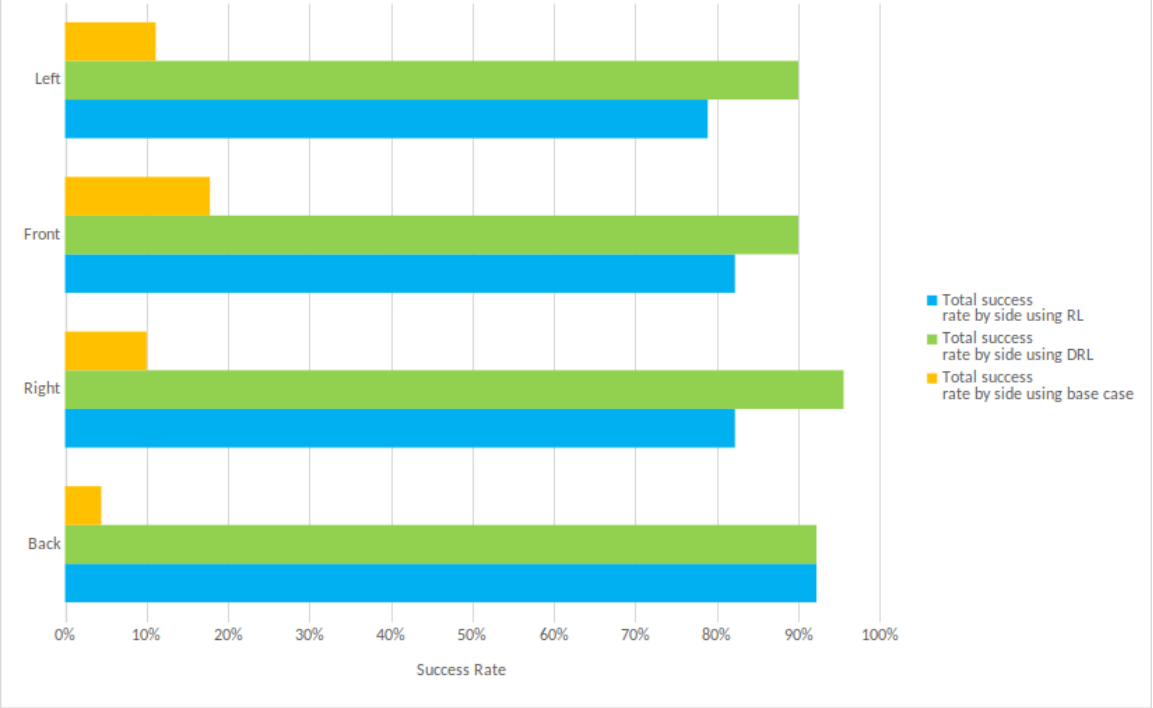


Figure 4.50: Comparison of my approaches vs the base case

The average recovery rate of all directions and impacts for the base case is 11%, improved by my RL approach with 84%. The best results were from DRL, with 92% successful recovery. Figure 4.51 shows this comparison.

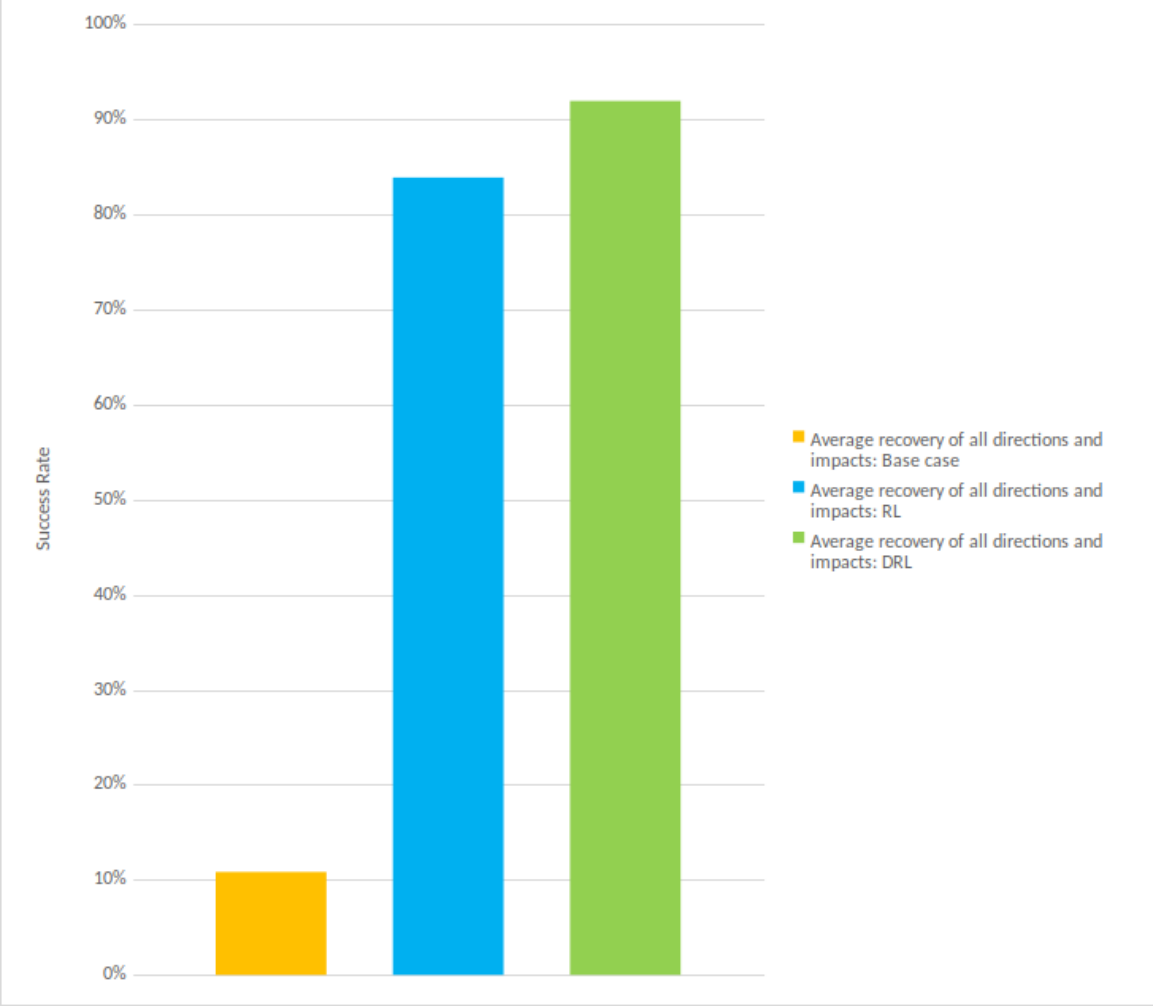


Figure 4.51: Comparison of my approaches vs the base case

4.7.5 Environment: Simulation - Turf Surface

This section describes the results I obtained from my experiments in simulation on the simulated soccer field (artificial turf). This involves the comparison of the results between reinforcement learning, deep reinforcement learning, and the base case. In the following subsections, first, the results of RL are discussed. Second, I discuss the results obtained from testing my deep reinforcement learning approach.

Third, I discuss the results of the base case. Finally, I describe the performance of my approach versus the base case.

Figure 4.52 shows a successful recovery from a strong push (50 CM). The force was applied to the front side of Polaris. Figures 4.53 - 4.55 show the entire simulation environment during a trial test.

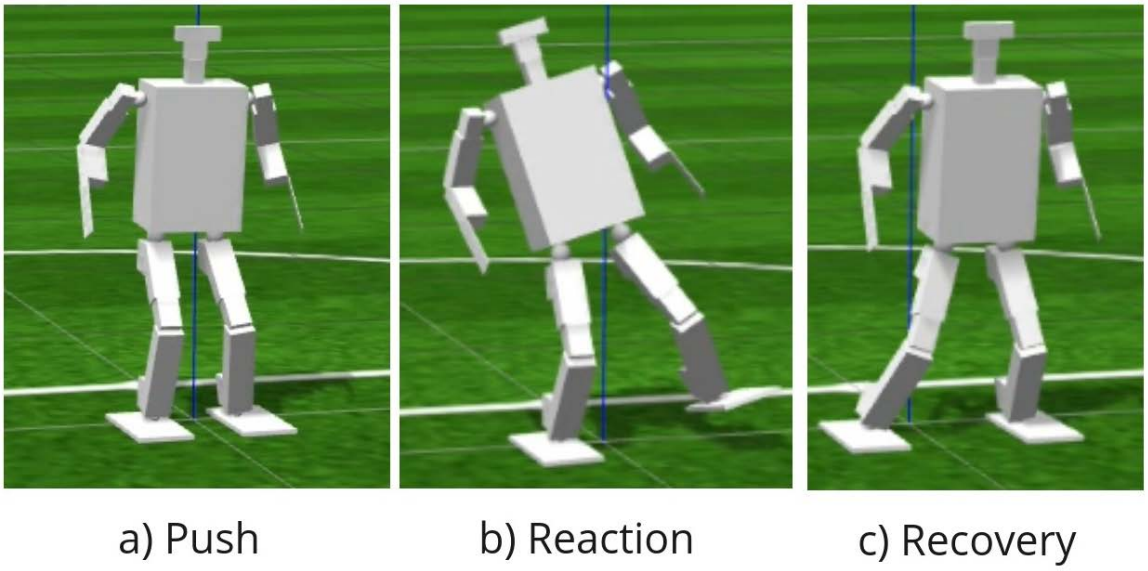


Figure 4.52: Push, Reaction, Recovery

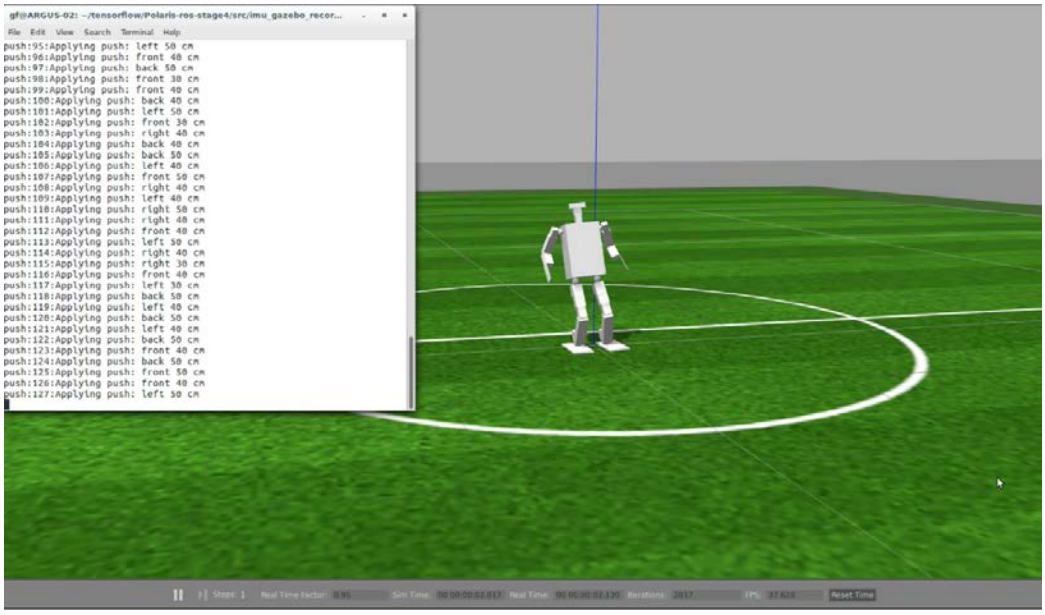


Figure 4.53: **Push**, Reaction, Recovery in the simulation environment on turf.

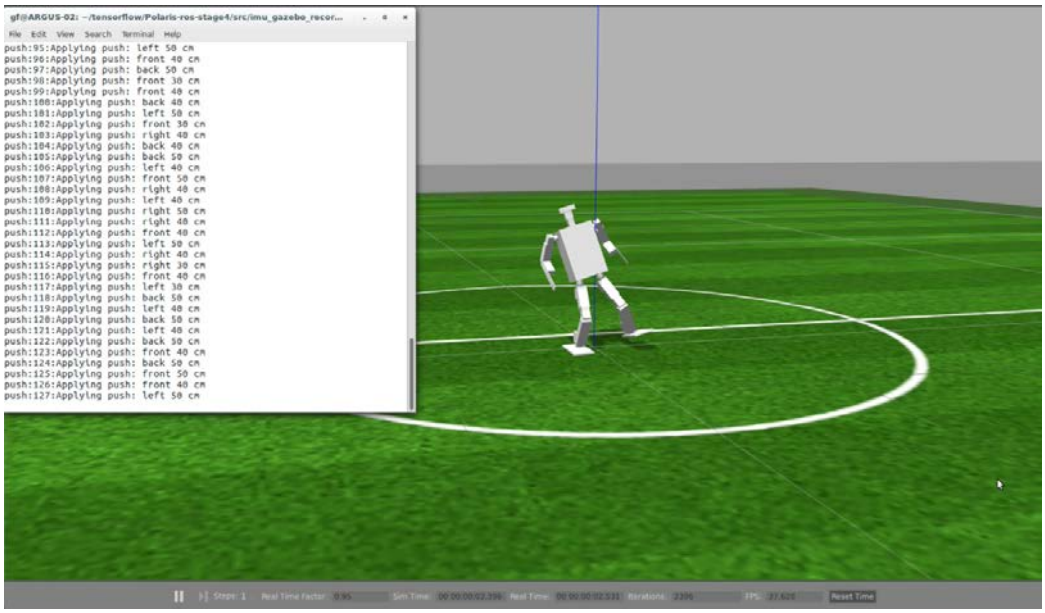


Figure 4.54: Push, **Reaction**, Recovery in the simulation environment on turf.

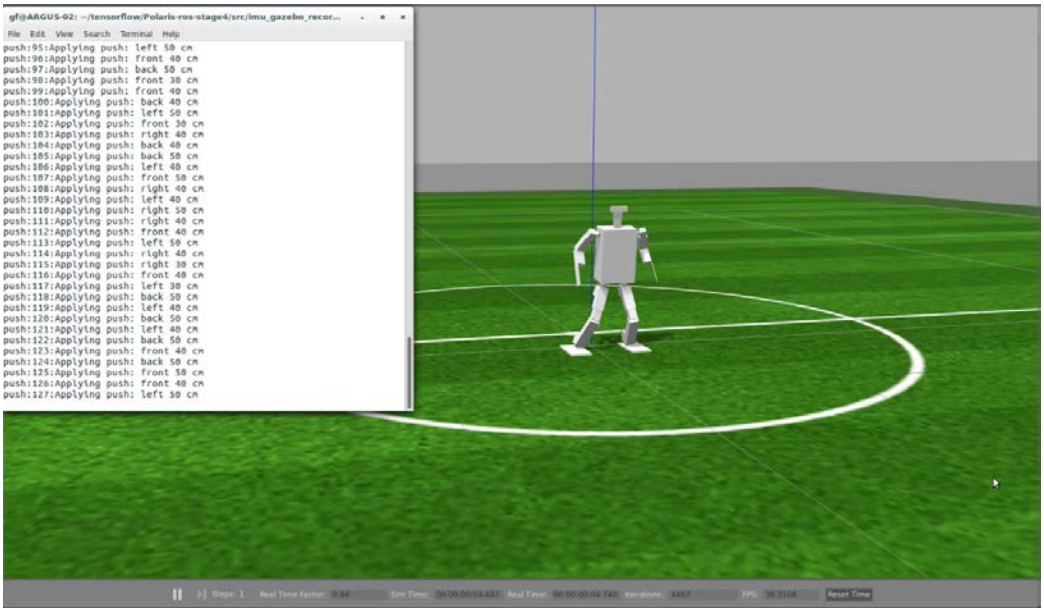


Figure 4.55: Push, Reaction, **Recovery** in the simulation environment on turf.

4.7.5.1 Approach: Reinforcement Learning

In this section I discuss the results of 360 pushes (12 * 30) on the simulated turf surface using my reinforcement leaning approach.

The results of all trials related to this experiment are shown in Table 4.13. For each case in which the robot successfully recovered from the impact, a score of 1 was given, otherwise no score was granted. Figure 4.56 shows the summarized results of my approach using RL. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The blue, orange, green, and purple columns represent 30 CM pushes, 40 CM pushes, 50 CM pushes, and total success rate over all directions in percentage form, respectively.

Turf surface: RL		Successful attempts			Total number of successful attempts by directions
Distance in centimeters		30	40	50	Out of 90
Directions	Back	23	25	25	73
	Right	23	21	23	67
	Front	26	26	26	78
	Left	21	24	26	71

Table 4.13: Results of trials for the experiment in the simulation on turf using RL.

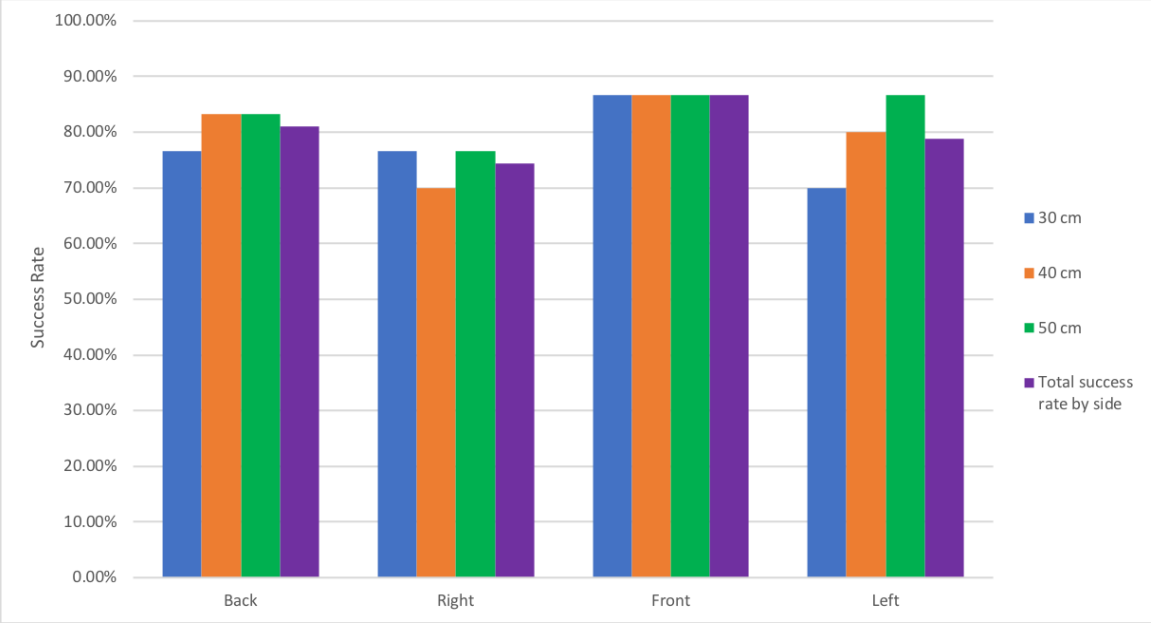


Figure 4.56: Experimental results in the simulation using reinforcement learning on simulated turf

My closed-loop control mechanism using the RL approach was able to recover from pushes with an average success rate of 78% for all of the directions at the low impact level (30 CM swing). Polaris was able to recover 87% of pushes received from the Front, 77% from the Back and Right, and 70% from the Left.

At the medium impact level (40 CM swing), the recovery average result was 80%,

with 83% from the Back, 70% from the Right, 87% from the Front, and 80% from the Left recovered successfully.

At the strong impact level (50 CM swing), the average success rate of all directions is 83%, with 87% of the pushes from the front and left sides, 83% of the pushes from the Back side, and 77% from the Right side recovered successfully.

The average success rate of recovery, based on the impact level, using RL approach is shown in figure 4.57.

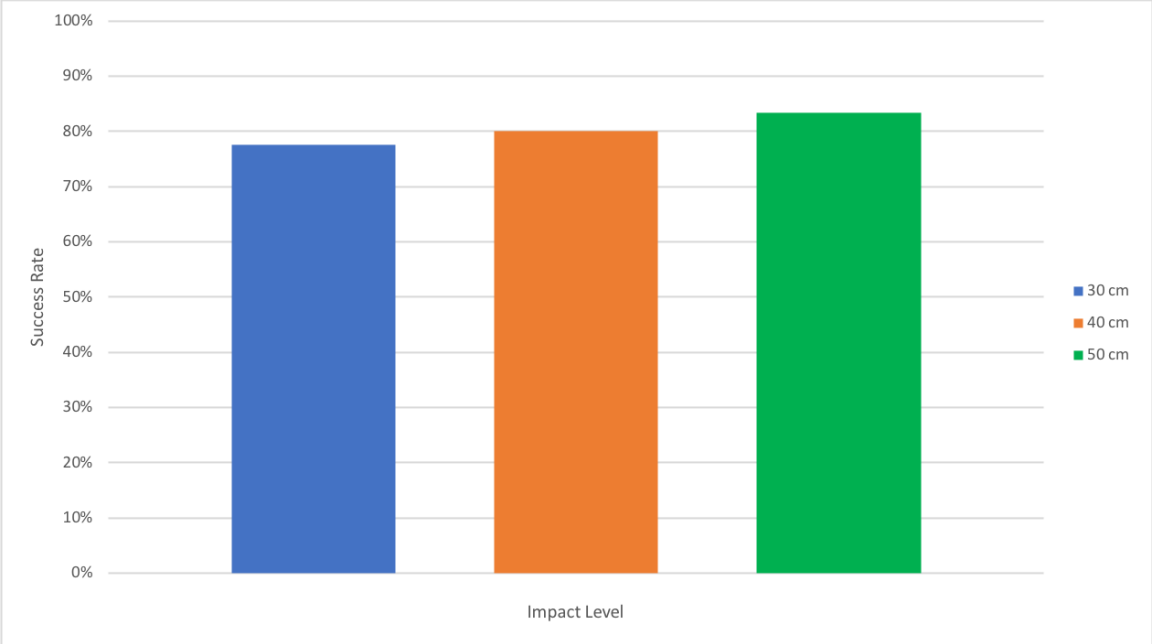


Figure 4.57: Average success rate of recovery based on the three impact levels using RL on simulated turf.

4.7.5.2 Approach: Deep Reinforcement Learning

In this section I discuss the results of 360 pushes (12 * 30) on the simulated turf surface using my deep learning approach. All the actions were the results of using

my deep neural network as the reward function.

The results of all trials related to this experiment are shown in Table 4.14. For each case in which the robot successfully recovered from the impact, a score of 1 was given, otherwise no score was granted. Figure 4.58 shows the summarized results of my approach using DRL. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The blue, orange, green, and purple columns represent 30 CM pushes, 40 CM pushes, 50 CM pushes, and total success rate by each direction in percentage form, respectively.

Turf surface: DRL		Successful attempts			Total number of successful attempts by directions
Distance in centimeters		30	40	50	Out of 90
Directions	Back	25	28	29	82
	Right	30	25	28	83
	Front	29	27	28	84
	Left	30	27	26	83

Table 4.14: Results of trials for the experiment in simulation on turf using DRL.

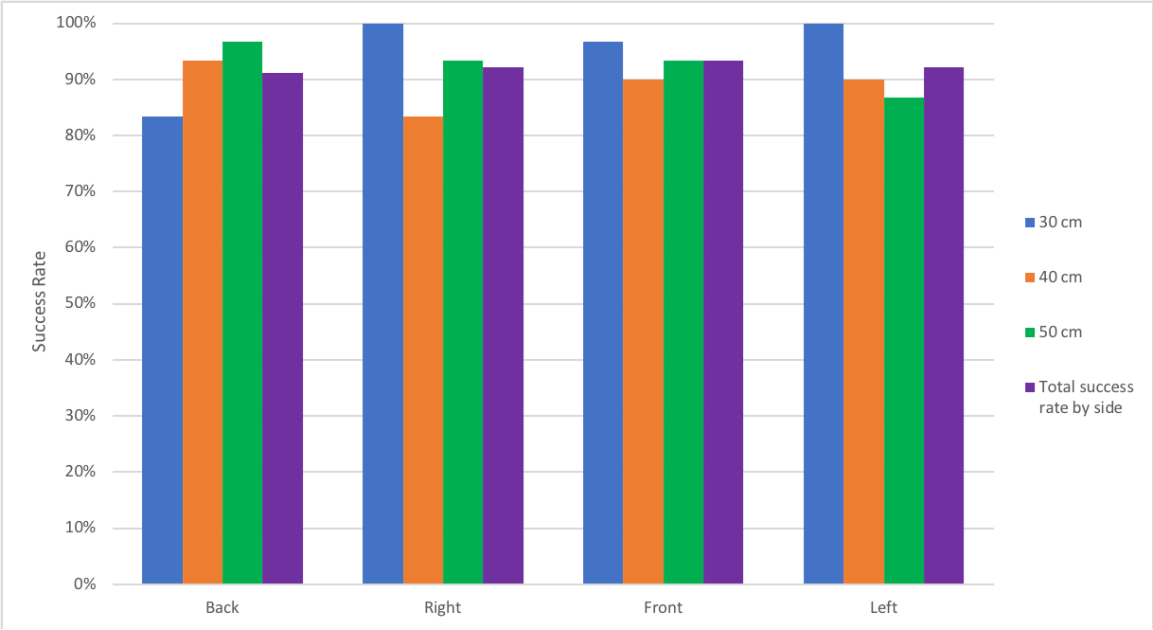


Figure 4.58: Experimental results in simulation using deep reinforcement learning on simulated turf

Similar to RL (Section 4.7.5.1), my closed-loop control mechanism using DRL approach was able to recover from a strong majority of pushes, with an average success rate of 95% for all of the directions at the low impact level (30 CM swing). Polaris was able to recover from 100% of pushes that were received from the Right and Left, 97% from the Front, and 83% from the Back.

At the medium impact level (40 CM swing), the recovery average result was 89% recovery for all the pushes over all directions. It recovered from 93% of the pushes from the Back side, 90% from the Front and Left sides, and 83% from the Right side.

At the strong impact level (50 CM swing), the average success rate over all directions is 93%, with 97% of the pushes recovered from the Back, 93% from the Right and Front sides, and 87% of the pushes from the Left side recovered.

The average success rate of recovery, based on the impact level, using the DRL approach is shown in Figure 4.59

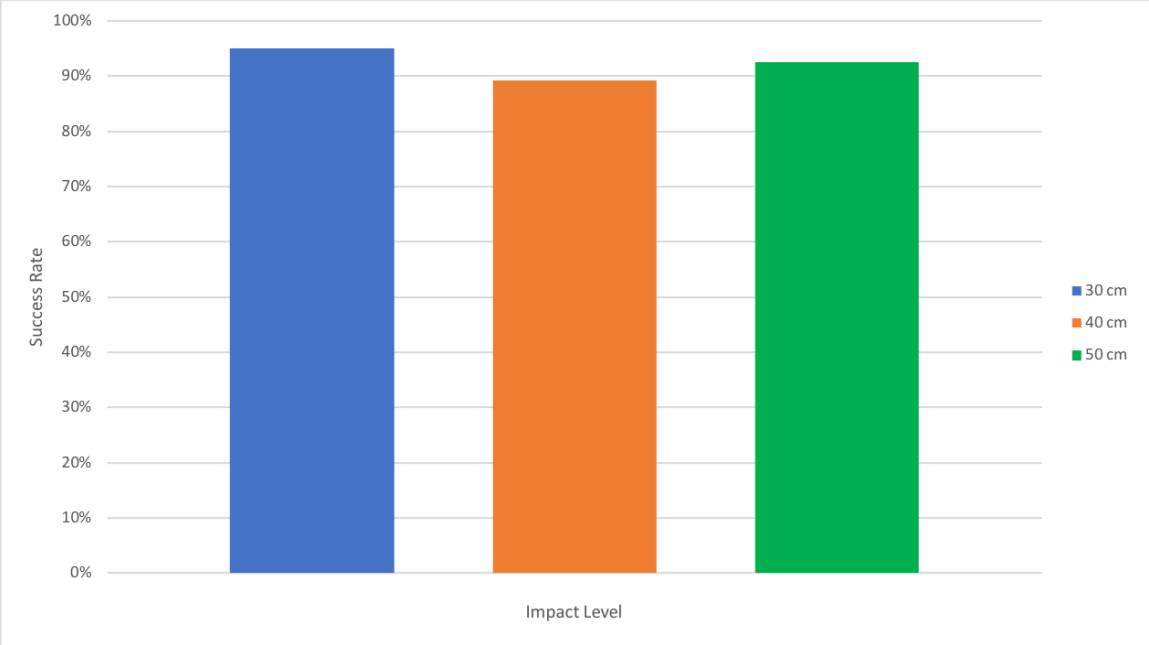


Figure 4.59: Average success rate of recovery based on the three impact levels using DRL on turf.

4.7.5.3 Approach: Base Case

In this section I discuss the results of 360 pushes (12 * 30) on the simulated turf surface using the base case.

The results of all trials related to this experiment are shown in Table 4.15. For each case in which the robot successfully recovered from the impact, a score of 1 was given, otherwise no score was granted.

Figure 4.60 shows the summarized results of the base case approach using the closed-loop feedback control and taking random actions in the walking engine's pa-

rameter thresholds. In this figure, all the pushes are grouped in four different directions (Back, Right, Front, Left). There are four columns in each direction. The blue, orange, green, and purple columns represent 30 CM pushes, 40 CM pushes, 50 CM pushes, and total success rate in each direction in percentages, respectively.

Turf surface: Base case		Successful attempts			Total number of successful attempts by direction
Distance in centimeters		30	40	50	Out of 90
Directions	Back	1	1	2	4
	Right	1	1	3	5
	Front	3	5	3	11
	Left	1	3	2	6

Table 4.15: Results of trials for the experiment in simulation on turf, using the base case.

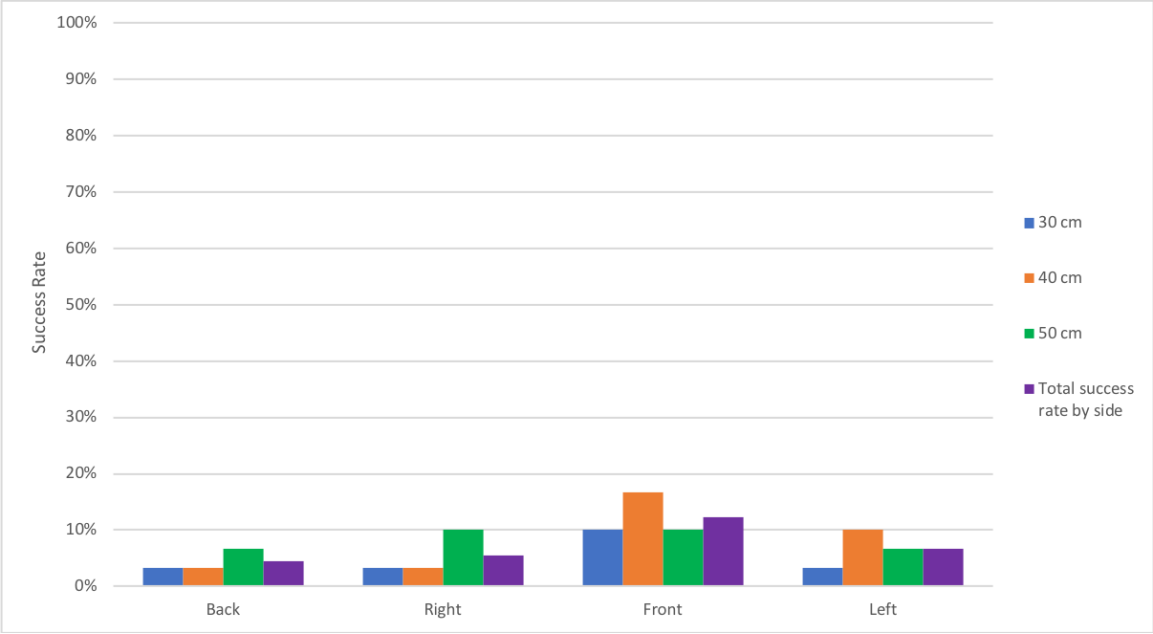


Figure 4.60: Experimental results in the simulation using random actions between the threshold on simulated turf.

The average recovery success rate using the base case is 5% for all of the directions at the low impact level (30 CM swing). Polaris was able to recover from 10% of the Front pushes, and 3% of all of the pushes that were applied to other sides.

At the medium impact level (40 CM swing), the average recovery results was 8%, with 3% from the Back and Right, 17% from the Front and 10% from the Left.

At the strong impact level (50 CM swing), the average success rate over all directions is 8%, with 7% recovery from the Back and Left, and 10% from the Right and Front.

The average success rate of recovery, based on the impact level, using the base case approach is shown in Figure 4.61.

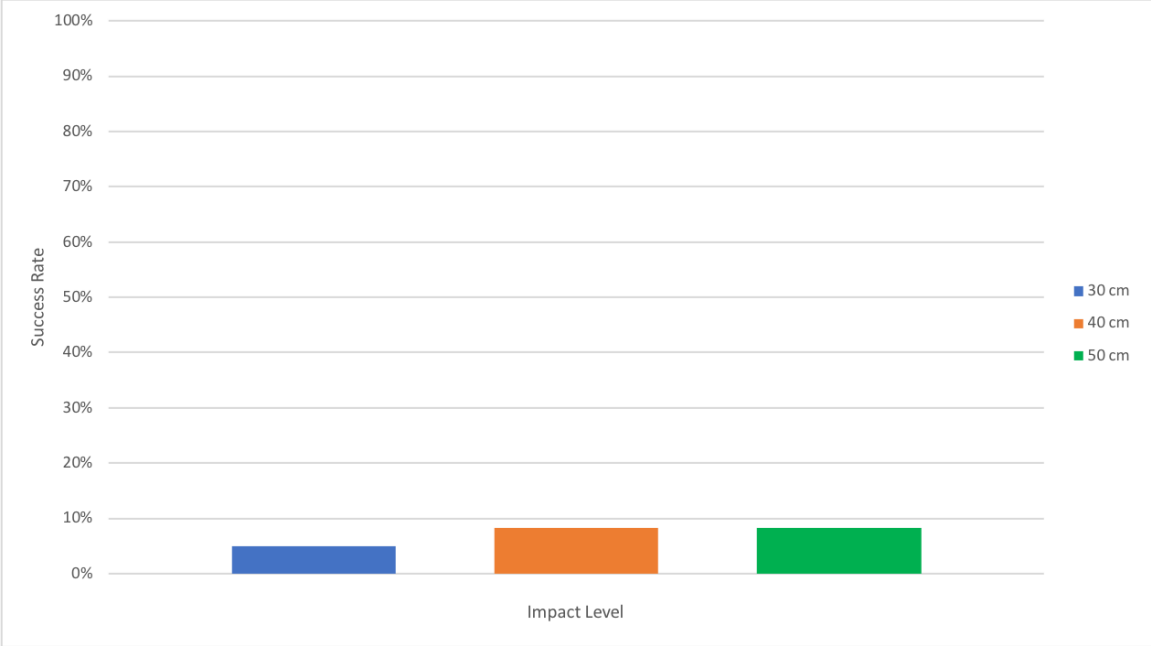


Figure 4.61: Average success rate of recovery based on the three impact levels using the base case on a simulated turf surface.

4.7.5.4 Summary

Figures 4.62 and 4.63 show the difference between the performances of my two approaches RL (Section 4.7.5.1) , DRL (Section 4.7.5.2), and the base case (Section 4.7.5.3).

Final results show that Polaris was able to recover from 81% and 91% of the pushes from the Back side, using RL and DRL approaches respectively. However, the recovery rate for the base case from the same side was 4%.

On the Right side, by using RL, Polaris could recover 74% of the pushes. DRL performed very well with the total success rate of 92%. The recovery for the base case was 6%.

When using the RL approach, Polaris was able to recover from 87% of the pushes from the Front. Using DRL makes the recovery results better, at 93%. For the base case, the success rate was 12%.

Finally, from the Left side, the RL approach led to 79%, DRL 92%, and the base case 7% total success recovery.

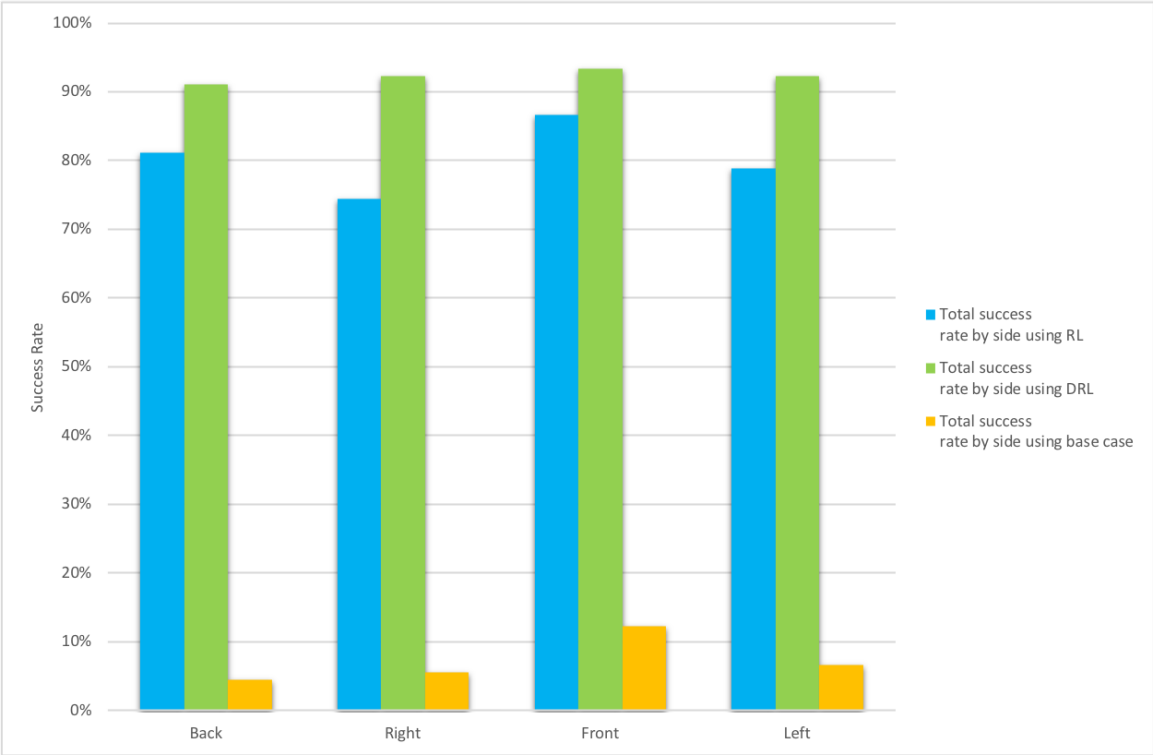


Figure 4.62: Comparison of my approaches vs the base case

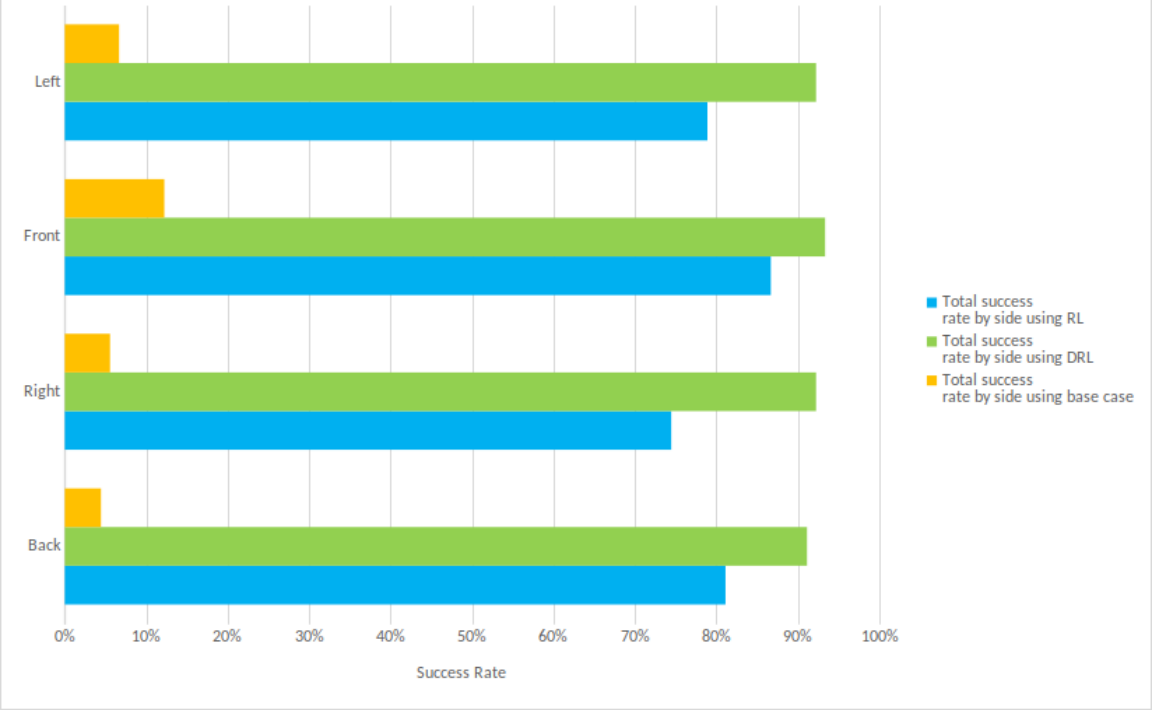


Figure 4.63: Comparison of my approaches vs the base case

The average recovery rate over all directions and impacts for the base case was 7%, improved by my RL approach (80%). Finally DRL, with 92% successful recovery. Figure 4.64 shows this comparison.

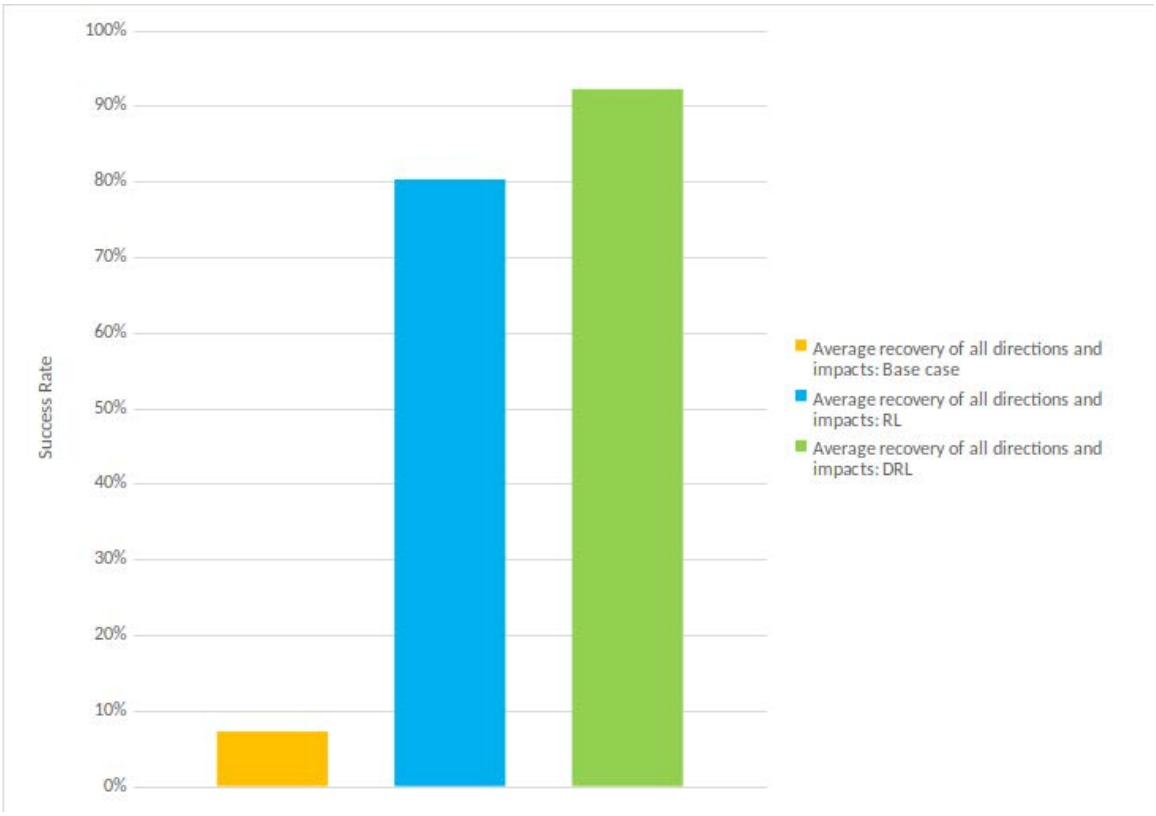


Figure 4.64: Comparison of my approaches vs the base case

4.8 Summary

This chapter provided an overview of the experimental environments that I used to evaluate my approach and implementation in both real world and simulation. Based on the experimental results, my RL and DRL approaches performed better, by a large margin, than the base cases for every surface in both simulation and the real world.

For the real world experiments, the results show that the best surface in which my approaches outperformed is the concrete surface. This is due to two main reasons, 1) all of the actions were learned on the concrete in the simulation during the learning

process. 2) Concrete is the stiffest surface and it is easier to recover on such a surface. The second easiest surface among the three is carpet. Finally, the hardest surface for recovery is the artificial turf. Figures 4.65 and 4.66 compare the results of RL, DRL, and the base case on three different surfaces based on four different directions that the forces were applied.

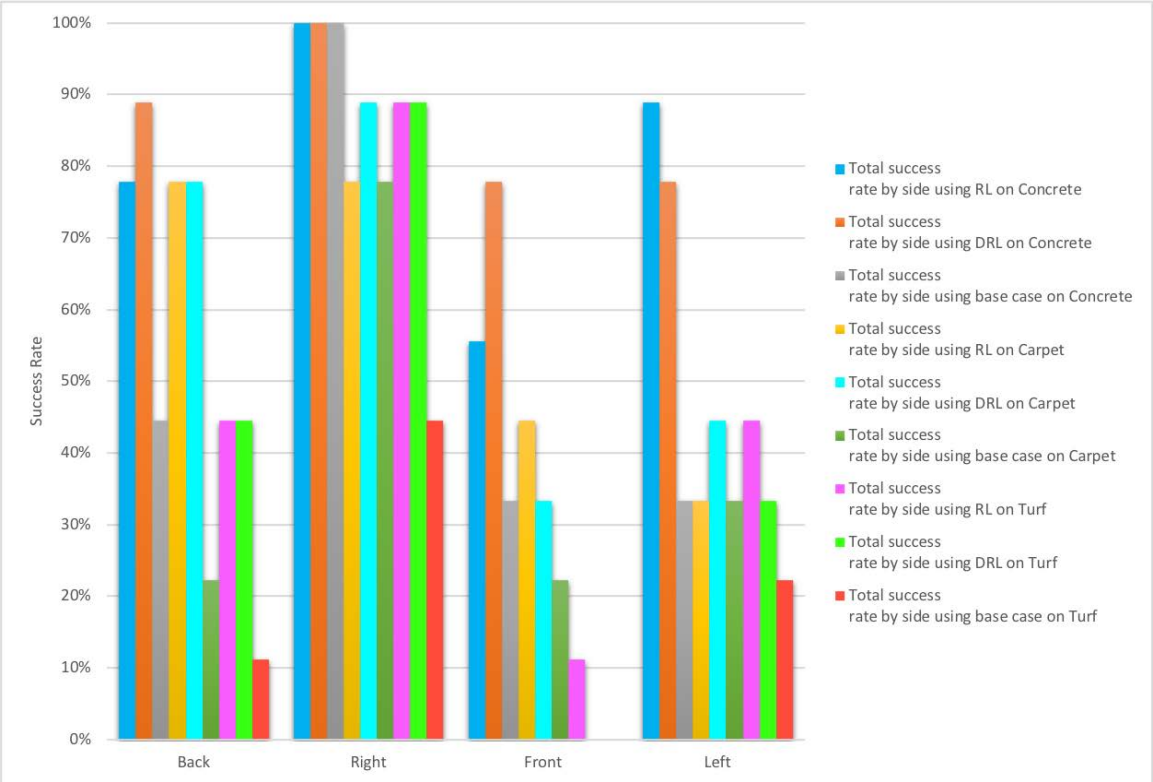


Figure 4.65: Performance comparison of all approaches on all surfaces in the real world

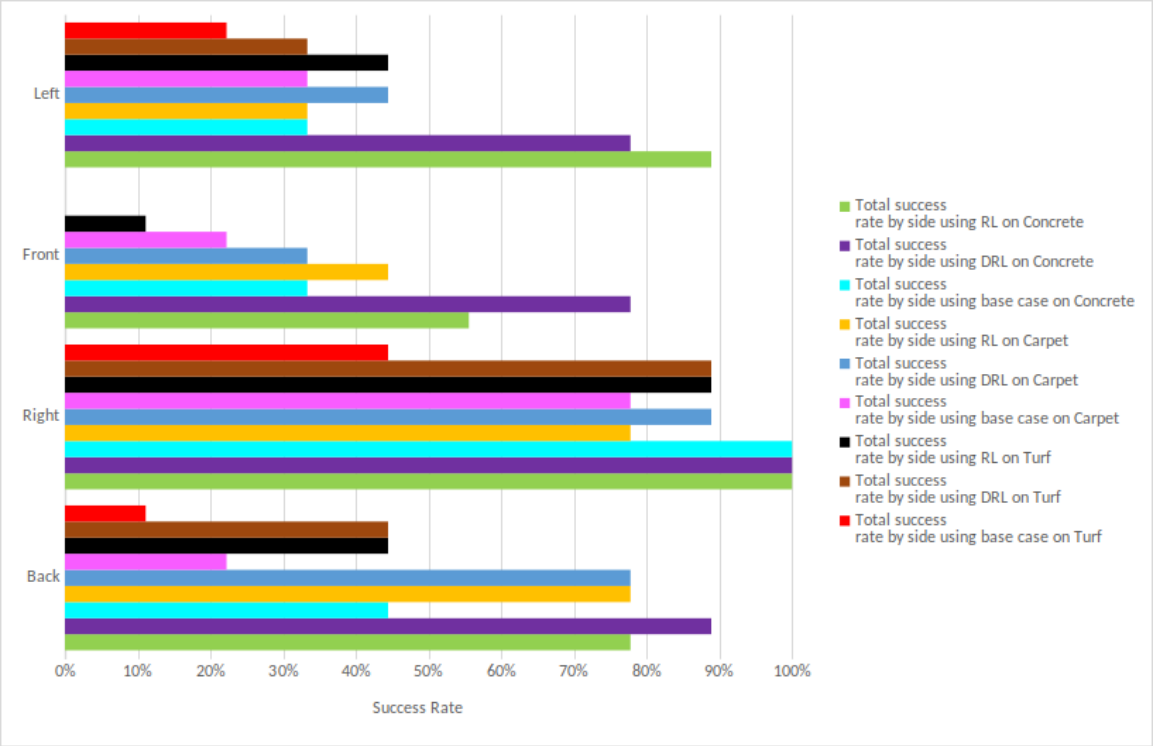


Figure 4.66: Performance comparison of all approaches on all surfaces in the real world

For the simulation experiments, the results show that the best surface on which my approaches performed is the concrete surface, which is similar to the real world experiment. Artificial turf is the next more difficult surface in simulation. Figures 4.67 and 4.68 compare the results of RL, DRL, and the base case on these two different surfaces.

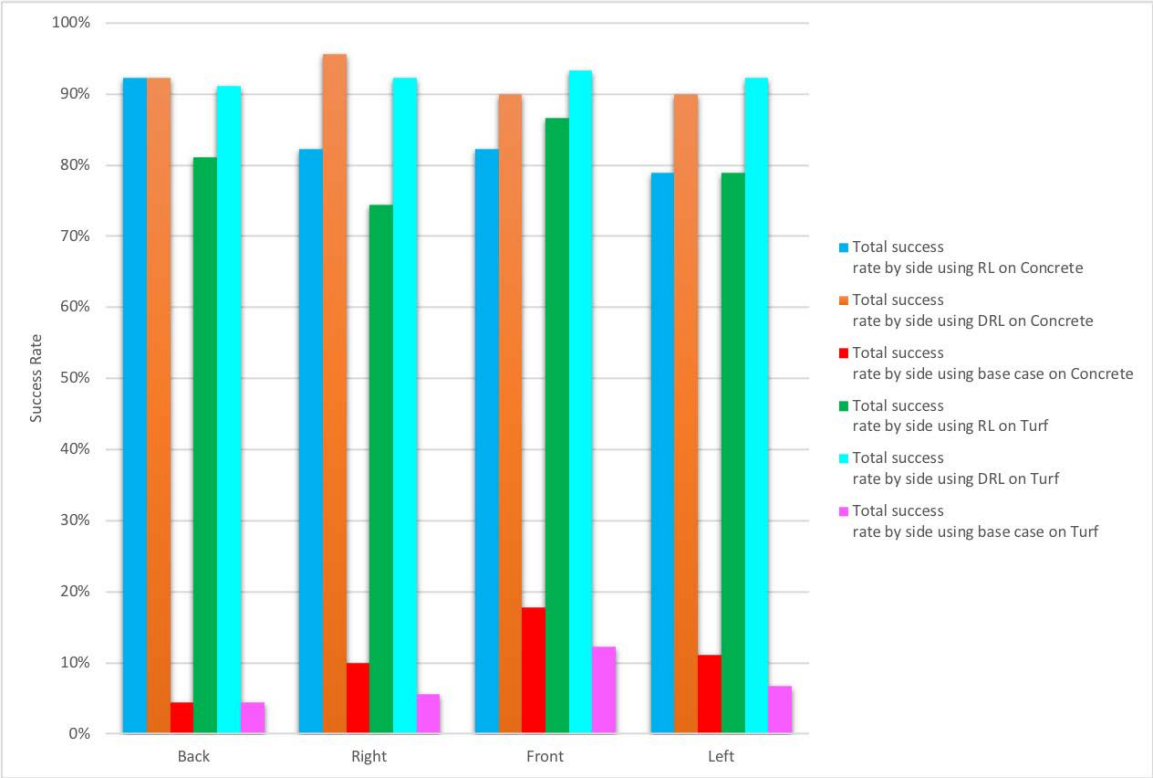


Figure 4.67: Performance comparison of all approaches on all the surfaces in the simulation environment

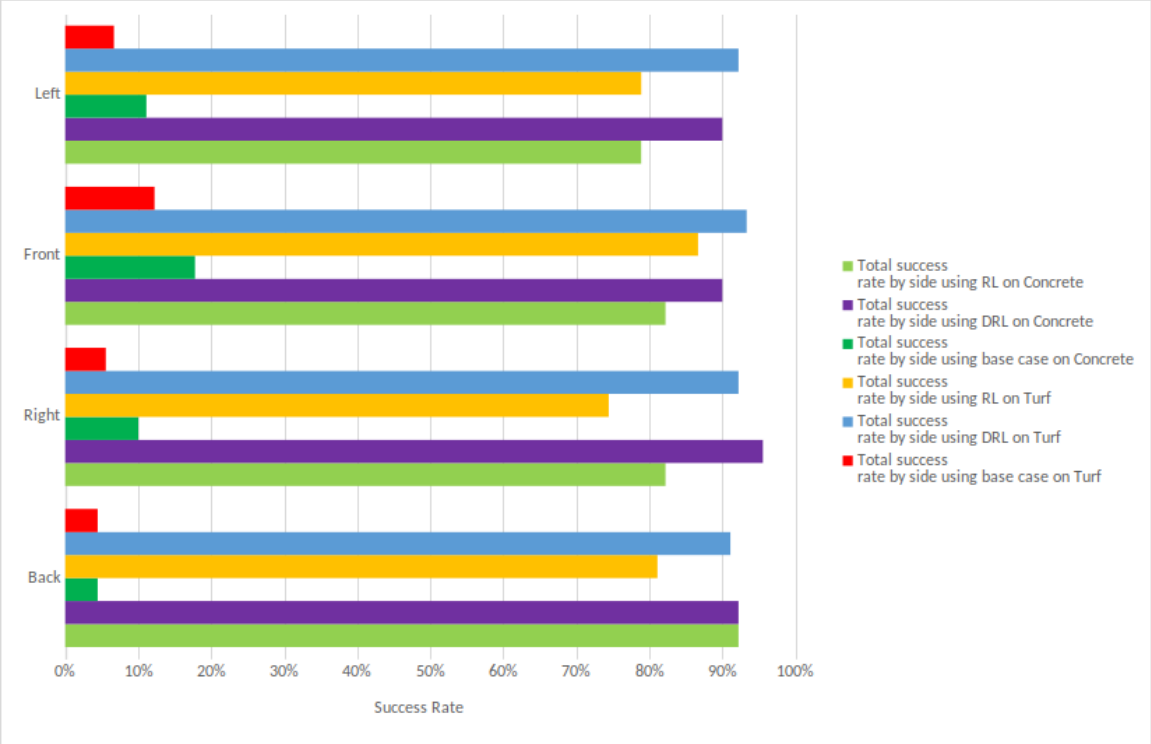


Figure 4.68: Performance comparison of all approaches on all the surfaces in the simulation environment

These experiments highlighted that my approach and implementation have advantages over the base case in both the real world and in simulation.

Chapter 5

Conclusion

5.1 Overview

I begin this chapter by discussing the results from Chapter 4 and how these answer my research questions (Section 5.2). I provide an outline of the contributions of my thesis in Section 5.3. Finally, I discuss future research directions of my research work in Section 5.4.

5.2 Answers to Research Questions

I discussed my research questions in Section 1.3. In this section, I will review and answer those questions based on the results described in Chapter 4.

1. **Is my closed-loop control going to be fast enough for inexpensive robots that are equipped with cheaper hardware for responding to an external push?**

To answer this question, I must only refer to the results that I achieved from the real world. Dealing with the robot in the real world is a very complex task compared to a simulation [Hotz and Gómez, 2004]: if the robot does a task perfectly in the virtual world, it does not necessarily mean that it will do the same in reality. Sensor noise, servo motor jitter and voltage oscillation (battery issues) are just a few examples of problems that are common in the real world. This way all the factors in the real world that could affect the robot are encountered. The results show that Polaris, a cheaper version of a humanoid robot compared to the robots that were discussed in Chapter 2, can successfully recover from different pushes with various impacts, including strong pushes, as well as pushes from different directions. This demonstrates that my control loop mechanism is responsive enough for the real world.

2. **Are any of my reinforcement learning and deep reinforcement learning push recovery techniques going to be able to replace the parameters that a human operator provides (hand-tuned) for recovery based on his/her experiences?**

   As the results show, both reinforcement learning and deep learning performed well enough to be a replacement to human knowledge. Through more training, the robot is able to learn better action choices, which are more likely to lead the robot to a successful recovery. This is very similar to the experience of an operator whose knowledge and experience are transferred to the robot. The more experience the operator has, the better parameters can be given to the robot.

3. **Which of the proposed approaches (RL, DRL), results in better recovery if the surfaces, directions, and impacts are unknown to the robot (i.e. with no prior knowledge)**

   I categorize the answer to this question into three categories: 1) real world, 2) simulation, 3) overall, considering both real world and simulation.

   In the real world, RL achieved 62% overall recovery success rate while DRL 63% (on concrete RL achieved 81%, DRL 86%. On carpet RL achieved 58%, DRL 61%. On turf RL achieved 47%, DRL 42%).

   In simulation, RL achieved 81.5% overall recovery success rate while DRL achieved 92% (on concrete: RL 84%, DRL 92%. On turf: RL 79%. DRL 92%.).

   In the real world and simulation, RL achieved 69.8% recovery success rate while DRL achieved 74.6%. In all the categories, DRL performed better in comparison to RL.

The results also show that in the simulation, there is not much difference between concrete and turf in comparison with the real world. Based on my observation, objects in the real world cannot be demonstrated identically in simulation. There are many factors that contribute to these differences, e.g., friction and texture of the surface (concrete, carpet, turf). I experimented by altering the friction value in the simulation to some high and low arbitrary numbers. However, the result showed that the differences between the frictions is not significant. For that reason, I continued my experiments with the default surfaces that are provided with the simulation.

5.3 Contributions

This research is concentrated on active balancing and push recovery for an inexpensive humanoid robot. This thesis makes a number of important contributions to the state of the art in push recovery and active balancing:

- My main contribution is a new approach to push recovery based on learning appropriate step responses using RL and DRL. It is important to note that this approach is specifically intended for inexpensive humanoid robots that use more fragile, low-torque, limited-power servos with low computational power. No research in the literature provides a solution that recovers from small, medium, and large disturbances for inexpensive robots. My methods were also tested in the real world using a physical robot, as opposed to other methods examined only in simulation or in restricted settings such as boom mounts.

- As part of my work I have developed and validated an extensive 3D model of Polaris in Gazebo. There are no existing Gazebo models of inexpensive humanoid robots, and only a few 3D models of expensive humanoids. This will further future research on inexpensive humanoids in simulation. In my work I used this model to train the RL and DRL approaches outlined in Chapter 3.

- I introduced a novel way of using RL and stepping for push recovery. This generates the walking engine parameters and uses these for push recovery of the humanoid robot. These parameters then will be used in the inverse kinematics model to generate a trajectory of a step. I replaced all the hand-tuned walking engine parameters, to support better push recovery.

- I also introduced a new approach in DRL that learns how good or bad a chosen action is. If the reward of an action is ≥ 90, the robot will invest in that action and try to learn more about it. Otherwise, it does not waste time on it.

In addition, there are a number of contributions made through my implementation choices. For example, in order to make my program compatible with many different available hardware and software modules, I have made my walking engine and the push recovery compatible with the ROS platform. If other researchers use ROS, they can also add my modules easily to their programs. Other researchers would be able to adapt my active balancing and push recovery techniques to their inexpensive humanoid robots as long as the robot's walking engine has the listed parameters in Table 3.4 (these parameters are very common in every walking engine). Also, depends on the robot's height the range of some of these parameters (Table 3.4) needs to be altered. For example, if the length of the lower part of the robot's body is 25 CM, the threshold for the z-offset needs to be altered to a range such as Minimum:18 CM, Maximum 25 CM. There might be a need to change some other threshold for other parameters such as Step-x, Step-y. The minimum and maximum values of these parameters can be chosen based on trial and error. Because it also depends on the structure of the links on the robot's body. However, if a walking engine does not have or have extra one or more parameters (Table 3.4) these parameters can be removed or added to the learning module respectively.

5.4 Future Work

Although the evaluation of my research has successfully highlighted the benefit of my approach, my research and implementation also raise a number of areas in which improvement could be made.

As future work, I would like to improve the design of my approach by including the direction of the walking gait (forward or backward) and simplifying it by using symmetry between the left and right side of the robot. These two additional features might reduce the learning process. This could be extended by employing the ZMP method [Kajita et al., 2003] for the walking engine along with stepping, using RL and DRL.

As future work, I would like to use the newly-trained RL component to generate a larger dataset for training the DNN, and possibly improve performance further.

Another interesting direction of future work would be to have a deep neural network that outputs walking engine parameters. This does not necessary mean that we do not need a deep neural network for the reward function. After generating a set of walking engine parameters, this set could be the input to a DNN and its output would be an altered (tuned) version of the given input. Also, I would like to investigate other types of neural networks for my framework, such as using a Long short-term memory (LSTM) neural network [Sak et al., 2014; Hochreiter and Schmidhuber, 1997]. One of the strengths of LSTM network is that it is well suited for classification areas.

Another interesting direction of future work would be to replace the look-up tables that I used for RL and DRL with DNNs. Algorithms such as Deep Q-Learning (DQN) [Mnih et al., 2015], Double Deep Q-Learning (DDQN) [Van Hasselt et al., 2016],

Dueling Q-Learning [Wang et al., 2015], and Asynchronous Advantage Actor Critic (A3C) [Mnih et al., 2016] would be able to replace the look-up tables. However, the main important factor for employing any of the above algorithms is a need for powerful Graphical Processor Units (GPUs) with a lot of Random Access Memory (RAM). For my research since the system is designed for inexpensive humanoid robots, I did not use any GPU nor lots of RAM. The total amount of RAM that the robot had access to was 4GB. The robot used 4GB for the operating system, simulation software, and the learning part. Table 3.2 lists the robot hardware specification. With this specification it is almost impossible to replace the look-up tables. So it is essential to have a hardware upgrade.

The Triple Jump event [Baltes et al., 2019] from HuroCup [Fira, 2019] is a new research challenge that was introduced in 2018. This event is related to dynamic balancing of a humanoid robot. Generating an appropriate set of steps with enough force to lift off the ground and let the robot land carefully enough to avoid breaking servos is a very hard task, especially for an inexpensive robot. As another line of future work, my framework can be extended to be used for the triple jump research and the generation of the necessary steps.

Another interesting direction for future research would be to design different 3D models of surfaces in the simulation and let the robot learn how to walk on outdoor surfaces. Gravel, snow, and mud, are the examples of common outdoor surfaces. Although walking on such surfaces is very challenging for an inexpensive robot, I believe my framework could be extended to teach the robot to perform well under such conditions. As of now, my simulation environment does not provide many different

models of the ground with different textures and frictions.

In my current implementation, I have assigned a relatively large range of possible values for the nine walking engine parameters. The bigger the range is, the more time the robot requires to explore and exploit. The main issue with this is that when using a computer with low computational power and memory space, many of the possible actions might not be explored with the current ranges. Another possible improvement for the future implementation is to start the learning process with the smaller range.

As I mentioned earlier (Section 3.4.3.1), I did not let the robot know what surface it is walking on during all the experiments.. This makes the push recovery problem a lot harder, since a set of generated actions must be suitable for all the surfaces. In Chapter 4, the final results show that among the three surfaces, artificial turf is the most challenging surface for push recovery, and the total recovery percentage was lower compared to concrete and carpet, respectively. Based on my observation, one of the main challenges for the robot to recover on this surface is the surface thickness. The thickness of the artificial turf surface that was used during my experiments was 4 CM to 5 CM. As was discussed in Section 3.4.2, Table 3.4, the minimum and maximum values for the step-height are 1 CM and 8 CM, respectively. Many of the generated steps, had the step-height between 1 CM and 5 CM. This thickness does not allow the majority of the good actions that had the range for the step-height between 1 CM and 5 CM, to be successful on the turf. Based on the final results, this range of step-height was suitable for the concrete surface. Another improvement that could be applied to my implementation is to change the possible step-height range from 1 CM to 8 CM, to 5 CM to 8 CM. By changing the minimum value to 5 CM,

the performance would be improved on both carpet and artificial turf surfaces.

Beyond improving technique, there is also future work applying the results from this thesis. The Spartan race challenge [Baltes and Tu, 2019] from HuroCup [Fira, 2019], is another robotics benchmark that is designed for humanoid robots. This event was introduced in 2017 to open a wide area of research, e.g., active balancing, push recovery, complex motion planning. This event consists of three challenges: 1) lift and carry 2) ladder climbing 3) rope climbing. In the lift and carry challenge, there are uneven terrains that the robot has to traverse without falling. This challenge in the Spartan race could be another direction of future research, and my framework could be extended for learning the walking steps.

5.5 Conclusion

Push recovery and active balancing of humanoid robots are important ongoing research topics in robotics. There is no perfect solution to all these problems even on expensive equipment, and the challenges are more significant for inexpensive humanoid robots that use more fragile, low-torque, limited-power servos along with low computational power. In this Ph.D. research, I have described the design, implementation, and evaluation of a novel push recovery approach using reinforcement learning and deep reinforcement learning [Yi et al., 2011] using stepping on an inexpensive 20-DOF humanoid robot.

To begin with, I implemented a parameterized walking engine that allows the robot to walk. Next, I implemented a hand-tuned closed-loop control that is coupled with the walking engine for recovering from external pushes. Meanwhile, I implemented

the stepping approach and added this functionality to the walking engine. With this closed-loop control, the robot was able to recover from small, medium, and large external disturbances.

My push recovery mechanism has also been demonstrated in the field with strong success. This mechanism for push recovery was our mechanism for entering the RoboCup Humanoid league technical challenge push recovery event (this event varies by year and normally has more than one challenge with scores totalled). We were awarded first place for this challenge in 2016 and 2018, came second in 2015, and had the highest results in the push recovery component in 2017. Recovery from unexpected impacts is an important part of many activities, and this approach also forms a core within our entries to other robotic sporting events such as the 2016, 2017, and 2018 FIRA HuroCup. Also, I have designed and implemented an extended version of my hand-tuned closed-loop control that uses machine learning techniques for making the push recovery fully autonomous.

Finally, my push recovery closed-loop feedback control mechanism was published in [Hosseinmemar et al., 2018] and I won the second best award among 146 accepted papers. I have also published a follow-up journal paper for push recovery for inexpensive humanoid robots [Hosseinmemar et al., 2019].

Bibliography

T. Assman, H. Nijmeijer, A. Takanishi, and K. Hashimoto. Biomechanically motivated lateral biped balancing using momentum control. *Eindhoven : Eindhoven University of Technology, Traineeship report. - DC 2011.035*, 2012.

J. Baltes and K.-Y. Tu. Hurocup laws of the game spartan race (pro), 2019. URL `https://goo.gl/fR15RM`.

J. Baltes, C. Iverach-Brereton, and J. Anderson. Human inspired control of a small humanoid robot in highly dynamic environments or jimmy darwin rocks the bongo board. In *Robot Soccer World Cup*, pages 466–477. Springer, 2014.

J. Baltes, K. Anhel Camarillo Gomez, and K.-Y. Tu. Hurocup laws of the game triple jump, 2019. URL `https://goo.gl/sQnQcd`.

G. Bebis and M. Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13 (4):27–31, Oct 1994. ISSN 0278-6648. doi: 10.1109/45.329294.

S. Behnke. Online trajectory generation for omnidirectional biped walking. *Proceedings - IEEE International Conference on Robotics and Automation*, 2006(May): 1597–1603, 2006. ISSN 10504729. doi: 10.1109/ROBOT.2006.1641935.

J. A. Boyan and A. W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pages 369–376, 1995.

A. Buja, W. Stuetzle, and Y. Shen. Loss functions for binary class probability estimation and classification: Structure and applications. *Working draft, November*, 3, 2005.

S. R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, 17 (1-19):16, 2004.

A. R. Cassandra. Exact and approximate algorithms for partially observable markov decision processes, 1998.

T. Chai and R. R. Draxler. Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific model development*, 7(3):1247–1250, 2014.

G. C. Chow and A.-l. Lin. Best linear unbiased interpolation, distribution, and extrapolation of time series by related series. *The review of Economics and Statistics*, pages 372–375, 1971.

J. J. Collins and C. J. De Luca. Open-loop and closed-loop control of posture: a random-walk analysis of center-of-pressure trajectories. *Experimental brain research*, 95(2):308–318, 1993.

S. Cousins. ROS on the PR2. *IEEE Robotics and Automation Magazine*, 17(3):23–25, 2010. ISSN 10709932. doi: 10.1109/MRA.2010.938502.

P. Cunningham, M. Cord, and S. J. Delany. Supervised learning. In *Machine learning techniques for multimedia*, pages 21–49. Springer, 2008.

C. Dismuke and R. Lindrooth. Ordinary least squares. *Methods and Designs for Outcomes Research*, 93:93–104, 2006.

S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson. Optimization based full body control for the atlas robot. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 120–127. IEEE, 2014.

S. Ferrari and R. F. Stengel. Smooth function approximation using neural networks. *IEEE Transactions on Neural Networks*, 16(1):24–38, 2005.

Fira. Leagues - fira sports - hurocup, 2019. URL `http://www.firaworldcup.org/VisitorPages/Show.aspx?ItemID=2106,0`.

K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

Gazebo. Ros gazebo, 2017. URL `http://wiki.ros.org/gazebo`.

Gigabyte. Mini-pc barebone (brix), 2018. URL `https://www.gigabyte.com/Mini-PcBarebone/GB-BSi3H-6100-rev-10#ov`.

C. R. Gil, H. Calvo, and H. Sossa. Learning an efficient gait cycle of a biped robot based on reinforcement learning and artificial neural networks. *Applied Sciences*, 9 (3):502, 2019.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Google. Tensorflow, 2017. URL `https://www.tensorflow.org`.

C. Graf and T. Röfer. *A Center of Mass Observing 3D-LIPM Gait for the RoboCup Standard Platform League Humanoid*, pages 102–113. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-32060-6. doi: 10.1007/978-3-642-32060-6_9. URL `https://doi.org/10.1007/978-3-642-32060-6_9`.

S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

J. Han and C. Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pages 195–201. Springer, 1995.

W. Hardle and E. Mammen. Comparing nonparametric versus parametric regression fits. *The Annals of Statistics*, pages 1926–1947, 1993.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

A. Hofmann. Robust execution of bipedal walking tasks from biomechanical principles. 2006.

A. Hosseinmemar, J. Baltes, J. Anderson, M. C. Lau, and H. B. Wodi. Alternative control for teleoperation robots using a set-and-leave approach. In *International Conference on Advanced Humanoid Robotics Research (ICAHRR)*, 12 2016.

A. Hosseinmemar, J. Baltes, J. Anderson, M. C. Lau, C. F. Lun, and Z. Wang. Closed-loop push recovery for an inexpensive humanoid robot. In M. Mouhoub, S. Sadaoui, O. Ait Mohamed, and M. Ali, editors, *Recent Trends and Future Technology in Applied Intelligence*, pages 233–244, Cham, 2018. Springer International Publishing. ISBN 978-3-319-92058-0.

A. Hosseinmemar, J. Baltes, J. Anderson, M. C. Lau, C. F. Lun, and Z. Wang. Closed-loop push recovery for inexpensive humanoid robots. *Applied Intelligence. The International Journal of Research on Intelligent Systems for Real Life Complex Problems*, 49(173):1–14, 2019. ISSN 0924-669X (Print), 1573-7497 (Online). doi: 10.1007/s10489-019-01446-z.

P. E. Hotz and G. Gómez. The transfer problem from simulation to the real world in artificial evolution. In *Workshop and Tutorial Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX)*, pages 17–20, 2004.

Q. Huang, K. Yokoi, S. Kajita, K. Kaneko, H. Aral, N. Koyachi, and K. Tanie. Planning walking patterns for a biped robot. *IEEE Transactions on Robotics and Automation*, 17(3):280–289, 2001. ISSN 1042296X. doi: 10.1109/70.938385.

H. Ishiguro, T. Ono, M. Imai, T. Maeda, T. Kanda, and R. Nakatsu. Robovie: an interactive humanoid robot. *Industrial robot: An international journal*, 28(6): 498–504, 2001.

C. Iverach-Brereton, B. Postnikoff, J. Baltes, and A. Hosseinmemar. Active balancing and turning for alpine skiing robots. *The Knowledge Engineering Review*, 32, 2017.

C. J. Iverach-Brereton. Rocking the Bongo Board: Humanoid Robotic Balancing on Dynamic Terrain. Master's thesis, University of Manitoba, Winnipeg, Manitoba, Canada, 2015.

S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa. The 3d linear inverted pendulum mode: A simple modeling for a biped walking pattern generation. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 239–246. IEEE, 2001.

S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 1620–1626. IEEE, 2003. ISBN 0-7803-7736-2. doi: 10.1109/ROBOT.2003.1241826. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1241826`.

S. Kajita, M. Morisawa, K. Miura, S. Nakaoka, K. Harada, K. Kaneko, F. Kanehiro, and K. Yokoi. Biped walking stabilization based on linear inverted pendulum tracking. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 4489–4496. IEEE, 2010.

L. Kelmar and P. K. Khosla. Automatic generation of forward and inverse kinematics for a reconfigurable modular manipulator system. *Journal of Field Robotics*, 7(4): 599–619, 1990.

D. Kim, J. Lee, and L. Sentis. Robust dynamic locomotion via reinforcement learning and novel whole body controller. *arXiv preprint arXiv:1708.02205*, 2017.

J. Y. Kim, I. W. Park, and J. H. Oh. Walking Control Algorithm of Biped Humanoid Robot on Uneven and InclinedFloor. *Journal of Intelligent and Robotic Systems*, 48:457–484, 2007. doi: 10.1007/s10846-006-9107-8.

T. Kimoto, K. Asakawa, M. Yoda, and M. Takeoka. Stock market prediction system with modular neural networks. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 1–6. IEEE, 1990.

N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake. Optimization-based locomotion planning, estimation,

and control design for the atlas humanoid robot. *Autonomous Robots*, 40(3):429–455, 2016.

A. Kumar, N. Paul, and S. Omkar. Bipedal walking robot using deep deterministic policy gradient. *arXiv preprint arXiv:1807.05924*, 2018.

L. Kwek, E. Wong, C. K. Loo, and M. Rao. Application of active force control and iterative learning in a 5-link biped robot. *Journal of Intelligent and Robotic Systems*, 37(2):143–162, 2003.

M. C. Lau. *Betty: A Portrait Drawing Humanoid Robot Using Torque Feedback and Image-based Visual Servoing*. PhD thesis, Department of Computer Science, University of Manitoba, Winnipeg, Canada, April 2014.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

S. H. Lee and A. Goswami. Ground reaction force control at each foot: A momentum-based humanoid balance controller for non-level and non-stationary ground. In *Proceedings of IROS-2010*, pages 3157–3162, Taipei, 2010. doi: 10.1109/IROS.2010.5650416.

S.-H. Lee and A. Goswami. Reaction mass pendulum (rmp): An explicit model for centroidal angular momentum of humanoid robots. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4667–4672. IEEE, 2007.

C. S. Leung, K. W. Wong, P. F. Sum, and L. W. Chan. On-line training and pruning for recursive least square algorithms. *Electronics Letters*, 32(23):2152–2153, 1996.

M. Missura and S. Behnke. Omnidirectional capture steps for bipedal walking. In *Proceedings of Humanoids-2013*, pages 401–408, Atlanta, GA, 2013. ISBN 9781479926190.

M. Missura and S. Behnke. Online learning of foot placement for balanced bipedal walking. *IEEE-RAS International Conference on Humanoid Robots*, 2015-Febru: 322–328, 2015. ISSN 21640580. doi: 10.1109/HUMANOIDS.2014.7041379.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

J. Morimoto, G. Cheng, C. G. Atkeson, and G. Zeglin. A simple reinforcement learning algorithm for biped walking. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 3030–3035. IEEE, 2004.

M. Morisawa, F. Kanehiro, K. Kaneko, N. Mansard, J. Sola, E. Yoshida, K. Yokoi, and J. P. Laumond. Combining suppression of the disturbance and reactive stepping for recovering balance. *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pages 3150–3156, 2010. ISSN 2153-0858. doi: 10.1109/IROS.2010.5651595.

A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

N. C. I. NIH. Anatomical terminology, 2018. URL `https://training.seer.cancer.gov/anatomy/body/terminology.html`.

J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.

X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):143, 2018.

Phidgets. Phidgetspatial precision 3/3/3 high resolution, 2018. URL `https://www.phidgets.com/?&prodid=32`.

L. P. Pook. *Understanding pendulums: a brief introduction*, volume 12. Springer Science & Business Media, 2011.

J. Pratt, J. Carff, S. Drakunov, and A. Goswami. Capture point: A step toward humanoid push recovery. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 200–207. IEEE, 2006.

Quanmax. Qutepc-3000 series, 2017. URL `http://www.quanmax.com/site/product/qutepc-3000-series/`.

J. Quiñonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate

gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

S. Ramezani, M. Heydari, F. Shamshirdar, S. Behjou, M. Ahmadi, M. Karimi, B. Azari, S. Sadeghnejad, A. Hosseinmemar, J. Anderson, and J. Baltes. AUT-UofM Humanoid TeenSize Team Description Paper RoboCup 2015 Humanoid TeenSize Robot League. *Team Description Paper, Robocup*, 2015. URL `https://www.robocuphumanoid.org/qualification/2015/1d0f918aaddf0f852e4655e12ca06e999d5137a8/AUT_UofM_Humanoid_TeenSize_2015_TDP.pdf`.

Robocup. Robocup technical challenge, 2018. URL `http://www.robocuphumanoid.org/wp-content/uploads/RCHL-2018-Rules-Proposal_final.pdf`.

Robotis. Dynamixel, 2018a. URL `http://www.robotis.us/dynamixel/`.

Robotis. U2d2 robotis usb to dynamixel adapter, 2018b. URL `http://support.robotis.com/en/product/auxdevice/interface/u2d2.htm`.

Robotis. Usb2dynamixel, 2017. URL `http://support.robotis.com/en/product/auxdevice/interface/usb2dxl_manual.htm/`.

E. Rodriguez-Leal, J. S. Dai, and G. R. Pennock. Kinematic analysis of a 5-r sp parallel mechanism with centralized motion. *Meccanica*, 46(1):221–237, 2011.

H. Sak, A. Senior, and F. Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*, 2014.

T. Šalát. On statistically convergent sequences of real numbers. *Mathematica Slovaca*, 30(2):139–150, 1980.

A. Schmitz, M. Missura, and S. Behnke. Learning footstep prediction from motion capture. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6556 LNAI:97–108, 2011. ISSN 03029743. doi: 10.1007/978-3-642-20217-9{_}9.

S. Shojaeipour, B. Parhizkar, A. Hosseinmemar, A. Shojaeipour, H. Esfandiari, E. Mobasheri, B. Gebril, and Z. Mohana. Laser-pointer rangefinder between mobile robot and obstacles via webcam based. In *Key Engineering Materials*, volume 447, pages 609–613. Trans Tech Publ, 2010.

K. Sreenath, H.-W. Park, I. Poulakakis, and J. W. Grizzle. A Compliant Hybrid Zero Dynamics Controller for Stable, Efficient and Fast Bipedal Walking on MABEL. *The International Journal of Robotics Research*, 30(9):1170–1193, 2011. ISSN 0278-3649. doi: 10.1177/0278364910379882.

B. Stephens. Humanoid push recovery. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pages 589–595. IEEE, 2007.

B. Stephens. *Push recovery control for force-controlled humanoid robots*. PhD thesis, Carnegie Mellon University Pittsburgh, Pennsylvania USA, 2011.

B. J. Stephens and C. G. Atkeson. Push recovery by stepping for humanoid robots with force controlled joints. *2010 10th IEEE-RAS International Conference on*

Humanoid Robots, Humanoids 2010, pages 52–59, 2010. doi: 10.1109/ICHR.2010.5686288.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

R. O. System. Kinetic Kame robot operating system, 2016. URL `http://www.ros.org/`.

R. Tajima, D. Honda, and K. Suga. Fast running experiments involving a humanoid robot. *2009 IEEE International Conference on Robotics and Automation*, pages 1571–1576, 2009. ISSN 1050-4729. doi: 10.1109/ROBOT.2009.5152404.

T. Takenaka, T. Matsumoto, and T. Yoshiike. Real time motion generation and control for biped robot-1 st report: Walking gait pattern generation. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1084–1091. IEEE, 2009.

F. Tanaka, K. Isshiki, F. Takahashi, M. Uekusa, R. Sei, and K. Hayashi. Pepper learns together with children: Development of an educational application. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 270–275, Nov 2015. doi: 10.1109/HUMANOIDS.2015.7363546.

D. Tolani, A. Goswami, and N. I. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical models*, 62(5):353–388, 2000.

Trossenrobotics. Ax series dynamixel & bioloid accessories, 2018. URL `https://www.trossenrobotics.com/6-port-ax-mx-power-hub`.

S. G. Tzafestas, T. E. Krikochoritis, and C. S. Tzafestas. Robust sliding-mode control of nine-link biped robot walking. *Journal of Intelligent and Robotic Systems*, 20 (2-4):375–402, 1997.

H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

M. Vukobratović and B. Borovac. Zero-moment point—thirty five years of its life. *International journal of humanoid robotics*, 1(01):157–173, 2004.

P. R. Vundavilli and D. K. Pratihar. Balanced gait generations of a two-legged robot on sloping surface. *Sadhana*, 36(4):525–550, 2011.

S. v. d. Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2): 22–30, 2011.

H. Wang, H. Qi, M. Xu, Y. Tang, J. Yao, X. Yan, and M. Li. Research on the relationship between classic denavit-hartenberg and modified denavit-hartenberg. In *2014 Seventh International Symposium on Computational Intelligence and Design*, volume 2, pages 26–29, Dec 2014. doi: 10.1109/ISCID.2014.56.

Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989. doi: 10.1162/neco.1989.1.2.270. URL `https://doi.org/10.1162/neco.1989.1.2.270`.

S. J. Yi, B. T. Zhang, D. Hong, and D. D. Lee. Learning full body push recovery control for small humanoid robots. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2011. ISBN 9781612843865. doi: 10.1109/ICRA.2011.5980531.

S. K. Yun and A. Goswami. Momentum-based reactive stepping controller on level and non-level ground for humanoid robot push recovery. In *IEEE International Conference on Intelligent Robots and Systems*, 2011. ISBN 9781612844541. doi: 10.1109/IROS.2011.6048132.