**A TECHNICAL REPORT**

**ON**

**STUDENTS’ INDUSTRIAL WORK EXPERIENCE**

**SCHEME (SIWES)**

**HELD AT**

**GILEAD BIOMEDICAL ENGINEERING**

**BY**

**OMOLAYO ISAAC TEMILOLUWA**

**EEG/2015/097**

**SUBMITTED TO:**

**THE SIWES COORDINATOR**

**DEPARTMENT OF ELECTRONIC AND ELECTRICAL ENGINEERING**

**FACULTY OF TECHNOLOGY**

**OBAFEMI AWOLOWO UNIVERSITY,**

**ILE-IFE, OSUN STATE.**

**FEBRUARY 2019**

Department of Electronic and Electrical Engineering,

Obafemi Awolowo University,

Ile-Ife, Osun State.

25th February, 2020.

The SIWES Coordinator,

Department of Electronic and Electrical Engineering,

Obafemi Awolowo University,

Ile-Ife,

Osun Sate.

Dear Sir,

# LETTER OF TRANSMITTAL

In accordance with the Student Industrial Work Experience Scheme (SIWES) schedule to provide adequate industrial training for the student in their various field of study, I, OMOLAYO Isaac Temiloluwa with matric number: EEG/2015/097, of the Department of Electronic and Electrical Engineering hereby write to submit a technical report on Student Industrial Work Experience Scheme (SIWES) undertaken at Gilead Biomedical Engineering in partial fulfilment of the award of Bachelor of Science in Electronic and Electrical Engineering, Obafemi Awolowo University, Ile-Ife, Osun.

Yours faithfully,

………………………..

OMOLAYO Isaac Temiloluwa

EEG/2015/097

# CERTIFICATION

This is to certify that I, OMOLAYO Isaac Temiloluwa, with matric number EEG/2015/097, compiled this report based on my six months student industrial work experience scheme (SIWES) carried out at Gilead Biomedical Engineering, Ile-Ife, Osun.

……………………… ………………………

Student’s Signature Date

# DEDICATION

I hereby dedicate this report to the immaculate, immortal, only wise God and the giver of sound wisdom for, he alone is the strength of my life. I also dedicate this report to my lovely parents, Mr. and Mrs. Omolayo for their advice, financial and moral support

# ACKNOWLEDGEMENT

My regards and appreciation go to our heavenly father for his infinite mercies, protection and wisdom, he granted to me throughout the period of my attachment.

Also, to the student industrial work experience scheme (SIWES). They have helped a lot in assisting students to acquire relevant practical on the job experience.

My immeasurable thanks go to my parents for their effortless love and care, I cannot appreciate your consisted prayerful and loving care for me, I pray you will live to eat the fruit of your labor.

My profound gratitude goes to my industry-based supervisor, Dr. K. P. Ayodele for his support throughout my period of attachment.

Not forgetting Mr. Segun and Mr. Faremi Bolu who helped and guided me throughout the course of this industrial attachment and thanks to Mr. Adewale of communications lab for providing me with a work space.

To my colleague, Komolafe Elisha who worked hand in hand with me, it was a wonderful experience working with you.

# TABLE OF CONTENTS

[LETTER OF TRANSMITTAL ii](#_Toc34586230)

[CERTIFICATION iii](#_Toc34586231)

[DEDICATION iv](#_Toc34586232)

[ACKNOWLEDGEMENT v](#_Toc34586233)

[TABLE OF CONTENTS vi](#_Toc34586234)

[LIST OF TABLES x](#_Toc34586235)

[LIST OF ABBREVIATIONS xi](#_Toc34586236)

[LIST OF FIGURES xii](#_Toc34586237)

[LIST OF PLATES xiii](#_Toc34586238)

[ABSTRACT xiv](#_Toc34586239)

[CHAPTER ONE 1](#_Toc34586240)

[INTRODUCTION 1](#_Toc34586241)

[1.1 Objectives of SIWES 1](#_Toc34586242)

[1.2 Gilead Biomedical Engineering 2](#_Toc34586243)

[1.3 Scope of the Report 4](#_Toc34586244)

[1.4 Justification of SIWES at the place of IT 4](#_Toc34586245)

[CHAPTER TWO 5](#_Toc34586246)

[LITERATURE REVIEW 5](#_Toc34586247)

[2.1 Machine Learning 5](#_Toc34586248)

[2.2.1 Categories of Machine Learning 5](#_Toc34586249)

[2.2 Reinforcement Learning 8](#_Toc34586250)

[2.2.1 Elements of Reinforcement Learning 8](#_Toc34586251)

[2.2.2 Components of a Reinforcement Learning Agent 9](#_Toc34586252)

[CHAPTER THREE 11](#_Toc34586253)

[DESIGN OF THE GRIPPER IN A FORCE TORQUE SENSOR 11](#_Toc34586254)

[3.1 What is a Force Torque Sensor? 11](#_Toc34586255)

[3.2 Components of the Force Torque Sensor 11](#_Toc34586256)

[3.2.1 Load Cell 11](#_Toc34586257)

[3.2.2 HX711 amplifier 11](#_Toc34586258)

[3.2.3 NI USB-8451 DAQ 12](#_Toc34586259)

[3.2.4 Atmega328p Microcontroller on Arduino Uno 12](#_Toc34586260)

[3.3 Design of the Force Torque Sensor 14](#_Toc34586261)

[3.3.1 Introduction to I2C Configuration 14](#_Toc34586262)

[3.3.2 I2C Interface 15](#_Toc34586263)

[3.3.3 I2C Addresses 15](#_Toc34586264)

[3.3.4 I2C Protocol 15](#_Toc34586265)

[CHAPTER FOUR 18](#_Toc34586266)

[REINFORCEMENT LEARNING IN ROBOTICS 18](#_Toc34586267)

[4.1 Reinforcement Learning for Industrial Robots 18](#_Toc34586268)

[4.1.1 Problem Statement 18](#_Toc34586269)

[4.1.2 Metrics 18](#_Toc34586270)

[4.1.3 Analysis 19](#_Toc34586271)

[4.1.4 The State Space 19](#_Toc34586272)

[4.1.5 The Actions 20](#_Toc34586273)

[4.1.6 The Reward Strategy 23](#_Toc34586274)

[4.1.7 Algorithms and Techniques 24](#_Toc34586275)

[4.1.8 Framework Setup 25](#_Toc34586276)

[4.1.9 Implementation 26](#_Toc34586277)

[4.1.10 Results and Conclusions 28](#_Toc34586278)

[4.1.11 Further Work and Improvement 28](#_Toc34586279)

[4.2 Reinforcement Learning for Efficient Gait Cycle of Biped Robots 29](#_Toc34586280)

[4.2.1 The NAO Robot 29](#_Toc34586281)

[4.2.2 Problem Statement 31](#_Toc34586282)

[4.2.3 Methodology 31](#_Toc34586283)

[4.2.4 Base Level Modules Definition 33](#_Toc34586284)

[4.2.5 First Level Modules Definition 33](#_Toc34586285)

[4.2.6 Learning the Poses 34](#_Toc34586286)

[4.2.7 Second Level Module Definition 37](#_Toc34586287)

[4.2.8 Tying it all together 38](#_Toc34586288)

[4.2.9 Algorithm 41](#_Toc34586289)

[4.2.10 Results and Conclusions 44](#_Toc34586290)

[4.2.11 Further Work and Improvements 44](#_Toc34586291)

[CHAPTER FIVE 45](#_Toc34586292)

[REPAIR AND SERVICING OF A HARMAR AL600 LIFT 45](#_Toc34586293)

[5.1 The Harmar AL600 Lift 45](#_Toc34586294)

[5.1.1 Components of the AL600 Lift 45](#_Toc34586295)

[5.2 The Problem 47](#_Toc34586296)

[5.3 Troubleshooting and Fixing 47](#_Toc34586297)

[CHAPTER SIX 49](#_Toc34586298)

[CONCLUSION AND RECOMMENDATION 49](#_Toc34586299)

[6.1 Skills Gained and Educational Goals 49](#_Toc34586300)

[6.2 Recommendation 50](#_Toc34586301)

[REFERENCES 51](#_Toc34586302)

# LIST OF TABLES

Table 4.1: Conditions and specified reward functions 23

# LIST OF ABBREVIATIONS

**AI:** Artificial Intelligence

**DOF:** Degree of Freedom

**IT:** Industrial Training

**ITF:** Industrial Training Fund

**ML:** Machine Learning

**MLP:** Multi-Layer Perceptron

**PULSR:** Platform for Upper Limb Stoke Rehabilitation

**SIWES:** Student’s Industrial Work Experience Scheme

**ZMP:** Zero Moment Point

# LIST OF FIGURES

Fig 1.1: Organization structure of the research group 3

Fig 2.1: Categories of Machine Learning 7

Fig 2.2: The action-reward feedback loop 7

Fig 3.1: Components used in the force torque sensor 13

Fig 3.2: Load Cell Wiring to HX711 Amplifier 17

Fig 3.3: I2C Protocol 17

Fig 3.4: LabVIEW VI showing connection using the I2C configuration 17

Fig 4.1: The VREP environment 21

Fig 4.2: The robotic arm with the active space discretized 21

Fig 4.3: Visual representation of states 22

Fig 4.4: The NAO robot with equipped sensors 30

Fig 4.5: NAO robot and degrees of freedom 30

Fig 4.6: First Level Q-Network 36

Fig 4.7: Second Level Q-Network 36

Fig 4.8: VREP environment with NAO robot 40

Fig 4.9: Fall Module Multi-Layer Perceptron (MLP) 40

Fig 5.1: Major parts of the AL600 lift 46

# LIST OF PLATES

Plate 5.1: Assembly and troubleshooting of the AL600 lift 48

Plate 5.2: Installation and testing of the AL600 lift. 48

# ABSTRACT

This report is a detailed reflection of the knowledge I was able to garner during my SIWES internship at Gilead Biomedical Engineering, Ile-Ife, Osun. I was assigned to Reinforcement Learning section under control systems research. The internship was more of a research experience with learning and working on state-of-the-art research with the help of the principal investigator and professional journals on the internet.

I conducted research on the application of reinforcement learning as a control scheme in robotics. This research exposed me to how work is being done on new topics in research and development setting.

I also worked on implementing an improvised force-torque sensor by working on the gripper which measures the force.

I also worked on repair and maintenance of a Harmar AL600 lift which exposed me to repair and maintenance of equipment as used in the industry.

My overall experience during the attachment was an eye-opening and educative one. I had the opportunity to work in a standard working environment, learnt a lot of engineering and work ethics suitable for the business world and apply theories that had been learnt in class.

# CHAPTER ONE

# INTRODUCTION

## Objectives of SIWES

SIWES which stands for Student Industrial Work Experience Scheme was initiated by the Industrial Training Fund (ITF) in 1973 so as to complement the theoretical knowledge acquired in higher institutions with practical experience.

The goal of SIWES is to promote industrialization in Nigeria, and am avenue between the world of teaching, learning, industry and work with reference to a filed of study such as engineering, science, agriculture, technology and other professional education programs. The minimum duration of the program is twenty-four weeks. It was funded by the federal government of Nigeria and jointly coordinated by the industrial training fund (ITF) and the National Universities Commission (NUC).

The Objectives of the student industrial training work experience schemes are:

* Provision of avenue for students in Nigerian universities to gain industrial skills and experience in their course of study.
* To prepare students for the work situation they are likely to meet after graduation.
* To expose students to work methods and techniques in handling equipment and machinery that may not be available in the universities.
* To make the transition from university to the world of work easier, and thus enhance students contacts for later job placement.
* To provide students with an opportunity to apply their theoretical knowledge in real work situation, thereby bridging the gap between university work and actual practice.
* To enlist and strengthen employers’ investment in entire educational process of preparing university graduates for employment.
* Abilities to design a system, components, or process to meet desired needs.
* Abilities to function on multidisciplinary teams.
* Broad education necessary to understand the impact of engineering solutions in a global societal context.
* To acquired knowledge of contemporary issues.

## Gilead Biomedical Engineering

Gilead Biomedical Engineering specializes in research and development of solutions in the following areas: biomedical engineering, control and instrumentation, machine learning and artificial intelligence. Established in 2019, the company uses engineering concepts in solving medically related issues. The company is located at 44 Olurounbi compound, opp. Bank of Agric building, Ibadan road, Ile-Ife, Osun state. The company aims at using engineering principles in studying and solving health problems.

The company mainly focuses on research. The research group utilizes a technique called vertical integration, this implies theoretical and research works are done mostly be PhD and M.Sc. students and crafting and technical works are done mostly by SIWES students. This also empowered my practical knowledge and maker skills. The organization structure is shown below.

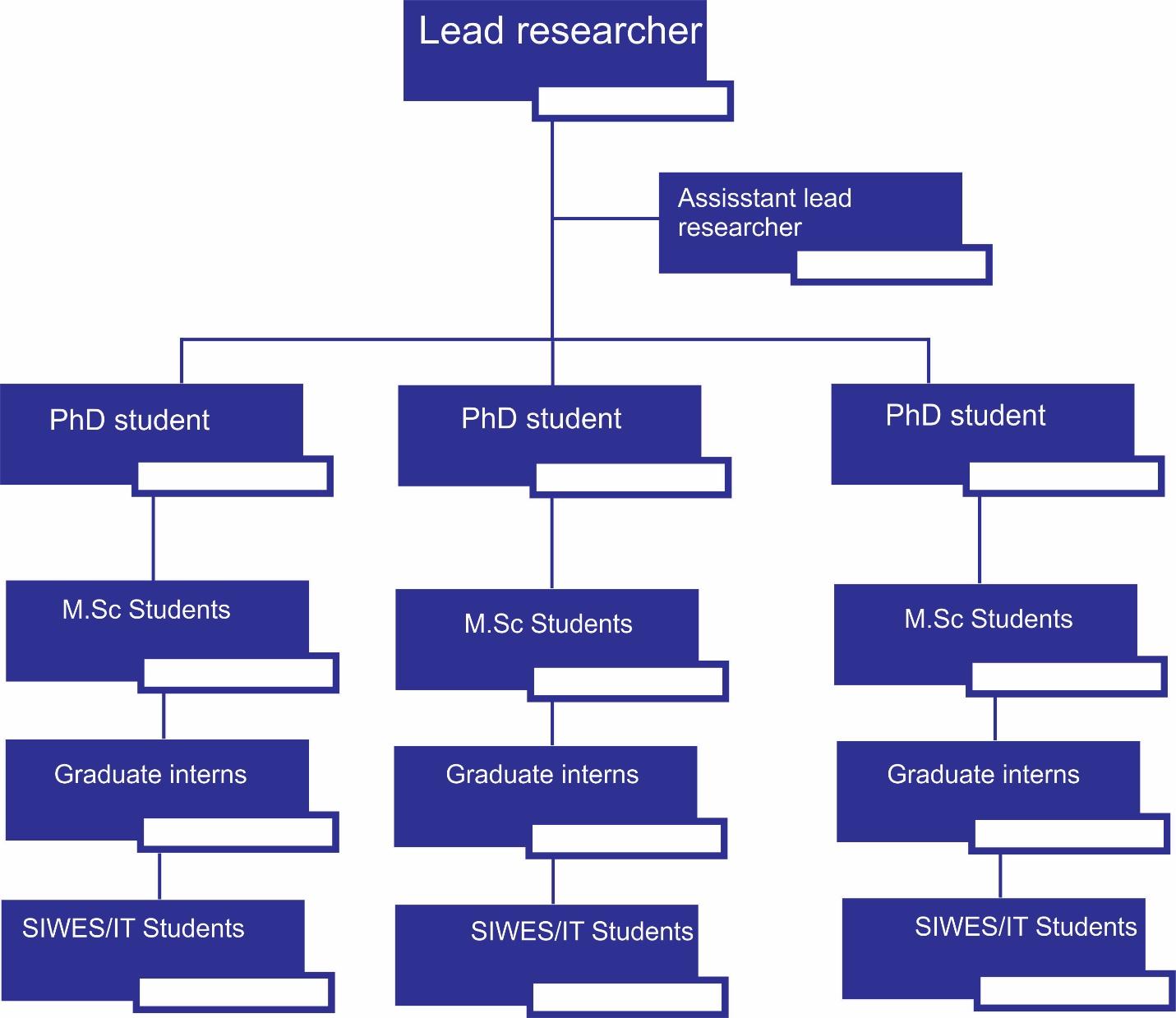


Fig. 1.1: Organization structure of the research group

## Scope of the Report

This report covers the extent of training undergone during my SIWES II program which was undertaken at Gilead Biomedical Engineering, Ile-Ife, Osun state, in partial fulfilment for the award of Bachelor of Science, Electronic and Electrical Engineering Technology.

## Justification of SIWES at the place of IT

Student Industrial Work Experience Scheme (SIWES) was designed to enable students experience the theoretical ideas acquired so far in their respective institutions in real practical forms as related to their respective course of study.

To this end, as a student of Electronic and Electrical Engineering, it thus fits to undergo this program at a related organization, hence the choice of Gilead Biomedical Engineering, Ile-Ife, Osun which specializes in application of engineering concepts to solve biomedical problems.

At Gilead Biomedical Engineering, a lot of electronic and electrical engineering related works were being carried out ranging from design of prosthetic hand, hand orthosis, epileptic seizure detector to design of brain-controlled wheelchairs and rehabilitation robots. These operations to a higher degree have inculcated in me, some required engineering ethics and principles which is one of the purposes of SIWES program.

# CHAPTER TWO

# LITERATURE REVIEW

## 2.1 Machine Learning

According to (Wikipedia, 2020), Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people.

Although machine learning is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve. Machine learning algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs.

### 2.2.1 Categories of Machine Learning

There are three major categories of machine learning:

* Supervised Learning
* Unsupervised Learning
* Reinforcement Learning

Supervised Learning is the most popular paradigm for machine learning. Given data in the form of examples with labels, we can feed a learning algorithm these example-label pairs one by one, allowing the algorithm to predict the label for each example, and giving it feedback as to whether it predicted the right answer or not. Over time, the algorithm will learn to approximate the exact nature of the relationship between examples and their labels. When fully-trained, the supervised learning algorithm will be able to observe a new example and predict a good label for it.

Unsupervised Learning features no labels. Instead, the algorithm would be fed a lot of data and given the tools to understand the properties of the data. From there, it can learn to group, cluster, and/or organize the data in a way such that a human (or other intelligent algorithm) can come in and make sense of the newly organized data. We can say unsupervised learning is data driven.

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. In machine learning, the environment is typically represented as a Markov Decision Process (MDP). Many reinforcement learning algorithms use dynamic programming techniques. Reinforcement learning is very behavior driven. (Heidenreich, 2018) describes reinforcement learning as learning from mistakes. Place a reinforcement learning algorithm into any environment and it will make a lot of mistakes in the beginning. So long as some sort of signal to the algorithm that associates good behaviors with a positive signal and bad behaviors with a negative one is provided, we can reinforce our algorithm to prefer good behaviors over bad ones. Over time, our learning algorithm learns to make less mistakes than it used to. During the course of the Industrial Attachment, I was mainly focused on Reinforcement Learning and as such; I would dive further into its description.

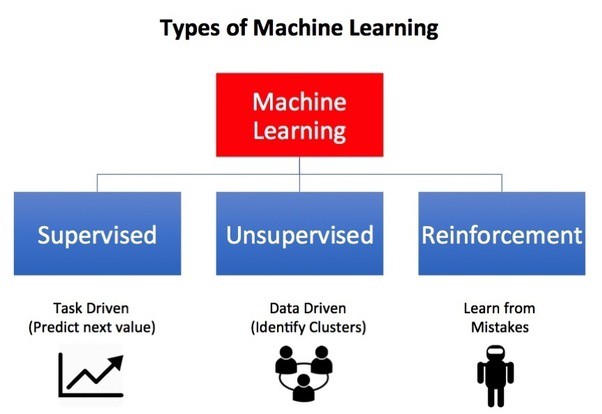


Fig 2.1: Categories of Machine Learning. Source: Towards data science.

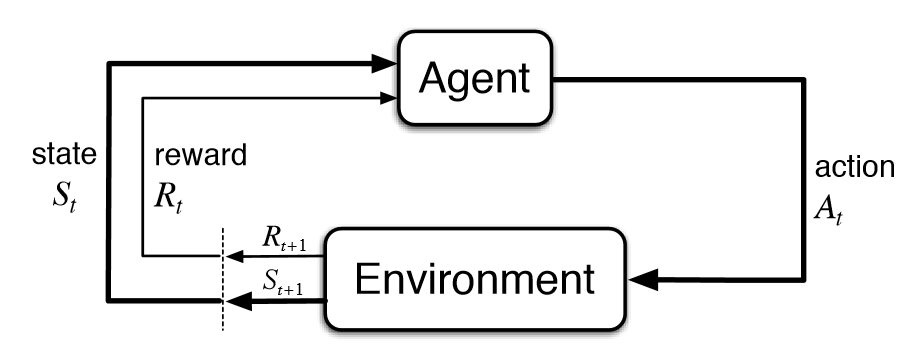


Fig 2.2: The action-reward feedback loop.

## 2.2 Reinforcement Learning

In reinforcement learning, the goal is to find a suitable action model that would maximize the total cumulative reward of the agent. Fig 2.2 above illustrates the action-reward feedback loop of a generic RL model.

According to (Ashraf, 2018), the key idea behind RL includes an environment which represents the outside world to the agent and an agent that takes actions, receives observations from the environment that consists of a reward for his action and information of his new state. That reward informs the agent of how good or bad was the taken action, and the observation tells it what is the next state in the environment.

The agent tries to figure out the best actions to take or the optimal way to behave in the environment in order to carry out his task in the best possible way.

In RL there is no supervisor, only a reward signal or a real number that tells the agent how good or bad an action was. Feedback from the environment might be delayed over several time steps, it’s not necessarily instantaneous e.g. for the task of reaching a goal in a grid-world, the feedback might be at the end when the agent reaches the goal. The agent might spend some time exploring and wandering in the environment until it finally reaches the goal after a while to realize what were the good and bad actions it had taken. The agent influences the environment through its actions which in turn affect the subsequent data it receives from the environment, it’s an active learning process.

### 2.2.1 Elements of Reinforcement Learning

Some key terms that describe the basic elements of an RL problem are:

* Agent – This is the algorithm being trained
* Environment — Physical world in which the agent operates
* State — Current situation of the agent
* Reward — Feedback from the environment

A **Reward** Rt is a scalar feedback signal that indicates how well the agent is doing at time step t. The agent’s job is to maximize the expected sum of rewards. Reinforcement learning is based on the Reward Hypothesis which states that: “All goals can be described by the maximization of expected cumulative rewards”.

In fig. 2.2, the **Agent** and the E**nvironment** interact at each other over a sequence of discrete time steps, t = 0, 1, 2, 3, …. At each time step t, the agent receives some representation of the environment’s state, St ∈ S, and on that basis selects an action, At ∈ A(s). All the sequence of observations, actions, and rewards during the agent’s life time up to time step t is called the history. It’s what the agent has seen so far, i.e. all the observable variables up to time step t. The environment emits next state and a reward associated with the taken action.

The E**nvironment State** is the information used within the environment to determine what happens next from the environment’s perspective, i.e. spit out the observation or next state and reward. The **Agent State** captures what happened to the agent so far, it summarizes what is going on and the agent uses this to pick the next action.

A **Markov State** contains all useful information from the history and is used to represent the agent’s state. An agent state is Markov if that state contains all the useful information the agent has encountered so far.

### 2.2.2 Components of a Reinforcement Learning Agent

* Policy

It’s a probability distribution over actions given states, i.e. the agent’s behavior function or how the agent picks his actions given that it’s in some certain state.

* Value Function

It’s a function that tells us how good is each state and/or action, i.e. how good is it to be in a particular state, and how good is it to take a particular action. It informs the agent of how much reward to expect if it takes a particular action in a particular state.

* Model

A model predicts what the environment will do next. It’s the agent’s representation of the environment, i.e. how the agent thinks the environment works.

# CHAPTER THREE

# DESIGN OF THE GRIPPER IN A FORCE TORQUE SENSOR

## 3.1 What is a Force Torque Sensor?

A force torque sensor detects the different forces that are applied on the robot end effector in the 3 geometric axes (X-Y-Z). The sensor also detects the torque applied around the 3 different axes. By doing so, the sensor gives feedback to the robot and can adapt its motion to feel the minimum of force that is applied to it.

## 3.2 Components of the Force Torque Sensor

The following materials were used in the design of the force torque sensor:

### Load Cell

A load cell is a type of transducer, specifically a force transducer. It converts a force such as tension, compression, pressure, or torque into an electrical signal that can be measured and standardized. As the force applied to the load cell increases, the electrical signal changes proportionally. The most common types of load cell used are hydraulic, pneumatic, and strain gauge.

### HX711 amplifier

It is a small breakout board for the HX711 IC that allows to easily read load cells to measure weight. By connecting the amplifier to a microcontroller, it will be possible to read the changes in the resistance of the load cell, and with some calibration it is able to get very accurate weight measurements. This can be handy for creating personal industrial scale, process control or simple presence detection. The HX711 uses a two-wire interface (Clock and Data) for communication. Any microcontroller’s GPIO pins should work, and numerous libraries have been written, making it easy to read data from the HX711.

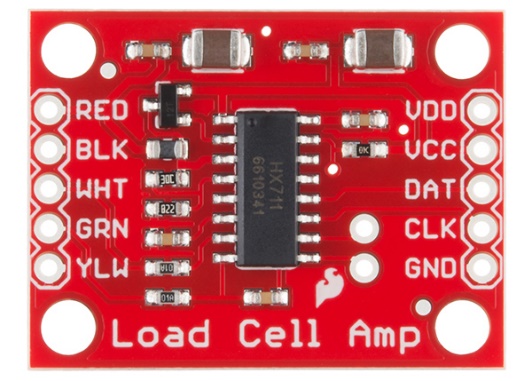
### NI USB-8451 DAQ

The NI USB‑8451 is a master interface for connecting to and communicating with inter-integrated circuit (I2C), System Management Bus (SMBus), and serial peripheral interface (SPI) devices. With plug‑and‑play USB connectivity, the USB‑8451 is a portable solution to communicate with consumer electronics and integrated circuits. It also includes eight general-purpose digital I/O lines for a variety of applications, such as configuring the address of I2C devices or toggling LEDs. The USB‑8451 can be physically located more closely to I2C/SPI devices than PCI interfaces, reducing I2C bus length and minimizing noise problems. Additionally, the interface provides +5 V and GND to power circuits with no external power supply.

### Atmega328p Microcontroller on Arduino Uno

The high-performance Microchip picoPower 8-bit AVR RISC-based microcontroller combines 32KB ISP flash memory with read-while-write capabilities, 1024B EEPROM, 2KB SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible timer/counters with compare modes, internal and external interrupts, serial programmable USART, a byte-oriented 2-wire serial interface, SPI serial port, a 6-channel 10-bit A/D converter (8-channels in TQFP and QFN/MLF packages), programmable watchdog timer with internal oscillator, and five software selectable power saving modes. The device operates between 1.8-5.5 volts. By executing powerful instructions in a single clock cycle, the device achieves throughputs approaching 1 MIPS per MHz, balancing power consumption and processing speed.

Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power jack, an ICSP header and a reset button.



(b)

 (a)

(d)

(c)

Fig 3.1: Components used in the force torque sensor. (a) hx711 amplifier, (b) NI USB-8451 data acquisition device, (c) Arduino Uno development board, (d) 5kg bar type load cell.

## 3.3 Design of the Force Torque Sensor

The force torque sensor is to be used in the end effector of the PULSR research robot being developed in the company.

I worked on measuring the force applied using two 5kg bar type load cells connected to a hx711 amplifier. Load cells use a four-wire Wheatstone bridge configuration to connect to the HX711 (see fig 3.2).

I then connected the hx711 to the Atmega328p microcontroller on the Arduino Uno using the digital I/O pins. I then proceeded to write the code in the Arduino IDE using the hx711 library.

The data acquired by the microcontroller is to be used in a LabVIEW VI to measure the signal rate of obtained forces. To do this, the data is transferred from the microcontroller to the NI 8451 data acquisition device. The I2C configuration bus of both devices is used to achieve this. The Arduino was used as the slave while the USB-8451 was the master. I connected both devices using their designated I2C pins and used a Master Reader/Slave Sender approach. The Arduino (slave) sent data over the I2C port to the USB-8451(master) which read the data.

### 3.3.1 Introduction to I2C Configuration

I2C is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems. It was invented by Philips and now it is used by almost all major IC manufacturers. Each I2C slave device needs an address – they must still be obtained from NXP (formerly Philips semiconductors).

Each slave device has a unique address. Transfer from and to master device is serial and it is split into 8-bit packets. All these simple requirements make it very simple to implement I2C interface even with cheap microcontrollers that have no special I2C hardware controller. You only need 2 free I/O pins and few simple i2C routines to send and receive commands.

The initial I2C specifications defined maximum clock frequency of 100 kHz. This was later increased to 400 kHz as Fast mode. There is also a High speed mode which can go up to 3.4 MHz and there is also a 5 MHz ultra-fast mode.

### 3.3.2 I2C Interface

I2C uses only two wires: SCL (serial clock) and SDA (serial data). Both need to be pulled up with a resistor to +Vdd. There are also I2C level shifters which can be used to connect to two I2C buses with different voltages.

### 3.3.3 I2C Addresses

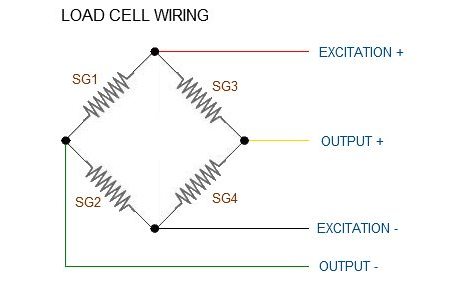
Basic I2C communication is using transfers of 8 bits or bytes. Each I2C slave device has a 7-bit address that needs to be unique on the bus. Some devices have fixed I2C address while others have few address lines which determine lower bits of the I2C address. This makes it very easy to have all I2C devices on the bus with unique I2C address. There are also devices which have 10-bit address as allowed by the specification.

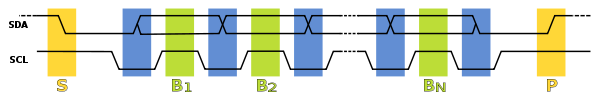
7-bit address represents bits 7 to 1 while bit 0 is used to signal reading from or writing to the device. If bit 0 (in the address byte) is set to 1 then the master device will read from the slave I2C device.

Master device needs no address since it generates the clock (via SCL) and addresses individual I2C slave devices.

### 3.3.4 I2C Protocol

In normal state both lines (SCL and SDA) are high. The communication is initiated by the master device. It generates the Start condition (S) followed by the address of the slave device (B1). If the bit 0 of the address byte was set to 0 the master device will write to the slave device (B2). Otherwise, the next byte will be read from the slave device. Once all bytes are read or written (Bn) the master device generates Stop condition (P). This signals to other devices on the bus that the communication has ended and another device may use the bus.



Fig 3.2: Load Cell Wiring to HX711 Amplifier

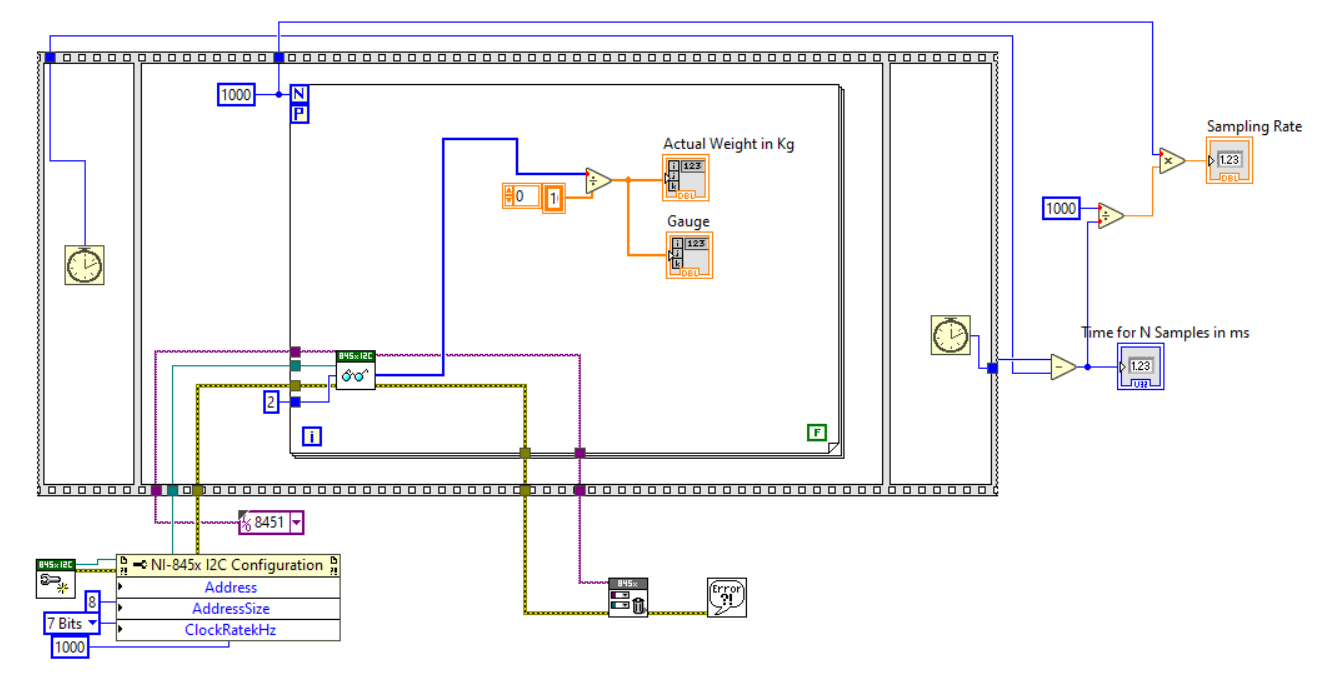
Fig 3.3: I2C Protocol

Fig 3.4: LabVIEW VI showing connection using the I2C configuration

# CHAPTER FOUR

# REINFORCEMENT LEARNING IN ROBOTICS

## 4.1 Reinforcement Learning for Industrial Robots

I applied knowledge gained on Reinforcement Learning during the Industrial Attachment to simulate an industrial robot. This project explores the process of training a robotic arm to accomplish a given task. The robotic arm which simulates the repetitive task of assembly robots was designed in a VREP environment (see Fig 4.1) and trained using a Reinforcement Learning algorithm.

### 4.1.1 Problem Statement

In this project, a **Uarm with Gripper** model was taken and taught how to hold a given object and place it in a given location. This is a common use case in many industrial and home applications. However, traditionally robots have been programmed assuming a controlled environment, making it impossible for them to adapt to new environments.

In this project, classic Q-learning was used as a learning technique. It is an online, model-free and off-policy reinforcement learning approach. The arm will have to learn to locate the object, grab it, lift it, maneuver it to top of the bin and release it. Appropriate reward strategy was designed to enable the agent to do so.

The task is episodic in nature and is considered successfully finished if the robot arm puts (drops) the object in the bin.

### 4.1.2 Metrics

The following metrics were tracked for gauging effectiveness of the solution:

* Success/failure per episode: When an episode terminates, it will either have failed in

it’s goal or succeeded. Initial episodes will see failures because of novelty of the task

and gradually should start succeeding. This metric will let us know how many episodes it took for training before the arm learnt the task.

* Cumulative reward per episode: How much reward did the agent collect before the

episode ended? We should see a gradual increase in accumulated reward: and it should

move from negative to positive. Even when positive, we will see failures initially if the

episode ends in one of the fail states. Gradually it should start succeeding.

* Actions per episode: The challenge of the arm is to take the least number of actions to

accomplish the task. This metric will track that.

### 4.1.3 Analysis

To solve a reinforcement learning problem, it is critical to segregate the agent from its environment and define the actions, states and rewards.

In this solution:

* The Agent is the entity which controls the robotic arm. It implements the learning

algorithm, takes the actions, perceives the state and receives the rewards.

* The Environment consists of the robotic arm, the object and the bin. It reacts to actions

taken by the agent.

### 4.1.4 The State Space

The state space of our environment consists of:

* Position of the gripper of the robotic arm: x, y, z coordinates
* Position of the object: x, y, z coordinates
* State of the gripper: engaged or disengaged

The active space of the problem was discretized for this project. As shown in fig 4.2, the space around the arm, object and bin was discretized along the x, y and z axis.

The min and max locations along x, y and z axis as well as the step size has been parameterized and was set in code.

Therefore, the number of states the environment can be in, is the product of (a) the number of discrete locations where the object can be located, (b) the number of discrete locations where the arm can be positioned and (c) number of states of the gripper (engaged/disengaged).

### 4.1.5 The Actions

There are two types of actions possible in our environment:

* Move the arm to a specific location in the active space.
* Engage / disengage the gripper

Therefore, the total number of actions possible is the product of (a) the number of discrete locations where the arm can be in and (b) number of states of the gripper (engaged/disengaged).

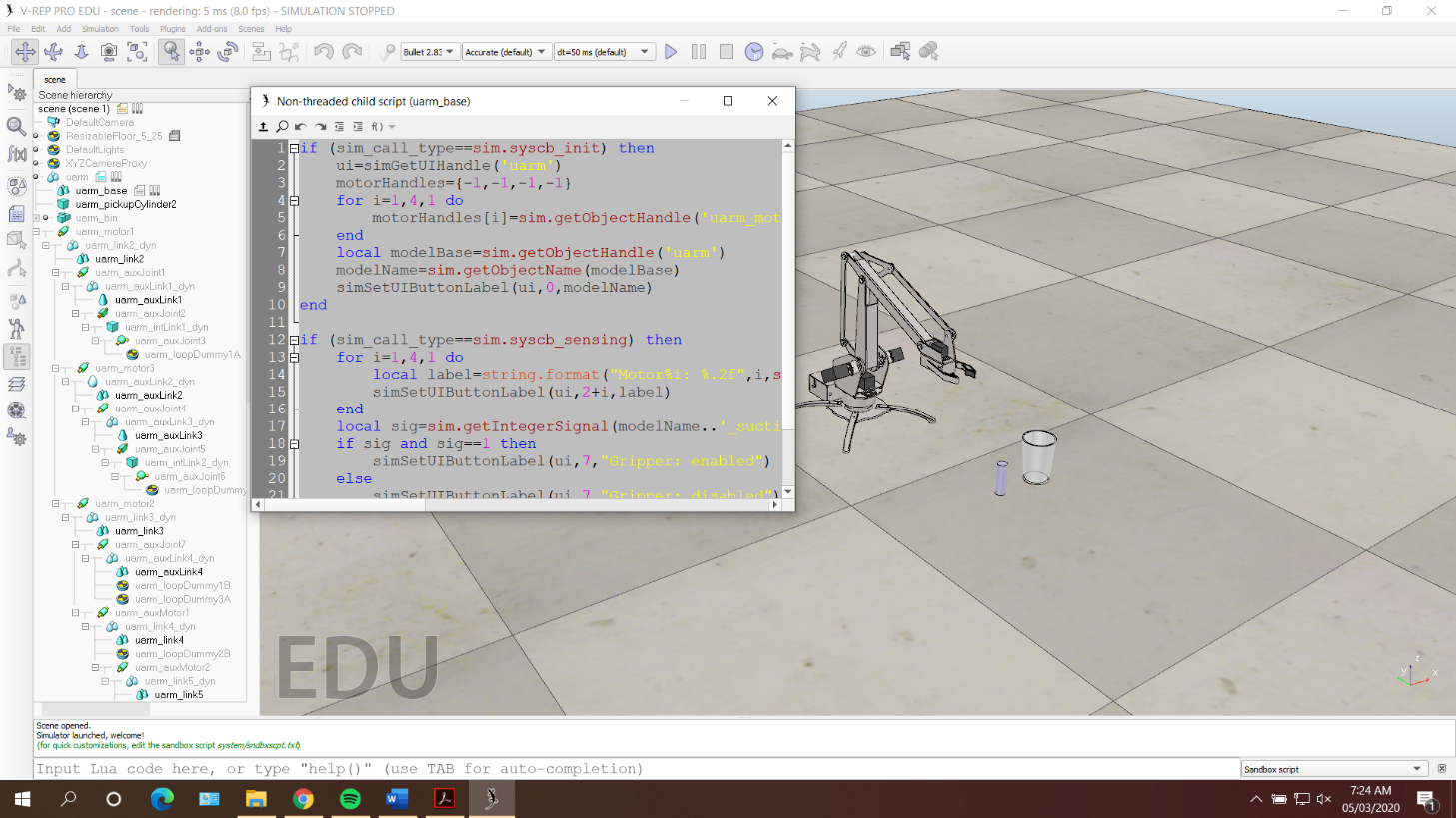


Fig 4.1: The VREP environment

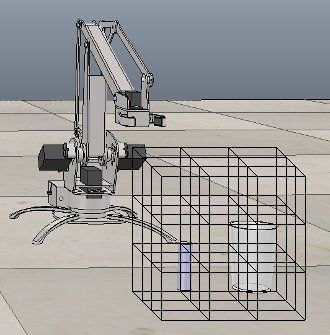
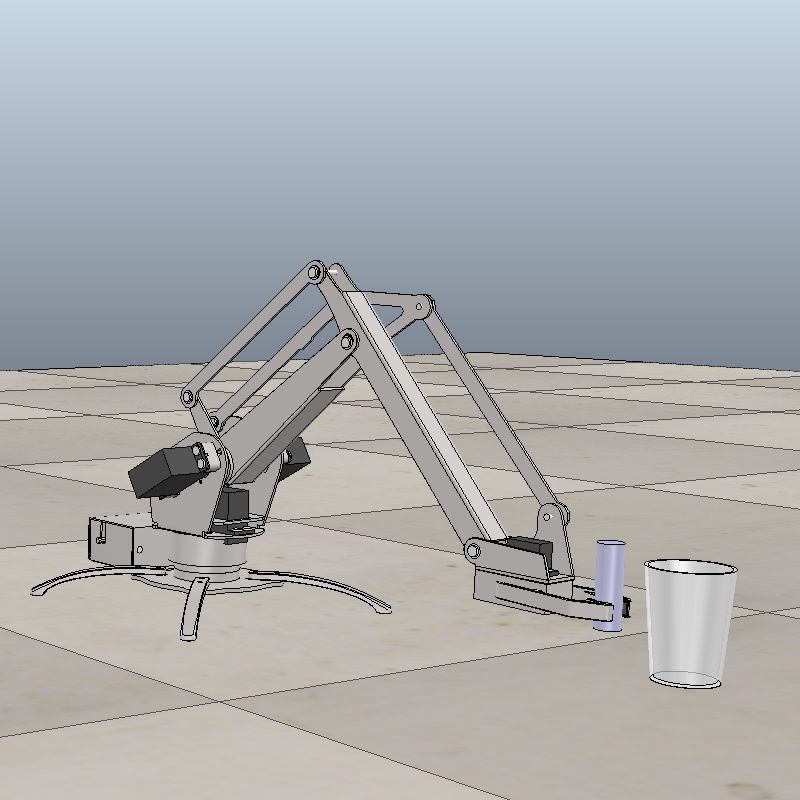
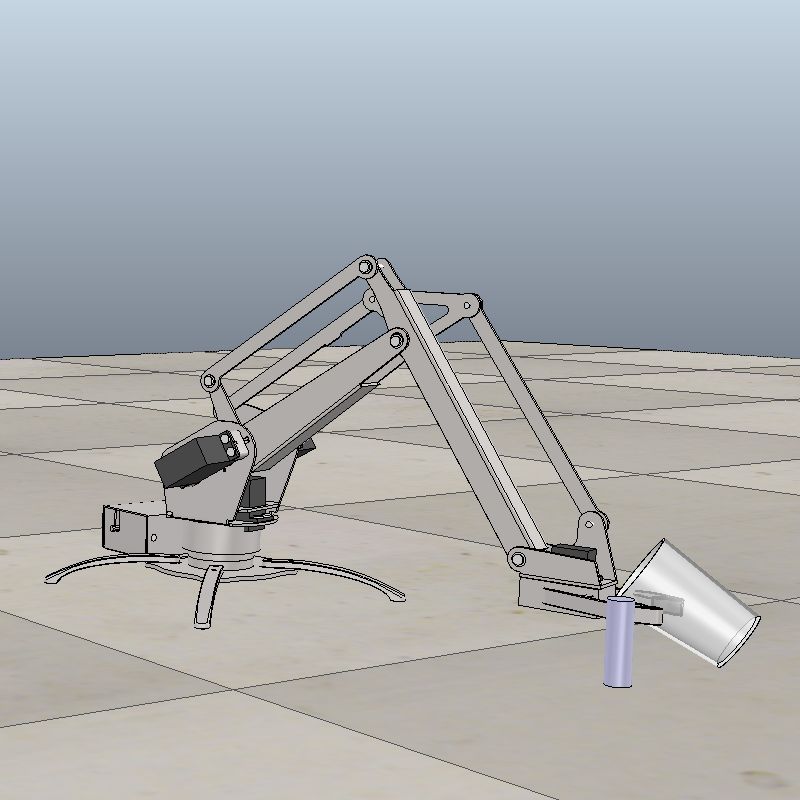


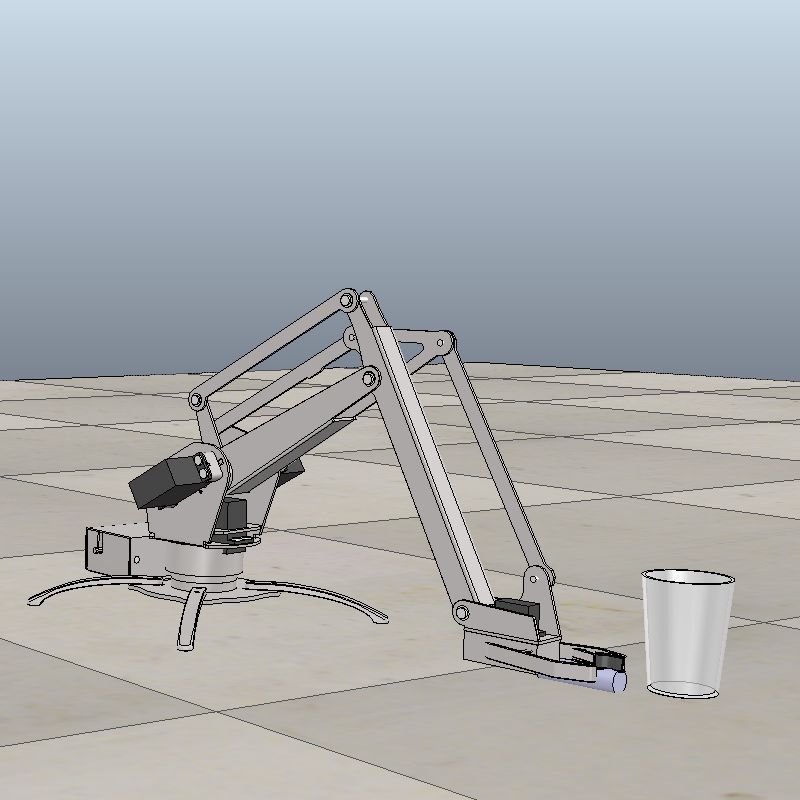
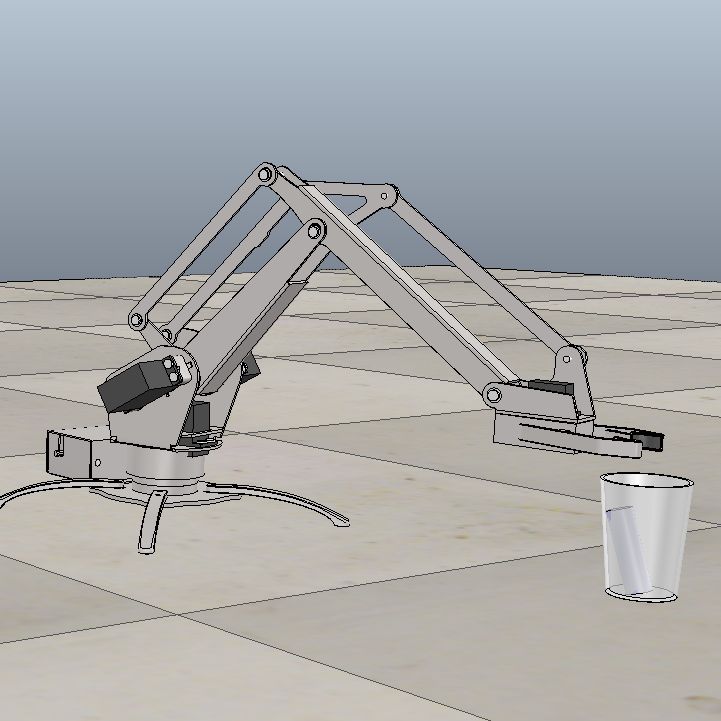
Fig 4.2: The robotic arm with the active space discretized.





(b)

(a)



(d)

(c)

Fig 4.3: Visual representation of states. (a) The grip could grab the object for the first time in an episode, (b) The arm displaces the bin, (c) The robotic arm accomplished the task, (d) The arm topples the object.

### 4.1.6 The Reward Strategy

Devising a reward strategy was challenging and was arrived at by observing multiple iterations for this specific problem.

Table 4.1: Conditions and specified reward functions

|  |  |  |
| --- | --- | --- |
| **S/N** | **Condition** | **Value** |
| 1 | The environment reached invalid state | -10 |
| 2 | The arm topples the object | -10 |
| 3 | The arm displaces the bin | -10 |
| 4 | The grip was engaged when there was no object in it | -3 |
| 5 | With the object in grip, the grip was disengaged. But  the object did not fall in the bin. | -3 |
| 6 | The grip could grab the object for the first time in an  episode | 5 |
| 7 | The grip could grab the object and drop it in the bin  (objective achieved) | 10 |
| 8 | Default reward | -1 |

Rewards (1), (2) and (3) were most negative because the arm movement would cause the environment to reach a state from where no further actions would be possible.

Rewards (4) and (5) were put in place to suggest the arm what not to do.

Reward (6) was to encourage the gripping of the object.

Reward (7) was the maximum that any action could get. This suggests achieving the goal.

Reward (8) was the default any action would get. A negative value would encourage the arm to reach the goal in as less actions as possible.

### 4.1.7 Algorithms and Techniques

Classic Q-Learning technique was used to solve this problem. Several approaches exist to solve reinforcement learning problems, but Q-learning has some useful properties:

* Unlike dynamic programming, Q-learning does not need complete model of the environment (state transition probabilities).
* Unlike dynamic programming, Q-learning learns from experience.
* Unlike Monte Carlo, Q-learning doesn’t need to wait for the episode to end to update the values. It can do so online as the actions are taken and rewards obtained.
* Can also be applied to non-episodic and continuous tasks.

**Exploitation vs Exploration: Epsilon-Greedy with Decay**

Q-learning is an off-policy algorithm. This means that it learns the values of a state-action pair by choosing and taking actions on its own, and doesn’t depend on any policy.

However,

* If the random actions are chosen, it will take enormous time to converge and may not yield optimal solution. Choosing actions randomly is called exploration, since we choose to ignore the current best action in favor of a probable better future action.
* If only the best actions (based on the best q-value available at current state) is always chosen, we may not get the optimal solution. Choosing the best action available is called exploitation, since we choose to exploit our current knowledge base of what is known to be best.

With the epsilon-greedy approach, we allow the agent to sometimes explore rather than exploit. The rate of exploration is determined by a percentage called epsilon.

In this solution, we also decay the epsilon percentage value with each episode to allow more explorations at the start of the learning but reduce the rate as the learning progresses.

**When to Stop Learning**

The learning process would keep playing episodes and update the Q table in the process. Four conditions were chosen to decide when to stop the learning process. We would stop learning after an episode if:

* Minimum episodes: A minimum number of episodes were executed (parametrized)
* Success state: And When an episode would end with task accomplished
* Least number of actions: And when the episode would do so with least number of actions
* All steps exploitative: And when the episode would do so with all actions taken using exploitation (and not exploration)

The last condition is important. It was observed that episodes could accomplish the goal using exploration early on. However, it would be premature to stop the learning, since the exploration might not yield a positive q-value. Such a Q table may not lead to goal state.

### 4.1.8 Framework Setup

V-REP was chosen to simulate a reinforcement learning environment. Towards that,

* V-REP version 3.4 ref 1 was installed.
* A scene with the robotic arm, a cylindrical object and a bin was created.
* Custom child scripts were written to act as hooks into the framework.
* V-REP python remote client scripts were imported into the project

### 4.1.9 Implementation

To start the simulation, one would invoke the main.py script with the argument ‘--train’

python main.py --train

Once training would end, one can test the final model by invoking main.py script with the

argument ‘--test’

python main.py --test

The main loop starts executing episodes and gathering statistics. It checks for the conditions to

stop the learning.

**if** train\_mode:  
 *# Here we train the agent and generate the qtable.  
 # qtable will be dumped in qtables/qtable.txt.npy* **while** episodes > 0:  
 log\_and\_display(**'=============================================> Episode '** + str(episodes))  
 agent.reset()  
 total\_reward, total\_steps, success, total\_explorations = agent.execute\_episode\_qlearn(config.NUM\_MAX\_ACTIONS)  
 episode\_num = config.NUM\_EPISODES - episodes + 1  
 *# Reduce exploration rate with each episode* agent.epsilon = np.maximum(0.0, config.EPSILON - episode\_num \* config.EPSILON\_DECAY)  
  
 stat\_file = open(config.PLOT\_FILE, **'a'**)  
 stat\_file.write(**"{0}, {1}, {2}, {3}, {4}, {5}\n"**.format(episode\_num, success, total\_reward,  
 total\_steps, total\_explorations, agent.epsilon))  
 stat\_file.close()  
 agent.save\_qtable()  
 episodes -= 1  
  
 **if** success **and** total\_steps == config.MIN\_ACTIONS\_EXPECTED \  
 **and** total\_explorations == 0 **and** episode\_num > config.MIN\_EPISODES\_TO\_RUN:  
 log\_and\_display(**'Optimal moves learnt. Terminating training. Now run agent with epsilon as 0.'**)  
 **break**

The execution of one episode consists of choosing an action, executing that action, and

updating the Q-table.

**def** execute\_episode\_qlearn(self, max\_steps: int):  
 *"""Invoking this method will execute one episode of training.  
 """* total\_reward = 0  
 total\_steps = 0  
  
 **while** max\_steps > 0 **and not** self.env.is\_goal\_achieved() **and not** self.env.environment\_breached:  
 action\_id = self.select\_action(self.current\_state\_id)  
 reward = self.execute\_action(action\_id)  
 new\_state\_id = self.env.actionstate\_curr[**'current\_state\_id'**]  
 self.update\_q\_table(self.current\_state\_id, action\_id, reward, new\_state\_id)  
 self.current\_state\_id = new\_state\_id  
  
 max\_steps -= 1  
 total\_reward += reward  
 total\_steps += 1  
  
 **return** total\_reward, total\_steps, self.env.is\_goal\_achieved(), self.total\_explorations

While choosing an action, we apply the epsilon-greedy approach. Epsilon is set in the config file. Also, its value is decayed per episode run so that as time progresses, we decrease exploring.

**def** select\_action(self, current\_state\_id):  
 *"""This method returns an action based on current state. It returns a mix of  
 exploratory and exploitative actions based on epsilon value.  
 """* **if** np.random.uniform() < self.epsilon:  
 log\_and\_display(**'Exploring...'**)  
 self.total\_explorations += 1  
 action\_id = np.random.choice(self.env.total\_actions)  
 **else**:  
 log\_and\_display(**'Exploiting...'**)  
 action\_id = np.argmax(self.q\_table[current\_state\_id])  
 **return** action\_id  
  
**def** execute\_action(self, action\_id):  
 action = self.env.actions[action\_id]  
  
 **if** action[0] == self.env.action\_type1:  
 log\_and\_display(**'Action: Moving claw '** + str(action[1]))  
 **return** self.env.move\_arm(action[1], action\_id)  
 **elif** action[0] == self.env.action\_type2:  
 log\_and\_display(**'Action: Engaging/Disengaging claw '** + str(action[1]))  
 **return** self.env.enable\_grip(action[1], action\_id)

The training of the arm generated three files:

* log.txt: The execution log with details on how the Q-values were updated
* qtable.txt.npy: The Q-table dumped in file
* episodes.txt: Details of each episode

### 4.1.10 Results and Conclusions

The simulation was allowed to run for 2500 episodes and data was collected. Initial episodes mostly ended in failures. There were some successes below 1400 episodes.

Post 1700 episodes, we see more frequent successes, suggesting (a) the model has gradually learnt the task (b) we are doing more exploitations than explorations.

The maximum reward achieved was 12, and minimum was -47. The initial successful episodes received less than 12, mostly because they accomplished the task with more steps, which is penalized by default reward of -1.

The initial successful episodes took more steps/actions than later ones. The task could be accomplished with minimum of 5 steps.

Initially the arm makes many mistakes but learns along the way. The various conditions which were considered for which reward/penalty was necessary.

### 4.1.11 Further Work and Improvement

Several improvements could be made to the solution:

* The training could have happened with the object kept at random locations. This would have made the robot arm learn to pick it from any location in the active space. This would have of course taken more time to train. I choose not to do it for time constraints.
* As can be seen, the state space can become very large as we increase the active space. To tackle this, we can either do function approximation, or implement DQN.
* Asynchronous methods could have been used to parallelize the training to decrease training time.

## 4.2 Reinforcement Learning for Efficient Gait Cycle of Biped Robots

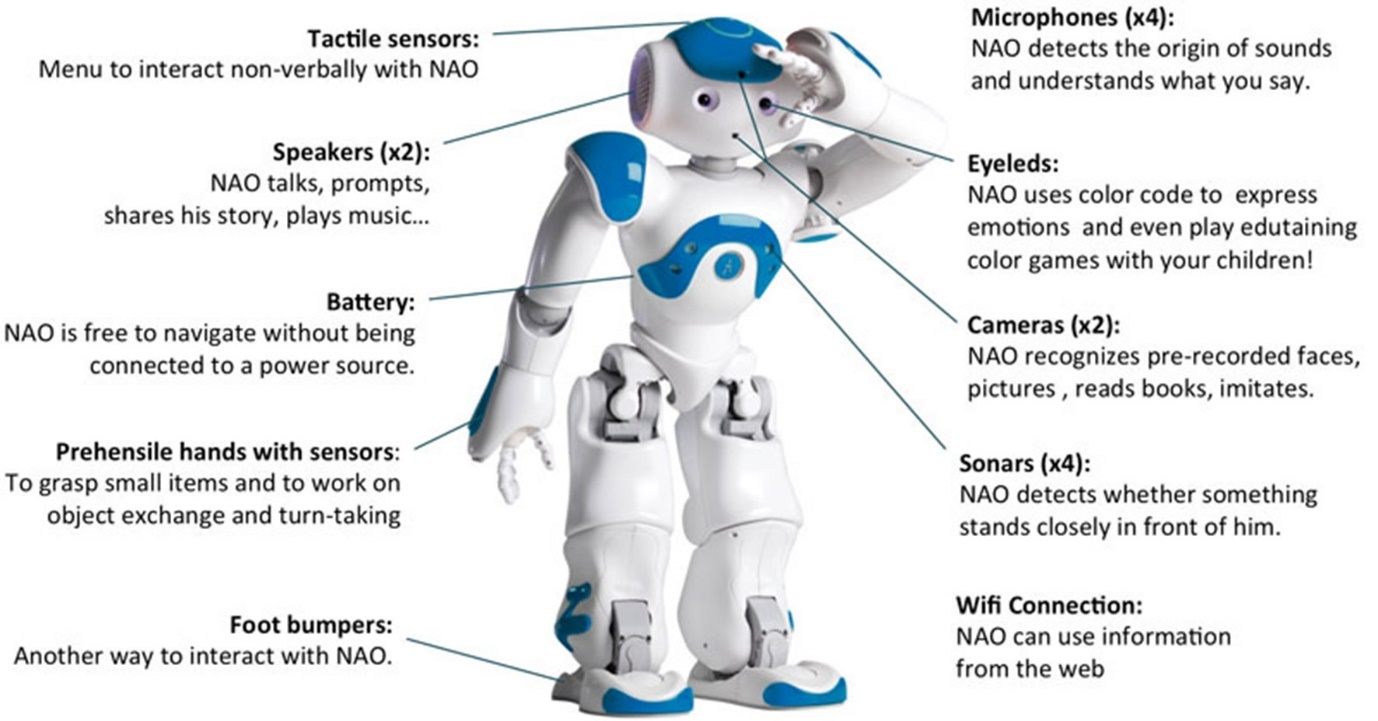
Programming robots for performing different activities requires calculating sequences of values of their joints by taking into account many factors, such as stability and efficiency, at the same time. Particularly for walking, state of the art techniques to approximate these sequences are based on reinforcement learning (RL). In this work, a multi-level system where the same RL method is used first to learn the configuration of robot joints (poses) that allow it to stand with stability, and then in the second level, we find the sequence of poses that let it reach the furthest distance in the shortest time, while avoiding falling down and keeping a straight path. The model is implemented in a simulated environment using q-learning. The NAO robot by Softbank Robotics is used in this simulation.

### 4.2.1 The NAO Robot

NAO is an autonomous, programmable humanoid robot developed by Aldebaran Robotics. The NAO robot is controlled by a specialized Linux-based operating system, dubbed NAOqi. The OS powers the robot's multimedia system, which includes four microphones (for voice recognition and sound localization), two speakers (for multilingual text-to-speech synthesis) and two HD cameras (for computer vision, including facial and shape recognition). The robot also comes with a software suite that includes a graphical programming tool dubbed Choregraphe; a simulation software package and a software developer's kit.

**Kinematics**

NAO has 25 degrees of freedom (joints), 11 DOF for the lower part, including legs and pelvis, 14 DOF for the upper part including it's strunk, arms and head. Each leg has 2 DOF at the ankle, 1 DOF at the knee and 2 DOF at the hip. A mechanism made of two coupled joints at each hip equips the pelvis.



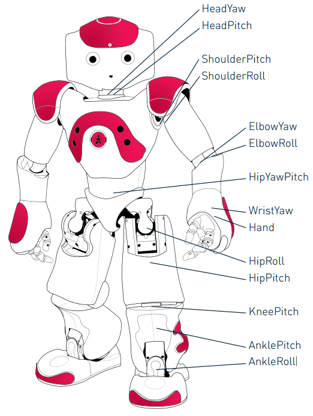
Fig 4.4: The NAO robot with equipped sensors

Fig 4.5: NAO robot and degrees of freedom

### 4.2.2 Problem Statement

Biped robots are designed with a physical structure that tries to emulate a human being with the purpose of providing versatility in terms of movement in such a way that these robots can move through irregular terrains and are better adapted in comparison with robots with wheels. This kind of robots is easily adapted to the human environment due their locomotion, and humans can adapt easily to their interaction because of their similarity in terms of physical structure. Given their kinematic complexity (more than twenty degrees of freedom), the methods to control them are highly complex, with this level of complexity and their extensive potential a recurrent theme of study for many researchers, both in the robotics area and the artificial intelligence area.

### 4.2.3 Methodology

To undertake this, I used an approach from structured programming called **decomposition**. Decomposition means breaking a complex problem or system into parts that are easier to conceive, understand, program, and maintain i.e. To divide a large problem into smaller ones. To teach the robot to learn complex behavior, it involved learning in a multiple-level system using a framework made of different levels: base level, first level, second level, and third level.

At the lowest level of this framework, the base level, lies the direct interaction with the angles of all the joints of the robot, and in the first level, we have specific configuration of the joints to conform a pose. The second level consists of simple activities, and finally, at the upmost, third level, more complex activities (tasks) are defined. In this framework each level uses the knowledge of the previous level to build a new knowledge level.

**Joint Angle (Base Level)**

A joint angle is a change in the angle of a particular joint. In other words, calling a joint angle module will interact directly with the increase or decrease of the angle of a joint. This level is the base of the decomposition framework and its modules (extension or flexion) are directly defined.

**Pose (First Level)**

A pose is a specific configuration of the robot joints in an instant t, which is a list that contains the information about the position of the whole body of the robot. For instance, a pose can be written as:

*Pose = [Left-Shoulder, Left-Elbow, Right-Shoulder, Right-Elbow, Left-Hip, Left-Knee, Left-Ankle, Right-Hip, Right-Knee, Right-Ankle]*

This pose definition has ten different values to be filled. This level of the decomposition framework is learned using the information that the previous level provides

**Activity (Second Level)**

An activityis a combination of poses that correspond to a specific action. For instance, an activity can be a movement of a robot. That is the essence of this level, the idea (as in all the levels) is to use the previous knowledge—the poses—to complete an activity. To accomplish it, the system must combine the available poses, in different orders, until it is able to reach its goal. Some examples of activities that can be completed with poses are walking forward, walking backward, sitting down, standing up, turning left, turning right, taking an object etc.

The goal is to train the robot to walk. I will begin by establishing each level in details

### 4.2.4 Base Level Modules Definition

In the base level, we have the joint angle modules. These modules have to be defined because this is the basis for learning in the next levels. For this work, I considered the joints that are utilized when walking on the sagittal plane, that is, 10 of the 25 joints of the NAO.

Each of the 10 joints considered has 2 movements: extension and flexion. Taking this into account, there’s a total of 20 modules defined for the base level

### 4.2.5 First Level Modules Definition

Once I set the base method, I was able to define the poses, that is, the modules of the first level in the decomposition framework. Based on previous study, many of the works on biped walking use ZMP trajectories

**Zero Moment Point (ZMP)-Based Poses**

In order to learn poses, I selected them based on the ZMP criterion because from previous study, following a ZMP trajectory enables the robot to walk. I defined 12 modules in the first level of knowledge, where each of them was calculated by considering the ZMP criterion, i.e., following a specific position of it. The ZMP coordinates can be calculated as follows:

where W is the width and L the length of the foot sole and f1 + f2 + f3 and f4 are the four sensors located in the sole of a robot’s foot; the NAO robot already comes with these sensors built-in.

The x,y-coordinates of the ZMP of each pose from 1 to 6 located in the sole of the right foot and are defined as:

Pose 1: –0.25 ≤ Y ≤ 0.25 and −0.25 ≤ X ≤ 0.25

Pose 2: 1.25 ≤ Y ≤ 1.75 and −0.25 ≤ X ≤ 0.25

Pose 3: 2.75 ≤ Y ≤ 3.25 and −0.25 ≤ X ≤ 0.25

Pose 4: 4.25 ≤ Y ≤ 4.75 and −0.25 ≤ X ≤ 0.25

Pose 5: 1.75 ≤ Y ≤ 2.25 and 0.75 ≤ X ≤ 1.25

Pose 6: 1.75 ≤ Y ≤ 2.25 and -1.25 ≤ X ≤ −0.75

Poses from 7 to 12 are exactly the same but the ZMP coordinate is computed using the force sensors in the left foot instead of the right foot.

### 4.2.6 Learning the Poses

The poses are learned using q-learning and artificial neural networks. The use of artificial neural networks allows to generalize the states. That is, instead of stating that a specific state corresponds to certain action, I generalize groups of states that correspond to that particular action.

The actions of the Q-network in this level of knowledge are the methods of the previous level, in this case, the 20 joint angle methods defined in the previous level.

The state has a length of four values and it is the combination of the ZMP coordinates of both feet, because we want the robot to have a precise coordinate of the ZMP.

The reward proposed was 10 if the robot reached the goal coordinates, and −10 if the robot falls down. It is 0 in any other case. The purpose was to avoid falling down by giving a negative reward; remember that for q-learning, the goal is to maximize reward.

**First Level Q-Network**

The input of the Q network is the state and the output is a layer of several neurons, one neuron for each available action. The output yields, as a result, a q-value for each action. When the training is finished, the maximum value is selected. To update the weights of the Q-network, I used backpropagation with the difference that there’s no static target y vector, so I use the following equation to compute our target for every state-action pair;

*Q* (*St*, *At*) *← rt*+1 + *γ·maxQ*(*St*+1, *At*+1)

except when a terminal state is reached where the reward update is simply rt+1.

**First Level Q-Network Size**

The only thing left is the size of the Q-network. I used a Multi-Layer Perceptron (MLP) model with four neurons in the input layer, two hidden layers with hyperbolic tangent (tanh) activation function (one with 165 neurons and the other with 120 neurons), and finally an output layer with 20 neurons, each of them corresponding to one available action.

The input layer and the output layer are defined for the problem. We have chosen hyperbolic tangent for the activation functions of the hidden layers because we expect, in some neurons, negative values (see Figure 4.6 for the architecture of this network).

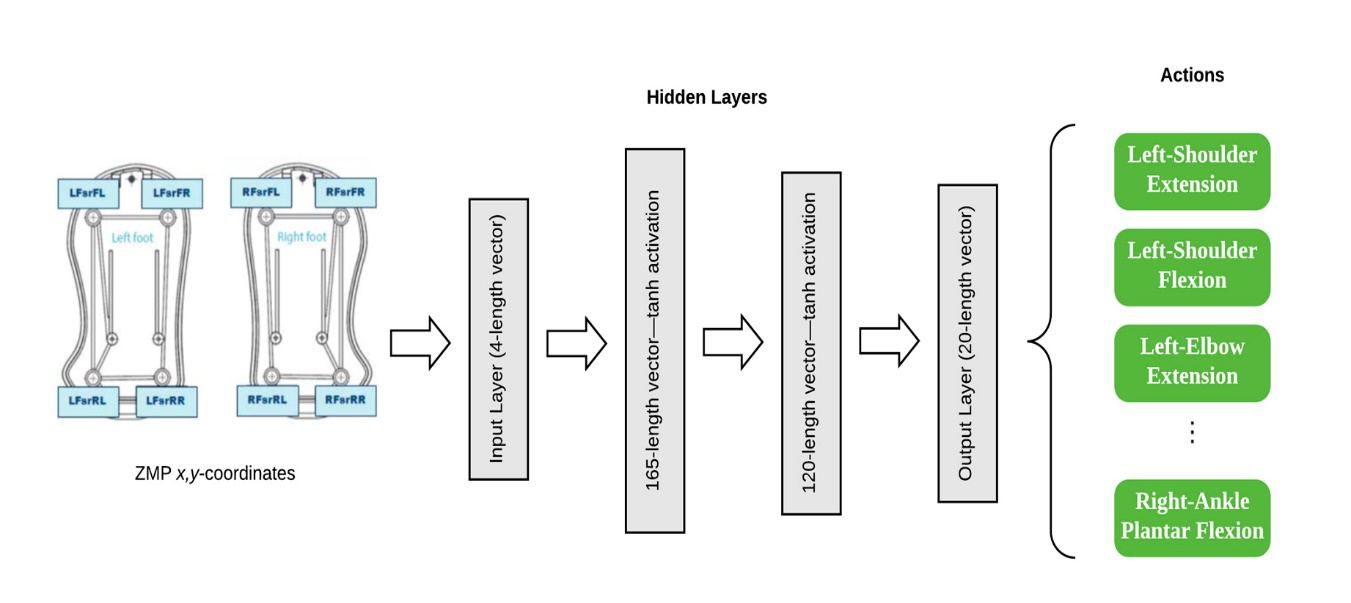


Fig 4.6: First Level Q-Network

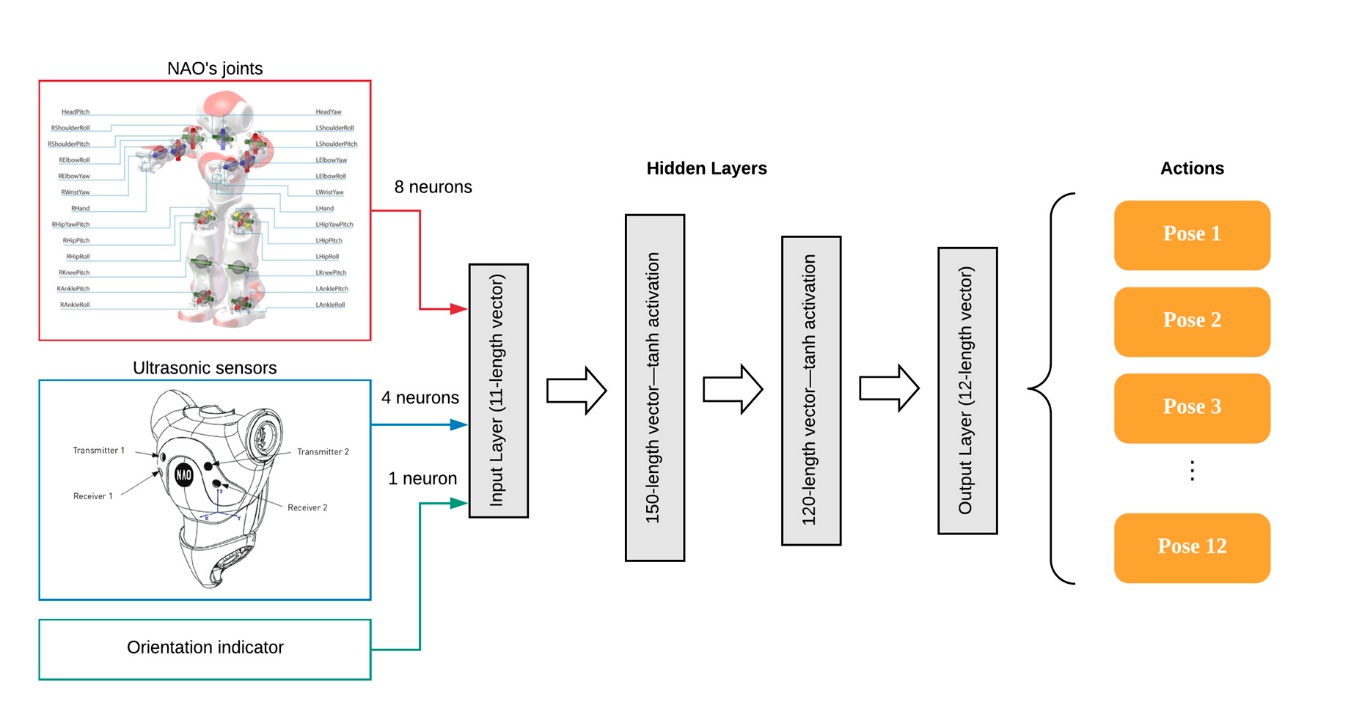


Fig 4.7: Second Level Q-Network

### 4.2.7 Second Level Module Definition

Once we have the poses and the modules of the first level, we are able to obtain the next level method (action). In the previous level we used Q-networks to learn the poses, so in this level we will use the same technique in order to have a uniform way of learning in each level. The goal is to find a combination of poses that allows the robot to walk without falling down, as fast as possible.

**Actions of the Second Level**

The actions available for this network are the modules learned in the previous level, the poses, where each of the poses learned before are options to accomplish our goal in this level. The problem is reduced to finding which poses and what order of these poses make the robot walk.

**States of the Second Level**

States at this level are conformed according to the real value of the joints in both legs of the robot by considering the hip, the knee, and the ankle. These values correspond to the current position of the robot.

Additionally, information about the distance is needed. This is important because our system needs the notion of displacement. To obtain this information I used the ultrasonic sensors of the NAO robot (see Fig 4.7 above).

In order to reduce the time of convergence of the algorithm, walking was restricted to only walking in a straight line; that is, if the robot deviates to the left or to the right by more than 10◦, the algorithm was penalized, with the expectation that it avoids the repetition of the movement that led to that angle in future.

**Reward of the Second Level**

The reward was set in a way that, if the robot advanced 8 cm, it will receive a reward of 10. If the robot fell or if the robot deviated from the straight path it will receive a −10 reward. The q-learning framework aimed to maximize the reward. With a negative reward, the system would try to avoid any path that led to this condition. In any other case, the reward was 0.

**Second Level Q-Network**

The proposed Q-network in this level is composed of one input layer with 11 neurons, where 8 of them are taken from the joints of the robot: (1) left hip pitch, (2) left knee pitch, (3) left ankle pitch, (4) left ankle roll, (5) right hip pitch, (6) right knee pitch, (7) right ankle pitch, and (8) right ankle roll. In this state, only both legs of the robot are considered, four values for each one; this is again, to reduce the complexity.

The remaining neurons correspond to the ultrasonic sensors (two neurons) and the indicator of the deviation of the robot. If the robot deviates in an angle of more than 10 degrees to the left or to the right, this indicator value is −1, otherwise the indicator gives a value of +1.

The proposed net has two hidden layers, the first one has 150 neurons with a tanh activation function, and in the second hidden layer, there are 120 neurons with the same tanh activation function. Finally, an output layer of 12 neurons with the activation function ReLU (rectifier linear unit) is created, with each of the neurons corresponding to one action. In this level, the actions available are the poses that were learned before; remember in the first layer the system learned 12 poses, so these are the available actions for this level. This architecture is shown in Fig 4.7.

### 4.2.8 Tying it all together

A brief overview of the software and algorithms used for simulating and training the networks used in this method.

**Simulation Software**

I used a simulator software called VREP by Coppelia Robotics. It allows for the launch of a simulated NAO moving in a virtual world. In this simulator, we can read all the sensors of the robot, the force sensors to compute the ZMP, the joints sensors that give the real value of the joints, the ultrasonic sensor needed to measure the distance covered, etc. The simulator allows the user to interface with C, C++, and Python.

**Q-Network Algorithm**

To communicate with the NAO robot, I used the NAOqi Framework. NAOqi is the name of the main software that runs on the robot and controls it. The NAOqi Framework is the programming framework used to program NAO.

**Programming Environment**

The whole program is written in Python 2.7 with a Python Application Programming Interface (API) developed by Aldebaran robotics to communicate with the NAOqi framework.

I designed and trained the artificial neural networks with the Python library Keras; this is a high-level neural network API, written in Python, and is capable of running on top of either TensorFlow or Theano. I used the Keras backend with Theano to take advantage of the GPU.

The library contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers, and a lot of tools to make working with images and text data easier. From these, we selected and loaded only the required modules in order to have an efficient use of memory.

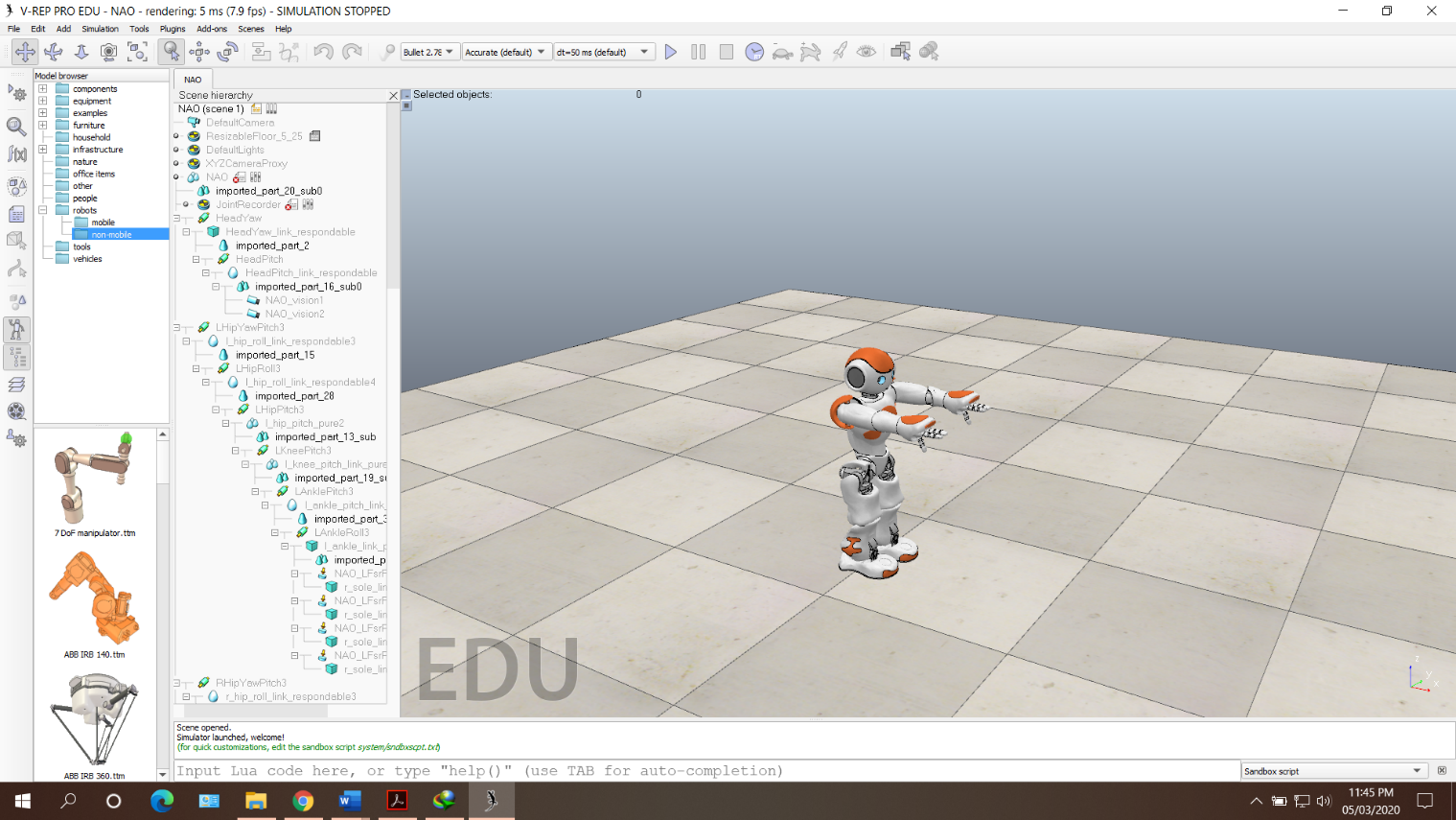
****

Fig 4.8: VREP environment with NAO robot

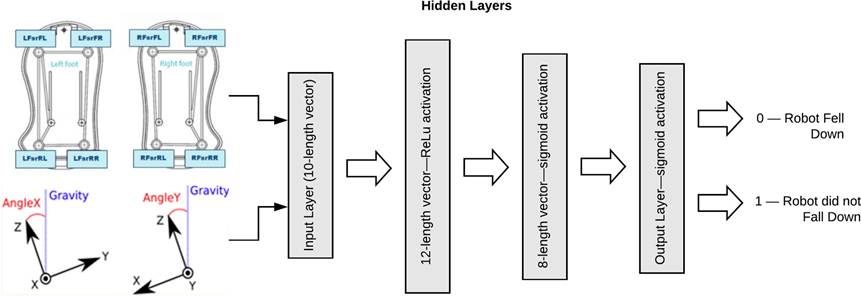
****

Fig 4.9: Fall Module Multi-Layer Perceptron (MLP)

**Fall Module**

A key element in the algorithm is the capacity to detect the falling of a robot with the fall module. This module is a pre-trained MLP that allows one to identify whether the robot has fallen or not.

In order to implement this module, I acquired data from the simulation and stored in a dataset of 790 values, and used that data in a previously labeled format to train a Multi-Layer Perceptron (MLP) with an input layer of ten neurons; from these ten neurons, eight belonged to the force sensors in the soles of the feet, and the other two belonged to the inertial unit of the NAO robot. This inertial unit told us the torso angle of the robot with respect to the ground; it was computed using the accelerometer and the gyrometer of the NAO.

The fall-module MLP has two hidden layers, one with twelve neurons with the ReLU activation function, and another with eight neurons with a sigmoid activation function. Lastly, there was an output layer with only one neuron, which outputs 0 if the robot fell down or 1 if it did not. This architecture is shown in Figure 4.9.

### 4.2.9 Algorithm

I initialized the NAO proxies, the motion proxy that allows to send signals to the NAO motors; the posture proxy that allows one to set the robot in a “home position”; the sonar proxy, which tells us the information of the ultrasonic sensors; and the memory proxy to access the data recorded, v. gr. the angle of the joints. I activated the robot by setting the stiffness of the body to 1 using the instruction motion.setStiffnesses (“Body”, 1.0).

I initialized the number of epochs to 50 and initialized gamma to 0.9, which means experience is considered.

I used a technique to let the algorithm converge via *experience replay.*  Before training I needed to define a variable for this purpose with *replay* as an empty list; however, as I was running the algorithm multiple times, it is not an empty list, but it was the list previously filled in a previous run, i.e., *replay = lastreplay.*

I opened a loop from *i* = 0 until *i* = epochs, set the robot in the “home position,” which was the NAO posture “StandInit,” then read sonars with the memory proxy and stored it in *sonar*. I then read the state.

Then, I opened a while loop with the condition that the NAO had not reached a terminal state.

I ran the Q-network forward and stored the result in *qval* (a vector). I wanted to explore more options in order to not fall in a local minimum, so with a probability of *epsilon*, I chose between the max action or a random action. This epsilon gradually decreased in such a way that after many iterations it became 0.

Then, I applied the actiont+1 and observed the *reward* and the *new state*. At this point, I had [*state*, *action*, *reward*, *new state*]. I stored this tuple in *replay* and repeat the process until the length of replay was the same as *buffer*; I set buffer as 60. Once the buffer was filled, I got a mini batch, which was a random sample of length 30 from the replay array.

Thereafter, I looped over each element of the mini batch. Each element was a list of four values that I used to set (*old state, action, reward2, new state2*), then ran forward the Q-network using as input the *old state* and stored the result in *old qval,* where the greatest value of *old qval* was selected.

At this point, I needed to check whether the *reward2* belonged to a terminal state, i.e., if the variable *update* was equal to the value of the *reward2*; if not, *update = (reward2 + (gamma × maxQ))*. This variable *update* was the rule used to compute the update of the neural network.

The value of the variable *update* needed to replace the value in the *Y* vector in the position of *index*. This will be the target for the Q-network, which should be done for all the elements in the minibatch. Once the loop was finished, I used the entire *Y* and *X* vector to train the Q-network using backpropagation.

The Q-network Algorithm is shown below:

Load fall module

Initialize NAO proxies

Define the size of the Q-network

Set NAO stiffness on

Initialize *epochs*

Set *gamma*

Set *batchSize*

Set *buffer*

*replay* ← empty list

**for** *i* = 1, epochs:

**read** state *st*

**while** not terminal state:

Run Q-network forward and store result in *qval*

With probability ε, select random action *a*

Otherwise, select *at* ← max (*qval*)

Execute action *a*

Read new state *st+1*

Read reward *r*

*replay* ← (*st*, *a*, *st+1*, *r*)

**if** *replay* length is equal to *buffer* length:

Set *minibatch* as a random sample of *replay* of length *batchSize*

**for** *memory*, *minibatch*:

Extract from *memory* (*old\_state*, *action*, *reward2*, *new\_state2*)

Run Q-network forward. Store result in *newQ*

*maxQ ←* max(*newQ*)

**if** *reward2* is not -10 or 10:

update ← *reward2* + *gamma*·*maxQ*

**else**:

update ← *reward2*

**end if**

*Y*[*action*] ← *update*

*X\_train* ← *old\_state*

*Y\_train* ← *Y*

**end for**

Use *X\_train* and *Y\_train* to train the model

**end if**

**end while**

**end for**

### 4.2.10 Results and Conclusions

After training the whole system, the robot was able to walk while displaying a stable behavior, which was to walk with balance. In addition, the robot did walk in straight line, i.e., it did not deviate. On the other hand, the algorithm provided for the robot allowed me to find: (i) a collection of twelve poses based on a ZMP criterion, and (ii) a combination of these poses in different action times that allow the robot to reach the goal distance without falling down and without deviating.

### 4.2.11 Further Work and Improvements

On one hand, different stages of walking can be explored, that is, walking on an irregular floor, walking on ramps, or omnidirectional walking, i.e., to walk in any direction, not only in a straight line. On the other hand, I can keep adding activities, i.e., modules of level two of the learning framework. Until now, only one activity, i.e., walking, has been proposed and tested but the idea is to learn more activities. Of course, more poses are needed to achieve such new activities. For instance, I can add “sit down,” “turn around,” or “take an object.” Lastly, to transfer the simulated algorithm to physical robots is left as future work.

# CHAPTER FIVE

# REPAIR AND SERVICING OF A HARMAR AL600 LIFT

## 5.1 The Harmar AL600 Lift

The Harmar AL600 is a hybrid lift platform developed by Harmar Mobility. This AL600 power chair lift features a 350 lbs. lifting capacity, 28.5 x 38-50 inch wheelchair platform, remote hand control and manual crank backup. The AL600 is compatible with most enclosed vehicles, including SUVs, and no drilling is required on most applications as this lift connects to existing third row seat handle.

### 5.1.1 Components of the AL600 Lift

The AL600 consists mainly of three sections. The tower section, base section and the platform which is where the weight is loaded.

The base section consists of:

* A 12V DC motor that drives a gear assembly which moves a chain link which in turn moves the platform and tower section forward or backward as decided by the operator.
* The rear and front limiting switches which are used to control the movement of the assembly.

The tower section consists of:

* A 12V DC motor that moves the platform (supported by a strap) up or down as needed by the operator.
* An up-limit switch which is also used to control the assembly.

The platform is where the wheelchair or scooter is loaded onto and strapped in.

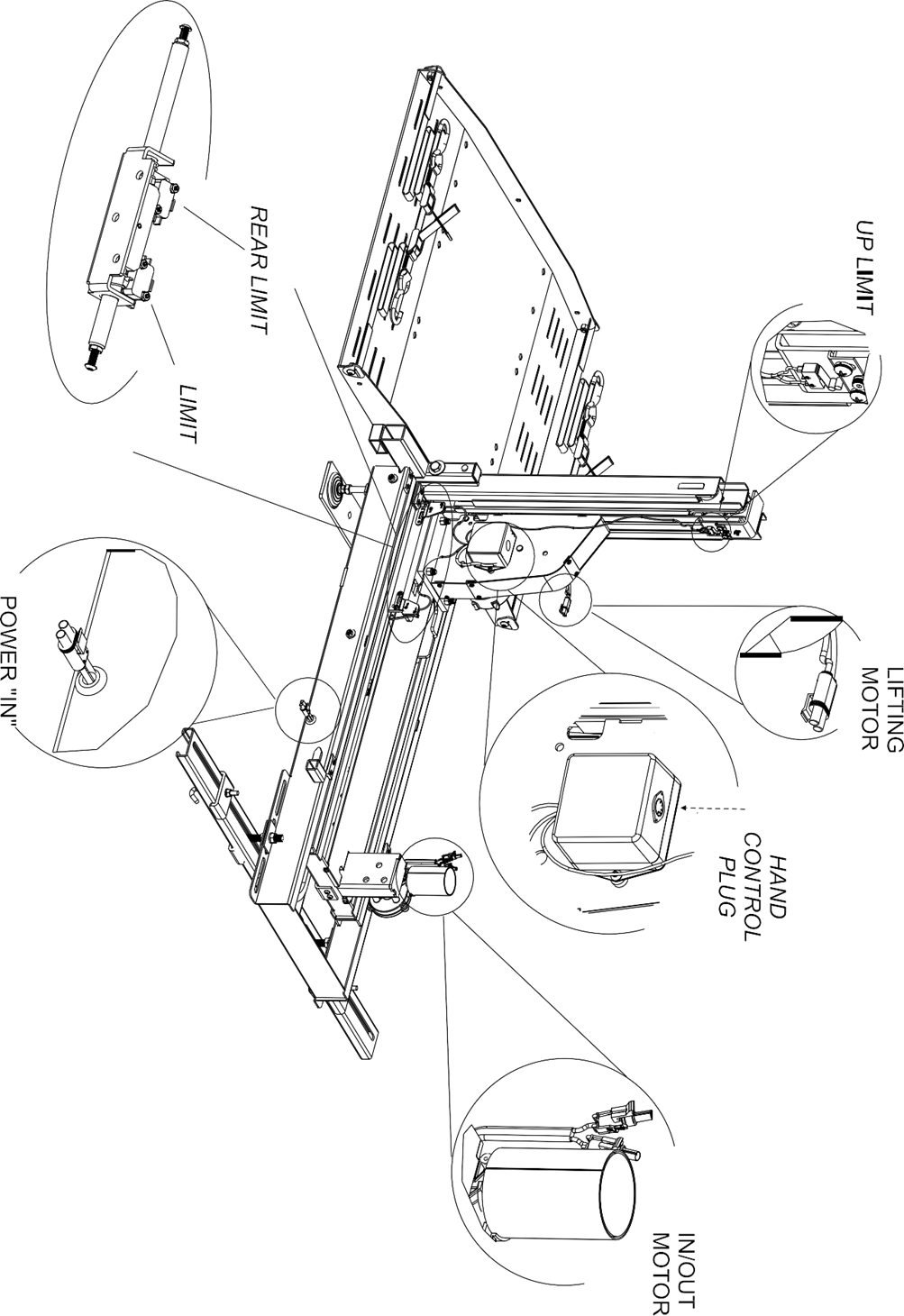


Fig 5.1: Major parts of the AL600 lift

## 5.2 The Problem

The lift was brought in disassembled and not in working condition. The owner suspected that one of the motors wasn’t in good condition.

## 5.3 Troubleshooting and Fixing

To check the owner’s claims of one of the motors being faulty, tests were performed on the motors to confirm. We hooked up a 12V Deep Cycle battery to the motors and checked if they ran at full torque. Both ran successfully and the motors were cleared.

We obtained the remote and checked the internal circuitry. On testing, it was discovered that some of the wires had burnt inside due to the load it carried as the control scheme of the machine has issues. After replacing the wires and confirming that the remote was good, we moved on to the base section of the lift.

We discovered that rear limiting switch had been damaged and had to be replaced. A new one was obtained and affixed to it. After ensuring that all was right with the base section, we moved on to the tower section.

The tower section housed the control box which housed the relays used to control the lift. We checked that all was right with the control box and affixed it back.

Then we coupled the base and tower section together following the instructions and wiring diagram from the manufacturer’s manual. On coupling, the system worked but there were glitches. The buttons didn’t function as expected.

On troubleshooting, we interchanged the terminals of the lifting motor and this fixed the problem. Then the lift was ready to be installed into the car.

The client brought the car and the lift was installed which was the end of a successful repair of the AL600 lift.



Plate 5.1: Assembly and troubleshooting of the AL600 lift.



Plate 5.2: Installation and testing of the AL600 lift.

# CHAPTER SIX

# CONCLUSION AND RECOMMENDATION

After six months of actively participating in the SIWES program, it is quite apparent that the introduction of the program into the undergraduate academic curriculum is a necessary inculcation by the respective authorities and is highly worthy of commendation.

At Gilead Biomedical engineering, there was a very good platform for students on industrial attachment to sharpen their cognitive skills to fit perfectly into the business world. In the control systems section where I was attached, they believe that technical knowledge should be learnt in schools and industrial attachment is meant for learning what the students need to face life after school, which they work towards. Technical knowledge was not left out in the process also.

## 6.1 Skills Gained and Educational Goals

After reflecting on my six-month internship, I looked back on the requirements SIWES and I discovered that I have been able to meet up some expectations.

Some of which are:

1. I was able to apply the knowledge of science, technology, engineering and mathematics during my assignments of carrying out research applications of reinforcement learning to robotics.

2. I was able to read and understand research papers written by another researchers on topics related to engineering and technology.

3. I was able to understand professionalism and ethical responsibility during my attachment.

4. I learnt and was able to function on multi-disciplinary projects, working with different people from different field of science and engineering and I got commendations.

5. I got broad education necessary to understand the impact of engineering solutions and practices in a global and societal context.

6. I already recognized to invest in life-long learning but it was further made solid, both in terms of academics and life values.

7. I was able to collaborate with other researcher across the globe on providing an engineering solution industrial robotics scope.

8. I learnt technical skills used in the repair and maintenance of industrial machines and equipment through the Harmar AL600.

## 6.2 Recommendation

This program has not only given room for great work experience but has given more exposure to the challenges that one may face while working in the industry. It is becoming increasingly apparent that it is only through an idea like SIWES that tertiary institutions can produce employable graduates to meet the needs of a technologically dynamic society thus drastically reducing the rate of unemployment in Nigeria.

I would like to suggest that the Industrial Training Fund should liaise with companies to make arrangements for absorption of students into their field of specialization and if there are no openings available companies should make it known to student long before commencement of the training. This is due to fact that getting placement is most times very difficult and consumes valuable time.

I therefore suggest that this program should be continued for all undergraduates with the stipulated six months or more, so that student can get the adequate training required.

# REFERENCES

Ashraf, M. (2018, April 7). *Towards Data Science*. Retrieved March 4, 2020, from https://towardsdatascience.com/reinforcement-learning-demystified-36c39c11ec14

Heidenreich, H. (2018, December 4). *Towards Data Science*. Retrieved March 4, 2020, from https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f

Wikipedia. (2020). Retrieved March 4, 2020, from https://en.wikipedia.org/wiki/Machine\_learning