3-10-2021

Learning Control of Robotic Arm Using Deep Q-Neural Network

Seyed Navid Mellatshahi
University of Windsor

Learning Control of Robotic Arm Using Deep Q-Neural Network

By

Seyed Navid Mellatshahi

A Thesis
Submitted to the Faculty of Graduate Studies
through the Department of Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science
at the University of Windsor

Windsor, Ontario, Canada

2020

Learning Control of Robotic Arm Using Deep Q-Neural Network

by

Seyed Navid Mellatshahi

APPROVED BY:

_____

M.J. Ahamed

Department of Mechanical, Automotive

and Materials Engineering

_____

A. Ahmadi

Department of Electrical and Computer
Engineering

_____

Sh. Alirezaee, Co-Advisor

Department of Electrical and Computer
Engineering

_____

M. Saif, Co-Advisor
Department of Electrical and Computer
Engineering

December18, 2020

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Enabling robotic systems for autonomous actions such as driverless systems, is a very complex task in real-world scenarios due to uncertainties. Machine learning capabilities have been quickly making their way into autonomous systems and industrial robotics technology. They found many applications in every sector, including autonomous vehicles, humanoid robots, drones and many more.

In this research we will be implementing artificial intelligence in robotic arm to be able to solve a complex balancing control problem from scratch, without any feedback loop and using state of the art deep reinforcement learning algorithm named DQN.

The benchmark problem that is considered as case study, is balancing an inverted pendulum upward using a six-degrees freedom robot arm. Very simple form of this problem has been solved recently using machine learning however under this thesis we made a very complex system of inverted pendulum and implemented in Robot Operating System (ROS) which is very realistic simulation environment.

We have not only succeeded to control the pendulum but also added turbulences on the learned model to study its robustness. We observed how the initial learned model is unstable at the presence of turbulence and how random turbulences helps the system to transform to a more robust model. We have also used the robust model in different environment and showed how the model adopt itself with the new physical properties.

Using orientation sensor on the tip of the inverted pendulum to get angular velocity, simulation in ROS and having inverted pendulum on ball joint are few highlighted novelties in this thesis in compare previous publications.

DEDICATION

I dedicate this dissertation to my lovely wife Sara, my little princess Alma and strong son, Arsha.

ACKNOWLEDGEMENTS

My deep gratitude goes first to my lovely smart wife, who supported me through all these years with her hardworking. I am so thankful to her, for being always my motivation and supporting me in all aspects of my life.

My appreciation also extends to Dr. Alirezaee and Dr. Saif, who guided me through my research projects with their valuable comments and feedbacks. Their support helped in transforming this paper into a more meaningful and valuable document. Their immense passion for teaching has further inspired me to pursue this path in the future.

I would also like to thank my committee members, Dr. Jalal Ahamed and Dr. Arash Ahmadi, for their comments and their time in reviewing my work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

LIST OF ABBREVIATIONS/SYMBOLS

RL	REINFORCEMENT LEARNING
AI	Artificial Intelligence
AGI	Artificial general Intelligence
MLP	Multi-layer perceptron
A3C	Asynchronous Advantage Actor-Critic Algorithm
TD3	Twin Delayed Deep Deterministic Policy Gradient
SAC	Soft Actor-Critic
TRPO	Trust Region Policy Optimization
NAF	Q-Learning with Normalized Advantage Functions
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q Network
SARSA	State–action–reward–state–action
IMU	Inertial Measurement Unit
DOF	Degree of Freedom

CHAPTER 1 INTRODUCTION

1.1 Problem Definition

The Inverted pendulum control is a benchmark control problem that is used by researchers to test the new control strategies over past 50 years. It has a simple structure but wealthy model to test control strategies. Solving this type of problem with machine learning is a promising approach because it does not require dynamic model of system but instead the machine learning algorithm can generate autonomous actions based on the experience.

The purposed case study in this thesis is balancing an inverted pendulum on a ball joint using a robotic arm and machine learning technique instead of normal control loop such as PID. The inverted pendulum can freely fall in any direction because it is connected to the end-effector using a ball joint. The robotic arm that has been chosen for this case study has 6 degrees of freedom. We use an Inertial Measurement Unit (IMU) for obtaining angular velocity and orientation of the pendulum to use these data in the learning algorithm.

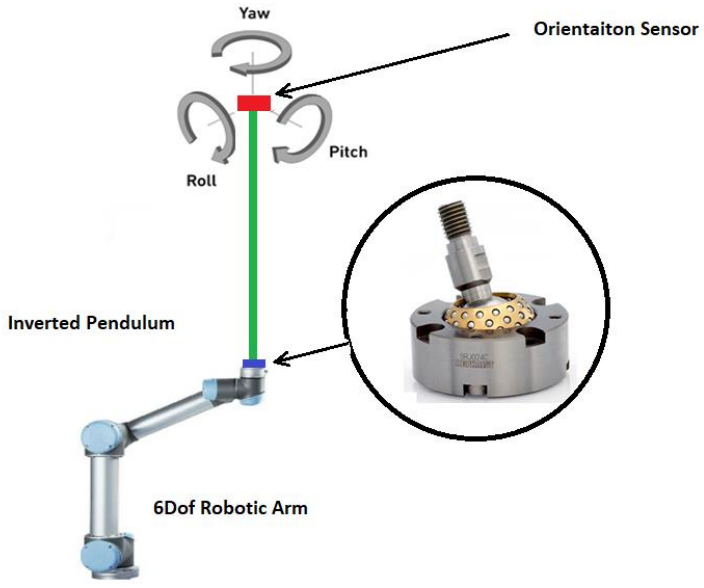


Figure 1.1: **3DoF Inverted Pendulum on Robotic Arm**

1.2 Applications

The Inverted pendulum control is a benchmark control problem that is used by researchers to test the new control strategies over past 50 years. It has a simple structure but wealthy model to test control strategies. There are many robotic applications also based on the inverted pendulum in term of their stabilization principle. [1] For example:

- Control of under-actuated robotic systems
- Design of mobile inverted pendulums
- Gait planning of humanoid robots

The control of humanoid robots is challenging task due to having dynamic constrains and uncertainty. Gait pattern generation is key problem and in order to simplify the trajectory generation many studies use analogy between bipedal gait and the inverted pendulum motion[1].

Design and implementation of mobile wheeled pendulum emerged in commercial products too, Segway and self-balancing scooter are two examples of the commercial products that uses the inverted pendulum controller approach.[1]

Figure 1.2: **Commercial products using Inverted Pendulum controlling techniques**

Shuujit Kajita and his team developed a novel framework for biped stabilization control for humanoid robot with 42 degree of freedom using a simple linear inverted pendulum dynamic for walking stabilization[2].

Figure 1.3: **Humanoid Robot 42 DoF Walking on a Pavement[2]**

Solving these types of problems with machine learning is a promising approach because it does not require dynamic model of system but instead the machine learning algorithm can generate autonomous actions based on the experience. It not only helps robot to make decision on unseen situations but also the learned model can be used on the robots with different physical properties.

1.3 Overall Challenges

Enabling robotic systems to do tasks autonomously is a very complex task in real-world scenarios because of uncertainty. Uncertainties cannot be programmed by IF & THEN, so sort of general artificial intelligence requires for these systems to enable them to make human like decisions

Although this combination enables robots to act in a situations where constrains are dynamic[3] but there are many challenges, some of the major challenges to train and build an autonomous model is:

- Data for specific task
- Computation time required for learning
- Robustness of a model
- Safety during learning process

When a machine learning algorithm runs on simulation environment, it is runs at high speed and few days or months may reach to a desired level of learning, however one of the main challenges is once the learnt model transferred to a real robot does it provide the same results. The Sim to Real transfer id big challenge and researchers are working on solutions to overcome the challenges some of them are listed in table 1.1

Table 1-1: **Possible solutions to overcome common challenges**

Solution/ Challenges	Data	Time	Safety	Robustness	Comments
Fast learning algorithms	✗	✓	✗	✗	
Robot Farm	✗	✗	✗	✓	
Realistic simulation	✓	✓	✓	✓	It is highly depending on robot, task, and nature of problem
Hybrid Training Method	✗	✓	✓	✓	Digital Twin (Sim to Real & Real to Sim)
Reinforcement Learning	✓	✗	✗	✓	

1.4 Methodology

Our approach to solve the proposed problem is using state of the art reinforcement learning algorithm in Robot Operating System (ROS) environment.

We will be using reinforcement learning (RL) to overcome challenge of data gathering, because reinforcement learning does not need any data for solving a problem instead it learns by reward and punishment technique. There are many reinforcement learning algorithms, but we will be using Deep Quality Network (DQN) which is recently attract many applications and emerged in recent research for variety type of application where human type of decision is needed.

We will be using ROS in combination with Gazebo simulation which is highly realistic environment and highly used in commercial and research problems. Using the realistic environment helps on computation time and overcome challenge of safety. Safety is a key problem in reinforcement learning because agent or robot does not have any understanding of its environment at the beginning and needs to explore and find right action in right situation over time. So, it may break or do very unsafe actions during learning process.

We will also make the learning model very robust by applying forces randomly during training process. So, the robust model can be used in real robot and adopt itself with the new dynamic model. Our proposed solution overcome all challenges that mentioned in the table 1.1 for this specific problem.

We will be using a sensor on the top of the inverted pendulum to sense the angular velocity and orientation of the inverted pendulum, these data used as input to the learning algorithm. We use sensor fusion technique to make this sensor and it is explained in detail on hardware configuration in chapter 3. Since majority of recent publications are single degree freedom inverted pendulum, to better analyzing and proper apple to apple comparison with their results, we have broken down the complex problem into 3 phases, from simple system which is one degree freedom inverted pendulum to more complex system, which is 3 DoF inverted pendulum on a 6 degrees robot arm.

1.5 Novelties

Multiple novelties are introduced in this thesis that makes it unique and wealthier among the others that are published on the same topic:

- We made an orientation sensor using sensor fusion technique and placed it on top of the pendulum to measure angular velocity, pitch, and roll. This approach has not been done in earlier papers so far and they measured the angle of pole from the pivot point as input to learning program. Their approach is not practical for two reasons; firstly, the speed on top of the pendulum changes faster than the button and secondly measuring angle from the joint itself is not practical because it cannot be easily implemented in hardware. In our proposed solution, the base bearing, ball joint and inverted pendulum is block that can be placed on any robot or hardware for experimental trial. We used sensor fusion technique that is explained in chapter 3 section 4.
- We used ROS and GAZEBO simulation environment which is very popular for robot simulation applications and very realistic. This platform allows smooth transfer from simulation to real robot without changing any major changes in the program. In addition, the learnt model would be very close to the reality since the simulation environment is one of the top robotic simulation in the world with well-known kinematic engine.
- we use inverted pendulum on a ball joint attached to the end-effector of robot arm with 6DoF, this complex system has not been published in any paper before to the best of our knowledge in the time of writing this thesis.

1.6 Structure

The structure of the thesis is in a sequence. First, in state-of-the-art chapter we discuss about reinforcement learning, simulation environment, robotic arm details and at the end close it by reviewing latest literatures in that topic.

In Chapter 3 we will discuss details of all 3 phases of proposed methodology for the single, 3 DoF and robotic arm inverted pendulum and finally in Chapter 4 we will review results of all 3 phases. Figure 1.4 shows the structure of this thesis from beginning to the end.

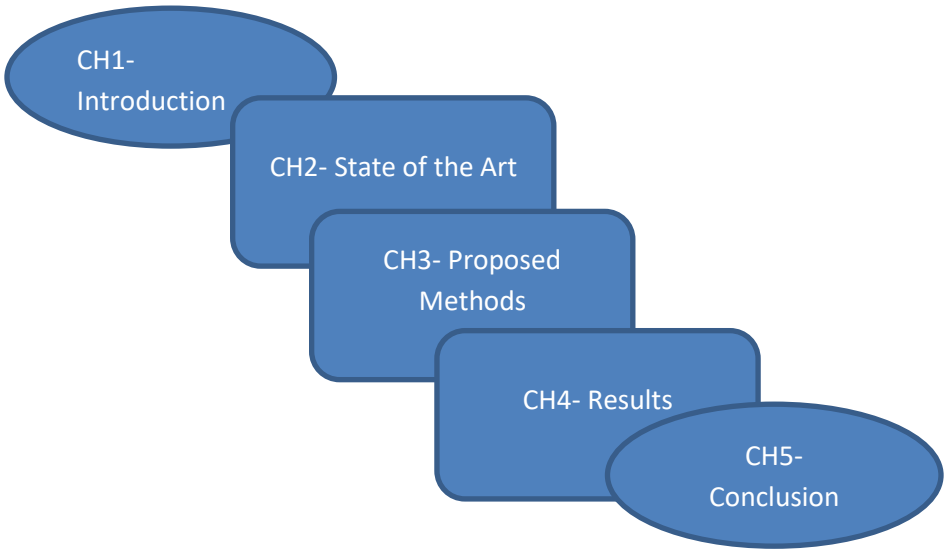


Figure 1.4: **Overall structure of the thesis**

1.7 Conclusion

In this chapter we provided an overview of the thesis and its structure, proposed methodology and solution methods to solve the problem of inverted pendulum with machine learning. In next chapter we will discuss in detail about state of the art that are used in this thesis.

CHAPTER 2 STATE OF THE ART

2.1 Introduction

In this chapter we will review discuss about reinforcement learning algorithm, robot operating system and its simulation environment, robot arm that is used for our case study and will do literature review at the end.

2.2 Machine Learning

Machine Learning (ML) has attracted more attention nowadays and found many applications in every sector of industry, from big data pattern recognition to automation and entertainment. It plays an important role in medical field such as heart, liver, and cancer early detection systems. ML has opened a way to finance and businesses too and enabling better data-driven approaches, from stock prediction to finance. Machine learning algorithms are getting improved and developed, new methods and algorithms are emerging every day.

ML technology is enabling a paradigm shift in problem-solving from analytical to powerful data-driven approach. High speed processing units, availability of big data and labeled data, enables computer programs learn models from training data and predict results from new data.[4]

Main categories of machine learning are:

- Supervised learning
- Unsupervised learning
- Reinforced learning

In supervised learning, with help of labeled data we look for an approximation function that can represent data at the end of training process. Supervised learning has many applications however for our case study we are not using this method since it is requiring data and secondly for the balancing pendulum billions of data point might needed. There are many algorithms for supervised learning such as Naive Bayes, Decision Trees, Linear Regression, Support Vector Machines (SVM), Neural Networks and many more.

Unsupervised learning method is mainly used for pattern recognition and deceptive modeling, basically we have data, but they are not labeled and learning algorithm try to find patterns and relation between data. Again, this category of machine learning is not proposed on this case study since we do not have data for balancing and inverted pendulum either labeled or unlabeled.

Reinforcement learning algorithm is the one we are proposing to use in this case study because it does not require any data and it enables robot learns task by its own without any supervision. Reinforcement leaning used reward technique to learn a task.

2.2.1 Reinforcement Learning Algorithm

Reinforcement Learning (RL) is a class of machine learning (ML) models where the learning process is based on evaluative feedbacks without any supervised signals.

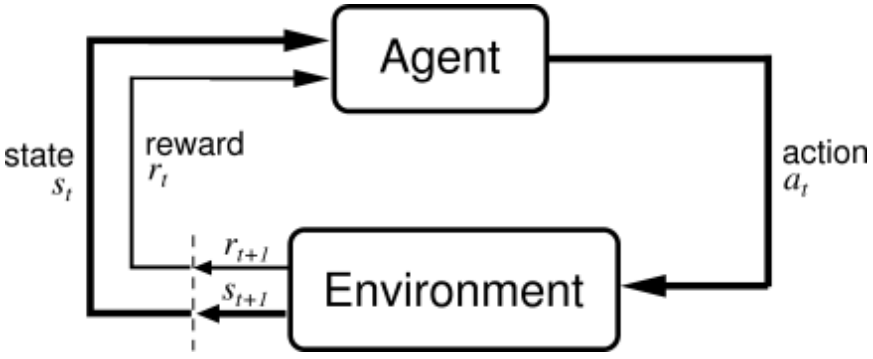


Figure 2.1: **Overview of Reinforcement Learning Algorithm**

RL comes from the mammal learning theory. it does not require any prior knowledge and in fact it can autonomously get optional policy with the knowledge obtained by trial-and-error and continuously interacting with dynamic environment. [5]

As seen in Figure 2.1, Agent chooses action based on each data point and later learns how good decision was. Over time, the algorithm changes its strategy to achieve the best reward.

The mathematical framework for defining a solution in reinforcement learning scenario is called Markov Decision Process. This can be designed as:

- Set of states, S
- Set of actions, A
- Reward function, R
- Policy, π
- Value, V

Agent takes an action (A) to transition from the start state to the end state (S) and in return, gets reward (R) for each action. actions can lead to a positive reward or negative reward. The set of actions that agent takes, define the policy (π) and the rewards it get in return, defines value (V). [5]

The task here is to maximize rewards by choosing the correct policy. So, we must maximize for all possible values of S for a time t that can be seen in equation 2.1, where π is policy, r is reward and s is state and E is a function that needs to be maximized:

$$E(r_t \mid \pi . s_t) \qquad \textit{Equation 2.1}$$

The objective in reinforcement learning algorithm is to find optimum policy to achieve the goal, however the challenge is how to maximize the summation of reward, so exploration and exploration is dilemma for the RL system. Let assume the RL agent did an action and got 100 rewards in the state space, but the main question is if it is it the best that hope for. In fact, exploitation is about the agent stick to what understood from the environment so far and accept that is good policy, but the risk is missing other opportunities out of the learnt policy that may lead to get more reward. On the other hand, exploration is about the

agent look for exploring environment and hoping to hunt more rewards but of course the risk is wasting time and getting negative feedback.

Many algorithms have been developed for reinforcement learning but most of them use Epsilon Greedy strategy to balance between exploitation and exploration. The gamma factor that is a number between 0 and 1 helps agent to discover its environment at the beginning but over the time when learning is progressing it reduces the exploration probability and move more toward exploitation.

Reinforcement learning systems have many applications such as self-driving cars, humanoid robots, game playing, automatic trading etc. and popular algorithms are Q-Learning, SARRA, DQN, A3C and Genetic Algorithm.

2.2.2 Q Learning

Q-learning is off-policy and model-free reinforcement learning algorithm. off-policy methods evaluate or improve a policy different from that used to generate the data, in contrast On-policy methods attempt to evaluate or improve the policy that is used to make decisions. In model free type algorithms RL make no assumption of the dynamic model of the environment.

The 'q' in Q-Learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward. In Q-Learning we make a Q-table that represents quality value of each action and each state. To briefly explain how Q-learning works, let assume a robot must cross a maze and reach end point but there are several mines and power up points in the area. Figure 2.2 represent the problem.

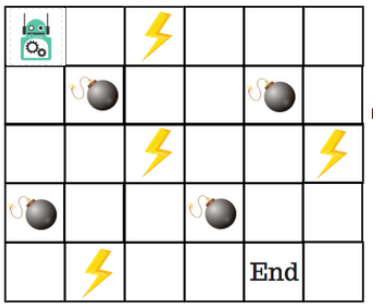


Figure 2.2: **Robot in Maze - Q-Learning**

The Q-table consists of four actions in 4 columns and 5 states in 5 rows. In this example. This table is first initialized with zero values but then using a Bellman Equation (2.2) will get updated each time robots do an action.[6]

$$V(s) = max(\, R((s,a) + \gamma V\,(s') \qquad \textit{Equation 2.2}$$

- s = a particular state
- a = action
- s′ = state to which the robot goes from s
- γ = discount factor (we will get to it in a moment)
- $R((s,a)$ = a reward function which takes a state s and action a and outputs a reward value
- $V(s)$= value of being in a particular state (the footprint)

The Bellman Equation in simple form says the current value of Q is related mostly to immediate reward plus a portion of future reward. Overall algorithm for Q-learning can be seen in figure 2.3

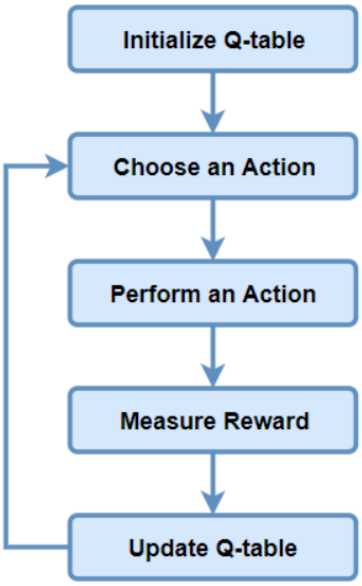


Figure 2.3: **Q-Learning Overall Algorithm**

As we saw in above example, in Q-learning, we build a memory table Q[s, a] to store Q-values for all possible combinations of states and actions. Memory requirement for Q-Learning is an array of states times actions. If the combinations of states and actions are too large, then memory and the computation requirement for Q will be extremely high or impossible. This constrain impose a problem for large scale situations, for example in chess the state space is about $10^{120,}$ which make Q-Learning algorithm useless. This main problem leads us toward selecting a better learning algorithm suited our case for controlling an inverted pendulum.

Instead of storing all Q values in a table what if we approximate it on each state. This is leveraging Neural Network in Q-Learning that is explained in next section.

2.2.3 Deep Q Learning

After DeepMind's paper published on 2015 "Human-level control through deep reinforcement learning" [7] , google company acquired DeepMind and few years later

Alpha Go has been released. This Deep RL algorithm mastered the game of GO without human knowledge and from scratch just by knowing the rules of game and playing against itself for 4 days. This version of Alpha Go managed to win 10 to zero against world Go champion. Go is abstract strategy board game for two players with number of legal board position of approximately 2×10^{170}. Few years after a new AlphaGo Zero emerged with newly developed Deep RL algorithm and achieved superhuman performance, winning 100–0 against the previously published champion-defeating AlphaGo.[8]

Combination of neural network with reinforcement learning opened new horizon for autonomous robots and recently topic of many researchers around the world. Just in IEEE from 2018, 390 papers are published that used this algorithm on various applications include robotics, autonomous vehicles, game play and all sort of control where human type of decision making is needed, or uncertainty are involved. DQN shows a great success in this type of control and that is main reason we have chosen this RL algorithm among the others too.

Figure 2.4 shows the difference between Q learning and Deep Q Learning in a simple problem, in fact instead of having a table for all states and actions we estimate Q values for each action with Neural Network and approximate the Q values. With this approach we minimize the amount of memory requirement. However, this approach imposes some other problems as well.

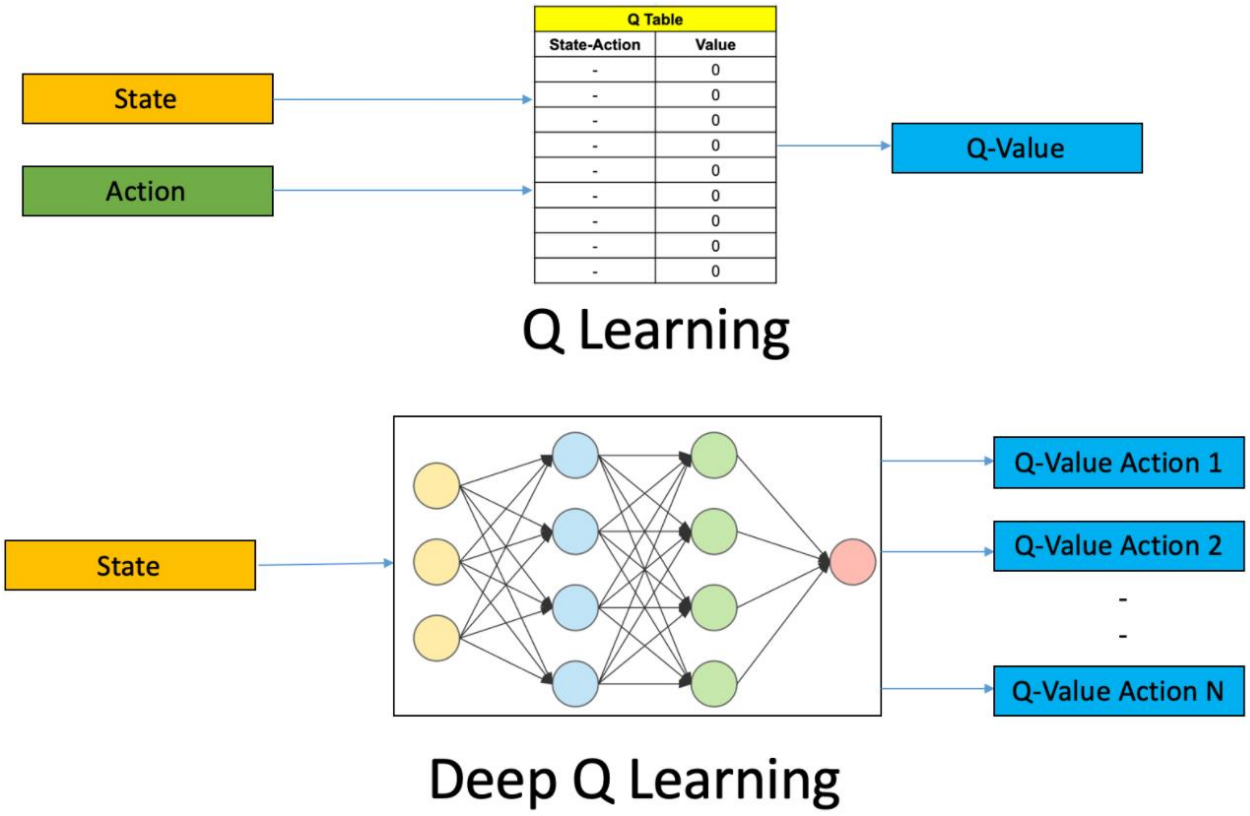


Figure 2.4: **Q-Learning Vs Deep Q Learning**

In reinforcement learning, both the input and the target change constantly during the process and make training unstable.[9] In contrast in the supervised learning, samples are randomized among batches and each batch has similar data distribution, also samples are independent. If these conditions are not fulfilled during training of neural network, then we may end up over fitting the network. We build a deep network to learn the values of Q but its target values are changing, basically the target values for Q depends on Q itself, so we are chasing a moving target.

The solution for DQN is using experienced replied memory. For instance, we put the last 50,000 transitions into a buffer and sample a mini batch of samples from this buffer to train the deep network. This forms an input dataset which is stable enough for training. As we randomly sample from the replay buffer, the data is more independent of each other.

As part of this thesis, we are not going to program DQN from scratch but instead we focus on implementation of this algorithm on a Robot Arm and experience the results on a more complex environment. For that reason, we will be using OpenAI baseline DQN algorithm.

OpenAI is an AI research and deployment company that is governed by the board of OpenAI Nonprofit with Microsoft as investor. Their mission is to ensure that artificial general intelligence (AGI)—by which we mean highly autonomous systems that outperform humans at most economically valuable work. One of their main products is High-quality implementations of reinforcement learning algorithms that are widely used by many researchers around the world.

2.3 Simulation Environment

There are many robotic simulation platforms that enables offline programming using the model of robots. However, among all there are only few that are very popular among researchers. Robot Operating System (ROS) in combination with Gazebo Robot Simulation is the top among all for about 10 years. ROS is a middleware software that connect high level programming to low level hardware extremely fast and in a very reliable way using subscription and publishing method.

2.3.1 Robot Operating System (ROS)

Robot Operating System (ROS) started late 2000s at Stanford university, widely becoming a common, standard tool among robotics researchers and industry, since its initial release in 2010. ROS is a tool for offline robot programming and provide great packages and support for artificial intelligence programming as well. Some of the highlighted features of ROS are [10]:

- Open source, big community, and continuous support

- Fast subscribing / publishing techniques
- Code reuse in robotics research and development
- Ready-to-use development environment with Comprehensive tools and client API libraries (MATLAB, C++, Python, Lisp, Java, …)
- Scalable (distributed network of processes)
- ROS enables connect ML algorithms to the actual robot or in simulated environment

Over the past 10 years, ROS has become the industry's most popular robot software development framework. According to ABI Research, by 2024 roughly 55% of the world's robots will include a ROS package. ROS supports many libraries some of the major ones are:

- OpenCV: computer vision
- Gazebo: Robot Simulator
- KDL: Kinematics and Dynamics
- TREX: High Level Planning

A ROS distribution is a versioned set of ROS packages and dependent to Linux distributions (e.g. Ubuntu). The purpose of the ROS distributions is to let developers work against a relatively stable codebase until they are ready to roll everything forward [10]

ROS starts with the ROS Master; the ROS Master allows all other ROS pieces of software (Nodes) to find and talk to each other. We can tell Node 1 to send messages to Node 2 or node 3 subscribe to Node 1 directly. Figure 2.5 shows how subscription and publishing message works in ROS environment. The camera node publishes messages then image processing node subscribe to image processing node and image display node on another laptop but at the same network can register to the same channel and listen to image data.

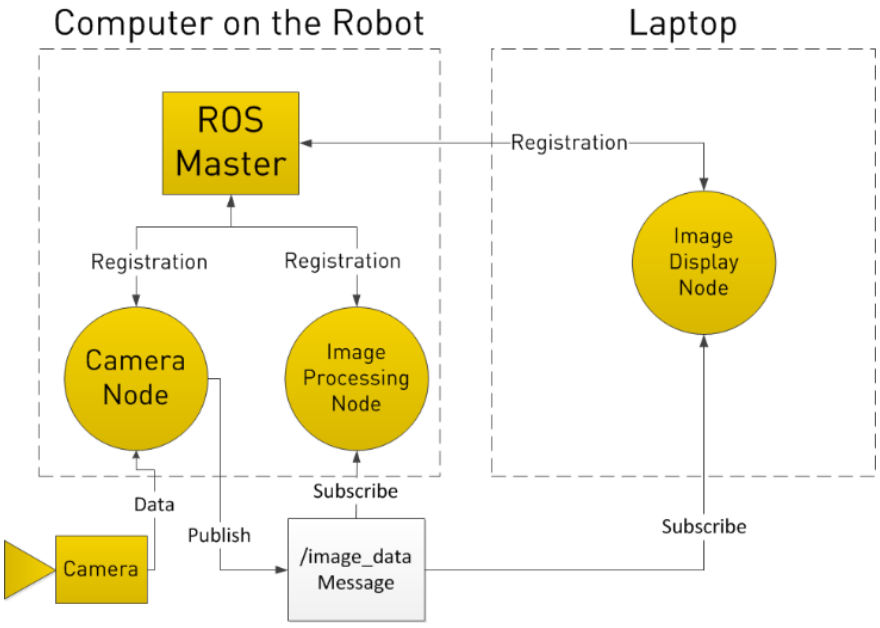


Figure 2.5: **Overview of ROS messaging system**

2.3.2 Gazebo Simulator / URDF File Structure

Gazebo is an open-source robotics simulation software and part of ROS software because it is being acquired by same company in 2011. Gazebo can use multiple high-performance physics engines, such as ODE (default), Bullet, etc. It provides realistic rendering of environments including high-quality lighting, shadows, and textures. It can model sensors that "see" the simulated environment, such as laser range finders, cameras (including wide-angle), Kinect style sensors, etc.

Gazebo highly used in research and industry and won many challenges and competition in world including NASA, Toyota Pirus, DARPA, Sub T, VRX etc.[11], [12]

Modeling for gazebo to be done using XML or Xacro which is macro language for XML format. In addition, for more sophisticated simulation we can export design directly from SolidWorks. This file is in specific format and called Unified Robotic Description Format (URDF). URDF is a file that describes the robot kinematic and basic physics, it has tree

structure, but order does not matter. We can define Links, joints, transmissions, collision, visual etc. An example of URDF shown below for a joint that has one child. [13]

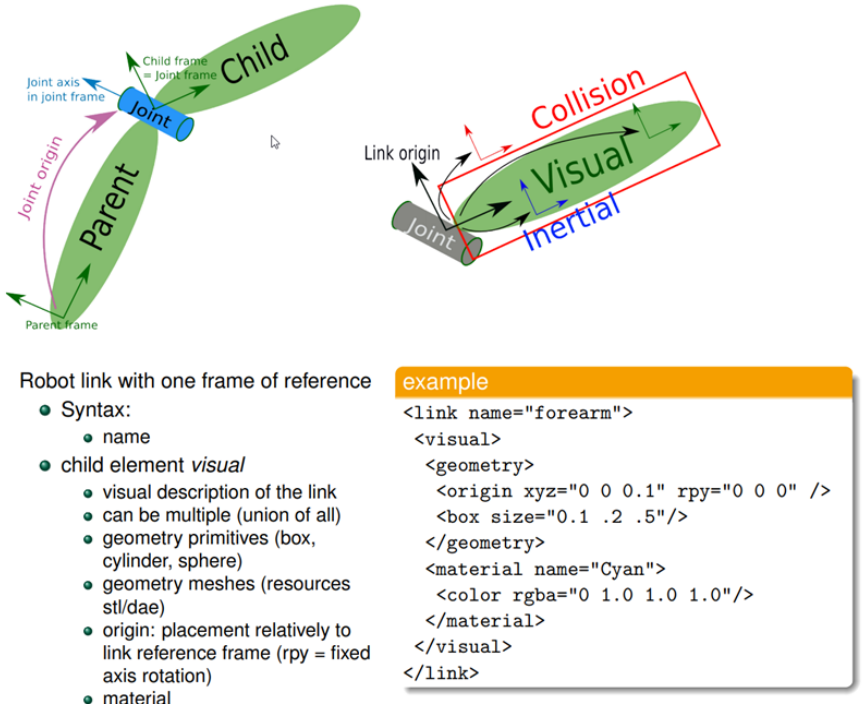


Figure 2.6: **A Simple Joint and XML Code of URDF**

2.4 Robotic Arm

Industrial robots are automated, programmable, and capable of movement on three or more axes. There are at least six type of industrial robots:

- Articulated Robots
- Cartesian Coordinate Robots
- Cylindrical Coordinate Robots
- Spherical Coordinate Robots
- SCARA Robots
- Delta or Parallel Robots

Articulated robots are the most common industrial robots. They are very similar to human arm and usually they have several degrees of freedom. Their articulations with many degrees of freedom allow the articulated arms a wide range of movements [14].

In practical industrial applications, there are two main categories of robotic programming methods:

- Online programming
- Offline programming

In online programming, the handheld programmer or joystick that is called teach pendant in industrial robot is used to manually move the end-effector to the desired position and orientation at each stage of the robot task, then robot controller save an calculate relevant frames, coordination and configurations for each step and then can repeat it at its maximum speed and accuracy step by step.

Offline programming method, which is based on the 3D model of the complete robot work cell and is becoming more popular. This type of programming highly related to the simulation model that is usually provided by manufacturer or third party has its strength on programming complex systems.

The robot arm that is used in this thesis is called Universal Robot which is one of the top manufacturers in making collaborative robot. University of Windsor recently made an automation lab with some of these robots. This robot has 6 joints and comes at 3 different payloads, 3kg, 5kg and 10 kg. Figure 2.7 shows all six joints of this robot

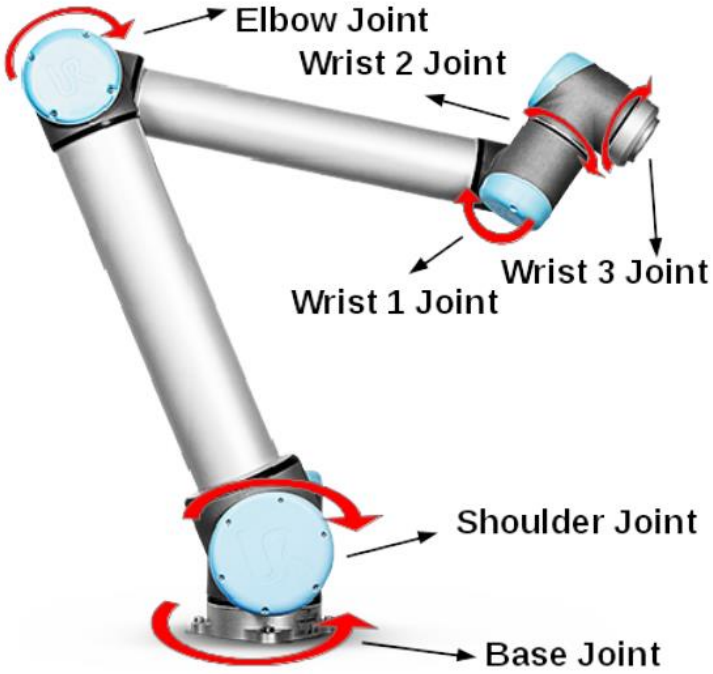


Figure 2.7: 6DoF Robot Arm and its joints [15]

According to the specification of this robot, all 6 joints can move 360 degrees with speed of 180°/sec. the repeatability of this robot is +/- 0.1 mm and apart from hardware Input/Output ports, it has TCP/IP 100 Mbit IEEE 802.3u protocol which make it suitable to communicate to the ROS environment.

Simulation model of this robot is available as open source and made by the manufacturer, that includes ROS driver. Figure 2.8 shows the simulation of this robot in Gazebo under the Robot Operation System platform.

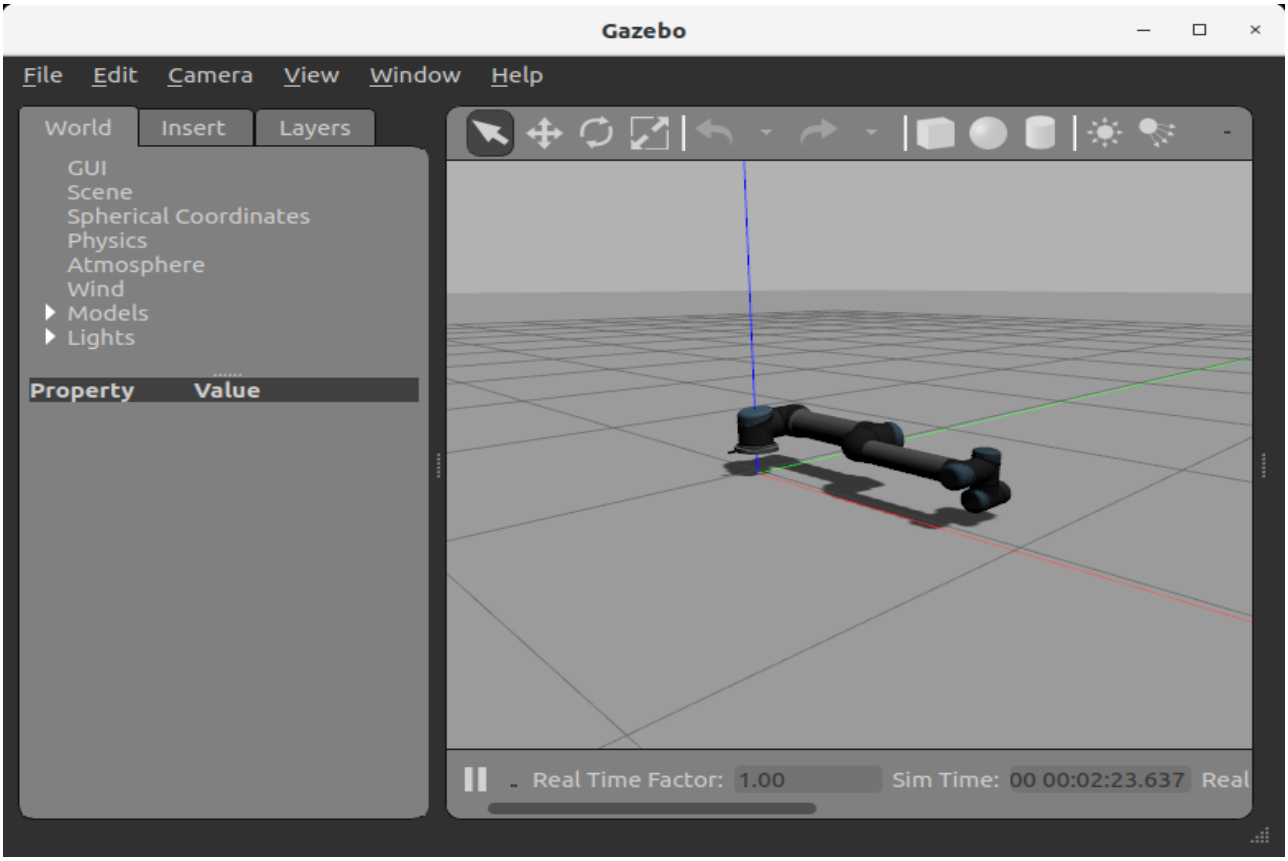


Figure 2.8: **Simulation model of Universal Robot in Gazebo**

2.5 Literature Review

For at least fifty years, the inverted pendulum has been the most popular benchmark problem among others, for teaching and research in control theory and robotics. According to a survey done on 2012 more than 13 different types of control designs are used in IEEE published journals to design a control system for inverted pendulum [1]. These control strategies included PID, predictive control, Hybrid control, Fuzzy logic control, RL and many more. To narrow down our literature review more specific, we will be review only very recent that used the reinforcement learning algorithm to solve inverted pendulum.

Yue Chao and his team on 2018 has published a paper in IEEE and used double layer back propagation neural network for inverted pendulum [16]. In this paper they used the straight-line single inverted pendulum as see in figure 2.5

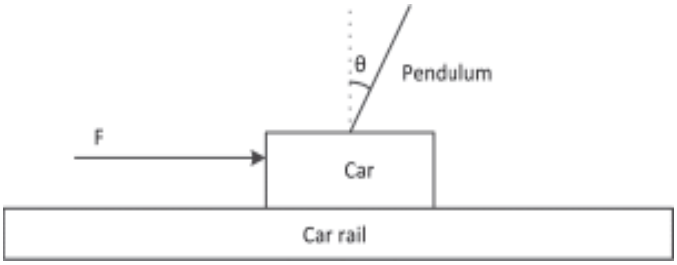


Figure 2.9: **straight-line single inverted pendulum**

They have used simulation model from Google Technology that included all physical properties of the environment included angular velocity, gravitational acceleration, angle of swing arm and so on. Their proposed neural network on this paper is shown below which takes 4 inputs as the states of the inverted pendulum and gives one output. The output is the speed of the cart in left or right direction. After 300 attempts, the learning system successfully implemented the swinging up of a single inverted pendulum. The entire time of reinforcement learning was 131s. [16]

The robustness of their model has not been verified in the paper also the physical properties of the inverted pendulum has not identified. As we explain later in this paper, physical properties of the model impact the robustness for example fraction between the swinging arm and the base is one of the key factors in the control.

Q-Learning approach also has been implemented by Alessio Ghio and his team and their paper has published in IEEE on 2019 [17]. In this paper they used two algorithms for balancing the inverted pendulum, the first algorithm used to swing up the pendulum upward and the second one is used to control it in upward position. A simple reward function has implemented to give 100 points when the pendulum is placed upward at each iteration. The physical properties that considered for the inverted pendulum is m = 1 kg, L = 1 m, b = 0.01 kg/s and g = 9.8 m/s 2. As per the chart published in the paper at the

beginning phase control is extremely unstable but over the 600 Episodes system was able to learn to control.

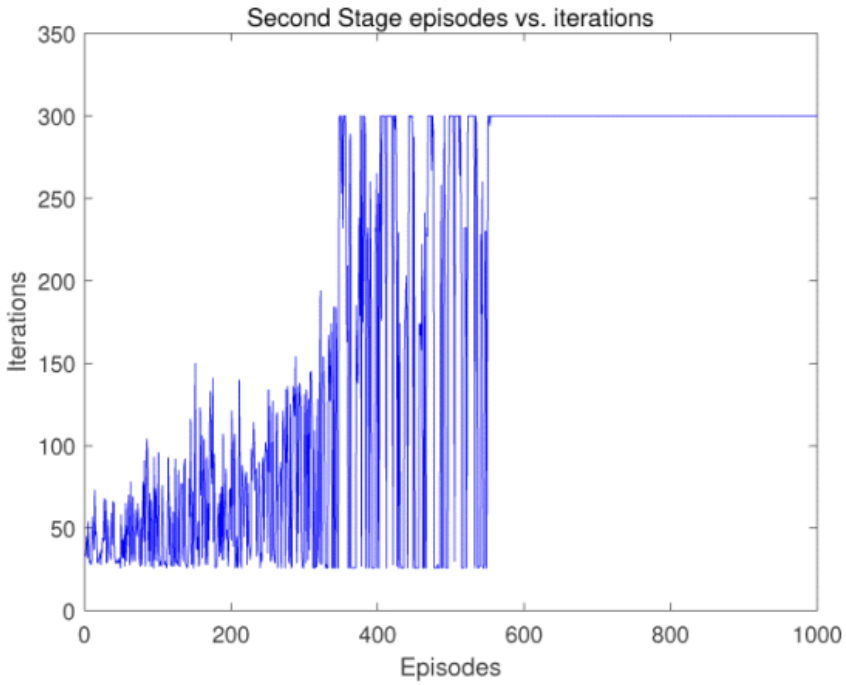


Figure 2.10: RL learning curve for single inverted pendulum [11]

In this research same as previous one, single inverted pendulum used with minimal physical property in the testing environment. Fraction, motor power and its transmission to move the base cart is ignored, the pivot point friction is not defined, and main point is the robustness of system has not been tested against any turbulence.

On December 2019, a paper is published in IEEE by X.Li & H. Liu that used DQN OpenAI baseline cartpole environment and implemented same in V-REP (Virtual Robot Experiment Platform). They have used V-REP because it is very rich in kinematic analysis and more realistic simulation rather than OpenAI Gym Cartpole environment.[18]

Their paper shows how physical properties impact on the RL model and to overcome instability during learning process they made a model with following characteristics that shows in figure 1.7:

- Changing the inverted pendulum to specific height
- Considering joint speed to specific value
- The masses of base and wheels set to larger value compare to inverted pendulum itself.
- The physics engine selected was Vortex, because of its stability in VREP environment.
- To overcome latency issue of Python in VERP, they have used a synch method in VERP, otherwise there will be large interference and huge impact on simulation unless reducing the gravity.

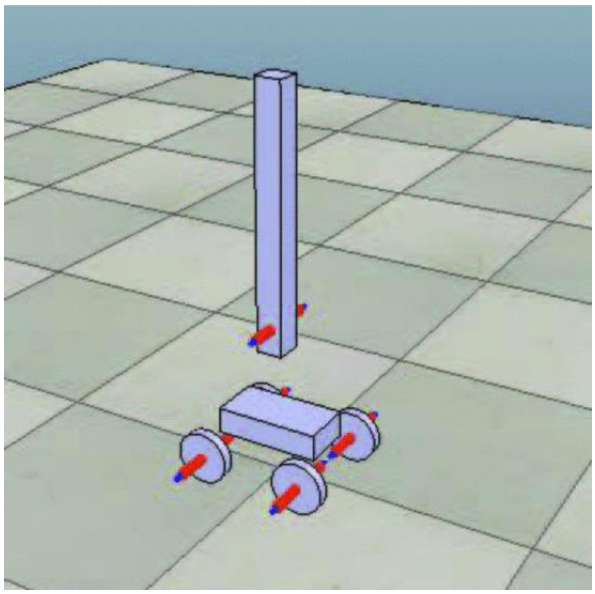


Figure 2.11: **V-REP Model of Cartpole [12]**

Although they have used only single inverted pendulum, had many challenges, and preset their physical properties to specific values however in our view this paper was the best among all recent published ones. It was very helpful to understand expectations on this this thesis too because it shows how difficult is to simulate a simple problem in a more realistic simulation environment and how different results can be achieved when parameters are not

accurately set. As seen in figure 1.8 their cost function chart is totally different than the results of other recent papers on solving the exact same problem.

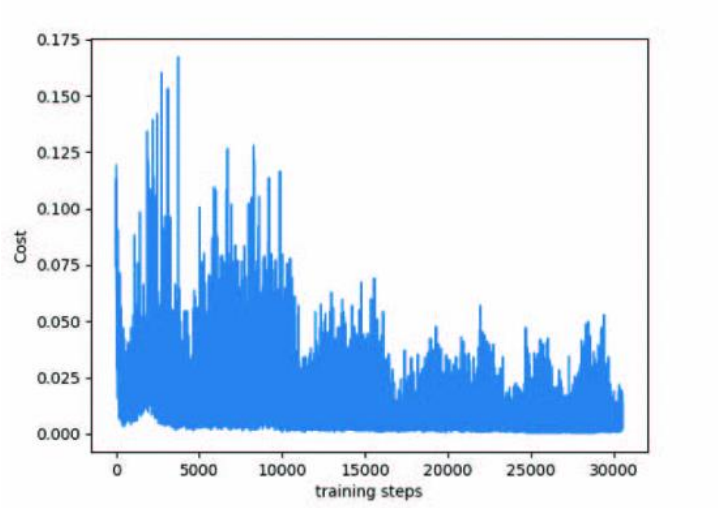


Figure 2.12: **Cost function of DQN Training in V-REP environment**

Simulations provide an abundant source of data and alleviate safety issues during the learning process however the model that is generated in simulation environment are often specific to the characteristics of the simulator. Due to modeling error, strategies, lack of proper kinematic model, physical properties of model etc. a simulation test might be successful but won't be when transferred to another simulation environment or real-world environment. [19]

Almost in none of the published paper the robustness of the RL model has been tested fully, and majority of them used non-realistic simulation environment where fraction, joint definition, power, efforts and acceleration have not been simulated with a proper kinematic engine because they have not been simulated in proper robot simulation platform unless the last one which is done in VERP. They have shown good result, but the outcome model only works in their environment and outside that the model is useless.

For our thesis we will be using Robot Operating System (ROS) in combination with Gazebo which are one of the top Robot Simulation Platform in the world that are widely used in research as well as industry.

CHAPTER 3 PROPOSED METHOD

3.1 Introduction

To implement reinforcement learning in the most complex system of inverted pendulum, we have broken down the case study in 3 sections. We start with single degree freedom inverted pendulum that is mainly used in recent published paper, this phase enables us to check and compare our results with the published ones, find our mistakes and solve all challenges for the next step.

Then we take step further and make it 3-degree freedom inverted pendulum where the inverted pendulum can free fall in any direction. To control this system, we place it on few chassis that can move freely on any directions. We tried to develop the entire program in such a way that expanding the project from one phase to another phase does not require re-structuring the entire program. We use classes that are written in Python language, libraries and configuration files to enable us quickly to transform one program to another.

3.2 One Degree Freedom Inverted Pendulum

The single straight-line one-degree freedom of inverted pendulum consists of a cart or a box that moves only in one axis and a pole that is hanged on the middle of it and where it swing freely on the same axis that cart or box moves. As seen in figure 10 a force F is required to move the cart to the left and right and based on the acceleration, speed, and its position the inverted pendulum can swing to the left or right too. The objective is by moving the cart (brown box), keep the angle φ as small as possible.

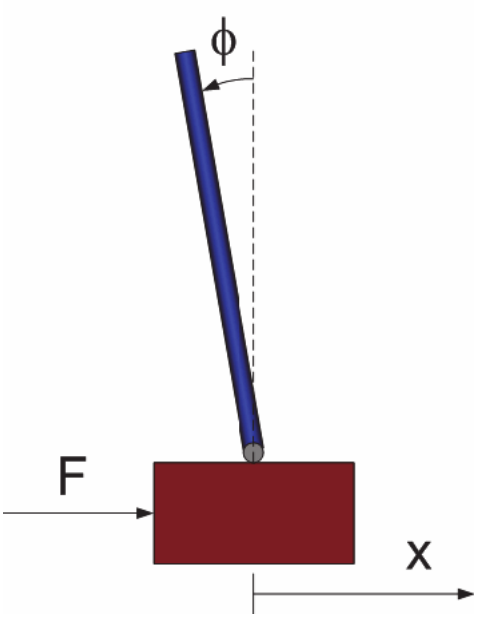
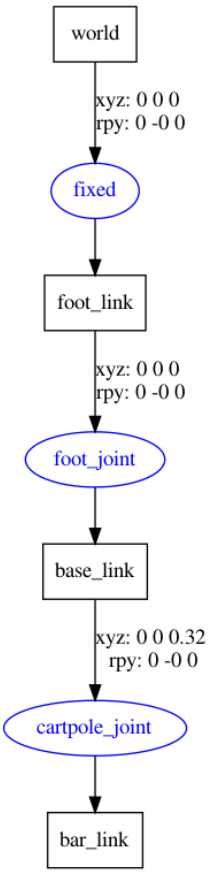


Figure 3.1: **Simple Single inverted pendulum on a cart**

To implement this environment in ROS and Gazebo two main parts to be done:

- Designing URDF, control related parameters and loading model to Gazebo
- RL algorithm, programming and storing the results

The model has 2 main links and some other links for sensors and movement rail. The URDF for the two main links are shown in detail in APPENDIX A, the overall links and main joints definition are shown in figure 2.2:




```
<!-- * * * Joint Definitions * * * -->
    <joint name="cartpole_joint" type="revolute">
        <parent link="base_link"/>
        <child link="bar_link"/>
        <origin xyz="0 0 0.32" rpy="0 0 0"/>
        <dynamics damping="0.0" friction="0.1"/>
        <limit lower="-1.57" upper="1.57" effort="1" velocity="100"/>
        <axis xyz="1 0 0"/>
    </joint>

    <joint name="foot_joint" type="revolute">
        <parent link="foot_link"/>
        <child link="base_link"/>
        <limit lower="-2.0" upper="2.0" effort="2000000" velocity="100000"/>
        <axis xyz="1 0 0"/>
    </joint>
```

Figure 3.2: **Overall URDF file structure and Joint definitions**

Some of the main characteristics of our model are listed below:

- 2.5 kg for main moving cart
- 1.5kg for vertical pole
- Prismatic joint moving in x direction with limit 2.5 meters on each side of X axis
- Revolute joints for vertical pole with limit of +/- of 1.57 radians
- The Rigidness factor Kd has considered higher number to make all parts very rigid
- Bouncing factor Kp and Mu1 &Mu2 (friction factors) have been set as initial values of 1 & 0.5 respectively
- Velocity Controller and Effort Transmission are used to move the cart

Figure 2.3 shows the 3D model of the cart in the Gazebo environment that include a green bar with weight of 2.5kg, rectangular with dimensions of 0.8m x 0.05mx 0.05, a Sensor as a red box with neglectable weight of 100 gram and main moving cart in black color with weight of 5kg and dimensions of 0.4mx0.2mx0.2m.

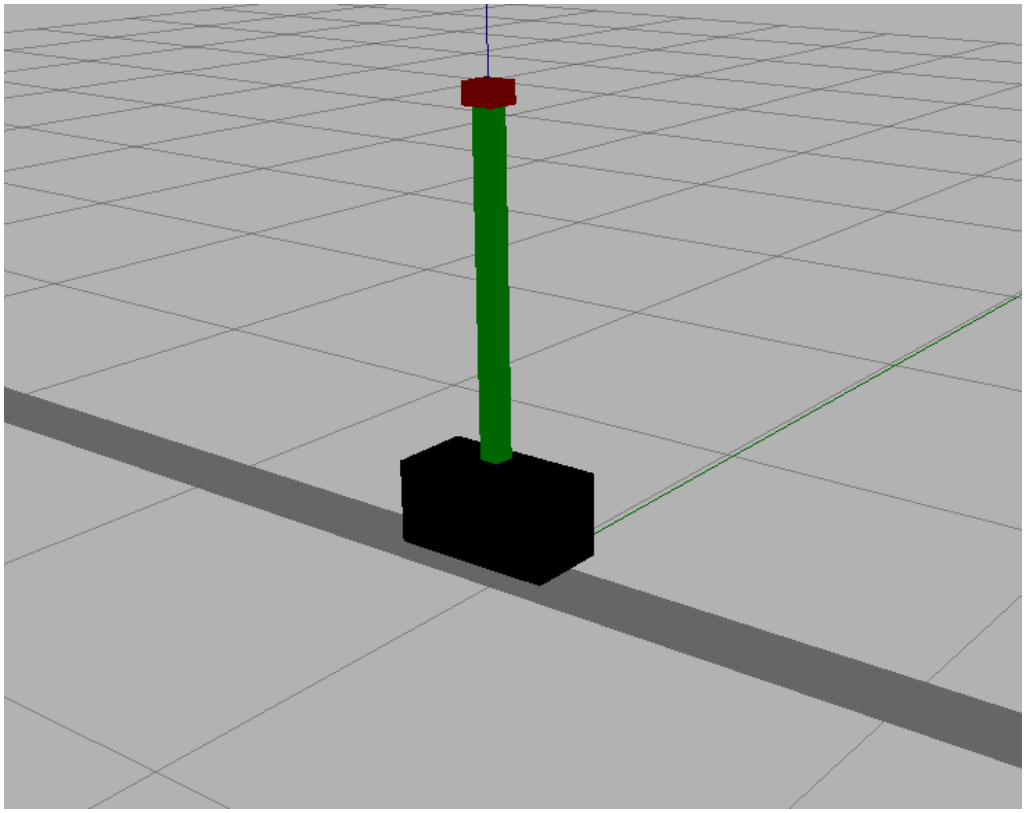


Figure 3.3: **Single Inverted Pendulum in Gazebo Environment**

There are multiple nodes defined for orientation sensor and joint publishers that automatically publishes joint states. Below is the diagram that shows all topics and nodes once we run only gazebo simulation. After running the RL program the node diagram changes because many publishers and subscribers pay roles.

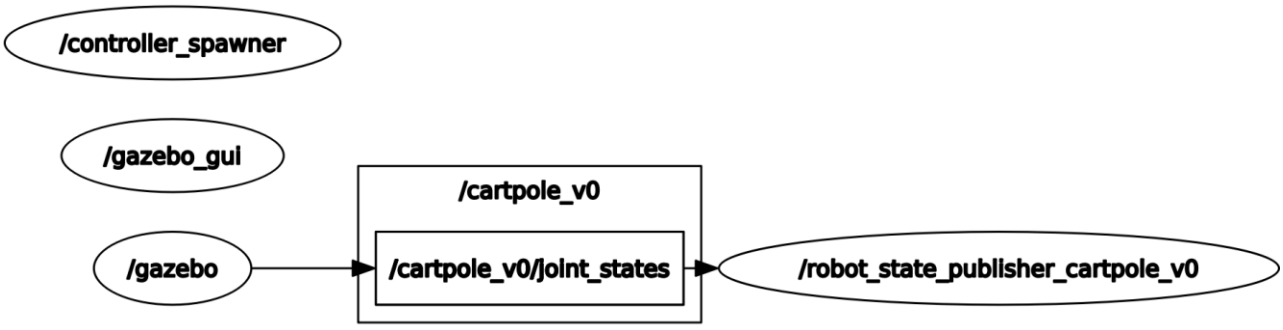


Figure 3.4: **Node list before launching the learning program**

To implement reinforcement learning algorithm that can interact with the Gazebo environment we use Python 3.7 language with some dependencies and libraries, some of the major libraries that used in programming are numpy, rospy, openai_ros, tensorflow. We have also used the latest release of DQN algorithm from the OpenAI baselines.

We have done some changes in existing cartpole environment of OpenAI_ROS to facilitate reusing the code in later stage and more complicated environment. The program consists of 3 main scripts:

- The “Robot Script” that takes care of the communication with the cartpole back and forth to get IMU data for angular velocity, pitch, and roll, and sending velocity commands to the cartpole
- The “Task Script” that take care of reward computation, collect observations, generate actions and initialization the simulation after each episode of training.
- The “Main Script” that loads the baseline DQN algorithm and set all parameters.

These 3 scripts written in class form and Python language and inherit all data and functions from each other. The OpenAI_ROS is a version of OpenAI_Gym that helps a lot implementation of RL algorithm for this case study[20]. The overall structure of the program shows in the figure 3.5.

We consider two factors for the reward function of the reinforcement learning, one is the angle of the pole that is coming from the sensor on the top and the other is the position of

the cart (black box) that is moving left or right. Each time an action is given to the system then reward is calculated, and we check if we must reset the environment to the initial stage or continue learning.

Equation 3.1 shows the reward function, the angle can vary from -1.57 to +1.57 radian as per physical properties so the best reward is when the pole is exactly in vertical position, the cart position can vary from -2.5m to +2.5m so the best reward is given when the cart is close to the center. We used Cosine of the two inputs including weight to calculate total reward. More weight is considered for keeping the pole in vertical position rather than being outside of the zone.

$$Total\ Reward = 1.7 * Cos(Pole\ Angle) + 1.3 * Cos(\frac{Cart\_Posision}{2}) \qquad \textit{Equation 3.1}$$

We also have another limit function to detect when the environment to be reset to start over the next episode, this function checks the limits of the pole angle and position of the cart.

If position of the cart is over 2 meters from the center or angle of the pole is more than 0.35 radian then we considered as failed and environment to be reset, learning is still continue until the defined total steps and episodes reaches or problem solve which is reaching 300 reward as mean square.

For the output to the simulation environment or actions we consider 2 actions to increase the speed or decrease the speed. Negative speed means just speed to the other direction. After running the program all nodes and connection are updated and can be seen in figure 3.6. The **cartpole_gym** is a node that is made by main training program, it receives joint states and IMU data and then it generates velocity commands, commands are just publishing data on another topics that receives by Gazebo Simulation environment. Using ROS enable us quickly to switch from simulation environment to the real robot by just changing the system IP of Gazebo to actual physical environment.

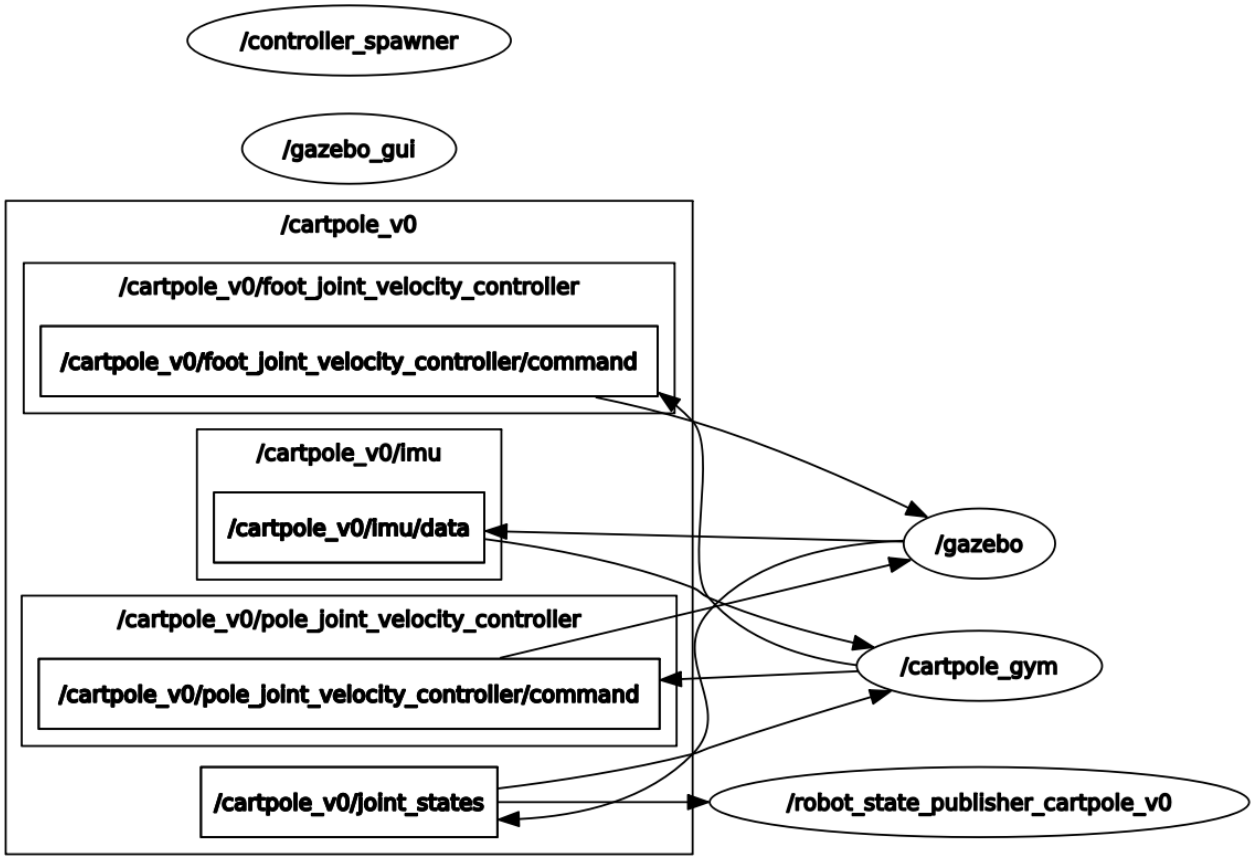


Figure 3.5: **Nodes connections, topics, publishers, and subscribers**

Parameters of the DQN are considered as default, below are the list of the parameters for the DQN algorithm from OpenAI baselines

- Neural Network type: Multilayers Perceptron (MLP)
- Size of the replay buffer = 50000
- Learning rate for Adam Optimizer: 0.001
- Gamma factor over which the exploration rate is annealed=0.1
- Final value of random action probability: 0.02

The observations are the inputs the neural network for estimating the Q values for the output. We have not considered any limit for the velocity of pendulum or cart because it may vary based on different physical characteristics of the system, the table 3-1 just shows

the limit of min max of the observation space but actual angular velocity and angle of the pole will be measured in real time and input the neural network of the learning algorithm.

Table 3-1 : **Observation space for the 1 DoF Inverted Pendulum**

Num	Observation	Min	Max
1	Cart Position	-2.5m	2.5m
2	Cart Velocity	No Limit	No Limit
3	Pole Angle	-0.7 Rad	+ 0.7 Rad
4	Angular Velocity at Tip	No Limit	No Limit

The observations are the inputs the neural network for estimating the Q values for the output. We have not considered any limit for the velocity of pendulum or cart because it may vary based on different physical characteristics of the system, the table 2-1 just shows the limit of min max of the observation space but actual angular velocity and angle of the pole will be measured in real time and input the neural network of the learning algorithm.

The action space is 2 Quality values estimation for moving the cart to the left or right, figure 3.7 shows the overall reinforcement model being created for this case study.

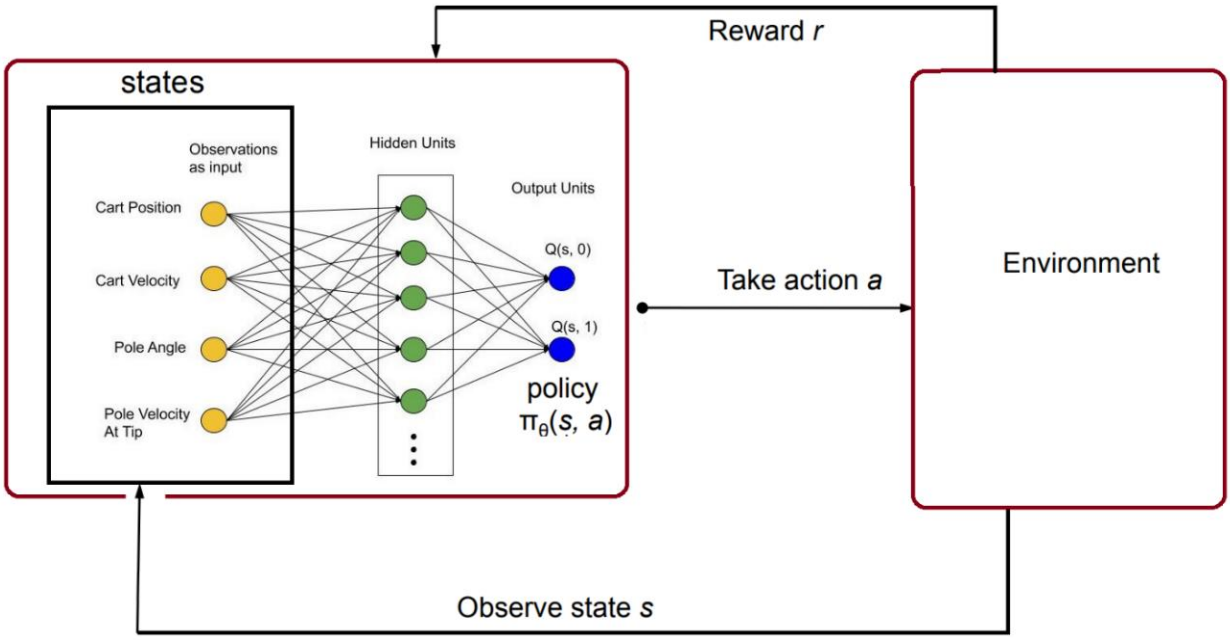


Figure 3.6: **Single DoF cartpole RL Model with 4 observation space and 2 outputs**

To run the entire simulation and training program, we must make some launch files. These launch files are responsible to load appropriate files and script at proper sequence and load preset settings.

The structure of the program is designed to separate learning and simulation in two different packages, that helps to debug and control version of the program much easier and gives ability to use packages for different purposes independently.

Launch files for simulation packages are responsible to load parameters for physical world, robot structure that includes cart and inverted pendulum and its controllers. Launch files for learning packages are responsible to load settings for the program, initiate nodes and run the main learning script. Details of launch files are provided in Appendix B. figure 3.8 shows the structure of the packages for both simulation and learning.

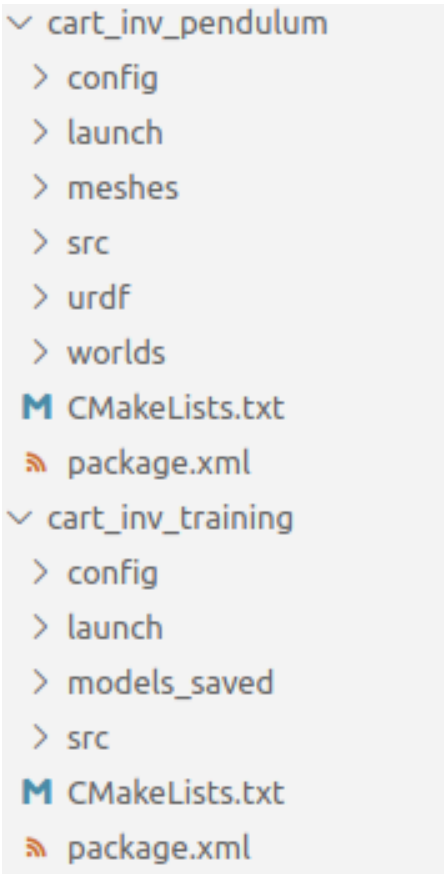


Figure 3.7: **Package structure for simulation and learning**

In addition, since the DQN baseline requires TensorFlow and Python 3 then we made a new Virtual Environment for Python 3.6 and ran the training on that environment. We used **rqt** ROS package for displaying the charts that basically subscribes to the published topics and display them as graph.

3.3 Three DoF Inverted Pendulum on Chassis

Implementation of the single DoF helped to prepare the second and more complex inverted pendulum system. The program structure has not been changed but the complexity of URDF and learning program has changed to accommodate the new environment. In this

phase we made an inverted pendulum that free fall from any direction and the base moves along x and y axis and can rotate along Z axis.

We have implemented 3 DoF inverted pendulum in two different ways. On the first way we considered no joint between the base and the inverted pendulum and in second implementation we used a 3 DoF revolute joint. Figure 3.9 shows the difference of this implementation.

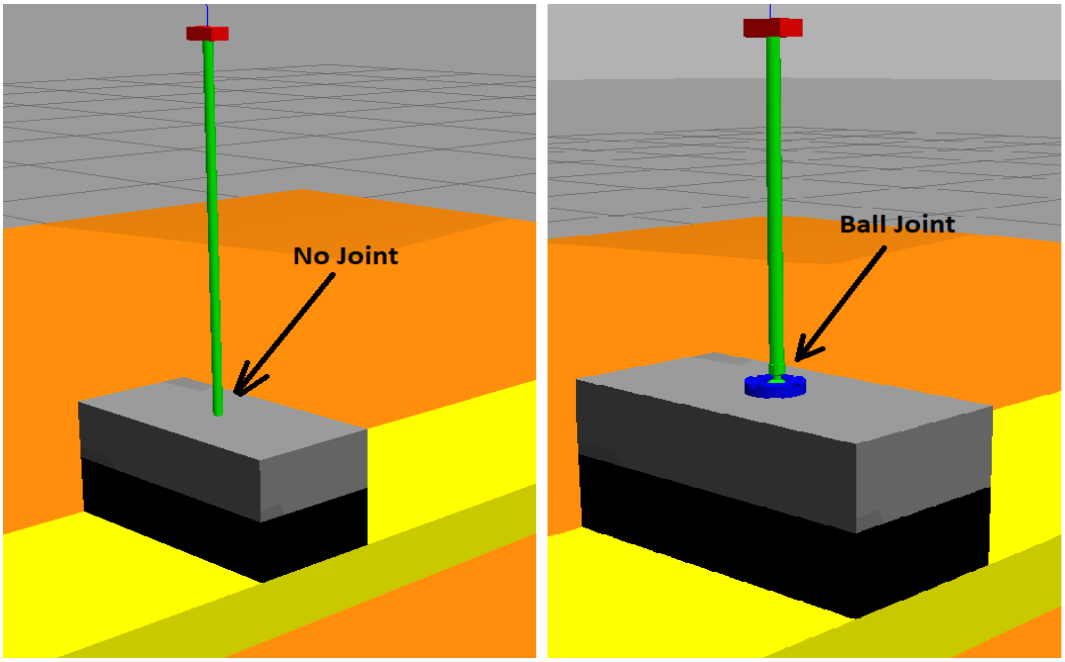


Figure 3.8: **Two ways of implementing 3DoF bar link**

For no joint option, we made two models, one was the chassis that loaded back to back in simulation environment, a bar link with weight of 1.5kg, length 0.8m and radius of 0.07m is standing on the middle of a chassis. The other design uses a ball joint that is designed in the SolidWorks and loaded into the URDF model using mesh identifier, it is defined as revolute joint.

```
<joint name="base_bearing_ball" type="revolute">
        <parent link="base_bearing"/>
        <child link="ball"/>
          <origin xyz="0.0023 0 -0.0005" rpy="0 0 0"/>
          <dynamics damping="0.0" friction="0.1"/>
          <limit lower="-1.46" upper="1.46" effort="1" velocity="100"/>
          <axis xyz="1 1 1"/>
</joint>
```

Figure 3.9: **Ball Joint URDF file**

Ball joint for this experiment is great choice so we made the technical drawing to build this joint for real robot implementation. All technical drawings are in Appendix D and figure 3.10 shows the overview of 3d drawing for the ball joint.

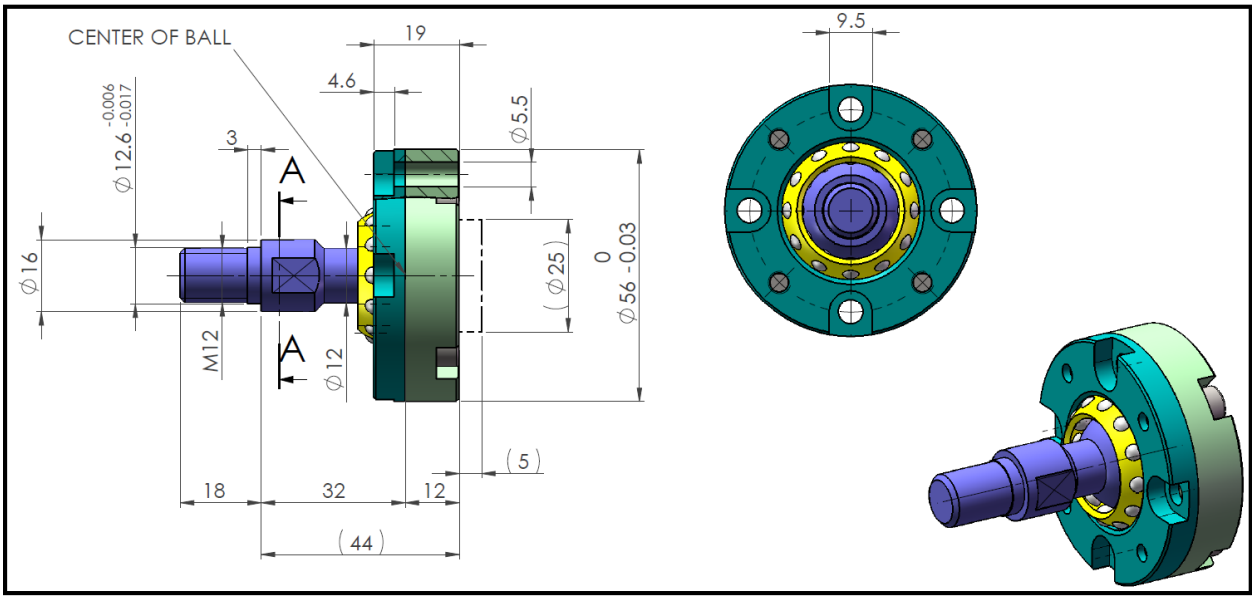


Figure 3.10: **Ball joint design in SolidWorks**

The friction parameters play an important role when using ball joint. If there is less friction then moving the base cannot apply force in any direction to inverted pendulum, ball slip in the bearing base so if inverted pendulum starts falling then there is no way to bring it back to the vertical position. On the other hand, once friction is set too high then it impacts on the nature phenomena of free fall, means when moving the chassis then pendulum moves on that direction.

Gazebos uses Open Dynamic Engine (ODE) by default that is not being changed during the entire case study. As per ODE documentation, the contact joint prevents body 1 and body 2 from inter-penetrating at the contact point. It does this by only allowing the bodies to have an “outgoing" velocity in the direction of the contact normal. Contact joints typically have a lifetime of one-time step. They are created and deleted in response to collision detection. Contact joints can simulate friction at the contact by applying special forces in the two friction directions that are perpendicular to the normal. Mu coefficient zero means no friction at all and infinite number for mu means no slippery.[21]

To control the inverted pendulum, we have extended the concept from previous design, we stacked 3 carts that are called them chassis 0, 1 and 2 in the program and they are on top of each other and joints together, as seen in figure 3.10.

The yellow chassis moves to the X axis, the black chassis moves on Y direction on top of the yellow chassis, and the gray chassis rotate 360 degrees on top of the black chassis with a revolute joint on Z axis. Each of these chassis are 2.5 kg and they are using prismatic joint with each other except the last one which has revolute joint and rotates along Z axis. The limit of 2.5 meters on each direction is applied for chassis that have prismatic joints.

Figure 3.12 also shows links and joints between each chassis.

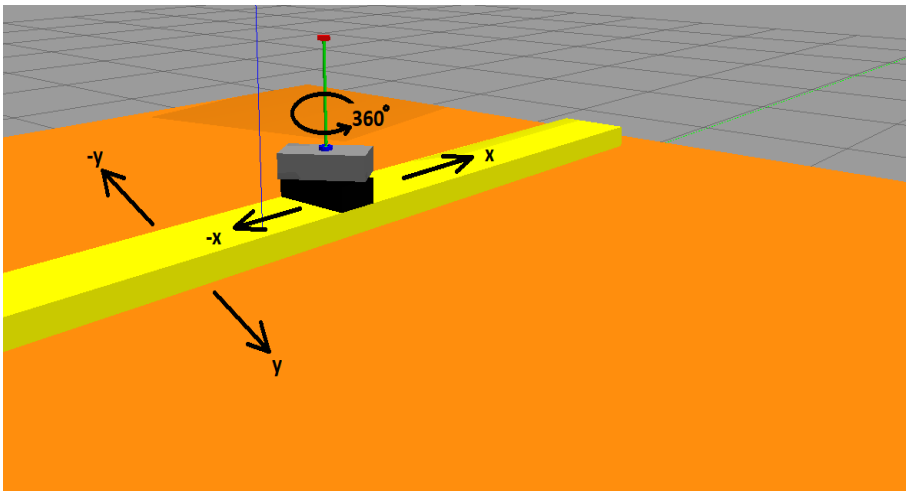


Figure 3.11: **Chassis to control the 3 DoF Inverted Pendulum**

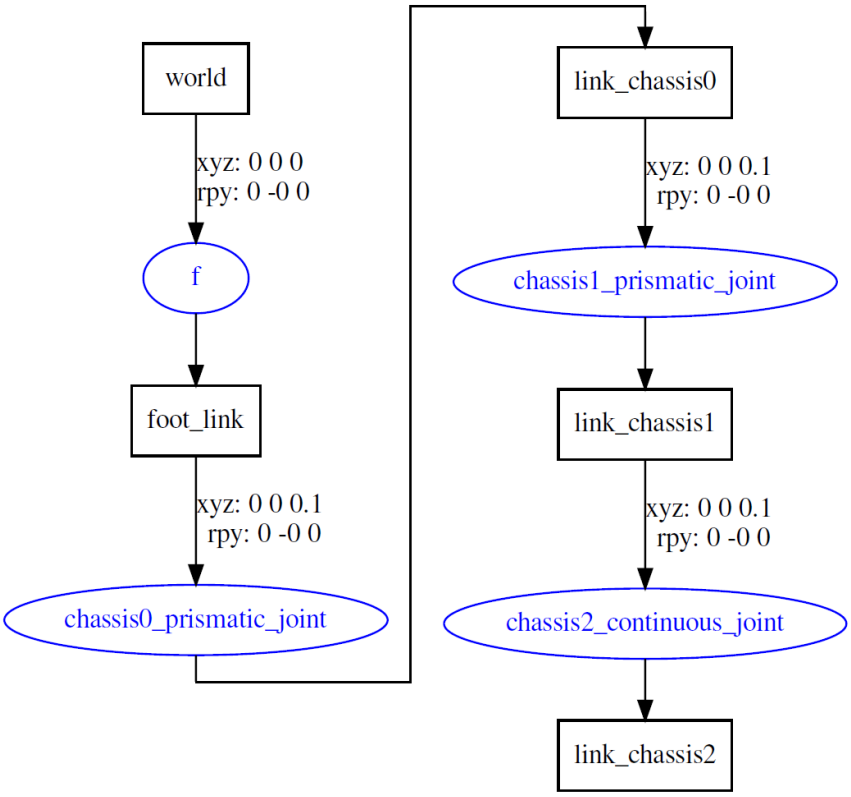


Figure 3.12: **URDF structure of the 3 chassis**

The physical properties in Gazebo such as Kd, kp, for softness and bumpiness of the materials, mu coefficient for frictions are considered in such a way that a force of 20N on any direction, causing inverted pendulum to fall on that direction for both cases; no-joint and ball joint.

The 20N force impact equally on both designs to make sure learning results of these two models are valid for comparison. The complete URDF file with details of all parameters are in Appendix C.

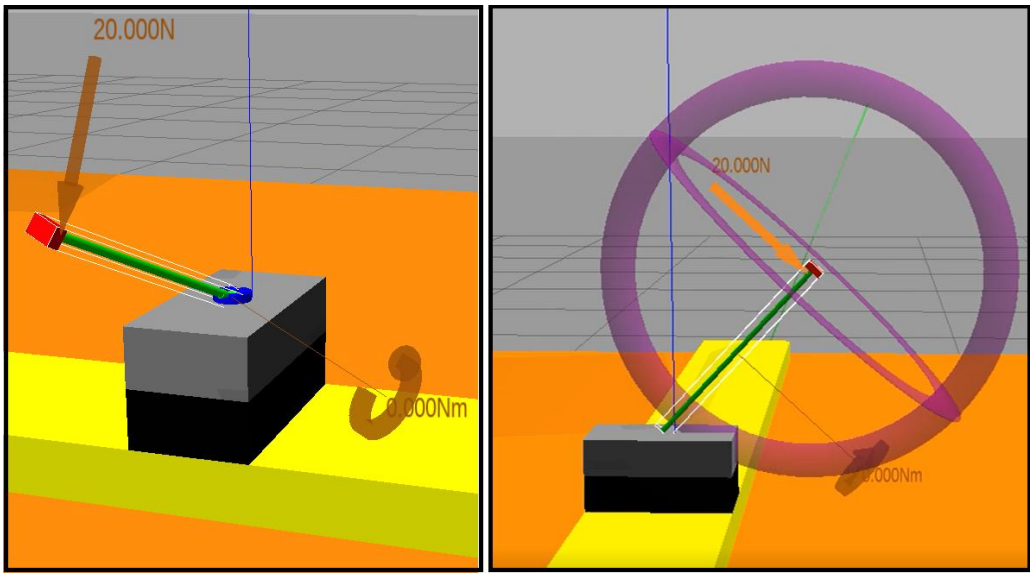


Figure 3.13: **Applying 20N force in any direction at initial state**

Implementation of DQN is almost same as previous single form of inverted pendulum, we have same IMU sensor on the top of the bar link that transmit the angular velocity, pitch, and roll. Observation and action space are different than single DoF. For observation we get position and speed of each chassis, pitch and roll and their angular velocity totally an array with 10 elements as per table 3.2.

Table 3-2: **Observation space for the 3 DoF Inverted Pendulum**

Num	**Observation**	**Min**	**Max**
1	Chassis 0 - Position	-2.5	2.5
2	Chassis 0 - Velocity	No Limit	No Limit
3	Chassis 1 - Position	-2.5	2.5
4	Chassis 1 - Velocity	No Limit	No Limit
5	Chassis 2 - Position	-2.5	2.5
6	Chassis 2 - Velocity	No Limit	No Limit
7	Pole-Pitch	-0.7 Rad	+ 0.7 Rad
8	Pole-Roll	-0.7 Rad	+ 0.7 Rad
9	Pole Angular Velocity	No Limit	No Limit
10	Roll Angular Velocity	No Limit	No Limit

For action space we command each chassis to increase, decrease or no changes to the chassis speed. In fact, 3 outputs command that each can have 3 different outputs that leads to totally 27 actions. Messages or data which are float64 for our case study are published as topic with different nodes.

The main Python algorithm has a node name **/Cart-Inv-Gym** that communicates with all inputs including IMU sensor and joints feedback and provides output to the controller that is eventually connected to the Gazebo Sim. Figure 3.14 shows all nodes and topics communication when script and Gazebo simulation are running.

As seen in this figure, the **cart_inv_gym** is main node from Python program that runs the DQN algorithm, it is responsible to get data from IMU sensor and compute and send command to the chassis. Chassis commands published in commands topic that are consumed by Gazebo, Gazebo perform action in the realistic environment and provide feedback through **join_state**.

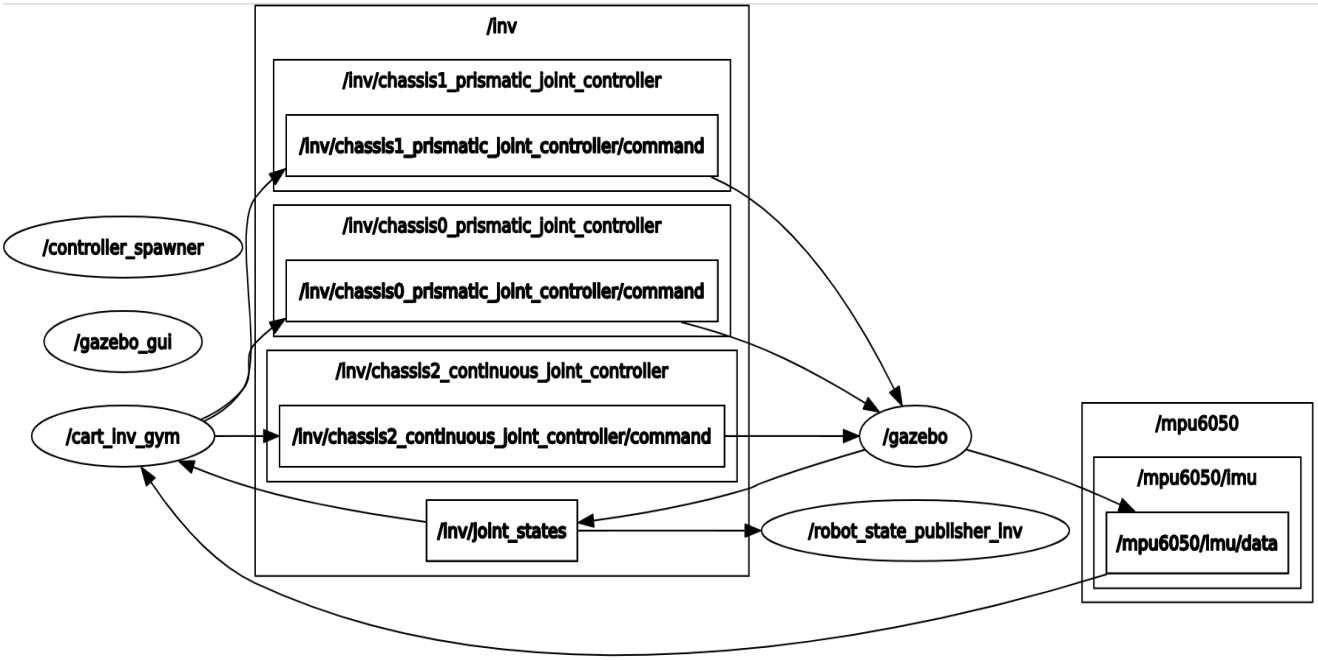


Figure 3.14: **ROS Nodes / Topics for the 3DoF system**

The structure of the Python scripts is also same as single DoF Inverted pendulum however with some changes in regard of sending commands to 3 chassis instead of one, checking extra incoming signals, reward and done functions. Figure 3.15 shows the overall structure and program block for the 3DoF inverted pendulum.

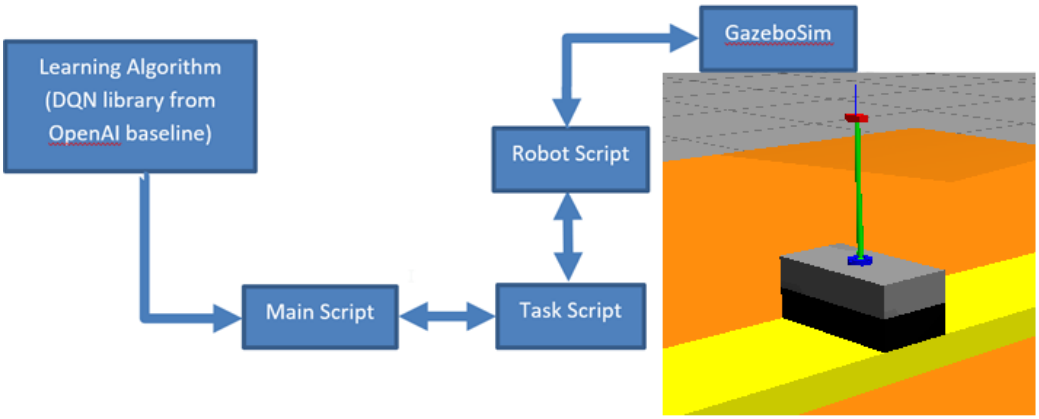


Figure 3.15: **3DoF Inverted Pendulum, Program Blocks**

For the reward, we are using Cosine of Chassis Position and angles of pitch and roll but considering some weight like previous case study. We consider more weight for keeping the pole upward and less for the limit of chassis. Figure 3.15 shows the reward function that is used for this system.

40 percent of weight goes for keeping the bar link vertical, but the 10 percent considered for keeping the chassis close to the center.

```
if not done:

    reward_pitch=math.cos(pitch)*1.4 #pitch_or_roll_reward_weight
    reward_roll=math.cos(roll)*1.4  #pitch_or_roll_reward_weight
    reward_x_position= math.cos(x)*1.1
    reward_y_position= math.cos(y)*1.1
    reward=reward_pitch+reward_roll+reward_x_position+reward_y_position
```

Figure 3.16: Reward function for 3DoF Inverted Pendulum

3.4 Three DoF Inverted Pendulum on Robotic Arm

Learning path and challenges of previous case studies, made the implementation phase much easier for the last and much more complex system. In this phase we used universal robot to implement inverted pendulum control. We have used same spherical joint with exact same physical properties and parameters for weight, frictions, and other coefficients. The base bearing attached to the universal robot end-effector as seen in figure 3.17

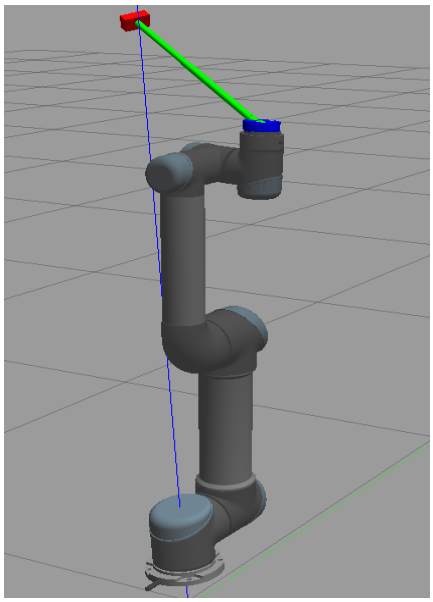


Figure 3.17: **Universal Robot with Inverted Pendulum on spherical joint**

One of the main challenges we faced at the beginning was selecting right version and had to change couple of times to find the best one that works for majority of libraries and in some cases had to change the direction of AI programming too because of some packages were not updated at the time of writing this thesis.

Our main driver to select the ROS version, was The Universal Robot package that is officially released by Universal Robot Company. The ROS package for UR is based ROS-Kinetic or ROS-Melodic which needs to be installed on Ubuntu 16.04 or Ubuntu 18.04 operating system. These operating system uses Python V2.7 while for some of the packages for example, AI, we had to use Python 3.6 as minimum. The version of ROS that is used for this thesis is Melodic that is installed on Ubuntu 18.04 however to use TensorFlow and OpenAI baseline for machine learning part of the thesis, we used Virtual Environment package in Python and runs those part of codes under a virtual environment that uses Python 3.6.

For the robot simulation in Gazebo environment, we used the library from ROS_Industrial that is open source and available for commercial or research purposes [22]. The only part that we changed in the Robot model was the initial position, limits on the angels and attached the inverted pendulum to the end effector. This is done by making a separate URDF file for the inverted pendulum, making mesh folders in the Redescription package and make a fix joint between bearing and end effector.

The robot model has trajectory controller to move all joints to the desired position with proper speed and direction. The trajectory controller is part of the simulation package made by the universal robot and we have not manipulated it. So, for the purpose of this study we used trajectory command that basically let robot find best way to move its arm for reach the desired angles for each waypoint.

In our use case we only want to move from a current angle of the joints to a new set of angles, so we only send command for 6 joints as an array. The 6 joints of the robots are shown in figure 4.2 and the array for joints are in radian and float64 format:

['elbow_joint', 'shoulder_lift_joint', 'shoulder_pan_joint', 'wrist_1_joint', 'wrist_2_joint', 'wrist_3_joint']

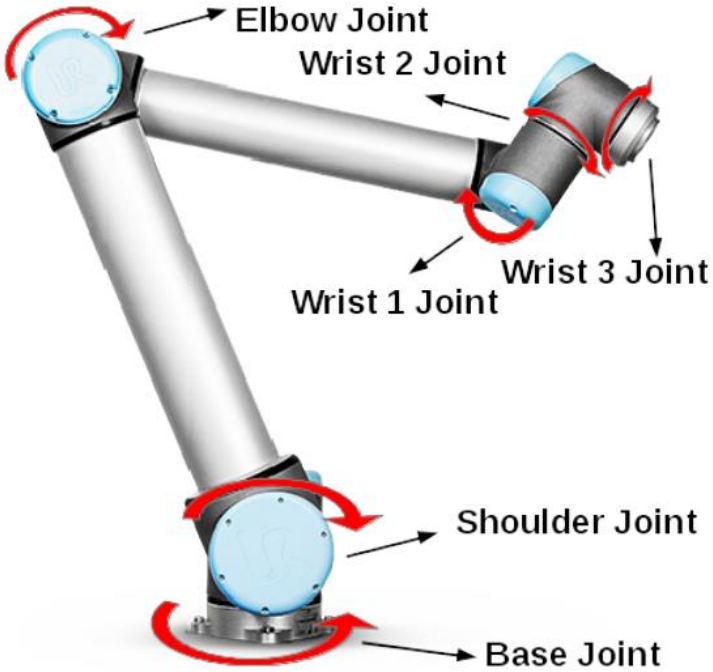


Figure 3.18: ***6DoF Robot Arm and its joints*** **[15]**

We have considered an angle limit for each joint during training because if joint freely moves 360 degrees then the inverted pendulum hits on the ground or other part of the robots arm and breaks, hence entire training to be restarted.

For Elbow and Shoulder Lift joint, we considered 0.78 radian or 45 degrees freedom. Wrist-1 joint can move between -1.57 radian to +1.57 or +/- 90 degrees, and wrist 2 has limit between 45 degree to 135 degrees. Figure 3.19 shows once all joints are in their limit, we see in this figure that there is no crash point for inverted pendulum and ground.

We have also lifted the robot by 10 centimeters because when robots loaded at each episode then it does not hit the ground. In real world, robot can be placed on a stand then on the table.

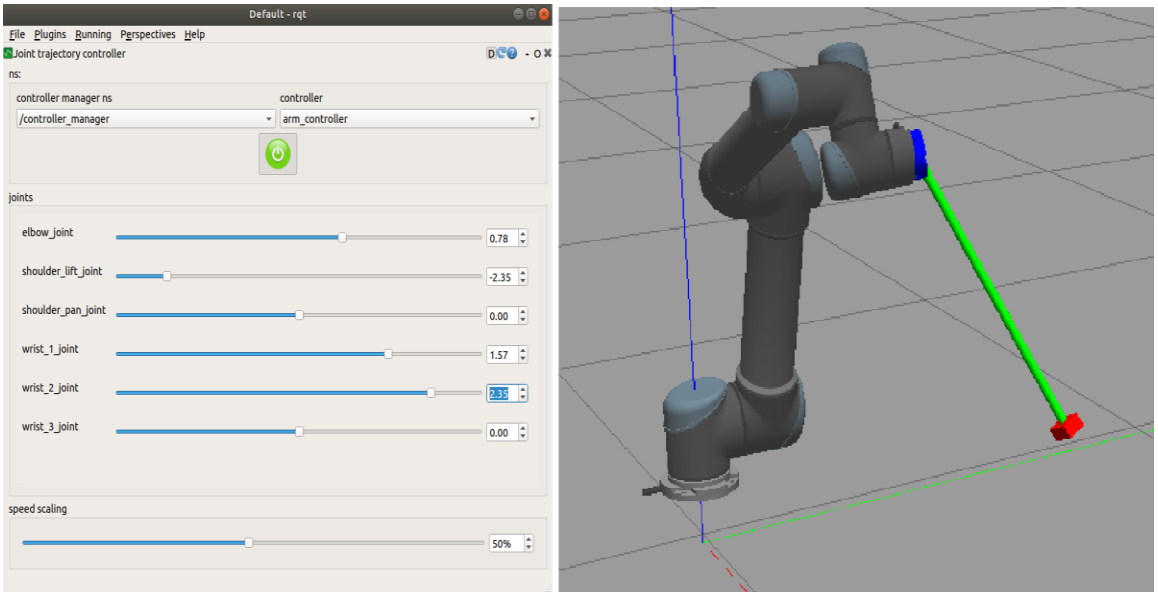


Figure 3.19: **Robot at its maximum joint limits**

Table 3.3 shows the input for the DQN algorithm, as seen in this table the information from the IMU sensor remains intact in compare of previous design and we still get the pitch, roll and angular velocity of each but we also get all joint positions of robot arm.

Table 3-3: **Input observation space for 3DoF on robot arm**

Num	**Observation**	**Min**	**Max**
1	Pitch Angle	-0.7	0.7
2	Pole Angle	-0.7	0.7
3	Pitch Angular Velocity	No Limit	No Limit
4	Pole Angular Velocity	No Limit	No Limit
5	Elbow Joint	-0.78	0.78
6	Shoulder Lift Joint	-0.78	0.78
7	Shoulder Pan Joint	No Limit	No Limit
8	Wrist1 Joint	-1.57	1.57
8	Wrist2 Joint	0.78	2.357
10	Wrist3 Joint	No Limit	No Limit

The output space is action to 6 joints either increasing, decreasing or no changes. So 3 different actions for 6 joints leads to have space of 3 power 6 which is 729 different of actions. Figure 3.20 shows all nodes and topics when the learning program and Gazebo are started.

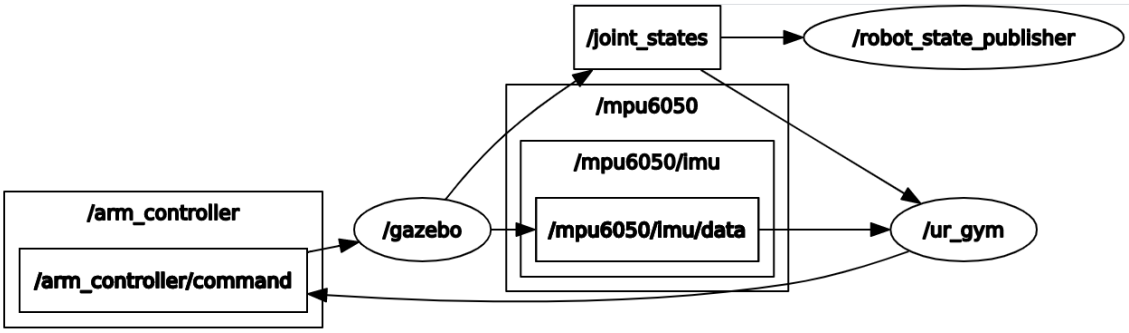


Figure 3.20: **Node diagram of robot arm and inverted pendulum**

As seen in figure 4.4, the joint state publisher, publishes all joints information from robot arm and **/arm_controller** node contains all actions to send command to the joints of the robot. It uses the trajectory command message format that contains joint angles and time of the goal.

The node **/ur_gym** is the main node of the Python learning program, it gets data from the joint states and IMU sensor and send commands to the controller of the robot, the commands perform action in the Gazebo and new action will be generated based on the new state.

The “done” function is like the previous case studies, but with the difference that we applied limit to the robot joints instead of 2.5 meters limit for chassis. If an action goes beyond the limit the training goes to “done” state or if the pendulum falls and have angle of more than 0.7 degrees then considered as done for that episode, accumulated reward will be shown in graph and new episode starts.

The reward function has not been changed from what shown in figure 3.16 in previous case study. We take pitch and roll and consider weighing factor to calculate total reward at each step.

3.5 Hardware configurations

In this section we describe in detail how to build an orientation sensor and integrate it to ROS environment to read roll, pitch and their acceleration of the inverted pendulum which are used in the learning algorithm as input.

We also explain details of communication to the universal robot and setting the hardware for ROS communication.

3.5.1 IMU Sensor

The inertial measurement unit (IMU) for our use case should have following properties in our case study as to minimize any negative impact on the learning process:

- Wireless communication
- Accurate reading
- Battery operated
- Small and light

There are many methods to get the orientation of a device in 3D space using combination of gyroscope, magnetometer, and accelerometer with combination of using filters such as Kalman or complementary.

The intention of this paper is not developing new method of measuring orientation but instead using an existing reliable method to build a simple and reliable sensor that can be used for this case study. For that purpose, we have chosen sensor fusion technique that is a simple method and mainly used in commercial quadcopters balancing and it is simple, reliable, and accurate enough for our case study.

The main idea of sensor fusion is to get more accurate reading by combining information from multiple sensors. On 2017 a paper published in IEEE and proposed an algorithm to get orientation of an object in 3D space using Gyroscope and Accelerometer which is used in our design [23].

Gyroscope measures rotational motion and its output is degree per second. If angular velocity is monotonous then we can say:

$$\theta = \omega \cdot t \quad \textit{Equation 3.2}$$

Where θ is angular displacement and ω = Angular Velocity and t is time. But in practical scenario, angular velocity is not constant with time and it varies so basically angular velocity is rate of change of angular displacement:

$$\theta' = \frac{d\theta}{dt} \quad \textit{Equation 3.3}$$

By integrating from both sides over a period we can find θ

$$\theta = \int_0^t \theta'(t)dt \cong \sum_0^t \theta'(t)T_s \quad \textit{Equation 3.4}$$

Because we cannot take a perfectly continuous integral, we must take the sum of a finite number of samples taken at a constant interval T_s. This approximation will introduce errors specially when gyroscope data changes faster than the sampling frequency, so the integral approximation will be incorrect and this error is called gyroscope drift and increases over time.[24]

So, gyroscope alone, cannot provide accurate angular velocity and angles on each axis (Pitch, Roll), mainly because of gyroscope drift.

In the proposed method a tri-axial gyroscope is used as a main source of information, and a tri-axial accelerometer is used to compensate drifting deviation on gyroscope measurements.[25]

MEMS accelerometer measures acceleration due to movement and gravity. For a static object acceleration is due to gravity (1g). We can find the orientation from the accelerometer and use that data to compensate the drift of Gyro. [26]

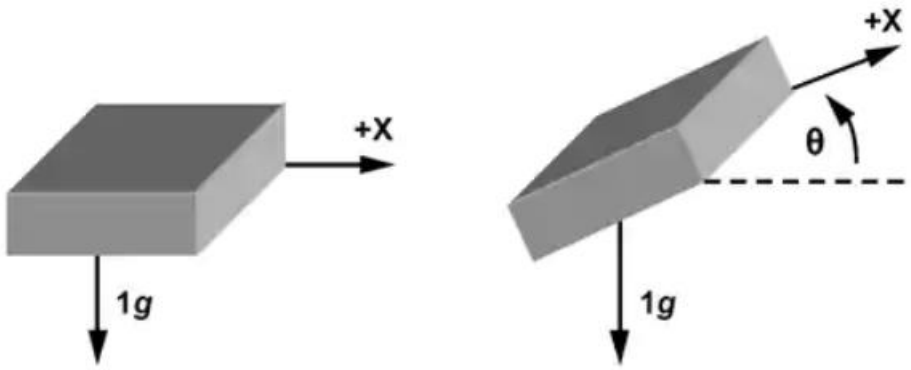


Figure 3.21: **Singe Axis Accelerometer**

Referring to basic trigonometry, the projection of the gravity vector on the x-axis produces an output acceleration equal to the sine of the angle between the accelerometer x-axis and the horizon, in figure 3.21 the θ can be calculated by following equation:

$$A_{x,out(g)=1g*\,Sin(\theta)} \qquad \textit{Equation 3.5}$$

When 3 dimensions accelerometer is used then the angles can be calculated using following formulas [24]

$$\theta = tan^{-1}\left(\frac{A_{x,out}}{\sqrt{A_{y,out}^2 + A_{z,out}^2}}\right) \qquad \textit{Equation 3.6}$$

$$\psi = tan^{-1}\left(\frac{A_{y,out}}{\sqrt{A_{x,out}^2 + A_{z,out}^2}}\right) \qquad \textit{Equation 3.7}$$

$$\varphi = tan^{-1}\left(\frac{\sqrt{A_{x,out}^2 + A_{y,out}^2}}{A_{z,out}}\right) \qquad \textit{Equation 3.8}$$

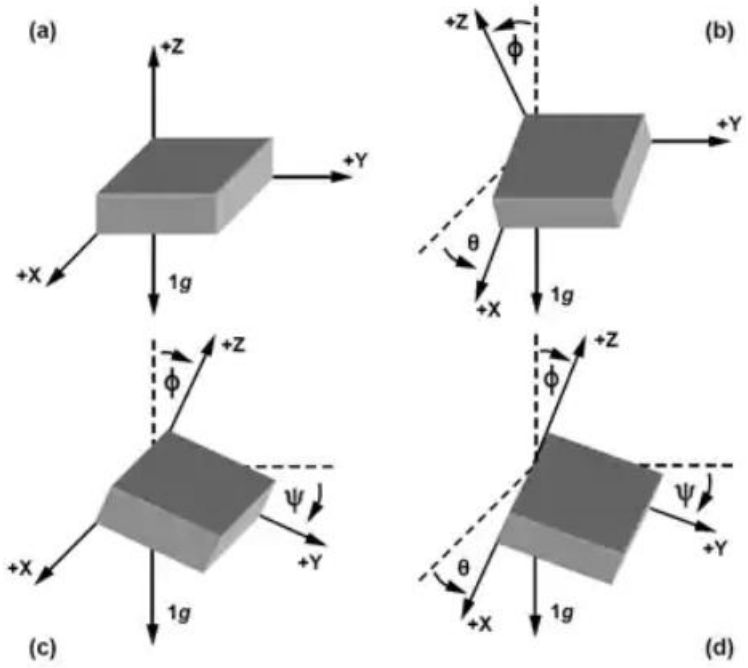


Figure 3.22: **Angels for independent inclining sensing**

We have used MPU6050 for making orientation. The MPU-6050™ is motion tracking module that is designed for the low power, low cost, and high-performance requirements of smartphones, tablets, and wearable sensors.

It is a micro-electromechanical system (MEMS) with 3-axis Gyro/Accelerometer combined into a single chip is also called six-axis motion tracking. We can adjust the reading range for both gyroscope and accelerometer which has direct impact to its accuracy. This chip has built in I2C communication port to communicate for external micro controller.

For micro controller part, we used ESP32 that is a wireless system on Chip (SOC) module with 4MB memory and 512KB Ram. This SOC works on battery voltage 2.7Vdc to 3.7

Vdc, and it has Bluetooth, WIFI, SPI and I2C communication port with some general-purpose Input/Output (GPIO) for interacting with other devices. This SoC fits to our requirement and with combination of ROS-Serial driver we can transfer data wirelessly between ESP32 and WIFI module on the laptop. Figure 3.23 shows the architecture of the entire system.

Figure 3.23: **Overall architecture for data acquisition**

With this MEMS sensor and a fusion algorithm, we can accurately measure pitch and roll of the inverted pendulum which is enough for our learning algorithm. It is good to mention that yaw cannot be calculated accurately with this sensor and a magnetometer as an additional measurement is required.

Registers in the initialization parts to be adjusted to give a proper range of reading. MPU6050 has 4 ranges for the accelerometer and 4 ranges for the Gyro. The accelerometer full scale range can be set to $\pm 2g$, $\pm 4g$, $\pm 8g$ *and* $\pm 16g$. We selected the lowest range $\pm 2g$ for our inverted pendulum application since we would not get even close to this acceleration, because the inverted pendulum is close to free fall. The sensitivity as per datasheet is calculated by least significant byte LSB divided by range so in our case for the range of 2g we get sensitivity of 32750/2g= 16384 LSB /g or 0.0006 m/s^2.

The gyro range is selected as minimum as well $\pm 250°/s$ or 41.6 RPM. As per datasheet, the sensitivity can be calculated by full scale reading value 32750 / 250 = 131 measurement units per degree per second.[27]

Table 3.4 shows the sensitivity of acceleration and gravity per different range setting.

Table 3-4: **Accelerometer and Gyro sensitivity and range**

Angular Velocity Limit	Sensitivity		Acceleration Limit	Sensitivity
250°/s	131 LSB/°/s		2g	16834 LSB/g
500°/s	65.5 LSB/°/s		4g	8192 LSB/g
1000°/s	32.8 LSB/°/s		8g	4096 LSB/g
2000°/s	16.4 LSB/°/s		16g	2048 LSB/g

The algorithm consists of two parts, first part is the initialization where we initialize communication and registers as well as measuring the gyro-drift at steady state condition for 1000 readings. Second part is main loop function of the Gryo & Accelerometer then combining them together. We use 96% of reading from Gyro and 4% reading from accelerometer to compensate. Figure 3.24 shows the algorithm in form of blocks:

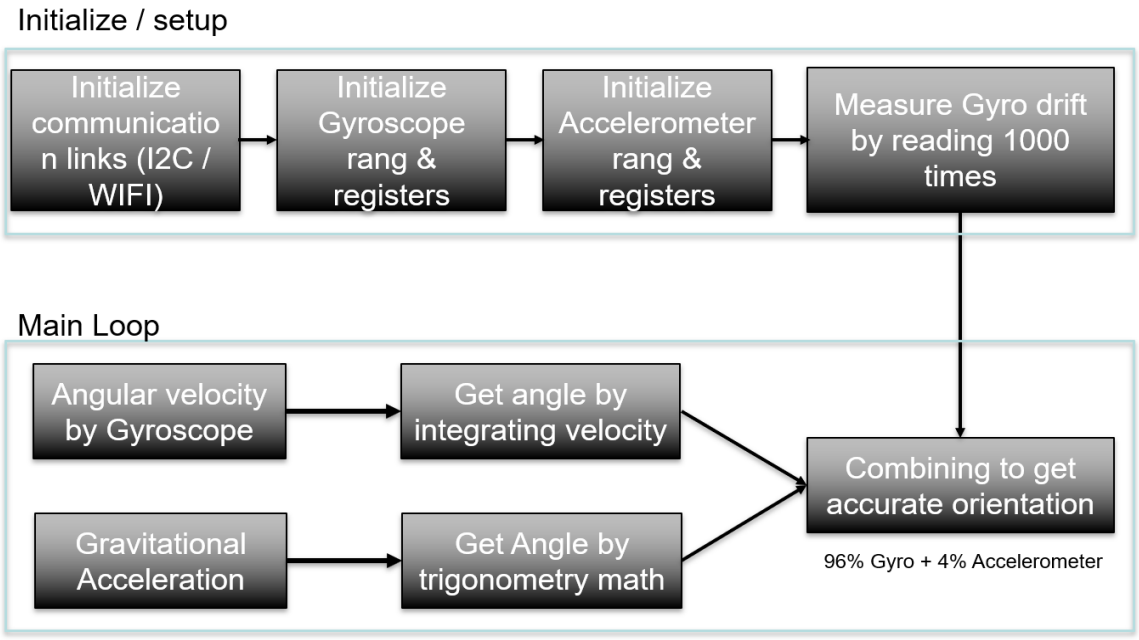


Figure 3.24: **Sensor fusion algorithm with Gyro-Accelerometer**

The rosserial protocol is a point-to-point ROS communication over a serial transmission line. This is one of the open-source packages under BSD license that enable serialization/de-serialization as standard ROS messages, simply adding a packet header and tail which allows multiple topics to share a common serial link. The number of Publishers and Subscribers are limited at 25, and the size of serialization and deserialization buffers are limited at 512 bytes by default for **rosserial_client.** This limitation is not a challenge since 4 values of float 32 bits are enough to get values of pitch, roll and their accelerations.

ROS Serial provides a library ros.h that can be used in C++ programs inside the micro-controller and enable to call the classes and functions to establish node communications for diagnostics, publishing and subscribing at speed of 57600 bits/sec.

Figure 3.25 shows the nodes that are made after establishing communication to the MPU6050 using microcontroller.

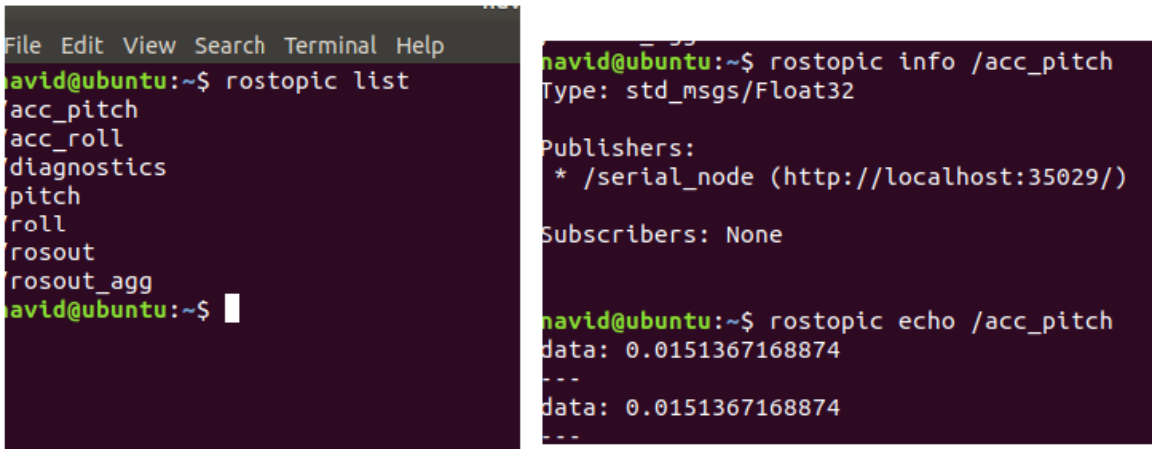


Figure 3.25: **List of all topics after and output of acceleration node**

Pitch and Roll are published in degree unit and acceleration on m/s^2. Figure 3.26 shows acceleration of the aitch as a curve with **rqt_multiplot** module of ROS platform. The x axis of the graph is in milliseconds and Y-axis is the acceleration of the Pitch. We have rotated the device 90 degrees in clockwise and counterclockwise and results shows the gravity acceleration that is sensed by sensor. There is some noise when device is in flat position for the pitch and roll acceleration however this amount of noise is neglectable for learning application since we have, or “reward” function set to give reward when the inverted pendulum is in -0.25 radian to +0.25 radian.

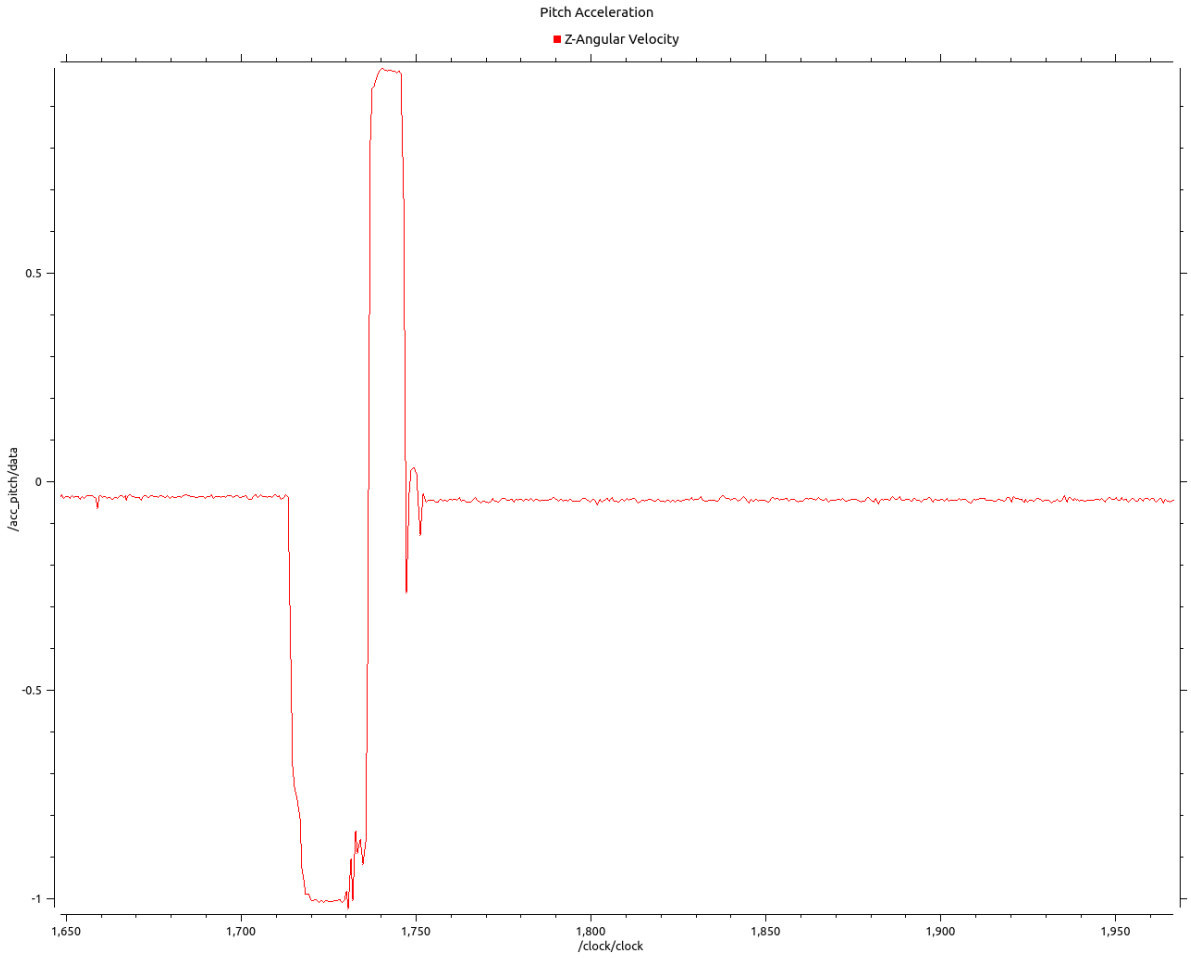


Figure 3.26: **Acceleration on X axis (Pitch)**

3.5.1 Robot Arm Setup

ROS communicates to many robots if the driver supplied from the robot manufacturer. There are more than 500 robots are supported and are listed in the ROS Wiki page [28], these are included Robot arms, humanoid robots, autonomous robots, drones and industrial robots such as Fanuc, Universal Robot, ABB and so many more.

To connected the program to the universal robot as an example, first the driver for ROS communication to be found in ROS-Industrial package, it communicates to the Universal Robot via Ethernet communication port [29]. When communication is established the

ROS-Industrial package start communication with a program inside the robot that is called URScript, which is universal robot own's Python-Like scripting language. This program has main duty to handshake with ROS and interpret ROS messages to hardware commands. ROS driver has a test command to bring the robot to the vertical position and starts with **rosrun ur_driver test.py**.

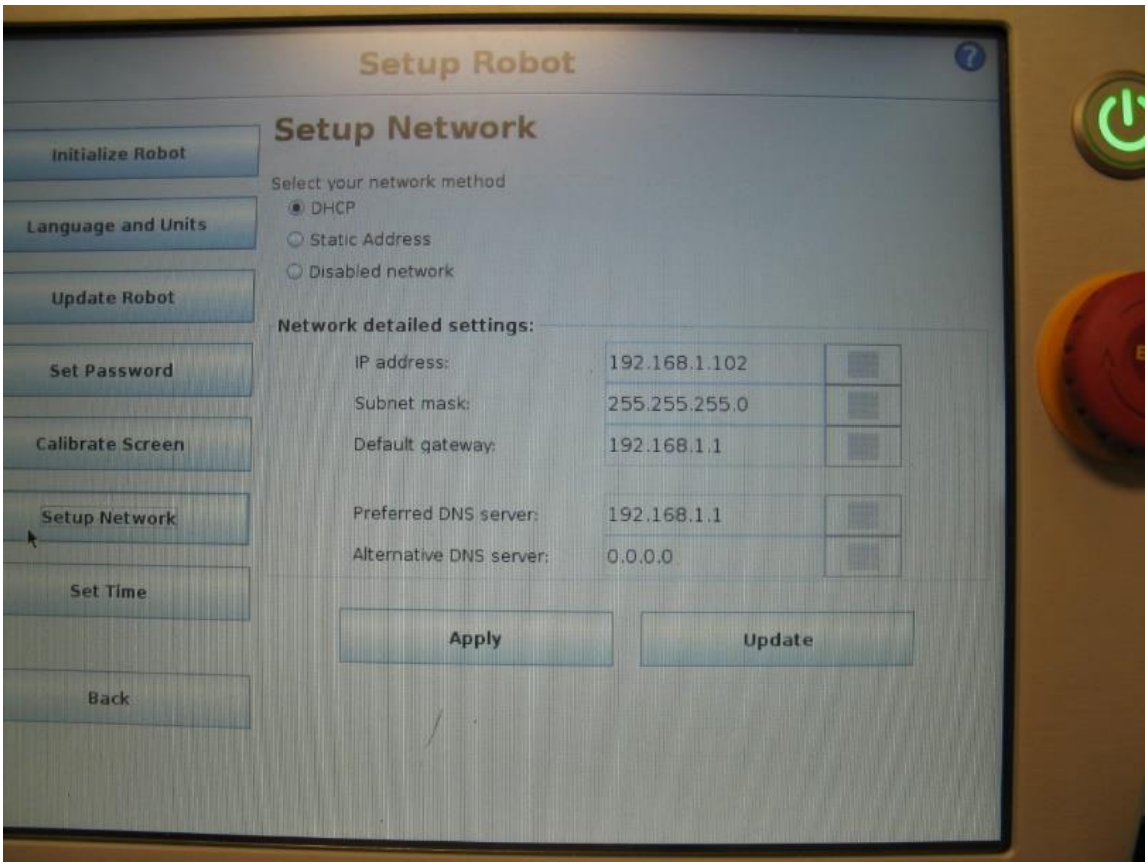


Figure 3.27: **Ethernet setting of Universal Robot**

CHAPTER 4 EXPERIMENTAL RESULTS

4.1 Introduction

In this section we review results for each experimental phase. Table 4-1 shows the detail of each case study and experiment that carried out on each phase.

Table 4-1: **Break down of main case study in 3 phases and expectations**

	Phases	Experiment on each phase
1	One DoF Inverted Pendulum on a single cart	1. Comparing results with previous papers 2. Apply turbulence to the model 3. Use the trained model on new inverted pendulum with different physical properties
2	Three DoF Inverted Pendulum on 3 chassis	1. Attach inverted pendulum with and without Ball Joint to Chassis 2. Apply turbulence to the pendulum and observer learning speed
3	Three DoF Inverted Pendulum on a 6 DoF Robot arm	1. Use exact same inverted pendulum design on 6 DoF robot 2. Train the model to balance the pendulum

4.2 Results of one DoF Inverted Pendulum

As per figure 4.1 is the curve of reward vs episode shows after about 400 episodes we reached to a stable level of control. Each episode means system controlled the pendulum until it fells (angle reaches more than 0.70 radian +/- 40 degrees) or the cart reaches to the end of its limit which 2.5 meter on each side.

Gazebo simulation automatically calculates a simulation time and real time by considering all wasted time for sending commands, receiving commends and spent time for calculations. Based on stops between each episode, that is set manually for initialization purposes, about 100 rewards is equal 5 seconds of real time. In other word, 1000 rewards means balancing the inverted pendulum for about a minute.

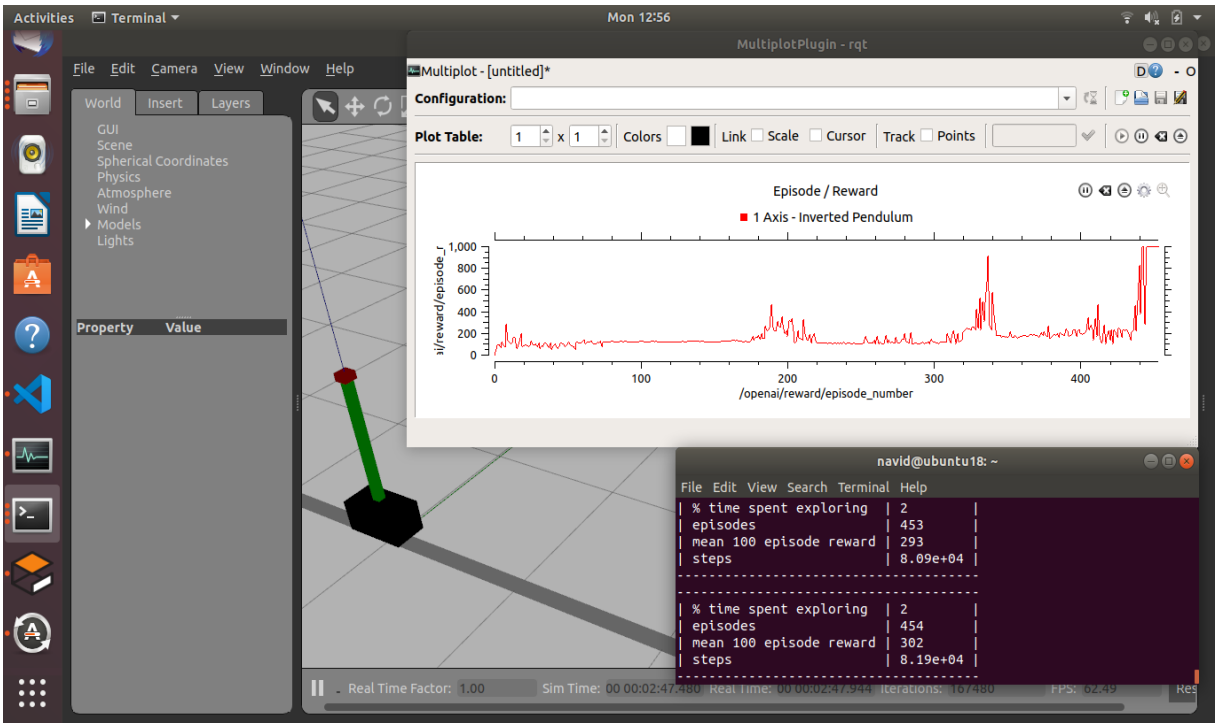


Figure 4.1: **Result of training RL for Single Inverted Pendulum without any disturbance**

This chart is very similar to what is published as result on the first literature review that is discussed on chapter 2.

To test the robustness of the learnt model we applied forces to the inverted pendulum. Gazebo simulation environment has feature to manipulate settings or apply forces to the robot and environment without stopping simulation. As seen in figure 4.2 force direction and its amount can be adjusted during simulation or at the beginning.

We applied forces at the steady state situation first, to see how much force pushes the pendulum in one direction. The physical parameters of the structure such as weight, length and friction at the pivot point are in such a way that applying only 50N at steady state situation causing the pendulum to fall off.

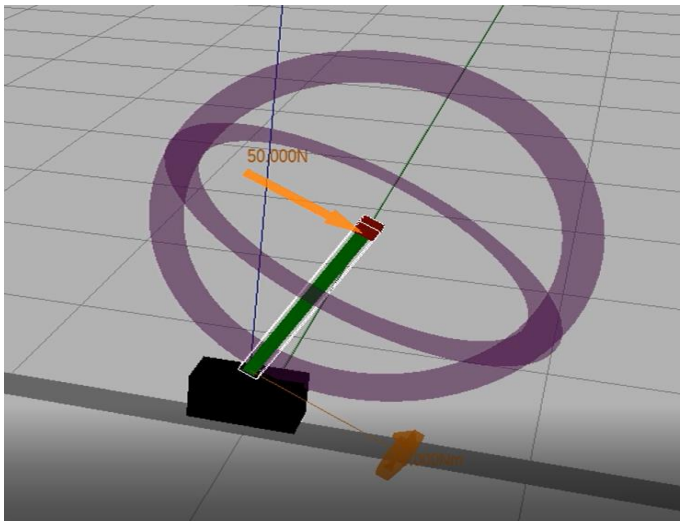


Figure 4.2: **Applying 50N force to the pendulum on the tip**

We started learning process for longer period and added turbulence randomly while training. We have started with 10N force applied to the tip of the inverted pendulum and increased until 50N over time and randomly. Figure 4.3 shows the chart of reward when force is applied as turbulence in the system. We see that at the beginning the system was confusing and unable to control the pendulum, a very wavy chart shows system was unstable but over time learnt to control.

As shown in Figure 4.3 after 400 episodes, we reached more stable control and reward stopped fluctuating. System starts learning to control while turbulence still applied randomly. After episode 800, we reached to reward 600 means 30 second keeping the inverted pendulum in upward position. After that applying 50N forces continuously but changing it directions, system still achieved good control.

This means we were able to make more robust model by applying forces during training and showed how initial learnt model was very primitive controller and unstable to any

turbulences. The leant model improved itself by being exposed to changes in environment and started adopts itself.

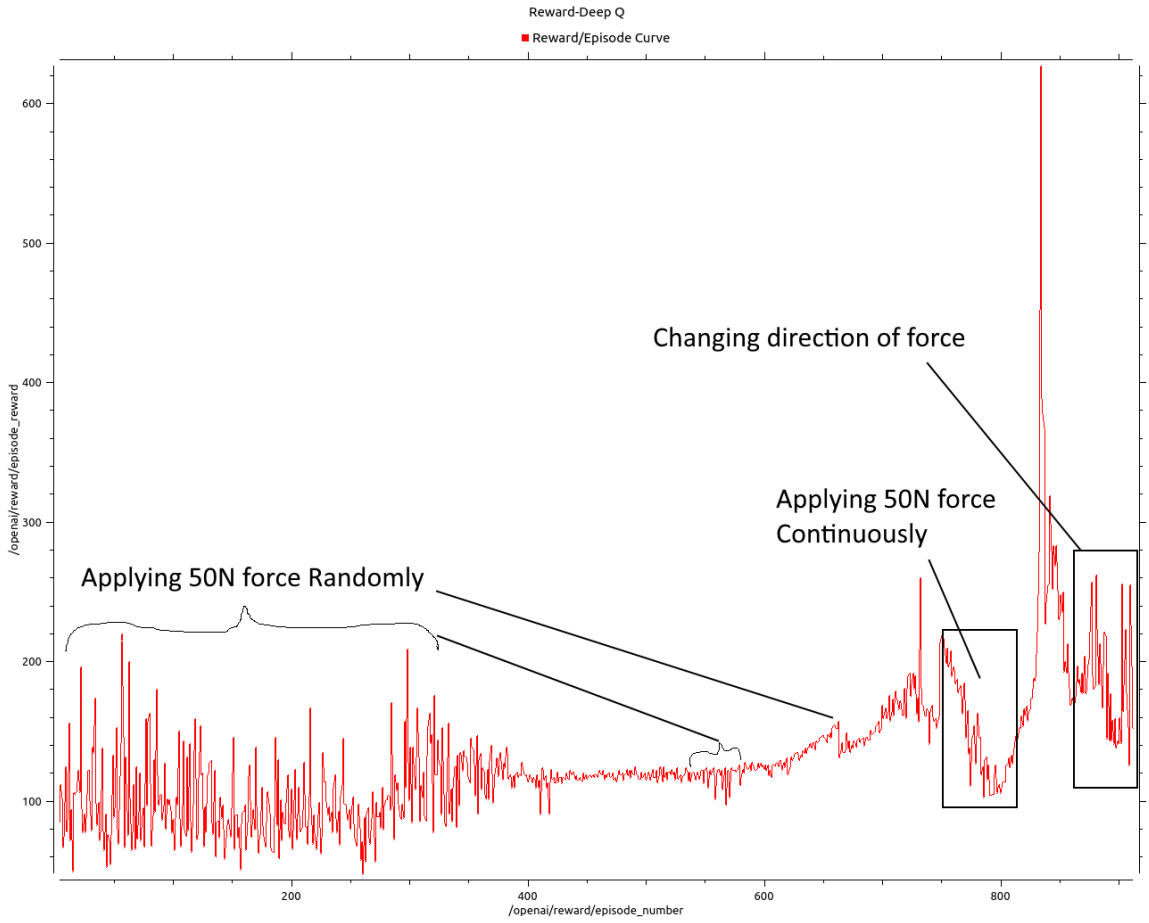


Figure 4.3 Applying forces randomly to the system and observe behavior

.

The other part of the study of this simple system was to understand how the learned model can adapt itself with the new physical model.

We trained two models, one that was trained without any turbulences and another with random turbulences during training. These models have been trained on normal laptop with Ubuntu OS for about a day. We loaded the models to train a new system with different physical properties and monitored how they behave new conditions.

Running a model is much faster than learning and running at the same time because many steps are skipped. When loading a trained model, 100 rewards equal about 20 seconds. We changed the weight of the pole from 1.5kg to 2.5kg and started observing the reward function. Figure 4.4 shows although trained model had average of 100 reward during 379 episodes, but it was instable. In contrast the robust model that is being trained by adding random turbulences, was able adopted itself with the new environment much quicker and after 300 episode reached stable reward of over 100. Figure 4.5 is showing the output data for this experiment.

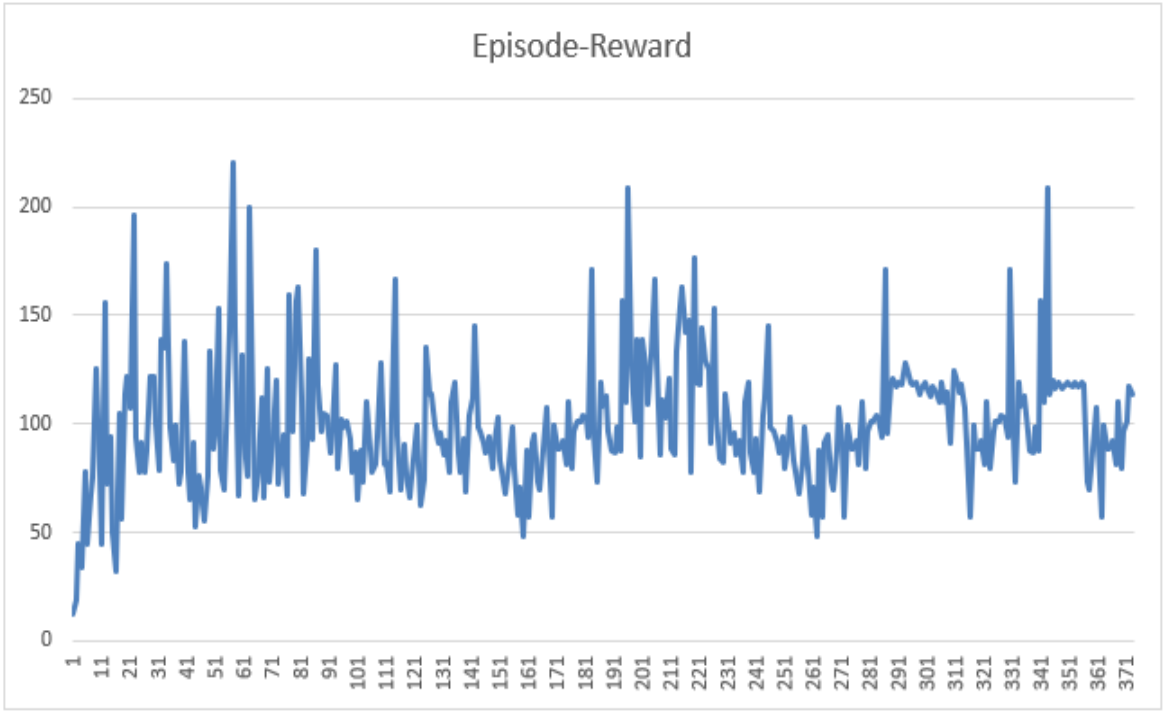


Figure 4.4: **Reward-Episode chart for the simple trained model**

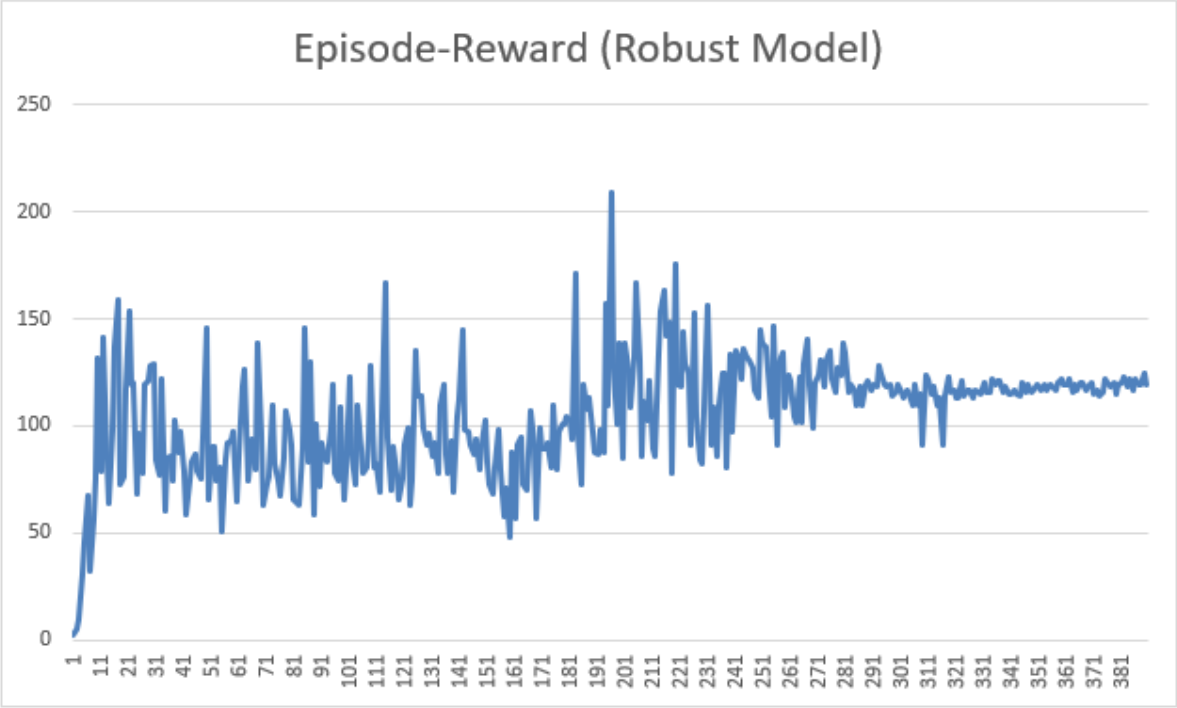


Figure 4.5: **Reward-Episode chart for the robust trained model**

4.3 Results of 3 DoF Inverted Pendulum on Chassis

In no joint experiment there were many challenges, first we had to make two Gazebo models and load them in sequence, if we place the bar link exactly on the surface of the top chassis in vertical orientation, it stands there and RL learns quickly that can hold the bar vertical with minimal changes in chassis positions. In fact, we got great reward, but the model is extremely poor and not resistance to any turbulences.

As seen in figure 4.6 we achieved over 1000 rewards at the beginning of the training while there was no turbulence however reward reduced tremendously to less than 100 in presence of turbulences.

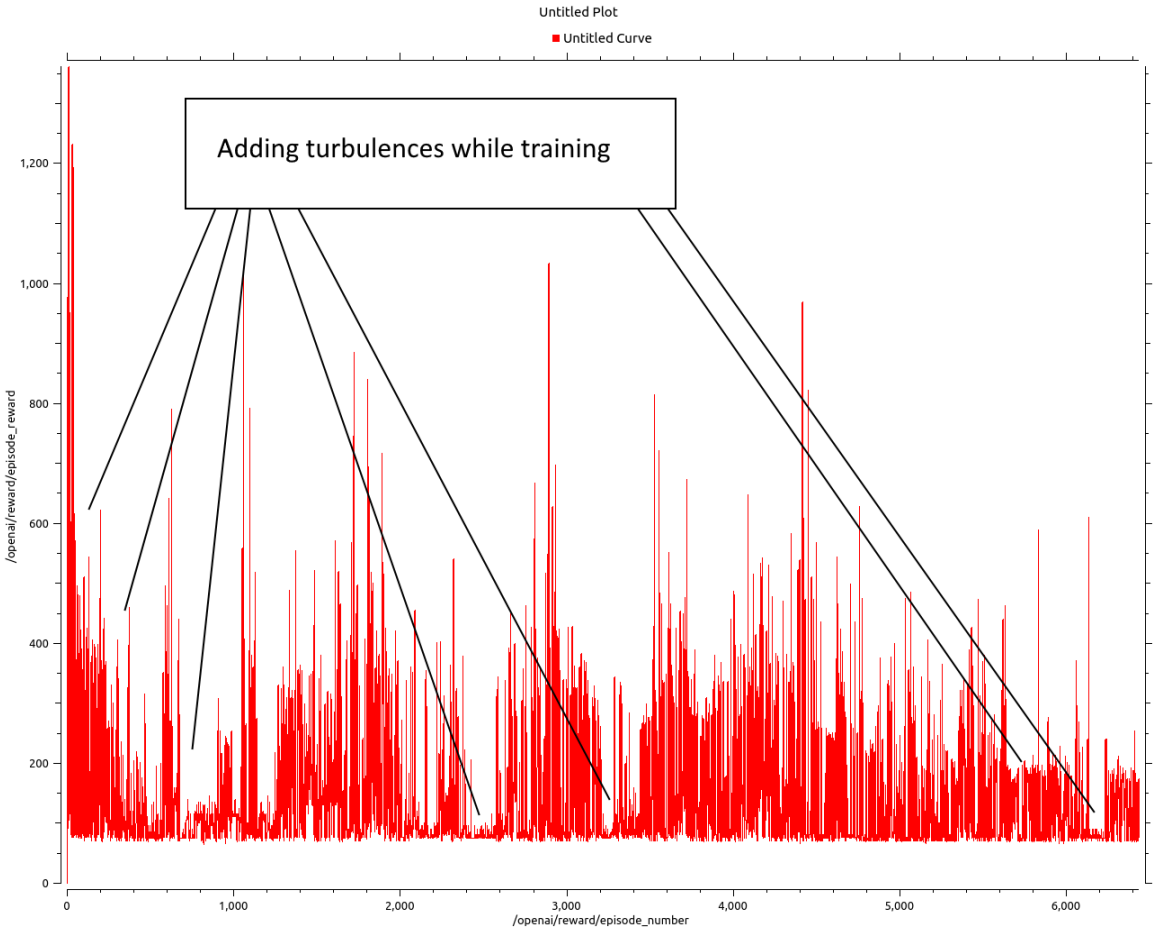


Figure 4.6: **Reward chart for 3DoF no joint model**

The challenge was because of no joint then turbulence was moving the bar link off its original position and causing falling off and losing control. A cylinder on the flat surface has more contact area than in angle position. Similar challenges reported in one of the papers mentioned in the literature review. [18]

The other challenge in no-joint design was the initialization after each failure, since there was no joint between the chassis and the bar link, and we had two models to load on sequence. We had to drop the bar-link from few centimeters above the chassis to give a bit of turbulence at initial stage otherwise RL decides to send actions with zero speed to the chassis which is failing the entire reinforcement learning approach.

On the other hand, ball joint approach showed a lot promises on learning the 3DoF Inverted pendulum. As seen in figure 4.7, it took more time for the system to learn how to control

in compare single inverted pendulum but was able to achieve better control and more reliable in compare of no-joint approach. The main reason for taking more time is the space of the observation and actions. In single inverted pendulum we had only 4 inputs and 2 outputs while in 3 DoF, we have 10 inputs and 27 outputs.

As seen in figure 4.7, after 400 episodes we achieved the reward of over 200 and then applying turbulence causing system slowly learns and adopts its model to more robust system. After about 800 episodes, system was able to achieve about 1000 reward points and applying turbulences did not bring the reward below its original level which was about 200. In addition, applying turbulence in different directions to the tip of the pendulum at the later stage was helping system to be more resistance and about 1300 episode the model achieved to learn and control the pendulum without hitting the sides and controlling it toward center point.

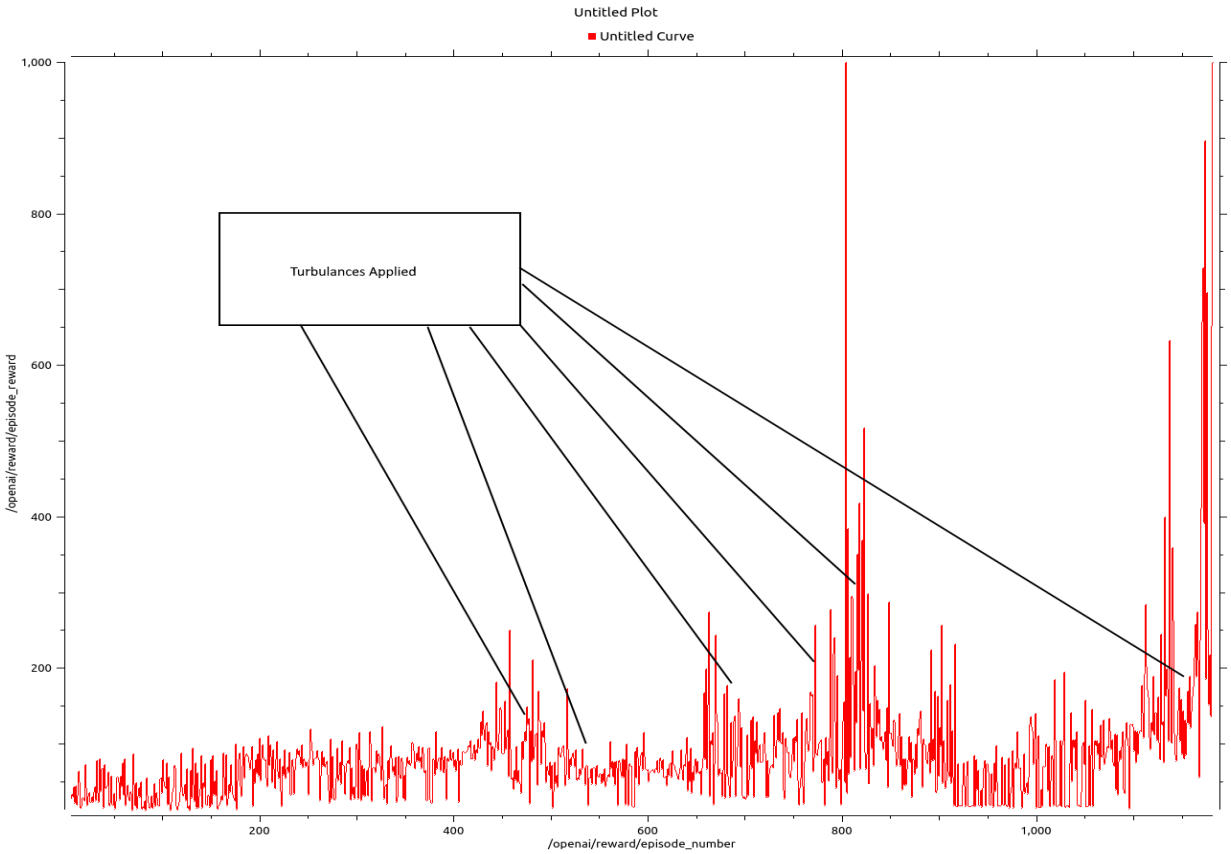


Figure 4.7: **3DoF reward chart with ball joint connection to chassis**

The friction parameters play an important role when using ball joint. If there is less friction then moving the base cannot apply force in any direction to inverted pendulum, ball slip in the bearing base so if inverted pendulum starts falling then there is no way to bring it back to the vertical position. On the other hand, once friction is set too high then it impacts on the nature phenomena of free fall, means when moving the chassis then pendulum moves on that direction.

The system achieves very high reward with too much friction on the joint but not robust at all. In fact, friction prevents it falls but when pendulum starts falling then there is no way to prevent it. Highest reward in this case is a fake value and unrealistic.

By changing the friction parameters to different values and keeping the training to learn for 2 days, we found the best values for mu1 and mu2 are 1.5 for the physical properties that we defined for this problem. Figure 4.8 shows reward function once we started learning with friction 0.5, at some point of time it shows it is learning but at the end fails to learn.

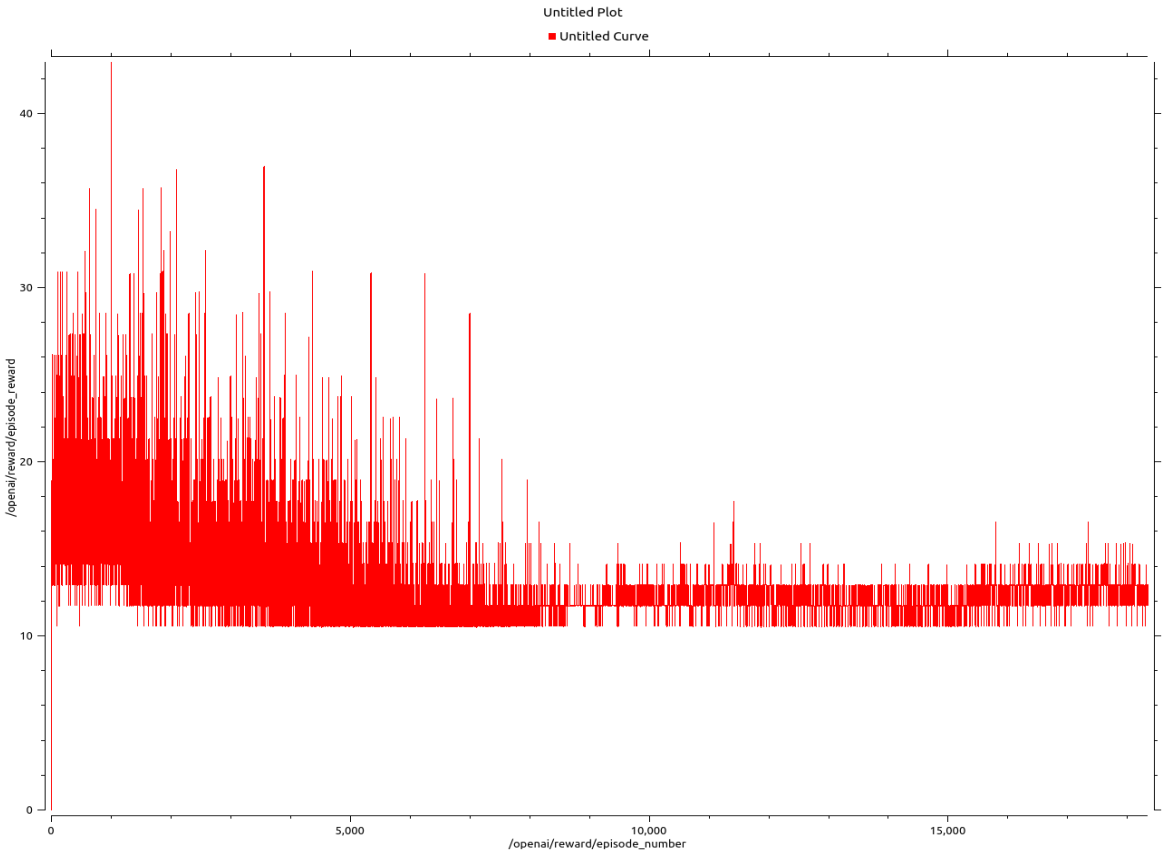


Figure 4.8: **Training for more 20,000 episodes, but no learning**

4.4 Results of 3 DoF Inverted Pendulum on Robotic Arm

As expected, learning process is more time consuming for the robotic arm mainly because of action space which is 729 in compare of previous case. So RL requires more time to learn the sequence of commands and make a proper approximation function to estimate Q values.

Figure 4.9 shows the improvement on the reward occurs and RL models started to learn control after 3000 episodes. Since the robot base is fixed and we have limitations on the angles for most of the joints, the amount of reward or time to balance the pendulum upward is not as equal as the previous case study.

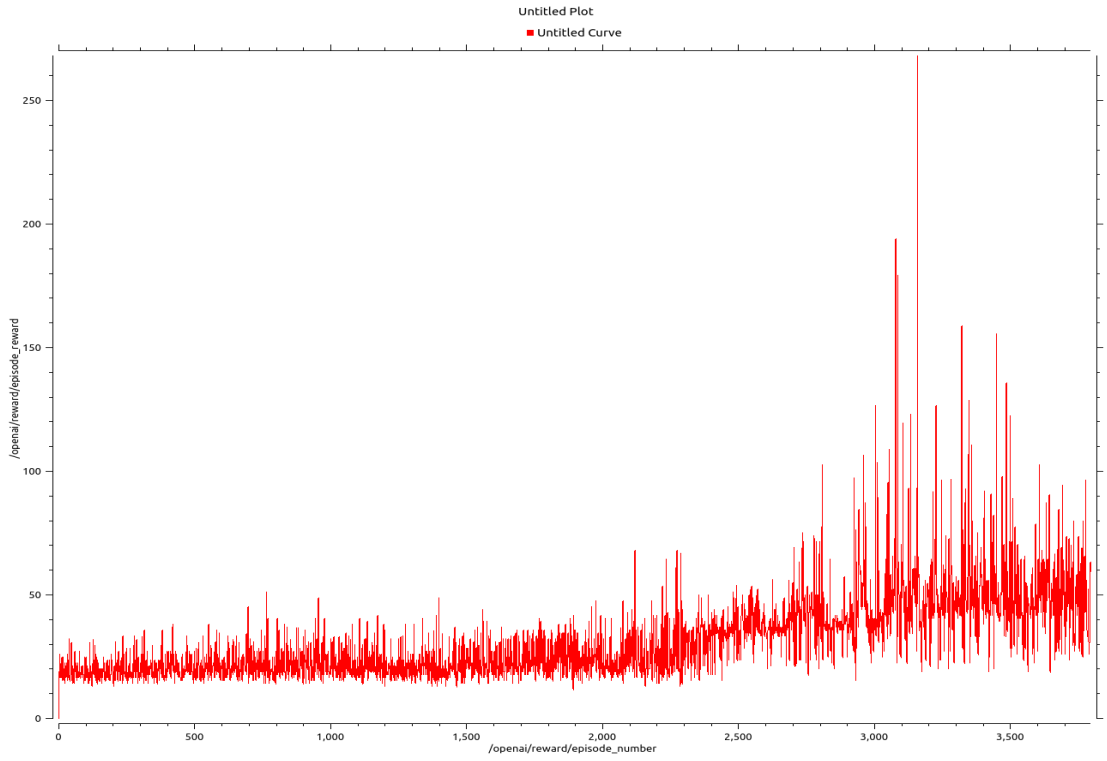


Figure 4.9: **UR learning to balance inverted pendulum but at much slower rate**

The number of changes in the angles for each action, play huge role in control and achieving learning. It was important in other case studies but not as much as in this case study.

In previous case studies we had velocity controllers for the robot, and we were changing speed of movement of the chassis however in the robot arm control we are not changing the speed of each angle and instead we let trajectory controller moves the robot arm within the specific time frame to the new position. This time frame is also constant value. We used different positional steps angle and tried to experiment how it impact the learning. As seen in figure 4.10, the position step of 0.005 radian and the amount of time 0.005 seconds leads to getting more reward in same number of episodes.

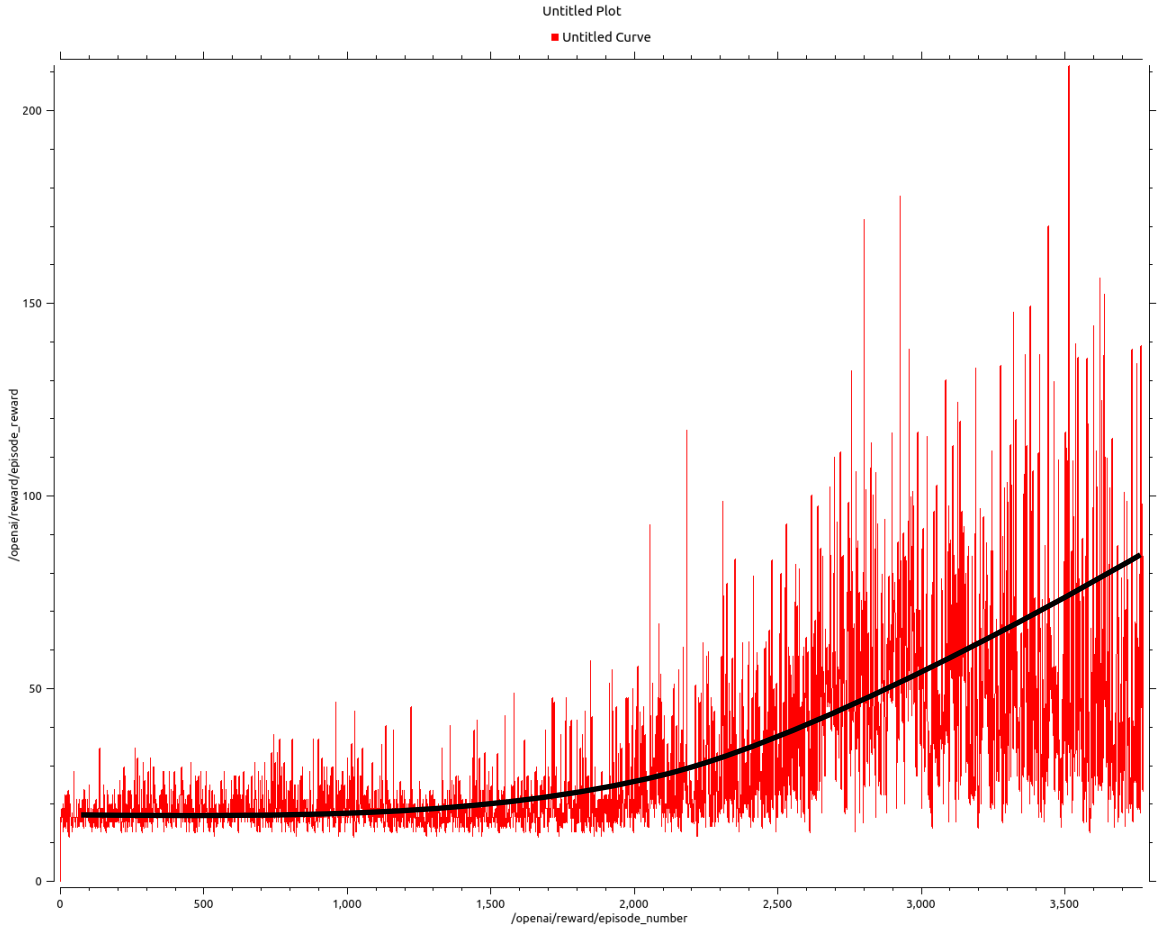


Figure 4.10: **Learning curve positional step is 0.005 radian and time is 0.005sec**

We have also experimented the learning rate and when we reduced the learning rate of the DQN algorithm and let robot to learn for longer period, then we could achieve much higher reward during training. The reason behind that is having a very smaller learning rate leads to have higher gamma factor in longer period. The gamma factor controls exploration and exploitation as explained in the chapter 1. Since the observation and space for the neural network of the DQN is huge in compare the other case studies, have more time for explorations helps RL to find better policy.

Figure 4.7 shows learning rate 0.00001 and after episode 14,000 which was about 2 days training on the laptop. We have reward of above 600 at some point of time. Although we get very high reward but as seen in figure 4.11 it is not consistent result in compare previous studies. The reward curve shows there is no overfit and system is still learning if kept for longer period. It is good to mention that in Deep Mind paper that is published for a DQN agent that learnt to play Atari games, they used 84 x84 size of image in 4 sequences which leads to have input space of 28,224. Their DQN algorithms learnt to play Atari games better than human in more than 50 million training episodes which is not possible under the normal laptops.[15]

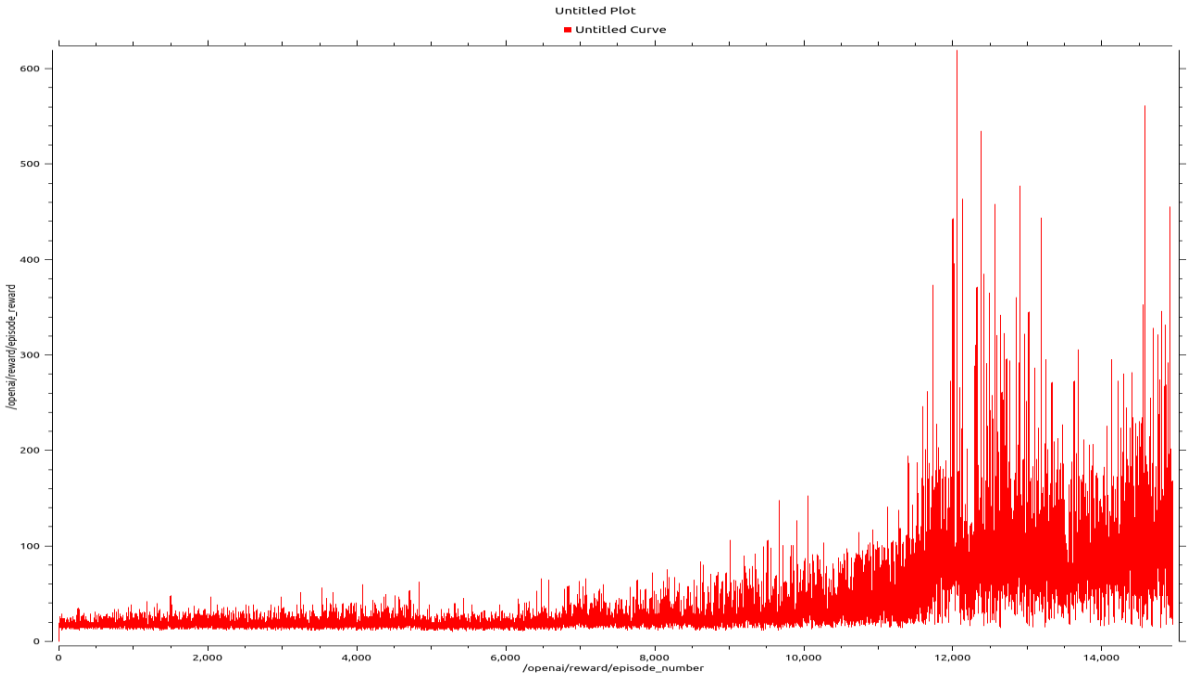


Figure 4.11: **Achieving reward of over 600 for balancing inverted pendulum**

CHAPTER 5 CONCLUSION AND FUTURE WORKS

In this thesis we have demonstrated that artificial intelligence is able to solve a complex balancing control problem from scratch, without any feedback loop and using state of the art deep reinforcement learning algorithm DQN. The benchmark problem that is considered as case study is balancing an inverted pendulum upward.

In our thesis we had multiple novelties in compare of published papers on similar subjects and some of them are:

- Six-degree freedom robot arm with ball joint inverted pendulum
- Using state of the art Robot Simulation platform, ROS and Gazebo
- Using Orientation sensor on the tip of the pendulum to get input data for learning

We have shown how different parameters impact the learning process and robustness of a model. We started from simple case study and slowly switched toward main goal which was controlling the inverted pendulum using robot arm. Simulation has been completed, tested and the experimental results showed the effectiveness of DQN algorithm in very complex problem. We also demonstrated that how applying turbulences during learning can help to achieve more robust model that can balance the inverted pendulum in more reliable way.

Once of the key achievement of this thesis was proving that a robust model can adopt itself with new environment much easier than a model that is being trained without presence of any turbulences. When physical parameters of a robot changes, a RL model has difficulty to adopt its model to the new environment so basically it is training the RL from scratch and nothing transferred from its past learning experience. While a robust model in same shows that after 300 episodes, it started to achieve a reliable control on the new robot environment.

We have designed and build a sensor for orientation using sensor fusion technique, successfully connected to the ROS to get the inputs. Guideline for communication between ROS and real robot provided as well as drawing to make a ball joint that is located in Appendix D however real robot testing has not being conducted and can be considered as future work.

In addition, this thesis can be considered a great source for testing similar problem with different reinforcement learning algorithms and comparing results in different scenarios.

REFERENCES/BIBLIOGRAPHY

[1] O. Boubaker, “The inverted pendulum: A fundamental benchmark in control theory and robotics,” *2012 Int. Conf. Educ. e-Learning Innov. ICEELI 2012*, 2012, doi: 10.1109/ICEELI.2012.6360606.

[2] S. Kajita *et al.*, “Biped walking stabilization based on linear inverted pendulum tracking,” *IEEE/RSJ 2010 Int. Conf. Intell. Robot. Syst. IROS 2010 - Conf. Proc.*, pp. 4489–4496, 2010, doi: 10.1109/IROS.2010.5651082.

[3] X. B. Jin, T. L. Su, J. L. Kong, Y. T. Bai, B. B. Miao, and C. Dou, *State-of-the-art mobile intelligence: Enabling robots to move like humans by estimating mobility with artificial intelligence*, vol. 8, no. 3. 2018.

[4] L. Huang and K. S. Ma, “Introducing Machine Learning to First-year Undergraduate Engineering Students Through an Authentic and Active Learning Labware,” *Proc. - Front. Educ. Conf. FIE*, vol. 2018-Octob, pp. 2018–2021, 2019, doi: 10.1109/FIE.2018.8659308.

[5] Q. Wang and Z. Zhan, “Reinforcement learning model, algorithms and its application,” *Proc. 2011 Int. Conf. Mechatron. Sci. Electr. Eng. Comput. MEC 2011*, no. 1, pp. 1143–1146, 2011, doi: 10.1109/MEC.2011.6025669.

[6] V. Mnih *et al.*, “Playing Atari with Deep Reinforcement Learning,” pp. 1–9, 2013, [Online]. Available: http://arxiv.org/abs/1312.5602.

[7] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, doi: 10.1038/nature14236.

[8] D. Silver *et al.*, “Mastering the game of Go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, 2017, doi: 10.1038/nature24270.

[9] F. Jiang, K. Dashtipour, and A. Hussain, “A Survey on Deep Learning for the Routing Layer of Computer Network,” *2019 UK/China Emerg. Technol. UCET 2019*, pp. 14–17, 2019, doi: 10.1109/UCET.2019.8881852.

[10] J. M. O’Kane, *A gentle introduction to ROS*, no. 2.1.3. 2016.

[11] W. E. Howlett, “Gazebo,” *Notes Queries*, vol. s3-X, no. 253, p. 352, 1866, doi: 10.1093/nq/s3-X.253.352-h.

[12] M. Mittal, “Introduction to Robot Simulation (Gazebo),” 2018.

[13] G. Walck, “Introduction to Robot Modeling in ROS Understanding URDF and XACRO,” 2015.

[14] Wikipedia, “Industrial Robot,” 2020, [Online]. Available: https://en.wikipedia.org/wiki/Industrial_robot.

[15] A. Topalidou-kyniazopoulou and S. Behnke, “Motion Planning Strategy For a 6-DOFs Robotic Arm In a Controlled Environment,” no. August, 2017.

[16] W. Linglin, “Single Inverted Pendulum Swing Control,” pp. 1558–1562, 2018.

[17] A. Ghio and O. E. Ramos, “Q-learning-based model-free swing up control of an inverted pendulum,” *Proc. 2019 IEEE 26th Int. Conf. Electron. Electr. Eng. Comput. INTERCON 2019*, 2019, doi: 10.1109/INTERCON.2019.8853619.

[18] X. Li, H. Liu, and X. Wang, “Solve the inverted pendulum problem base on DQN algorithm,” *Proc. 31st Chinese Control Decis. Conf. CCDC 2019*, no. 2, pp. 5115–5120, 2019, doi: 10.1109/CCDC.2019.8833168.

[19] X. Bin Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Sim-to-Real Transfer of Robotic Control with Dynamics Randomization,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 3803–3810, 2018, doi: 10.1109/ICRA.2018.8460528.

[20] M. A. R. and R. T. (of T. C. Alberto Ezquerro, “openai_ros,” [Online]. Available: http://wiki.ros.org/openai_ros.

[21] O. D. Engine, “ODE.”

[22] “ROS_Industrial / Universal Robot Package,” [Online]. Available: https://github.com/ros-industrial/universal_robot.

[23] L. A. Contreras-Rodriguez, R. Munoz-Guerrero, and J. A. Barraza-Madrigal, “Algorithm for estimating the orientation of an object in 3D space, through the optimal fusion of gyroscope and accelerometer information,” *2017 14th Int. Conf. Electr. Eng. Comput. Sci. Autom. Control. CCE 2017*, 2017, doi: 10.1109/ICEEE.2017.8108879.

[24] N. O-Larnnithipong and A. Barreto, “Gyroscope drift correction algorithm for inertial measurement unit used in hand motion tracking,” *Proc. IEEE Sensors*, no. 4, pp. 5–7, 2017, doi: 10.1109/ICSENS.2016.7808525.

[25] L. A. Contreras-Rodriguez, R. Munoz-Guerrero, and J. A. Barraza-Madrigal, “Algorithm for estimating the orientation of an object in 3D space, through the optimal fusion of gyroscope and accelerometer information,” *2017 14th Int. Conf. Electr. Eng. Comput. Sci. Autom. Control. CCE 2017*, pp. 1–5, 2017, doi: 10.1109/ICEEE.2017.8108879.

[26] C. J. Fisher, *Accelerometer*. .

[27] B. Ave, D. Number, and R. Date, “1 of 46,” vol. 1, no. 408, pp. 1–46, 2013, [Online]. Available: www.inversense.com.

[28] ROS, “ROS Supported Robots,” 2020, [Online]. Available: https://robots.ros.org/.

[29] ROS, “ROS & Universal Robot.” http://wiki.ros.org/universal_robot/Tutorials/Getting Started with a Universal Robot and ROS-Industrial.

APPENDICES

Appendix A Single DoF URDF

This is complete of the URDF file for single inverted pendulum, it includes all joints, links, transmission, and physical properties of the cartpole problem.

```
<?xml version="1.0" encoding="utf-8"  ?>
<robot name="cartpole_v0">

    <gazebo>
        <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.
so">
            <robotNamespace>/cartpole_v0</robotNamespace>
        </plugin>
    </gazebo>

    <!-- * * * Link Definitions * * * -->
    <!-- * * * Define World Link to fix the foot rail to the world * * * -
->
    <link name="world"/>
    <link name="bar_link">
        <visual>
            <origin xyz="0 0 0.4" rpy="0 0 0"/>
            <geometry>
                <box size="0.05 0.05 0.8"/>
            </geometry>
            <material name="green">
                <color rgba="0.9 0.8 0.6 1.0"/>
            </material>
       </visual>
        <collision>
            <origin xyz="0 0 0.4" rpy="0 0 0"/>
            <geometry>
                <box size="0.05 0.05 0.8"/>
            </geometry>
       </collision>
        <inertial>
            <origin xyz="0 0 0.4" rpy="0 0 0"/>
            <mass value="1"/>
```

```
            <inertia ixx="0.0535416666667" ixy="0.0" ixz="0.0" iyy="0.0535
416666667" iyz="0.0" izz="0.000416666666667"/>
        </inertial>
    </link>
    <gazebo reference="bar_link">
        <kp>1000.0</kp>
        <kd>1000.0</kd>
        <mu1>0.5</mu1>
        <mu2>0.5</mu2>
        <material>Gazebo/Green</material>
    </gazebo>

    <link name="base_link">
        <visual>
            <origin xyz="0 0 0.2" rpy="0 0 0"/>
            <geometry>
                <box size="0.4 0.2 0.2"/>
            </geometry>
            <material name="black">
                <color rgba="0 0 0 1.0"/>
            </material>
       </visual>
        <collision>
            <origin xyz="0 0 0.2" rpy="0 0 0"/>
            <geometry>
                <box size="0.4 0.2 0.2"/>
            </geometry>
       </collision>
        <inertial>
            <origin xyz="0 0 0.2" rpy="0 0 0"/>
            <mass value="2.5"/>
            <inertia ixx="0.0166666666667" ixy="0.0" ixz="0.0" iyy="0.0416
666666667" iyz="0.0" izz="0.0416666666667"/>
        </inertial>
    </link>
    <gazebo reference="base_link">
        <kp>1000.0</kp>
        <kd>1000.0</kd>
        <mu1>0.5</mu1>
        <mu2>0.5</mu2>
        <material>Gazebo/Black</material>
    </gazebo>

    <link name="foot_link">
        <visual>
```

```
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
            <geometry>
                <box size="6 0.025 0.1"/>
            </geometry>
            <material name="white">
                <color rgba="1 1 1 1.0"/>
            </material>
        </visual>
         <collision>
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
            <geometry>
                <box size="6 0.025 0.1"/>
            </geometry>
        </collision>
        <inertial>
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
            <mass value="5"/>
            <inertia ixx="0.00442708333333" ixy="0.0" ixz="0.0" iyy="15.00
41666667" iyz="0.0" izz="15.0002604167"/>
        </inertial>
    </link>
    <gazebo reference="foot_link">
        <kp>1000.0</kp>
        <kd>1000.0</kd>
        <mu1>0.5</mu1>
        <mu2>0.5</mu2>
        <material>Gazebo/White</material>
    </gazebo>



<!-- * * * Joint Definitions * * * -->
    <joint name="cartpole_joint" type="revolute">
        <parent link="base_link"/>
        <child link="bar_link"/>
        <origin xyz="0 0 0.32" rpy="0 0 0"/>
        <dynamics damping="0.0" friction="0.1"/>
        <limit lower="-1.57" upper="1.57" effort="1" velocity="100"/>
        <axis xyz="0 1 0"/>
    </joint>
    <joint name="foot_joint" type="prismatic">
        <parent link="foot_link"/>
        <child link="base_link"/>
        <limit lower="-
2.0" upper="2.0" effort="2000000" velocity="100000"/>
```

```
    </joint>
    <joint name="fixed" type="fixed">
        <parent link="world"/>
        <child link="foot_link"/>
    </joint>
<link name="sensor_box">
        <inertial>
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <mass value="0.01" />
            <inertia ixx="0.000001083" ixy="0.0" ixz="0.0" iyy="0.00000108
3" iyz="0.0" izz="0.0000015"/>
        </inertial>
        <collision>
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <geometry>
                <box size="0.1 0.1 0.08"/>
            </geometry>
        </collision>
        <visual>
          <geometry>
              <box size="0.08 0.08 0.05"/>
          </geometry>
          <material name="red">
            <color rgba="1.0 0 0 1.0"/>
          </material>
        </visual>
    </link>

<joint name="sensor_joint" type="fixed">
        <parent link="bar_link"/>
        <child link="sensor_box"/>
        <origin xyz="0 0 0.81" rpy="0 0 0"/>

</joint>

    <gazebo reference="sensor_box">
    <kp>10000000</kp>
    <kd>10000000</kd>
    <mu1>10.0</mu1>
    <mu2>10.0</mu2>
    <material>Gazebo/Red</material>
</gazebo>

    <!-- IMU sensor -->
    <gazebo>
```

```
        <plugin name="gazebo_ros_imu_controller" filename="libgazebo_ros_imu.so">
          <frameName>my-imu</frameName>
          <robotNamespace>/cartpole_v0</robotNamespace>
          <topicName>imu/data</topicName>
          <serviceName>imu/service</serviceName>
          <bodyName>sensor_box</bodyName>
          <gaussianNoise>0</gaussianNoise>
          <rpyOffsets>0 0 0</rpyOffsets>
          <!--<updateRate>50.0</updateRate>-->
          <alwaysOn>true</alwaysOn>
          <gaussianNoise>0</gaussianNoise>
        </plugin>
    </gazebo>
<!-- * * * Transmission Definitions * * * -->
    <transmission name="pole_joint_trans">
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="cartpole_joint">
        <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
      </joint>
      <actuator name="pole_jointMotor">
        <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
      </actuator>
    </transmission>

    <transmission name="foot_joint_trans">
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="foot_joint">
        <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
      </joint>
      <actuator name="foot_jointMotor">
        <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
      </actuator>
    </transmission>

</robot>
```

Appendix B Single DoF Inverted Pendulum Launch Files

This appendix is copy of all launch files, The main one is responsible to load other launch files, set the parameters and load controller of the single DoF inverted pendulum

Main Launch File:

```
<launch>
    <!-- load controller configuration to the ros parameter server -->
    <rosparam file="$(find cartpole_description)/config/cartpole_v0_velocity.yaml" command="load"/>

    <!-- launch the custom world -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch" >
        <arg name="paused" value="True"/>
        <!--arg name="use_sim_time" value="False" /-->
        <arg name="world_name" value="$(find cartpole_description)/worlds/cart_world.world"/>
        <env name="GAZEBO_MODEL_PATH" value="$(find cartpole_description)/models:$(optenv GAZEBO_MODEL_PATH)"/>
    </include>
    <!-- spawn the Cartpole_v0 construct -->
    <include file="$(find cartpole_description)/launch/spawn_cartpole_v0.launch"/>

  <node name="robot_state_publisher_cartpole_v0" pkg="robot_state_publisher" type="robot_state_publisher"
        respawn="false" output="screen">
            <param name="publish_frequency" type="double" value="5000.0" />
            <param name="ignore_timestamp" type="bool" value="true" />
            <param name="tf_prefix" type="string" value="cartpole_v0" />
            <remap from="/joint_states" to="/cartpole_v0/joint_states" />
        </node>


  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
        output="screen" args="--namespace=/cartpole_v0
                              joint_state_controller
                              pole_joint_velocity_controller
                              foot_joint_velocity_controller">

  </node>
  </launch>
```

Spawner Launch file that is responsible to spawn model into the simulation world

```
<launch>
    <arg name="x" default="0.0" />
    <arg name="y" default="0.0" />
    <arg name="z" default="0.0" />
    <arg name="roll" default="0.0"/>
    <arg name="pitch" default="0.0"/>
    <arg name="yaw" default="0.0"/>

    <!-- load controller configuration to the ros parameter server -->
    <rosparam file="$(find cartpole_description)/config/cartpole_v0_velocity.yaml" command="load"/>

    <!-- spawn the Cartpole_v0 construct -->
    <include file="$(find cartpole_description)/launch/spawn_cartpole_v0.launch">
        <arg name="x" value="$(arg x)" />
        <arg name="y" value="$(arg y)" />
        <arg name="z" value="$(arg z)" />
        <arg name="roll" value="$(arg roll)"/>
        <arg name="pitch" value="$(arg pitch)"/>
        <arg name="yaw" value="$(arg yaw)" />
  </include>

  <node name="robot_state_publisher_cartpole_v0" pkg="robot_state_publisher" type="robot_state_publisher"
        respawn="false" output="screen">
            <param name="publish_frequency" type="double" value="5000.0" />
            <param name="ignore_timestamp" type="bool" value="true" />
            <param name="tf_prefix" type="string" value="cartpole_v0" />
            <remap from="/joint_states" to="/cartpole_v0/joint_states" />
        </node>


  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
        output="screen" args="--namespace=/cartpole_v0
                              joint_state_controller
                              pole_joint_velocity_controller
                              foot_joint_velocity_controller">

  </node>
</launch>
```

Th Simulation world that is responsible to make the main simulation environment:

```
<launch>
    <!-- launch the custom world -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch" >
        <arg name="paused" value="True"/>
        <!--arg name="use_sim_time" value="False" /-->
        <arg name="world_name" value="$(find cartpole_description)/worlds/
cart_world.world"/>
        <env name="GAZEBO_MODEL_PATH" value="$(find cartpole_description)/
models:$(optenv GAZEBO_MODEL_PATH)"/>
    </include>

    <arg name="put_robot_in_world" default="false" />
    <arg name="put_robot_in_world_package" default="" />
    <arg name="put_robot_in_world_launch" default="" />

    <arg name="x" default="0.0" />
    <arg name="y" default="0.0" />
    <arg name="z" default="0.0" />
    <arg name="roll" default="0.0"/>
    <arg name="pitch" default="0.0"/>
    <arg name="yaw" default="0.0"/>

    <group if="$(arg put_robot_in_world)">
        <include file="$(eval find(put_robot_in_world_package) + '/launch/
' + put_robot_in_world_launch)">
            <arg name="x" value="$(arg x)" />
            <arg name="y" value="$(arg y)" />
            <arg name="z" value="$(arg z)" />
            <arg name="roll" value="$(arg roll)"/>
            <arg name="pitch" value="$(arg pitch)"/>
            <arg name="yaw" value="$(arg yaw)" />
        </include>
    </group>

</launch>
```

Appendix C CHASSIS URDF

This appendix shows the details of the URDF file for all chassis in the second study

```
<?xml version="1.0" encoding="utf-8"  ?>
<robot name="inv">


      <gazebo>
        <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.
so">
            <robotNamespace>/inv</robotNamespace>
        </plugin>
    </gazebo>

<link name="world"/>


<link name="foot_link">
    <visual>
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
        <geometry>
            <box size="6 6 0.1"/>
        </geometry>
        <material name="white">
            <color rgba="1 1 1 1.0"/>
        </material>
    </visual>
    <collision>
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
        <geometry>
            <box size="6 6 0.1"/>
        </geometry>
    </collision>
    <inertial>
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
        <mass value="5"/>
        <inertia ixx="15" ixy="0.0" ixz="0.0" iyy="15" iyz="0.0" izz="30"/
>
    </inertial>
</link>
<gazebo reference="foot_link">
    <kp>100000.0</kp>
    <kd>100000.0</kd>
```

```
    <mu1>0.5</mu1>
    <mu2>0.5</mu2>
    <material>Gazebo/Orange</material>
</gazebo>
    <joint name="f" type="fixed">
        <parent link="world"/>
        <child link="foot_link"/>
        <!-- <origin xyz="0 0 0.025" rpy="0 0 0"/> -->
  </joint>


<link name="link_chassis0">
 <visual>
           <origin xyz="0 0 0.05" rpy="0 0 0"/>
           <geometry>
               <box size="0.4 5.5 0.1"/>
           </geometry>
           <material name="black">
               <color rgba="0 0 0 1.0"/>
           </material>
       </visual>
        <collision>
           <origin xyz="0 0 0.05" rpy="0 0 0"/>
           <geometry>
               <box size="0.4 5.5 0.1"/>
           </geometry>
       </collision>
        <inertial>
           <origin xyz="0 0 0.05" rpy="0 0 0"/>
           <mass value="2.5"/>
           <inertia ixx="6.304" ixy="0.0" ixz="0.0" iyy="0.03542" iyz="0.
0" izz="6.335"/>
        </inertial>
</link>


<gazebo reference="link_chassis0">
    <kp>100000.0</kp>
    <kd>100000.0</kd>
    <mu1>0.5</mu1>
    <mu2>0.5</mu2>
     <material>Gazebo/Yellow</material>
    <!-- <material>Gazebo/Orange</material> -->
</gazebo>
```

```
<joint name="chassis0_prismatic_joint" type="prismatic">
  <parent link="foot_link"/>
  <child link="link_chassis0"/>
    <origin xyz="0 0 0.1" rpy="0 0 0"/>
    <limit effort="200000" velocity="100000" lower="-2.5" upper="2.5"/>
     <axis xyz="1 0 0"/>
</joint>



<link name="link_chassis1">
 <visual>
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
            <geometry>
                <box size="0.4 0.2 0.1"/>
            </geometry>
            <material name="black">
                <color rgba="0 0 0 1.0"/>
            </material>
       </visual>
        <collision>
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
            <geometry>
                <box size="0.4 0.2 0.1"/>
            </geometry>
       </collision>
        <inertial>
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
            <mass value="2.5"/>
            <inertia ixx="0.0166666666667" ixy="0.0" ixz="0.0" iyy="0.0416
666666667" iyz="0.0" izz="0.0416666666667"/>
        </inertial>
</link>


<gazebo reference="link_chassis1">
    <kp>100000.0</kp>
    <kd>100000.0</kd>
    <mu1>0.5</mu1>
    <mu2>0.5</mu2>
     <material>Gazebo/Black</material>
    <!-- <material>Gazebo/Orange</material> -->
</gazebo>

<joint name="chassis1_prismatic_joint" type="prismatic">
```

```
  <parent link="link_chassis0"/>
  <child link="link_chassis1"/>
    <origin xyz="0 0 0.1" rpy="0 0 0"/>
    <limit effort="200000" velocity="100000" lower="-2.5" upper="2.5"/>
     <axis xyz="0 1 0"/>
</joint>

<link name="link_chassis2">
 <visual>
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
            <geometry>
                <box size="0.4 0.2 0.1"/>
            </geometry>
            <material name="gray">
                <color rgba="0 0 0 1.0"/>
            </material>
        </visual>
        <collision>
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
            <geometry>
                <box size="0.4 0.2 0.1"/>
            </geometry>
        </collision>
        <inertial>
            <origin xyz="0 0 0.05" rpy="0 0 0"/>
            <mass value="2.5"/>
            <inertia ixx="0.0166666666667" ixy="0.0" ixz="0.0" iyy="0.0416
666666667" iyz="0.0" izz="0.0416666666667"/>
        </inertial>
</link>


<gazebo reference="link_chassis2">
    <kp>100000.0</kp>
    <kd>100000.0</kd>
    <mu1>0.5</mu1>
    <mu2>0.5</mu2>
     <material>Gazebo/Gray</material>
    <!-- <material>Gazebo/Orange</material> -->
</gazebo>

<joint name="chassis2_continuous_joint" type="continuous">
  <parent link="link_chassis1"/>
  <child link="link_chassis2"/>
    <origin xyz="0 0 0.1" rpy="0 0 0"/>
```

```
    <axis xyz="0 0 1"/>
</joint>


<link name="base_bearing">
   <contact>
    <rolling_friction value="0.005"/>
    <spinning_friction value="0.005"/>
  </contact>
  <inertial>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <mass value="0.17"/>
    <inertia ixx="0.0000491866666667" ixy="0" ixz="0" iyy="0.0000491866666
667" iyz="0" izz="0.00008704"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
        <mesh filename="package://cart_inv_pendulum/meshes/inverted_pendul
um/base.dae" scale="1 1 1"/>
    </geometry>
    <material name="green">
      <color rgba="0 1 0 1"/>
    </material>
  </visual>
  <collision>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
        <mesh filename="package://cart_inv_pendulum/meshes/inverted_pendul
um/base.dae" scale="1 1 1"/>
    </geometry>
  </collision>
</link>
<gazebo reference="base_bearing">
    <kp>1000000.0</kp>
    <kd>1000000.0</kd>
    <mu1>0.5</mu1>
    <mu2>0.5</mu2>
    <material>Gazebo/Blue</material>
</gazebo>



<joint name="base_chassis_joint" type="fixed">
    <parent link="link_chassis2" />
```

```
    <child link = "base_bearing" />
    <origin rpy="0 0 0" xyz="0.0 0 0.105" />

</joint>

<link name="ball">
  <contact>
    <rolling_friction value="0.005"/>
    <spinning_friction value="0.005"/>
  </contact>
  <inertial>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <mass value="0.17"/>
    <inertia ixx="0.00005883" ixy="0" ixz="0" iyy="0.00005883" iyz="0" izz
="0.00001224"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
        <mesh filename="package://cart_inv_pendulum/meshes/inverted_pendul
um/ball3.dae" scale="1 1 1"/>
    </geometry>
    <material name="red">
      <color rgba="1 0 0 1"/>
    </material>
  </visual>
  <collision>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
        <mesh filename="package://cart_inv_pendulum/meshes/inverted_pendul
um/ball3.dae" scale="1 1 1"/>
    </geometry>
  </collision>
</link>


<gazebo reference="ball">
    <kp>10000000</kp>
    <kd>10000000</kd>
    <mu1>0.5</mu1>
    <mu2>0.5</mu2>
    <material>Gazebo/Green</material>
</gazebo>

<joint name="base_bearing_ball" type="revolute">
```

```
        <parent link="base_bearing"/>
        <child link="ball"/>
        <origin xyz="0.0023 0 -0.0005" rpy="0 0 0"/>
        <dynamics damping="0.0" friction="0.1"/>
        <limit lower="-1.46" upper="1.46" effort="1" velocity="100"/>
        <axis xyz="1 1 1"/>
</joint>

<link name="pole">
  <contact>
    <rolling_friction value="0.005"/>
    <spinning_friction value="0.005"/>
  </contact>
  <inertial>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <mass value="2.5"/>
    <inertia ixx="0.0833639583333333" ixy="0" ixz="0" iyy="0.0833639583333
333" iyz="0" izz="0.00006125"/>
  </inertial>
  <visual>
    <origin rpy="0 0 1.57" xyz="0 0 0.015"/>
    <geometry>
         <cylinder radius="0.007" length="0.4"/>
    </geometry>
    <material name="green">
      <color rgba="0 1 0 1"/>
    </material>
  </visual>
  <collision>

    <origin rpy="0 0 1.57" xyz="0 0 0.015"/>
    <geometry>
         <cylinder radius="0.007" length="0.4"/>
    </geometry>
  </collision>
</link>
<gazebo reference="pole">
    <kp>1000</kp>
    <kd>1000</kd>
    <mu1>0.5</mu1>
    <mu2>0.5</mu2>
    <material>Gazebo/Green</material>
</gazebo>
<joint name="ball_pole" type="fixed">
        <parent link="ball"/>
```

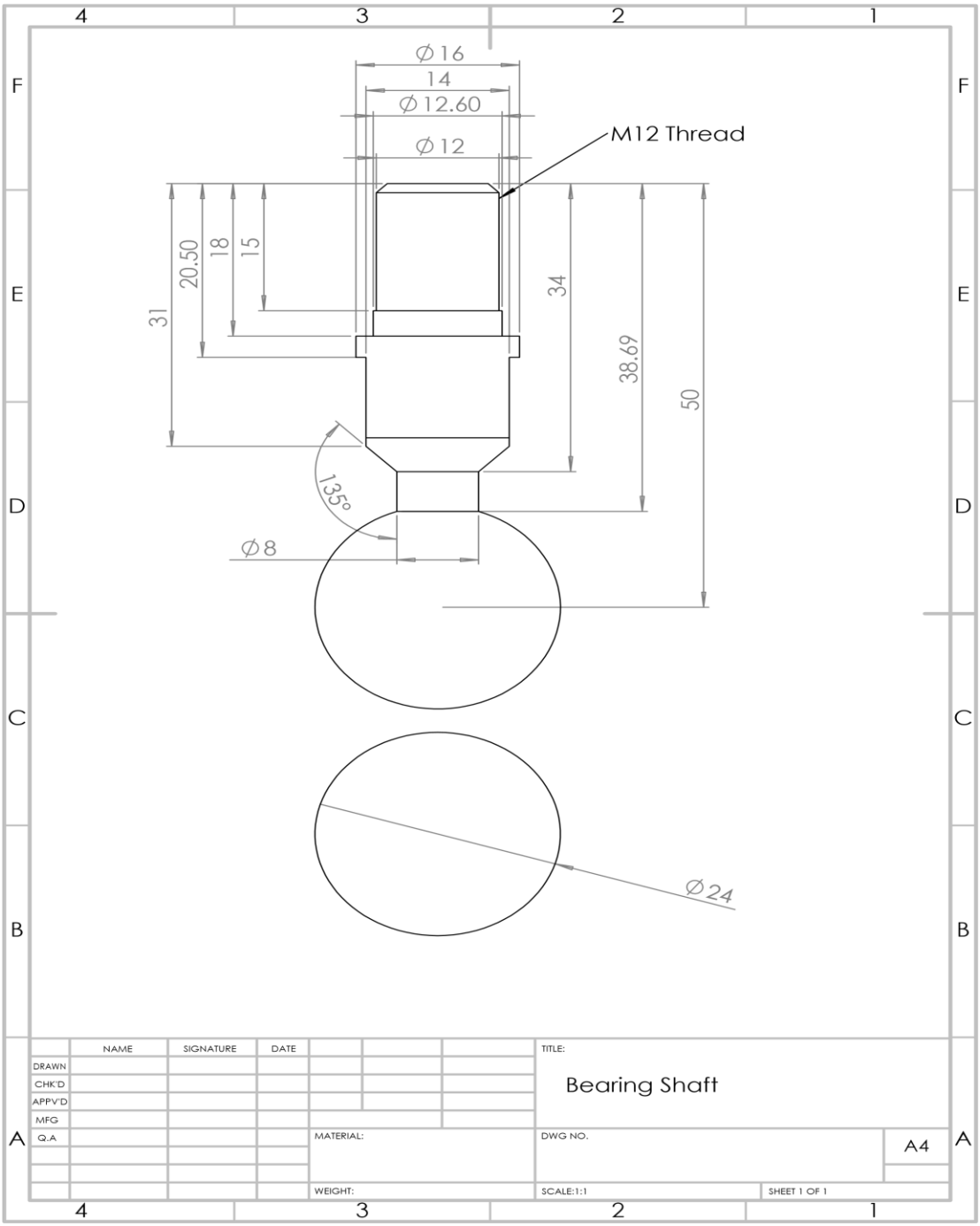
```
        <child link="pole"/>
        <origin xyz="0 0 0.217" rpy="0 0 0"/>

</joint>


 <link name="sensor_box">
        <inertial>
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <mass value="0.01" />
            <inertia ixx="0.000001083" ixy="0.0" ixz="0.0" iyy="0.00000108
3" iyz="0.0" izz="0.0000015"/>
        </inertial>
        <collision>
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <geometry>
                <box size="0.03 0.03 0.02"/>
            </geometry>
        </collision>
        <visual>
          <geometry>
              <box size="0.05 0.03 0.02"/>
          </geometry>
          <material name="red">
            <color rgba="1.0 0 0 1.0"/>
          </material>
        </visual>
    </link>
<gazebo reference="sensor_box">
    <kp>10000000</kp>
    <kd>10000000</kd>
    <mu1>0.5</mu1>
    <mu2>0.5</mu2>
    <material>Gazebo/Red</material>
</gazebo>
<joint name="sensor_joint" type="fixed">
        <parent link="pole"/>
        <child link="sensor_box"/>
        <origin xyz="0 0 0.21" rpy="0 0 0"/>

</joint>

    <!-- IMU sensor -->
    <gazebo>
```

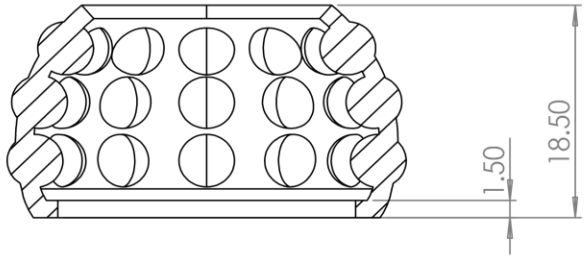
```
        <plugin name="gazebo_ros_imu_controller" filename="libgazebo_ros_imu.so">
          <robotNamespace>/mpu6050</robotNamespace>
          <topicName>imu/data</topicName>
          <serviceName>imu/service</serviceName>
          <bodyName>sensor_box</bodyName>
          <gaussianNoise>0</gaussianNoise>
          <rpyOffsets>0 0 0</rpyOffsets>
          <!--<updateRate>50.0</updateRate>-->
          <alwaysOn>true</alwaysOn>
          <gaussianNoise>0</gaussianNoise>
        </plugin>
    </gazebo>

    <transmission name="pole_joint_trans">
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="base_bearing_ball">
        <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
      </joint>
      <actuator name="pole_jointMotor">
        <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
      </actuator>
    </transmission>



<transmission name="chassis0_prismatic_trans">
        <type>transmission_interface/SimpleTransmission</type>
        <joint name="chassis0_prismatic_joint">
            <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
        </joint>
        <actuator name="motor0">
            <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
            <mechanicalReduction>1</mechanicalReduction>
        </actuator>
    </transmission>


    <transmission name="chassis1_prismatic_trans">
        <type>transmission_interface/SimpleTransmission</type>
```

```
        <joint name="chassis1_prismatic_joint">
            <hardwareInterface>hardware_interface/EffortJointInterface</ha
rdwareInterface>
        </joint>
        <actuator name="motor2">
            <hardwareInterface>hardware_interface/EffortJointInterface</ha
rdwareInterface>
            <mechanicalReduction>1</mechanicalReduction>
        </actuator>
    </transmission>

  <transmission name="chassis2_continuous_trans">
        <type>transmission_interface/SimpleTransmission</type>
        <joint name="chassis2_continuous_joint">
            <hardwareInterface>hardware_interface/EffortJointInterface</ha
rdwareInterface>
        </joint>
        <actuator name="motor3">
            <hardwareInterface>hardware_interface/EffortJointInterface</ha
rdwareInterface>
            <mechanicalReduction>1</mechanicalReduction>
        </actuator>
    </transmission>



</robot>
```

Appendix D Technical Drawing of ball joint

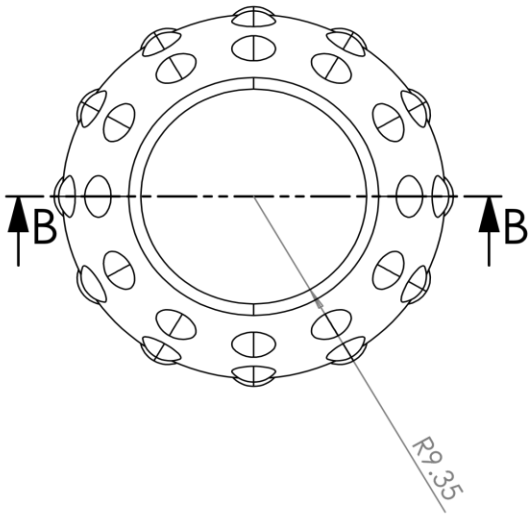
This appendix shows the details of the URDF file for all chassis in the second study

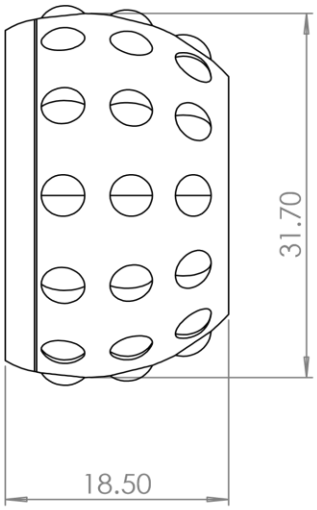




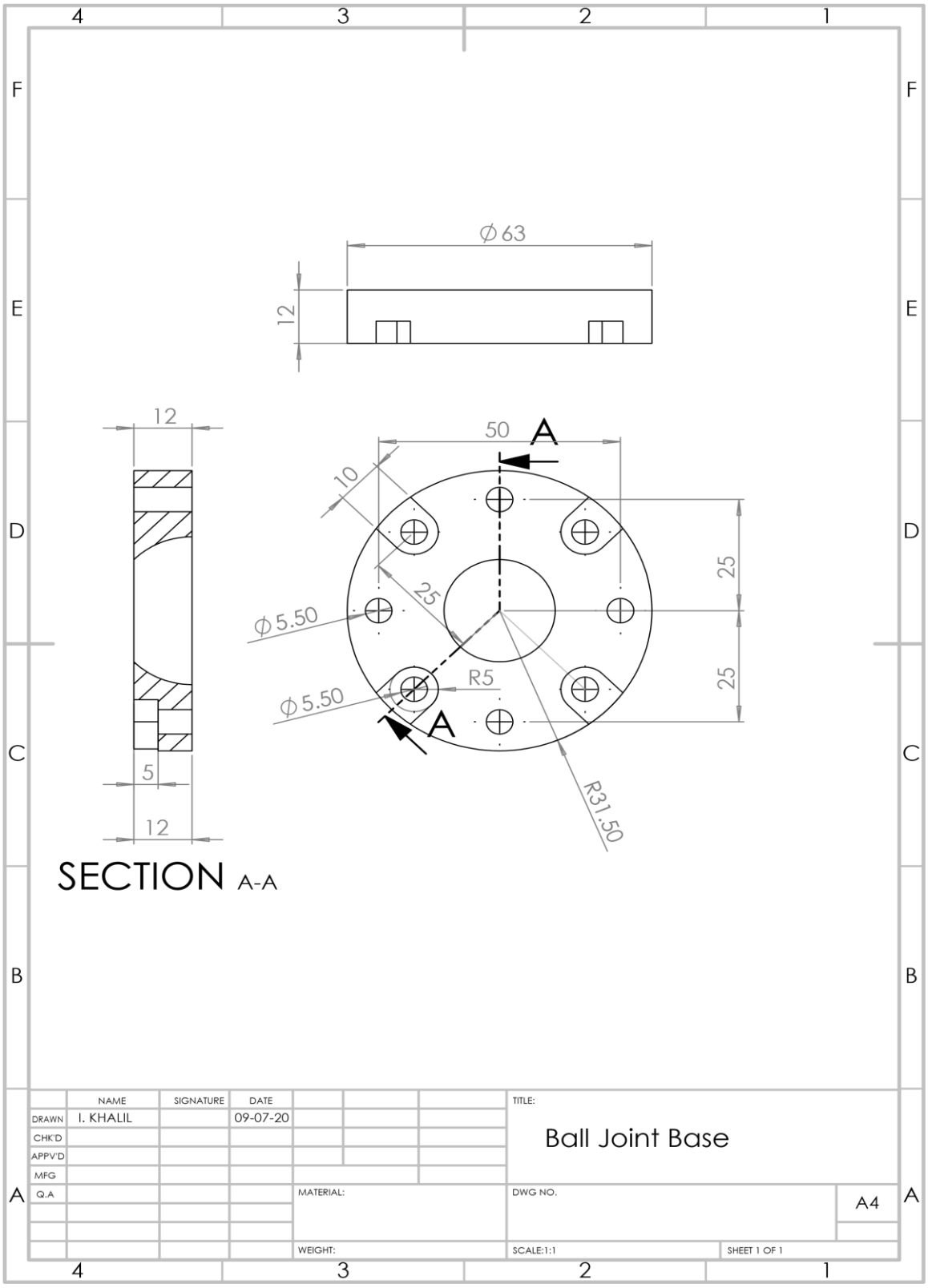
SECTION B-B

SCALE 2 : 1





	NAME	SIGNATURE	DATE				TITLE:	
DRAWN							Bearing	
CHK'D								
APPV'D								
MFG								
Q.A				MATERIAL:			DWG NO.	A4
				WEIGHT:			SCALE:1:1	SHEET 1 OF 1


4
3
2
1
F
E
D
C
B
A
Ø63
12
12
50
A
10
25
25
25
Ø5.50
Ø5.50
R5
A
R31.50
5
12
SECTION A-A
NAME
SIGNATURE
DATE
DRAWN
I. KHALIL
09-07-20
CHK'D
APPV'D
MFG
Q.A
TITLE:
Ball Joint Base
MATERIAL:
DWG NO.
A4
WEIGHT:
SCALE:1:1
SHEET 1 OF 1

VITA AUCTORIS

NAME:	Seyed Navid Mellatshahi
PLACE OF BIRTH:	Lahijan, IR
YEAR OF BIRTH:	1981
EDUCATION:	Roodaki High School, Lahijan, IR, 1998
	University of Shiraz, B.Sc., Shiraz, IR, 2004
	University of Windsor, M.Sc., Windsor, ON, 2020