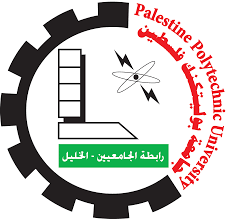
****

**Palestine Polytechnic University**

**College of IT and Computer Engineering**

**Department of Computer Engineering**

**Kinect based teleoperation of a humanoid robot**

**Supervisor**

Dr. Hashem Tamimi

**Team Members**

Mohannad Albustanji    181026

Eid Sonkrot                    217126

Huthaifa Rabae             237195

2023-2024

**Abstract**

Humanoid robots are sophisticated machines designed to mimic human appearance and behavior. Equipped with advanced technology like sensors, actuators, and artificial intelligence, they can interact with people and their surroundings in a manner akin to humans. An essential technology in this domain is motion detection, crucial for enabling these robots to perceive and respond to changes in their environment.

A prominent example of motion detection technology is the Kinect sensor, originally developed by Microsoft for the Xbox gaming console. This sensor combines cameras, infrared technology, and microphones to detect and track movement in real time. This ability to accurately sense motion has led to the Kinect sensor being utilized in various fields beyond gaming, including robotics, virtual reality, and healthcare.

In robotics, Kinect's motion detection capabilities are invaluable for developing more responsive and interactive humanoid robots. It helps these robots perceive human motion, enhancing their ability to mimic or respond to human actions. In virtual reality, the Kinect sensor enriches the user experience by allowing more natural and intuitive interaction with virtual environments. In healthcare, Kinect's motion tracking can aid in patient monitoring, physical therapy, and even surgical planning, offering a non-invasive way to analyze and track patient movement.

The widespread adoption of the Kinect sensor in these diverse applications underscores its effectiveness in motion detection. Its precision and real-time tracking capabilities make it a highly useful tool for research and industrial applications, contributing significantly to advancements in fields like robotics and healthcare.

Our project involves developing a system that captures motion data using a Kinect sensor, processes this data with a microcontroller, and then translates it into machine code compatible with a humanoid robot environment. This setup aims to enable real-time imitation between humans and a Nao robot, a humanoid robot with 25 degrees of freedom, allowing it to closely mimic slow human movements. The Nao Robot serves as a dynamic platform for exploring humanoid robotics and AI capabilities.

Contents

[1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7](#_Toc153453113)

[1.1 Motivation and importance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7](#_Toc153453114)

[1.2 Problem statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7](#_Toc153453115)

[1.3 Project Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8](#_Toc153453116)

[1.4 Aims and objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9](#_Toc153453117)

[1.5 Report outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10](#_Toc153453118)

[2 Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11](#_Toc153453119)

[2.1 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11](#_Toc153453120)

[2.2 Software Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11](#_Toc153453121)

[2.2.1 Programming Languages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11](#_Toc153453122)

[2.2.2 TensorFlow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12](#_Toc153453123)

[2.2.3 Webots . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12](#_Toc153453124)

[2.3 Algorithm and theoretical background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13](#_Toc153453125)

[2.3.1 machine learning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13](#_Toc153453126)

[2.3.2 Deep Reinforcement learning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13](#_Toc153453127)

[2.3.3 DDPG . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15](#_Toc153453128)

[2.3.4 Choosing DRL over Inverse Kinematics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17](#_Toc153453129)

[2.4 Literature Review . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17](#_Toc153453130)

[2.4.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17](#_Toc153453131)

[2.4.2 Studies . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18](#_Toc153453132)

[2.5 Hardware background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19](#_Toc153453133)

[2.5.1 Humanoid robot (Nao) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19](#_Toc153453134)

[2.5.2 Microsoft Kinect v2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21](#_Toc153453135)

[2.5.3 Microcontroller (raspberry pi 4 model B) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23](#_Toc153453136)

[2.6 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25](#_Toc153453137)

[3 System design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26](#_Toc153453138)

[3.1 Preface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26](#_Toc153453139)

[3.2 Requirements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26](#_Toc153453140)

[3.3 Hardware design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .27](#_Toc153453141)

[3.3.1 System overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27](#_Toc153453142)

[3.3.2 Block diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28](#_Toc153453143)

[3.3.3 System Pseudo Codes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28](#_Toc153453144)

[3.4 System software design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31](#_Toc153453145)

[3.4.1 Dataset Collection and Specifications . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31](#_Toc153453146)

[3.4.2 Pose Variations and Movements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31](#_Toc153453147)

[3.4.3 Data Collection Process . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32](#_Toc153453148)

[3.4.4 Training and Testing Data Split . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32](#_Toc153453149)

[3.4.5 Neural Network Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32](#_Toc153453150)

[3.4.6 Reward function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35](#_Toc153453151)

[3.4.7 Pseudocode for training process . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36](#_Toc153453152)

[3.4.8 Why we choose DRL over kinematics techniques . . . . . . . . . . . . . . . . . . . . . . . . . . .37](#_Toc153453153)

[3.4.9 Advantages of utilization RNNs in the project . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38](#_Toc153453154)

3.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38

[References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39](#_Toc153453155)

List of Figures

[Figure 2.1 Deep Reinforcement Learning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14](#_Toc135664507)

[Figure 2.2 Deep Neural network . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15](#_Toc135664508)

[Figure 2.3 Nao Component . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21](file:///D:\181026_Mohanad_PPU\9th%20Semester\Introduction%20to%20graduation%20project\Chapter123_last%20version.docx#_Toc135664509)

[Figure 2.4 Microsoft Kinect . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22](file:///D:\181026_Mohanad_PPU\9th%20Semester\Introduction%20to%20graduation%20project\Chapter123_last%20version.docx#_Toc135664510)

[Figure 2.5 Raspberry Pi 4 Components . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24](#_Toc135664511)

[Figure 2.6 Raspberry Pi 4 GPIO pinout . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25](#_Toc135664512)

[Figure 3.1 System Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27](#_Toc135664513)

[Figure 3.2 System Block Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28](#_Toc135664514)

[Figure 3.3 Actor and Critic Networks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34](#_Toc135664515)

List of Tables

[Table 1.1 Project Schedule . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10](#_Toc135664516)

# 

# Chapter 1: Introduction

## 1.1 Motivation and importance

The development of motion detection mechanisms enhances the development of new applications for human motion analysis and reproduction. Kinect sensor and humanoid robot are two different technologies with various trends and applications. We plan to create a new trend with this combination, developing an interface whose applications will have future implications.

## 1.2 Problem statement

Teleoperation of robots has become increasingly important in a wide range of industries and applications, from manufacturing and logistics to medicine and disaster response. The ability to remotely control robots to perform complex tasks can significantly improve efficiency and safety, particularly in hazardous or hard-to-reach environments.

However, accurately replicating human movements in real time can be a significant challenge for teleoperation systems. Human movements are complex and varied, and can be difficult to capture and reproduce in a way that is both accurate and responsive.

The goal of this project is to develop a teleoperation system that can accurately replicate complex human movements using a Kinect sensor and a Nao robot. The system will use machine learning techniques, including deep learning and reinforcement learning, to enable the robot to learn and replicate movements in real time.

Specifically, the system will use the Kinect sensor to capture 3D skeletal joint data in real time, which will be processed and used to control the movements of the Nao robot.

The machine learning algorithms will enable the robot to learn and replicate a wide range of movements, from simple gestures to more complex movements such as dance routines or surgical procedures.

The success of this project will enable the development of more efficient teleoperation systems for a variety of applications. The ability to accurately replicate complex human movements in real time has the potential to significantly improve efficiency and safety in industries ranging from manufacturing and logistics to medicine and disaster response, and could have a profound impact on how robots are used in the future.

## 1.3 Project Description

The proposed system is designed to capture human motion data using a Kinect sensor and reproduce the movements in real-time using NAO robots. The system consists of three main components: the Kinect sensor, the microcontroller, and the NAO robot.

The Kinect sensor is a device that uses depth sensing technology to capture detailed motion data from a human performer. The sensor emits infrared light and captures the reflections from the environment, creating a 3D model of the performer's body. The sensor can track the position and orientation of the performer's joints, providing information about their movements.

The motion data captured by the Kinect sensor is transmitted to a microcontroller using a wired USB 3 cable. The microcontroller processes the data and sends it to the NAO robot, which is equipped with software that can analyze and reproduce the movements in real-time. The NAO robot is a humanoid robot developed by Softbank Robotics, designed to mimic human movements and interact with humans in a natural way. The robot is equipped with sensors and cameras that allow it to perceive its environment and respond to external stimuli.

The system can be used to capture a wide range of human movements, from simple gestures to complex dance routines. The NAO robot can reproduce the movements in real-time, allowing for immediate feedback and analysis.

The system can be used in a variety of applications, such as in physical therapy, sports training, and entertainment. The system can also be used to analyze the movements of individuals with disabilities, providing valuable information for rehabilitation and therapy.

Overall, the proposed system offers a cost-effective and efficient solution for capturing and analyzing human motion data. The combination of the Kinect sensor and the NAO robot provides a flexible and versatile platform for real-time reproduction of the motion data, with potential applications in a wide range of industries. The machine learning algorithms used in the software allow for efficient analysis of the motion data, making the system an ideal tool for a variety of applications.

## 1.4 Aims and objectives

1. To deliver an affordable and efficient method for capturing and analyzing human motion data utilizing the Kinect sensor and NAO robot.
2. To achieve real-time replication of human motions with centimeter-level precision through machine learning and deep learning algorithms.
3. To foster the development of innovative applications in areas like physical therapy, sports training, and entertainment, focusing on human motion analysis and replication.
4. To bolster research in human-computer interaction by offering a dynamic and adaptable platform to explore interactions between humans and robots.
5. To advance assistive technologies for people with disabilities by real-time analysis and replication of their movements.
6. To enhance the application of robotics and AI in educational settings, providing practical learning opportunities for students and researchers in robotics and computer science.
7. To serve as an alternative to existing, more costly and labor-intensive technologies like traditional motion capture systems.

## 1.5 Report outline

Week 1 start from 1 Sep

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Task \ Week | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| QA plan |  |  |  |  |  |  |  |  |  |  |  |
| Project plan |  |  |  |  |  |  |  |  |  |  |  |
| Plan review |  |  |  |  |  |  |  |  |  |  |  |
| Project analysis |  |  |  |  |  |  |  |  |  |  |  |
| Requirements |  |  |  |  |  |  |  |  |  |  |  |
| Project design & implementation |  |  |  |  |  |  |  |  |  |  |  |

Table 1. Project Schedule

# Chapter 2: Background

## 2.1 Overview

This chapter serves as an introductory overview of the key components and theoretical underpinnings that form the basis of our project. We will begin by discussing the theoretical background, with a particular focus on the latest developments in deep learning, machine learning, and traditional robotics, and how these fields relate to teleoperation. Our project relies heavily on these fields, and a thorough understanding of their principles is necessary for its successful implementation. Specifically, we will explore the most up-to-date algorithms and techniques used in these fields, which are crucial for enabling our teleoperation system to operate with high accuracy and complexity.

This chapter aims to provide readers with a foundational understanding of the key concepts and methods used in our project, laying the groundwork for a more detailed exploration of our system in subsequent chapters.

## 2.2 Software Background

The software component of our project plays a crucial role in processing the data from the Kinect sensor and controlling the Nao robot's movements. This section provides an overview of the algorithms, programming languages and frameworks used in the development of our software.

### 2.2.1 Programming Languages

For our project, we have decided to use Python as the primary programming language. Python is a popular language for scientific computing, data analysis, and artificial intelligence due to its simplicity, readability, and vast library support.

### 2.2.2 TensorFlow

TensorFlow is an open-source machine learning framework developed by Google. It allows developers to create and train machine learning models using a wide range of tools and techniques. TensorFlow is particularly well-suited for deep learning applications, such as neural networks with many layers.

Some key features of TensorFlow include:

* + - 1. High performance: TensorFlow is optimized to take full advantage of modern hardware, including GPUs and TPUs.
      2. Flexibility: TensorFlow supports a wide range of model architectures, including convolutional neural networks, recurrent neural networks, and more.
      3. Large community: TensorFlow has a large and active community of developers, which means there are many resources available for learning and getting help.
      4. Easy deployment: TensorFlow models can be easily deployed to a wide range of platforms, including mobile devices, the web, and the cloud.

### 2.2.3 Webots

Webots is a robotics simulation software widely used in both academia and industry for robotics and machine learning applications. It offers a virtual environment where developers can design, program, and test their robot models before implementing them in real life.

Key features of Webots include:

* + - 1. Realistic Physics Simulation: Webots provides a realistic simulation of robot mechanics and environmental interactions, essential for accurate testing of control algorithms.
      2. Versatile Environment Customization: Users can design a variety of custom environments in Webots, allowing for extensive testing of robot performance under different conditions.
      3. Compatibility with Machine Learning Frameworks: Webots supports integration with popular machine learning platforms like TensorFlow, enabling the development and testing of advanced robot control algorithms.
      4. Open-Source Availability: As an open-source platform, Webots allows for code modifications and adaptations to fit specific project needs.

## 2.3 Algorithm and theoretical background

### 2.3.1 machine learning

Machine learning is a subfield of artificial intelligence that focuses on developing algorithms and models capable of learning from data and making predictions or decisions. It involves the design and development of mathematical and statistical models that can automatically improve and adapt through experience.

Machine learning techniques enable computers to learn patterns, extract insights, and make predictions or take actions without explicit programming. It finds applications in various domains, including image and speech recognition, natural language processing, recommender systems, and autonomous decision-making.

Machine learning algorithms utilize data to learn from examples, identify patterns, and generalize to new, unseen data, enabling automated data-driven solutions to complex problems.

### 2.3.2 Deep Reinforcement learning

Deep reinforcement learning (DRL) is a type of machine learning technique that enables agents to learn how to make decisions in complex environments by trial and error. It combines deep learning and reinforcement learning, two powerful techniques that have been successful in solving a variety of machine learning problems.

Reinforcement learning involves training an agent to make decisions based on feedback received from its environment. The agent learns to take actions that maximize a reward signal, which is a scalar value that indicates how well the agent is performing its task.

The goal of the agent is to learn a policy, which is a mapping from states to actions that maximizes the expected cumulative reward.

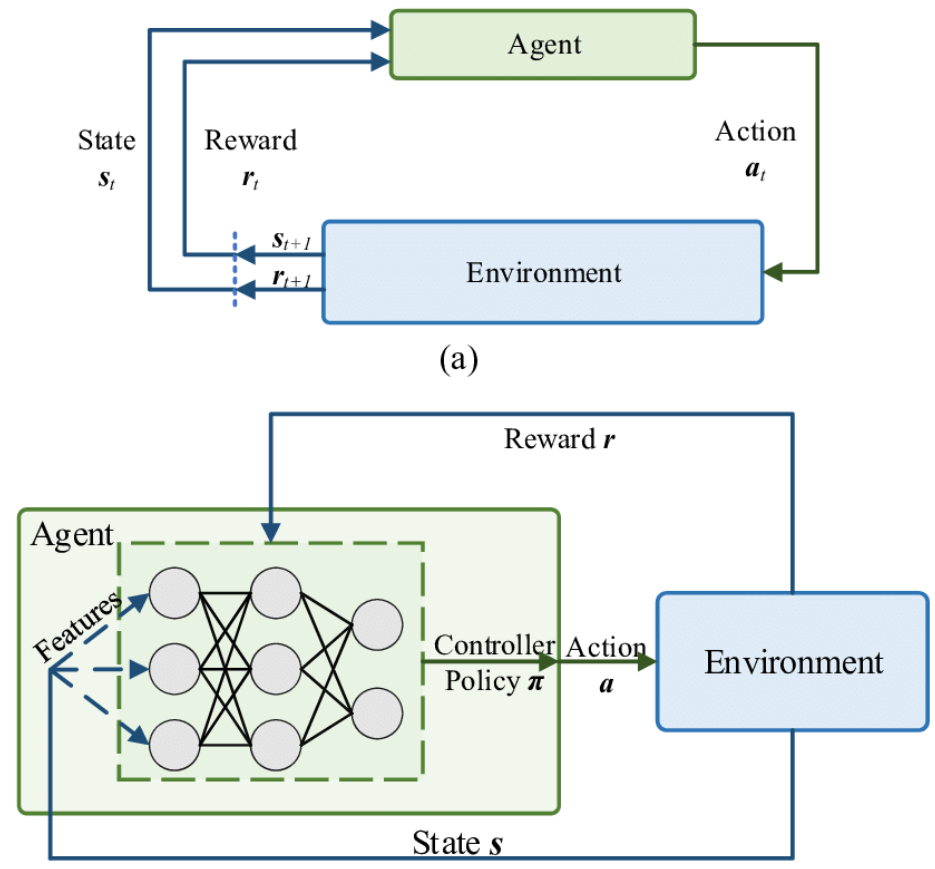


Figure 2. Deep Reinforcement Learning

DRL combines two techniques by using deep neural networks to approximate the value function or policy of the agent. The deep neural networks are used to represent the state-action value function, which is used to select the best action in each state, or the policy function, which is used to directly map states to actions. This enables agents to learn from high-dimensional inputs, such as images and sensor readings, and make decisions in complex and dynamic environments.

.

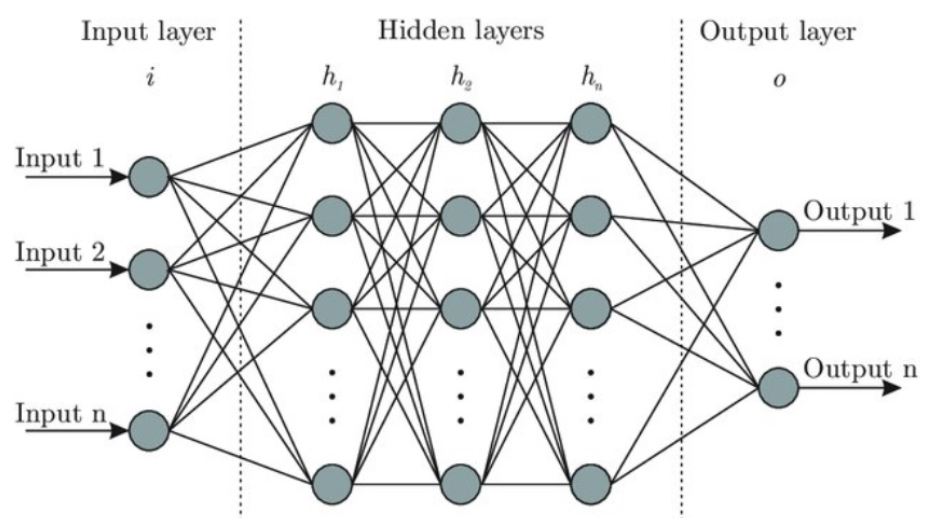


Figure 2. Deep Neural network

.

### 2.3.3 DDPG

Deep Deterministic Policy Gradient (DDPG) is a reinforcement learning algorithm that combines the ideas of Deep Learning and Policy Gradients with Q-Learning. The goal of the algorithm is to learn a deterministic policy, which maps states to actions directly, as opposed to a stochastic policy, which maps states to probability distributions over actions.

DDPG is based on the Actor-Critic method, which uses two neural networks: an Actor network and a Critic network. The Actor network is responsible for learning the policy, i.e., mapping the state to the action, while the Critic network is responsible for learning the Q-value of the state-action pairs. The Q-value represents the expected discounted future reward of taking a specific action from the current state.

DDPG has been shown to be effective in solving high-dimensional continuous control problems, such as robotic arm control, locomotion, and manipulation tasks.

*Algorithm DDPG:*

**while** (true){

**if** (ground contact)

{

if (s==none)

{

get state **s**

a = ∏ Ꝋ(s)

generate trajectory according to **a**

train critic by minimize **q**-(**r**+**Yq**)

train actor by maximize **q** using deterministic policy gradient

network parameter soft replacement

publish trajectory

**if** (fall down)

reset

**else**

continue

}

else

{

get **s**

calculate reward **r**

store transition [**s**, **a**, **s\_**, **r**]

s = s\_

}

else

{

stance control

publish trajectory

**if** (fall down)

reset

else

continue

}

}

### 2.3.4 Choosing DRL over Inverse Kinematics

We chose Deep Reinforcement Learning (DRL) over inverse kinematics techniques for our project due to its ability to handle complex and dynamic environments. Inverse kinematics techniques typically rely on solving mathematical equations to determine joint angles that can achieve a desired end-effector position. While these techniques work well for simple and static scenarios, they struggle with more complex movements and changing environments. DRL, on the other hand, allows an agent to learn optimal policies through interactions with the environment, without requiring explicit mathematical models or equations.

With DRL, I can train an artificial agent, such as a neural network, to mimic human-like movements by observing and imitating the data collected from the Kinect sensor. This approach offers greater flexibility and adaptability, as the agent can learn from diverse and dynamic situations, including variations in joint positions, body sizes, and movements. DRL also enables the agent to generalize its learned behaviors to new scenarios, making it suitable for real-world applications where the environment may be unpredictable or subject to changes.

Additionally, DRL allows for the exploration of more complex movements beyond what can be achieved through inverse kinematics. By combining reinforcement learning algorithms with deep neural networks, the agent can learn intricate patterns and coordination of joints, enabling it to perform not only predefined poses but also more nuanced movements, such as finger movements or fine-grained control.

## 2.4 Literature Review

### 2.4.1 Introduction

Human body tracking has become a critical aspect of many applications, including gaming, sports analysis, and medical rehabilitation. The use of robotics in such applications has also grown in recent years, with robots being used to assist in geriatric physiotherapy rehabilitation, imitating human actions for autism treatment, and lifting objects. The aim of this literature review is to explore studies that have utilized the Kinect sensor and the NAO robot to track human body movements.

### 2.4.2 Studies

The paper "Training NAO using Kinect" talks about how to control the movements of the NAO robot using Microsoft's Kinect sensor. However, in our project, we will be using deep reinforcement learning instead of Kinect inverse to control the robot's movements. Our goal is to mimic human movement with Nao robot and deep reinforcement learning, which is a more advanced technique than what was used in the paper. Nonetheless, this paper is still relevant to our project because it provides a foundation for controlling the movements of the robot and can help us build on existing knowledge and techniques. [1]

Our team is working on a project that involves using deep reinforcement learning to map human motion data captured by a Kinect sensor to the joints of a Nao robot. Our goal is to create an intuitive and user-friendly programming interface for teleoperating the robot. We came across "Dynamic-goal Deep Reinforcement Learning for Industrial Robot Telemanipulation" by Zhang et al., which also uses deep reinforcement learning to map human motion data to robot trajectories. Although their focus is on industrial robots and they use a custom hardware setup for teleoperation, we found their approach inspiring and relevant to our project. We are using a Nao robot and Kinect sensor, which are more accessible and affordable technologies. By leveraging deep reinforcement learning, we hope to create a more efficient and sample-efficient approach to teleoperating the Nao robot. Our project has potential applications in fields such as education, entertainment, and healthcare. [2]

## 2.5 Hardware background

### 2.5.1 Humanoid robot (Nao)

* NAO is a versatile humanoid robot created by SoftBank Robotics.
* NAO has become a benchmark in the world of research and education, as well as in healthcare, retail, and tourism.
* With his advanced sensors and interactive capabilities, NAO can adapt to any environment and interact with people in a natural way.
* He is fully programmable and customizable, making it possible to create application solutions that enable him to perform tasks in different areas based on all of his capabilities, including dialogue and motion.

#### 2.5.1.1 Nao hardware specifications:

1. Dimensions:
   * Height: 58 cm (22.8 inches)
   * Width: 27 cm (10.6 inches)
   * Depth: 22 cm (8.7 inches)
   * Weight: Approximately 4.3 kg (9.5 lbs)
2. Processing Unit:
   * Intel Atom Z530 Processor (1.6 GHz)
3. Memory:
   * 1 GB DDR2 RAM
4. Storage:
   * 2 GB Flash Memory
5. Sensors:
   * Two HD cameras (resolution: 1280x960 pixels)
   * Ultrasonic sensors for obstacle detection
   * Touch sensors on the head, hands, and feet
   * Inertial measurement unit (accelerometer and gyro-meter)
   * 4 Microphones for sound localization and voice recognition
6. Connectivity:
   * Wi-Fi (IEEE 802.11 b/g)
   * Ethernet port (10/100 Mbps)
   * USB 2.0 port
7. Power:
   * Battery: Lithium-ion battery
   * Battery capacity: 4800 mAh with approximately 60-90 minutes of continuous operation
   * Charging Time: Approximately 90 minutes
8. Operating System:
   * Aldebaran NAOqi OS (based on Linux)
9. Actuators:
   * 25 Degrees of Freedom (DOFs) for joint movement
   * Electric motors for actuation, including head, arms, hands, legs, and feet
10. Additional Features:
    * Text-to-speech synthesis
    * Voice and sound recognition
    * LED lights for visual feedback
    * Speaker for audio output

#### 2.5.1.2 Nao Components:

.

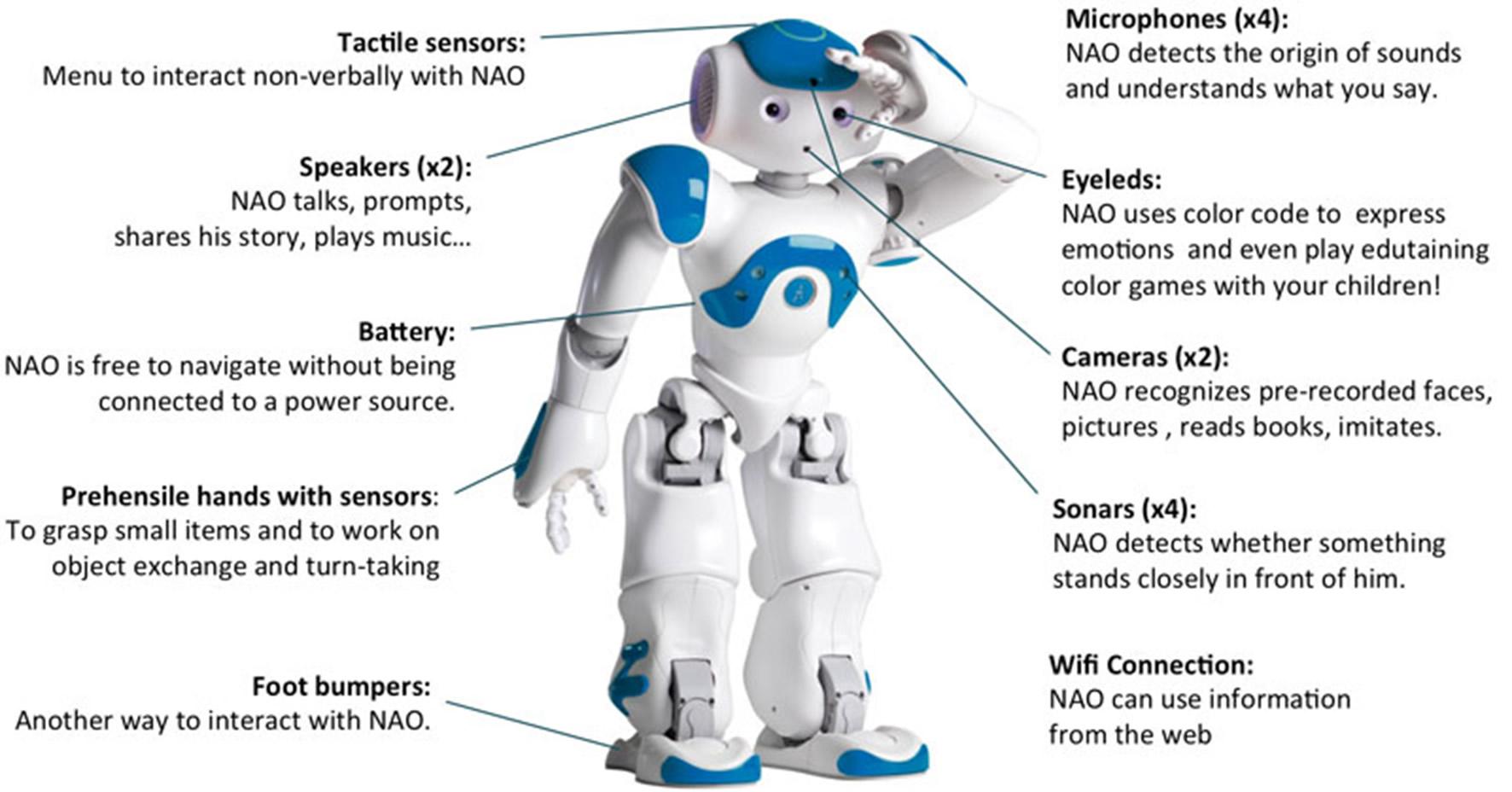


Figure 2. Nao Component

### 2.5.2 Microsoft Kinect v2

Kinect v2 is a depth-sensing camera developed by Microsoft, primarily used for capturing skeletal joint data and depth information. It offers an enhanced user experience in motion tracking, gesture recognition, and 3D depth sensing. The operation principle of Kinect v2 is ToF with modulation up to 130MHz, acquisition rate = 30Hz. In this section, we will explore the main components that Kinect work with it.

1. **Depth Sensing:** Kinect v2 utilizes an infrared depth sensor that captures the depth information of the surrounding environment. It can measure the distance between the sensor and objects in its field of view. It provides a depth resolution of up to 512x424 pixels, FOV (H=70.6o x V=60o), depth range (0.5-4.5) m, with ability to capture depth information with sub-millimeter precision.
2. **Color Camera:** In addition to depth sensing, Kinect v2 is equipped with a high-definition RGB color camera. It captures color images with a resolution of 1920x1080 pixels, FOV (H=84.1o x V=53.8o), enhancing the visual fidelity of the captured data.
3. I**nfrared Camera:** Kinect v2 includes an infrared (IR) camera that works in tandem with the depth sensor. It measures the distance between the camera and objects by projecting IR patterns and analyzing the distortion caused by their interaction with the environment.
4. **Microphone Array:** The device incorporates a built-in microphone array that enables voice recognition and audio processing. This feature facilitates natural language interaction and voice commands.

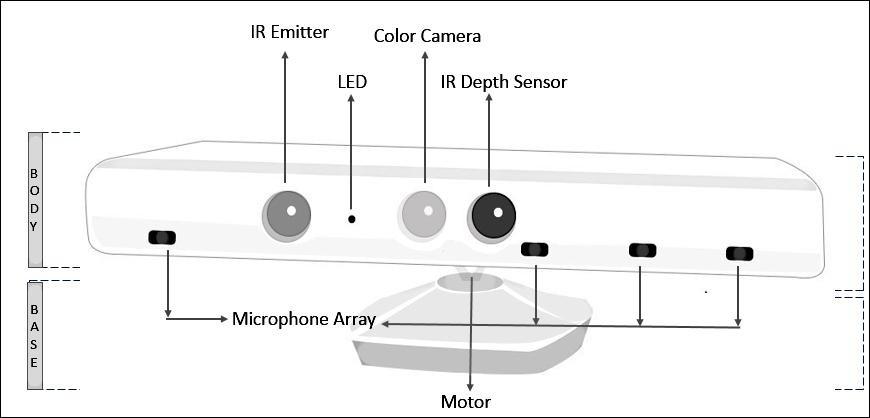


Figure 2. Microsoft Kinect

### 2.5.3 Microcontroller (raspberry pi 4 model B)

The Raspberry Pi 4 Model B is a versatile single-board computer that was released by the Raspberry Pi Foundation in June 2019. It is the fourth generation of the Raspberry Pi series, following the Raspberry Pi 3 Model B+. The Raspberry Pi 4 Model B offers significant improvements in terms of processing power, memory, connectivity, and multimedia capabilities compared to its predecessors.

1. Processor:
   * Broadcom BCM2711 quad-core Cortex-A72 (ARMv8) 64-bit SoC
   * Clock speed: 1.5 GHz
2. Memory:
   * Options: 4GB LPDDR4 SDRAM
3. Storage:
   * MicroSD card slot for the operating system and storage
4. GPU:
   * VideoCore VI
5. Connectivity:
   * 2 × USB 3.0 ports
   * 2 × USB 2.0 ports
   * Gigabit Ethernet port (RJ45)
   * 2.4 GHz and 5 GHz IEEE 802.11b/g/n/ac wireless LAN
   * Bluetooth 5.0
6. Display:
   * Two micro-HDMI ports
7. Audio
   * 3.5mm audio jack
   * HDMI audio output
8. Camera Interface:
   * MIPI CSI camera port for connecting a Raspberry Pi camera module
9. Display Interface:
   * MIPI DSI display port for connecting a Raspberry Pi touch display
10. GPIO:
    * 40-pin GPIO header (compatible with previous Raspberry Pi models)
11. Power:
    * USB-C power supply connector (5V DC)
12. Operating System:
    * Officially supported operating systems include various Linux distributions (Raspberry Pi OS, Ubuntu, etc.) and Windows 10 IoT Core.

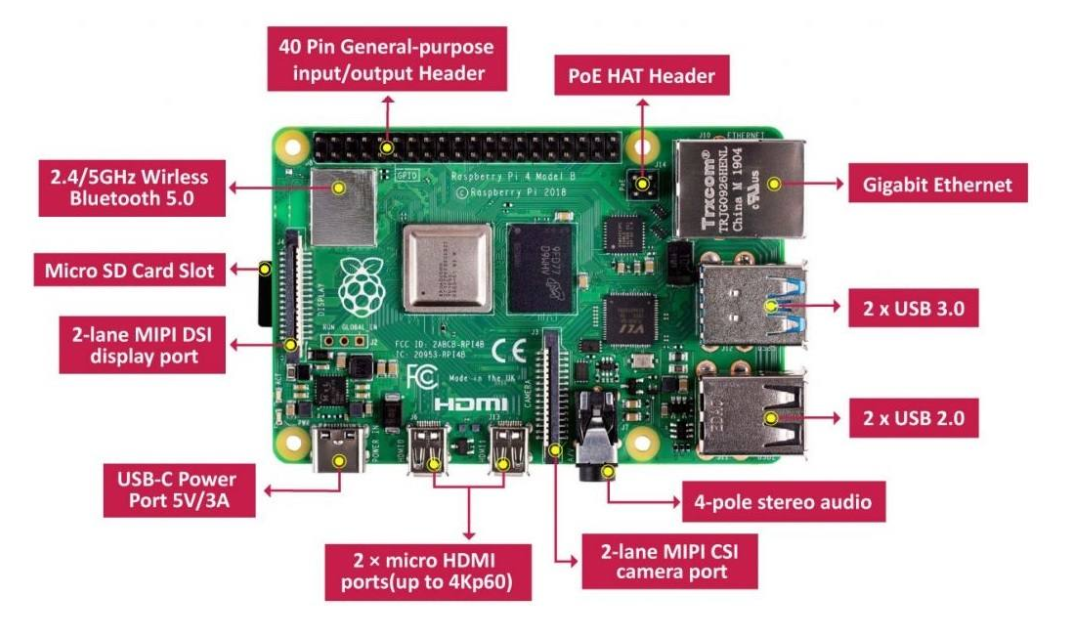


Figure 2. Raspberry Pi 4 Components

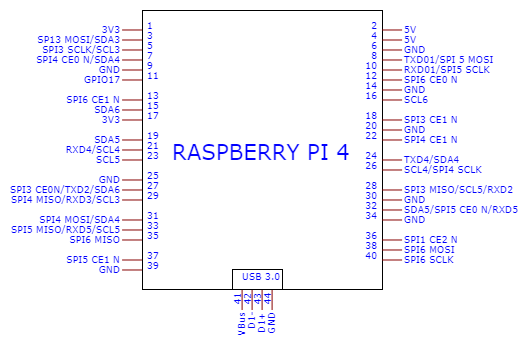


Figure 2. Raspberry Pi 4 GPIO pinout

## 2.6 Conclusion

The "Motion Capture and Analysis using Kinect and NAO Robot" project is designed to offer an affordable and effective method for capturing and analyzing human motion data. This project is set to enable the real-time replication of human movements with notable accuracy and fidelity. Additionally, it will play a significant role in the advancement of assistive technologies for people with disabilities by offering real-time analysis and reproduction of their movements. This project is also aimed at enhancing research in human-computer interaction by providing a versatile platform to explore interactions between humans and robots. It promises to encourage the integration of robotics and artificial intelligence in educational settings. Furthermore, this initiative is expected to serve as a cost-effective alternative to more traditional and expensive motion capture technologies. It will also pave the way for the development of new applications in various domains, including physical therapy, sports training, and entertainment, and will bring forth novel industry applications in these areas, as well as in the integration of realistic characters in video games.

# Chapter 3: System design

## 3.1 Preface

In this chapter, we will present the design of a system that uses a Nao 6 robot and Kinect v2 sensor to capture and mimic the movements of a human. The system is designed to enable the robot to learn and replicate the movements of a human, allowing it to perform tasks that require fine motor skills and dexterity. The chapter will cover the main features and components of the system, including the hardware and software components, the communication between the robot and the sensor, and the algorithms used for movement mapping and deep reinforcement learning. We will also discuss the challenges and limitations of the system, as well as future directions for improving its performance and expanding its capabilities.

## 3.2 Requirements

1. The system should enable human teleoperation of the Nao robot's motion using the Kinect sensor as an input device.
2. The system should track the user's movements and translate them into corresponding movements for the Nao robot within his possibilities and 25 degrees of freedom.
3. The system should perform complex movements and actions with accuracy in terms of centimeters, such as reaching for and picking up objects, navigating through a room, and interacting with the environment.
4. The system should incorporate closed-loop control, which entails utilizing sensors to provide feedback.
5. The system should be scalable and adaptable, allowing for future expansion and integration with other technologies and platforms.

## 3.3 Hardware design

The Hardwire design section provides an overview of the architecture and components of a robotic system designed to enable movement mimicking functionality.

### 

### 3.3.1 System overview

The robotic system is designed to mimic human movements using deep reinforcement learning. The system consists of various hardware and software components that enable the capturing and mapping of human movements to robot movements. The system architecture includes sensors to capture human movements, a controller to process the captured data, and actuators to control the robot's movements. The system uses deep reinforcement learning algorithms to improve the accuracy of movement mapping over time. The end goal of the system is to enable the robot to replicate human movements accurately in real-time.

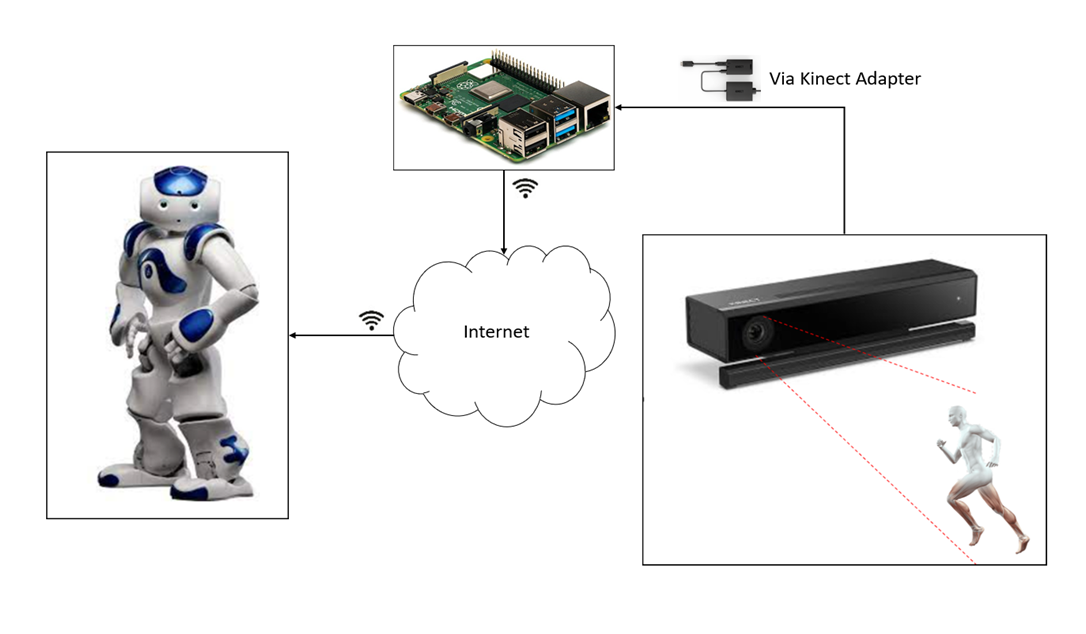


Figure 3. System Diagram

### 3.3.2 Block diagram

This figure shows the main parts of the system and how they interact.

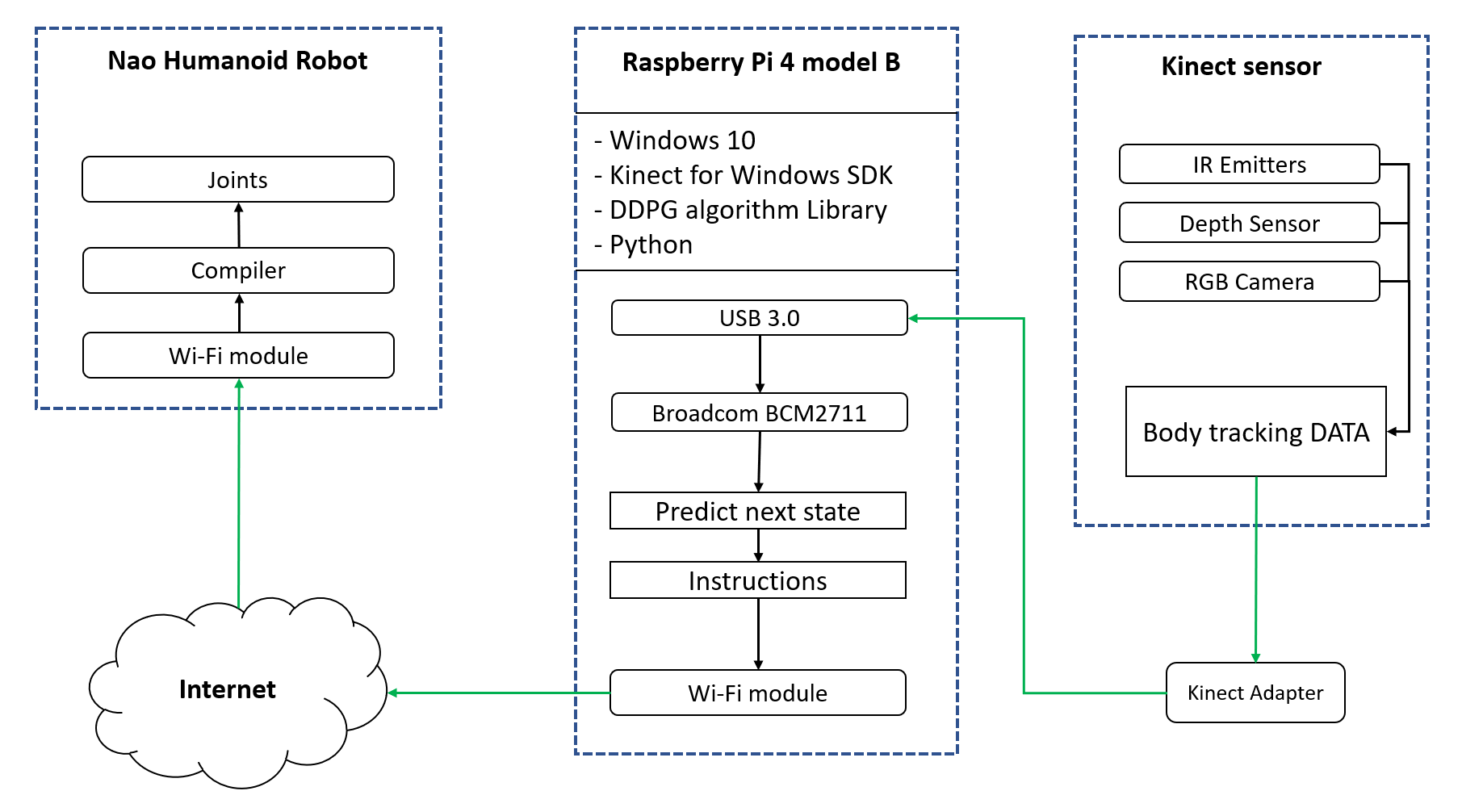


Figure 3. System Block Diagram

### 3.3.3 System Pseudo Codes

Algorithm 1 shown below runs on Microsoft Kinect V2. First, initialize Kinect, provide it a power supply, connect special adapter to it (USB B-type 🡪 USB 3.0). Second, from settings select a resolution: 512x424 px, frame rate: 30 fps and enable body tracking. Finally, inside loop, capture frames, read body tracking data, then send it to Raspberry Pi.

|  |  |
| --- | --- |
| Algorithm 1 Kinect sensor Pseudo code | |
| 1 | **Setup part:** |
| 2 | INITIALIZE Kinect |
| 3 | **CONNECT** Kinect to Kinect adapter |
| 4 | **SELECT** resolution, framerate |
| 5 | **ENABLE** Body Tracking |
| 6 | Loop part: |
| 7 | **WHILE** capturing frames: |
| 8 | **CAPTURE** frames (512x424@30fps) |
| 9 | **READ** Body tracking data |
| 10 | **MOVE** Body tracking data to Raspberry Pi |
| 11 | **ENDWHILE** |

Algorithm 2 runs on Nao humanoid robot. Initialize Nao and authenticate him to Raspberry Pi. In loop part, while raspberry pi sends Instructions; receive Instructions from Raspberry Pi, compile and regenerate executable Instructions for Nao. Finally Execute Instructions for Nao to change his state as like as possible to human state.

|  |  |
| --- | --- |
| Algorithm 2: Nao robot Pseudo code | |
| 1 | Setup part: |
| 2 | **INITIALIZE** Nao robot |
| 3 | **AUTHENTICATE** Raspberry Pi |
| 4 | Loop part: |
| 5 | **WHILE** raspberry pi send Instructions: |
| 6 | **RECV** Instructions from Raspberry Pi using Wi-Fi |
| 7 | COMPILE Instructions |
| 8 | **Execute** Instructions |
| 9 | **ENDWHILE** |

Algorithm 3 runs on Raspberry Pi 4 model B. At the beginning initialize Raspberry Pi, install Ubuntu, setup pyKinect2 library, setup python. Next step, add DDPG library to python and connect the Kinect adapter to USB 3.0 port on Raspberry Pi. Inside loop; receive body tracking data from Kinect via Kinect adapter, execute Deep Deterministic Policy Gradient (DDPG) algorithm to get prediction to next state, then convert the next state data set into python Instructions for Nao to change his state and send instructions to Nao using Wi-Fi.

|  |  |
| --- | --- |
| Algorithm 3: Raspberry Pi 4 model B Pseudo code | |
| 1 | Setup part: |
| 2 | **INITIALIZE** Raspberry pi |
| 3 | **SETUP** windows 10 |
| 4 | **SETUP** Kinect for Windows SDK |
| 5 | **SETUP** Python |
| 6 | **IMPORT** DDPG library |
| 7 | **CONNECT** Kinect adapter to USB 3.0 port in Raspberry pi |
| 8 | Loop part: |
| 9 | **WHILE** data != NULL: |
| 10 | **RECEIVE** Body Tracking DATA from Kinect via Kinect adapter |
| 11 | **EXECUTE** DDPG algorithm |
| 12 | next\_state = action predicted by DDPG |
| 13 | **CONVERT** next\_state 🡪 Instructions |
| 14 | **SEND** Instructions to Nao using Wi-Fi |
| 15 | **ENDWHILE** |

## 3.4 System software design

The system software design section provides an overview of the architecture and design principles underlying the development of our project. This section focuses on the high-level structure of the software system and how its various components interact with each other. By presenting the system software design upfront, we aim to provide readers with a clear understanding of the overall system organization and the rationale behind our design choices. This preface sets the stage for the subsequent discussions on training, execution, and other aspects of the project, enabling readers to navigate through the technical details with a solid grasp of the system's architecture and design principles.

### 3.4.1 Dataset Collection and Specifications

The dataset used for training the AI model consists of recordings captured by the Kinect sensor. The collected dataset comprises data from 127 volunteers, including both men and women. The volunteers have different heights, ranging from 1.45 to 1.91 meters, and varying body sizes.

### 3.4.2 Pose Variations and Movements

Each volunteer performed a series of eight predefined positions or poses, including:

1. Hands Up
2. Right Hand Up
3. Left Hand Up
4. Hands on Head
5. Arms Open
6. Stand Up Straight
7. Hands on Waist
8. Hands Forward

In addition to these predefined poses, the volunteers also performed a range of movements that fall under the same position category. This inclusion of movements allows for capturing more dynamic and natural pose variations.

### 3.4.3 Data Collection Process

For each volunteer, a total of 240 frames were recorded. Each frame consists of the 25 joint camera coordinates in X, Y, and Z dimensions.

This joint coordinate provides a comprehensive representation of the human body's pose and

movement.

### 3.4.4 Training and Testing Data Split

The recorded dataset consists of a total of 30,480 frames. To ensure an effective training process, the dataset was split into two subsets: a training dataset and a testing dataset.

Approximately 72% of the frames, amounting to 21,926 frames, were used as the training data. This larger portion of the dataset is dedicated to training the AI model to learn and imitate the desired poses accurately.

The remaining 28% of the frames, approximately 8,554 frames, were reserved for testing the trained model's performance. This testing dataset serves as an independent evaluation set to assess the AI model's ability to generalize and reproduce the poses accurately.

### 3.4.5 Neural Network Architecture

**Input Layer**

The input layer of the neural network receives the input data from the dataset, which consists of the recorded joint positions and movements of human subjects. Each input corresponds to a specific time step or frame in the sequence of movements.

**First Hidden Layer**

The input data from the input layer is then passed to the first hidden layer of the network. The first hidden layer consists of a set of neurons that perform computations on the input data. These computations involve weighted sums of the inputs and the application of the Tanh activation function.

**Second Hidden Layer**

The output from the first hidden layer is then fed into the second hidden layer. Similar to the first hidden layer, the second hidden layer performs computations on the inputs using weighted sums and the Tanh activation function.

The choice to incorporate two hidden layers in our project's neural network architecture was carefully considered to strike a balance between model complexity and the risk of overfitting. While adding more hidden layers can potentially increase the network's capacity to learn complex patterns, it also introduces a higher risk of overfitting, where the model becomes too specialized to the training data and performs poorly on unseen data. By limiting the number of hidden layers to two, we aim to mitigate the risk of overfitting while still allowing the network to capture and represent intricate relationships in the input data. This approach helps ensure a more balanced and generalizable model that can effectively learn and adapt to various movement patterns exhibited by the Nao robot.

**Output Layer**

The output from the second hidden layer is finally passed to the output layer of the neural network. The output layer consists of neurons that produce the final output of the network. In the case of the actor network, the output layer generates the policy that represents the desired movement or action. For the critic network, the output layer produces the estimated Q-value, which reflects the quality of the chosen action.

**The actor and critic networks**

The actor and critic networks are trained simultaneously, but they serve different purposes

**Actor Network**: The actor network is responsible for learning and generating the optimal probability for the desired movements. It takes the joint positions and movements as input and outputs the policy that determines the next action to be taken by the Nao robot.

**Critic Network**: The critic network evaluates the actions taken by the actor network and estimates the corresponding Q-value. The Q-value represents the expected cumulative reward associated with a particular action given the current state of the environment. The critic network helps guide the actor network by providing feedback on the quality of the chosen actions.

The neural networks, along with the DDPG algorithm, form an iterative process. During training, the networks interact with the environment (represented by the Nao robot) by receiving sensory input from the Kinect sensor and taking actions based on the learned policy. The resulting feedback and rewards from the environment are used to update the networks' weights and improve their performance over time.

By employing this flow of steps and the interaction between the actor and critic networks with the environment, we can train the neural network to learn and generate human-like movements based on the provided dataset and the reinforcement learning framework.

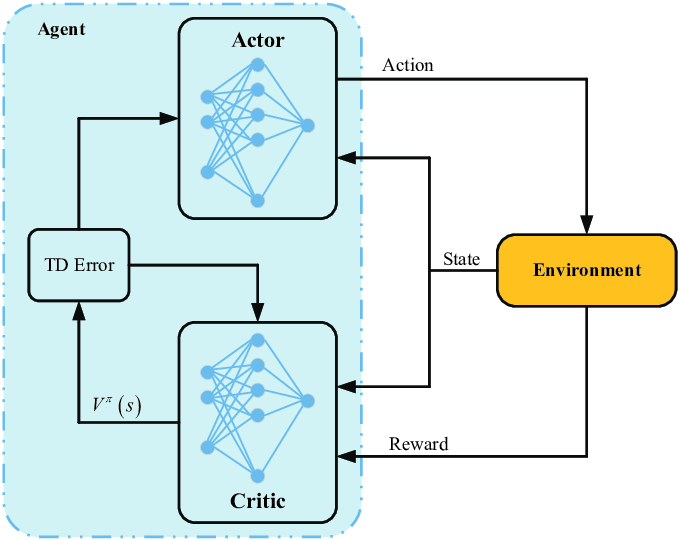


Figure 3. Actor and Critic Networks

### 3.4.6 Reward function

The reward function in our project plays a crucial role in shaping the learning process of the agent and guiding it towards the desired behavior. It quantifies the performance of the agent based on its actions and the current state of the environment. The goal is to design a reward function that encourages the agent to learn the desired movements and behaviors for the Nao robot.

In our project, we define the reward function to incorporate the concept of distance between the goal position and the current position of the Nao robot's joints in the virtual environment. We calculate the Euclidean distance between the goal position and the current position, and use it as a basis for assigning rewards.

The reward function can be defined as follows:

In this equation, alpha is a scaling factor that determines the influence of the distance on the reward. A higher alpha value results in a steeper reward decay as the distance increases, while a lower alpha value makes the reward decay more gradual. By using the exponential function, we ensure that the reward decreases exponentially as the distance between the goal and the current position increases.

The purpose of this reward function is to encourage the Nao robot to move closer to the goal position, as it will receive higher rewards for smaller distances. This incentivizes the agent to learn movements that bring it closer to the desired goal and helps in training it to mimic human-like movements.

### 3.4.7 Pseudocode for training process

|  |  |
| --- | --- |
| Training process Pseudocode | |
| 1 | Setup part: |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 | Loop part: |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 | reward = |
| 19 |  |
| 20 |  |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |
| 25 |  |
| 26 |  |

### 3.4.8 Why we choose DRL over kinematics techniques

We chose Deep Reinforcement Learning (DRL) over inverse kinematics techniques for our project due to its ability to handle complex and dynamic environments. Inverse kinematics techniques typically rely on solving mathematical equations to determine joint angles that can achieve a desired end-effector position. While these techniques work well for simple and static scenarios, they struggle with more complex movements and changing environments. DRL, on the other hand, allows an agent to learn optimal policies through interactions with the environment, without requiring explicit mathematical models or equations.

With DRL, I can train an artificial agent, such as a neural network, to mimic human-like movements by observing and imitating the data collected from the Kinect sensor. This approach offers greater flexibility and adaptability, as the agent can learn from diverse and dynamic situations, including variations in joint positions, body sizes, and movements. DRL also enables the agent to generalize its learned behaviors to new scenarios, making it suitable for real-world applications where the environment may be unpredictable or subject to changes.

Additionally, DRL allows for the exploration of more complex movements beyond what can be achieved through inverse kinematics. By combining reinforcement learning algorithms with deep neural networks, the agent can learn intricate patterns and coordination of joints, enabling it to perform not only predefined poses but also more nuanced movements, such as finger movements or fine-grained control.

Overall, DRL provides a more robust and versatile approach for teaching the Nao robot how to move, adapting to various conditions and expanding its capabilities beyond what traditional inverse kinematics techniques can offer.

### 3.4.9 Advantages of utilization RNNs in the project

The utilization of Recurrent Neural Networks (RNNs) in our project offers several advantages. RNNs are particularly well-suited for sequential data processing, making them an ideal choice for capturing the temporal dependencies in human motion. By incorporating RNNs into our neural network architecture, we can effectively model the time-varying nature of the joint angles and coordinates obtained from the Kinect sensor. This enables our system to learn and generate complex motion patterns, including smooth and natural movements. RNNs also possess the ability to retain and update internal states, allowing them to encode and remember past information, which is crucial for capturing context and long-term dependencies in motion sequences. Moreover, RNNs facilitate the training process by allowing backpropagation through time, enabling the network to learn from past time steps and make adjustments accordingly. Overall, the inclusion of RNNs enhances the expressive power of our system, enabling it to generate more realistic and dynamic motions.

Challenges

### 3.5 Summary

Chapter 3 presents a comprehensive design for a system that integrates a Nao 6 robot with a Kinect v2 sensor, aiming to replicate human movements with high precision. This intricate design encapsulates the interplay between sophisticated hardware components and advanced software algorithms, including deep reinforcement learning and recurrent neural networks. The system's core is its ability to accurately track and translate human movements into robotic actions, ensuring high fidelity in tasks requiring dexterity and fine motor skills. By detailing the requirements, hardware architecture, and software strategies, including dataset utilization, neural network configurations, and the rationale behind choosing deep learning over traditional kinematics, the chapter lays a solid foundation for a future where robots can seamlessly mimic and assist in human-centric tasks, demonstrating a significant leap in the field of robotics and human-robot interaction.

# Chapter 4: Implementation

### 4.1 Introduction

In this chapter, we delve into the practical aspects of implementing the system designed in Chapter 3. This includes setting up the hardware, configuring the software, and executing the algorithms necessary to enable the Nao 6 robot to mimic human movements using the Kinect v2 sensor. We will outline the steps taken, challenges encountered, and solutions applied during the implementation phase.

### 4.2 Hardware Setup

### 4.2.1 Assembling the Components

The assembly of the hardware components for the system implementation was conducted in a series of carefully planned steps, incorporating both Webots and TensorFlow platforms for enhanced simulation, training, and integration.

**Step 1: Connecting Kinect to Laptop for Simulation and Model Training**

**Initial Setup with Webots:** The first step involved setting up the Kinect v2 sensor with a laptop, incorporating the Webots robotics simulation software. This platform was crucial for creating a virtual environment to develop and test the model, ensuring a controlled and adaptable setting for initial trials.

**Software Installation and Configuration:** The Kinect for Windows SDK and relevant drivers were installed on the laptop. Simultaneously, TensorFlow, a powerful deep learning platform, was set up to work in tandem with Webots for simulation and data processing.

**Model Training and Simulation:** Utilizing TensorFlow within the Webots environment, the deep learning model was trained using data captured by the Kinect sensor. This integrated approach allowed for a robust simulation, enabling the testing of human movement recognition and interpretation in a virtual setting, which was essential for refining the model before actual hardware implementation.

**Step 2: Transitioning from Laptop to Raspberry Pi**

**Setting Up Raspberry Pi for TensorFlow and Webots:** The Raspberry Pi was prepared with necessary installations, including a Linux operating system and TensorFlow libraries. Special attention was given to ensure that the Raspberry Pi could support TensorFlow operations efficiently and communicate with the Webots simulation platform.

**Transferring and Optimizing the Model:** The trained TensorFlow model from the laptop was then transferred to the Raspberry Pi. This step required not only moving the model files but also optimizing them for compatibility and performance efficiency on the Raspberry Pi’s hardware.

**Step 3: Connecting Raspberry Pi with Nao Robot via Bluetooth**

**Bluetooth Integration for Wireless Control:** The Raspberry Pi was configured to establish a Bluetooth connection with the Nao robot. This wireless setup was essential for real-time control and data exchange, eliminating the limitations imposed by wired connections.

**Connectivity Testing and Troubleshooting:** Thorough testing was conducted to ensure that the Bluetooth connection was stable and reliable. This involved verifying the integrity of data transmission and the responsiveness of the Nao robot to instructions sent from the Raspberry Pi.

In summary, the assembly process involved an initial phase of simulation and model training using Webots and TensorFlow on a laptop, transitioning the system to the Raspberry Pi, and establishing a wireless connection with the Nao robot. This systematic approach, combining advanced simulation software with powerful deep learning tools, set the stage for a successful implementation of the system.

### 4.3 Software Configuration

The software configuration phase of the system implementation was crucial for ensuring that all software components worked seamlessly together. This phase involved installing and setting up various software tools, libraries, and frameworks on both the Raspberry Pi and the laptop used for simulation and training.

### 4.3.1 Installing Necessary Software

**Operating System Setup:** Both the Raspberry Pi and the laptop were configured with appropriate operating systems. The Raspberry Pi ran a Linux-based OS ( Ubuntu ) optimized for performance and compatibility with the hardware components.

**TensorFlow Installation:** TensorFlow, a key component for machine learning tasks, was installed on both the Raspberry Pi and the laptop. This installation was crucial for running the deep learning models and processing the data captured by the Kinect sensor.

**Webots Installation:** The Webots robotics simulation software was installed on the laptop. This platform was essential for creating a virtual testing environment, allowing for the development and refinement of the model without the need for constant physical testing.

**Kinect Software Setup:** The Kinect for Windows SDK and other necessary drivers were installed on the laptop to ensure proper communication and data capture from the Kinect v2 sensor.

### 4.3.2 Setting Up the Development Environment

**Development Tools and Libraries:**

**Simulation and Real-time Communication Testing:** The integrated development environment was tested in conjunction with the Webots simulation platform. This testing ensured that the code could communicate effectively with the virtual representation of the Nao robot in Webots.

4.2 Hardware setup

* 1. Raspberry Pi: Windows 10 IoT
  2. Kinect v2: install Kinect for Windows SDK, connect it to the adapter and connect the laptop with USB 3.0
  3. NAO: install Choregraphe, get NAO IP and confirm the connection

4.3 Software setup

* 1. Inverse kinematics (**equations** for Nao joints <theta> and Kinect Skeleton<sides o triangle> and **images**)
  2. Install webot and configure Nao
  3. Install python and c# with required libraries
  4. Choose dataset (Kinect v2)
  5. Implement DDPG using python
  6. Training
  7. Install Rabbit MQ to send data from Kinect to
  8. Read depth data from Kinect then send it to Nao robot

4.4 challenges

Balancing

Degree of freedom between Nao and Human

Training on PC not in Raspberry Pi

Chapter 5: Testing

* 1. Hardware
     1. Kinect v2
     2. Nao
     3. Raspberry pi
     4. System interconnection
  2. Software
     1. Inverse kinematics
     2. DDPG algorithm (Ask Eid how to test it)
     3. Webot
     4. Choregraphe
     5. C# and python libraries

Chapter 6: Conclusion and future work(one or two page maximum)

**Almost all subtitle should have image. We can add it later**

# References

1. Chartomatsidis, M., Androulakis, E., & Kavallieratou, E. (2016). Training NAO using Kinect.
2. Zhang, Y., Liu, Y., & Wang, J. (2019). Dynamic-goal Deep Reinforcement Learning for Industrial Robot Telemanipulation.
3. <http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring18/ujjawal/DDPG-Algorithm.pdf?fbclid=IwAR1YWz4Eji845dPpBwpaBc9RcyKUjOUHTL95H7Js33I0fLp8aH7otg6QrRY>
4. <https://keras.io/examples/rl/ddpg_pendulum/?fbclid=IwAR0FbjyRfz6Uw_oodZKQS3BMSz2_cjmHXJmMPgKz9NDP6x6aRoHC1CEBzK8>
5. <https://www.mathworks.com/help/reinforcement-learning/ug/ddpg-agents.html?fbclid=IwAR2Nwazkn_w6UNVsTQvhtDMYgFbaPmsetaGTUfqqMDr5_2fbD8xTYb7iaWk>
6. <https://roboticseabass.com/2020/08/15/introduction-to-deep-reinforcement-learning/?fbclid=IwAR0VHeBPJ6667VxP03ax9EujjhQJlhdNhAsXo9ic-Z07WMZBJzBl0fxB6fE>
7. <https://www.youtube.com/watch?v=GPjS0SBtHwY&t=1517s>
8. <https://www.youtube.com/watch?v=iAeis7j5LmE>
9. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9002889/?fbclid=IwAR0wfyCT3-8jToToxywHbGbDxrh27SUbEpvPo0PIB-_92-lXRz0OVoWIppA#:~:text=The%20Kinect%20v2%20depth%20sensor,4.5%20m%20range%20%5B13%5D>
10. <https://www.researchgate.net/profile/Md-Rasedul-Islam-2/publication/334848970/figure/fig1/AS:787192578322433@1564692744590/All-joints-in-NAO-robot-and-initial-position-50.png>
11. <https://www.researchgate.net/profile/Jean-Vanderdonckt/publication/333524942/figure/fig1/AS:773745962672129@1561486821911/The-25-joints-of-a-MS-Kinect-Skeleton.ppm>