ARCADA

Control of Robots

Sirius Hyleh

Degree Thesis

Mechanical and Sustainable Engineering

2023

Degree Thesis

Sirius Hyleh

Control of Robots.

Arcada University of Applied Sciences: Mechanical and Sustainable Engineering, 2023.

Identification number:

26017

Commissioned by:

Arcada University of Applied Sciences

Abstract:

The efficiency of modern manufacturing relies on industrial robots which are programmed to perform a number of different tasks that were previously done by humans. This thesis explores the possibilities of control of robots, focusing on the applications of six-axis robots in particular, as well as the safe implementation of an entry-level articulated robot to an educational institution. Niryo Ned2 robot was programmed to perform a pick-and-place task to grasp objects with both manual and computer vision-based control. By analyzing the programs, it was concluded that a lot of industrial applications demand the robot to be able to continuously adjust to the changes in environment, and conventional programming without the use of computer vision is often insufficient. ROS-based applications of commercially available Ned2 were then demonstrated and their capabilities and limitations discussed. The kinematics of the robot was then studied with both the Denavit-Hartenberg convention as well as by constructing a Jacobian matrix which was used to determine the relation between the end-effector velocities and joint velocities of the robot. The rules of the Denavit-Hartenberg convention were implemented successfully, and the actual position of the end-effector with the chosen joint angles were compared to the values of Ned2, which were very similar. The result was believed to be affected by the manual measurement of joint links and the oversimplified diagram.

Keywords:

Robotics, process automation, ROS, Python, Niryo Ned2, Denavit-Hartenberg convention

Contents

List of Symbols and Units

M_i	The input torque that comes from the motor (Nm)
r	The gear transmission ratio
μ	The gear efficiency (%)
{A}	The reference coordinate system A
${}^{A}P$	The reference coordinate system A in matrix form
{B}	The attached coordinate system B
${}^{A}_{B}R$	The rotated coordinate system in matrix form
J	The Jacobian determinant
$J(x_1, \dots, x_n)$	The Jacobian matrix for n-number of variables (x) and equations (y)
$\dot{X}$	The end-effector velocities of the robot
$\dot{\theta}$	The joint velocities of the robot
$\dot{x}$	The end-effector velocity in the x-direction (m/s)
$\dot{y}$	The end-effector velocity in the y-direction (m/s)
$\dot{z}$	The end-effector velocity in the z-direction (m/s)
ω_x	The end-effector angular velocity in the x-direction (rad/s)
ω_y	The end-effector angular velocity in the y-direction (rad/s)
ω_z	The end-effector angular velocity in the z-direction (rad/s)
a_n	The length of link n (m)
d_n	The link offset of link n (m)
θ_n	The joint angle of joint n (rad)
α_n	The link twist of link n (rad)
${}^{n-1}T_n$	The Denavit-Hartenberg homogeneous transformation matrix for joints n
R	The rotation matrix
T	The transformation matrix
${}^{A}\dot{\Omega}_B$	The angular acceleration relative to the frame {A} (rad/s^2)
${}^{B}\dot{V}_Q$	The linear acceleration relative to the frame {A} (m/s^2)
t	The instantaneous time (s)
Δt	The difference between two moments of time (s)
${}^{A}I$	The inertia tensor relative to the frame {A}
N	The moment (Nm)
${}^{C}I$	The center of mass (m)

ω	The angular velocity (rad/s)
$\dot{\omega}$	The angular acceleration (rad/s^2)
$\dot{v}_C$	The acceleration in the center of mass of an object (m/s^2)
F	The force (N)
m	The mass of an object (kg)

1 Introduction

This bachelor's thesis is focused on the Ned2 collaborative articulated robot and its practical applications. In this chapter, the problem of the thesis is defined, objectives are presented, and its relevancy to the bachelor's degree is discussed. The disposition section of this chapter introduces the main sections of the thesis.

1.1 Problem Definition

A wide variety of industrial tasks have been automated with the help of industrial robots for maximum precision, efficiency, and speed. However, it is often not a simple process to automate a task that was previously done by humans. To automate a complex task for a robot, one needs to go through a lot of trial and error to perfect the actions of the robot, as well as to be able to program the robot appropriately. Often, there are limitations, such as the accuracy of the robot, or problems with other capabilities of the robot's hardware and software, and the fact that the robot may not possess sapient-level thinking capabilities when it comes to judgement.

1.2 Objectives

The aim of this bachelor's thesis was to investigate the practical applications and capabilities of the Ned2 articulated robot, and the comparison of it to other entry-level articulated robots that are currently available. The thesis task was divided into four parts:

- Setting up the Ned2 robot arm with add-on components
- Controlling the robot arm and programming it to perform tasks
- Using matrix manipulations and the theory of kinematics to determine the positions and velocities of the end-effector of the robot

1.3 Compliance with Degree

The degree "Mechanical and Sustainable Engineering" is a bachelor's degree offered at Arcada University of Applied Sciences which focuses on mechanical engineering, sustainability, manufacturing processes, product design, digitalization, and functional materials; the degree is quite broad in scope and gives students knowledge of many different aspects of modern-day topics in engineering, which is relevant since engineers are expected to have a wide variety of skills in the contemporary labor market.

Robotics is a big part of modern manufacturing and is therefore very relevant and important for today's society. The degree does not include courses that are directly related to robotics, but many skills that are obtained with the tuition can be applied to robotics, such as mechanical design and programming. With the Fourth Industrial Revolution (Industry 4.0) being a recent topic in manufacturing, it is expected that the demand for engineers who have cross-disciplinary knowledge is going to increase.

1.4 Disposition

This subsection introduces the main six sections of the thesis, which are the following:

1. *Introduction*. This chapter introduces the thesis, its problem definition, objectives, and justifies its relevance to the themes of the academic degree.
2. *Literature Review*. This chapter discusses all the theory behind the motivations of the thesis and is used as supporting material for the method.
3. *Method*. This chapter aims to fulfill the objectives of the thesis by showing the reader what kind of methods were implemented.
4. *Results*. This chapter concentrates on the results that were obtained by the methods used for achieving the objectives of the thesis.
5. *Discussion*. This chapter discusses the results and their legibility.
6. *Conclusion*. This chapter concludes the thesis. It also reflects on the success of the thesis when it comes to obtaining the desired results defined by the objectives of the thesis.

2 Literature Review

2.1 History of Robotics and Computing

Nowadays, robots are part of everyday life and a functioning society for most people who live in the developed world. They, for example, manufacture goods and do dangerous jobs that are not suited for the well-being of humans, but also perform tasks that require extreme precision. In the early 20th century, the concept of robotics was still widely unknown. For example, the world of manufacturing looked completely different, and humans performed all tasks in a factory, no matter how risky the tasks were; there was no other way to get them done, after all.

The history of computing may sound very recent to many unaware of its extensive history beyond recent digital computers. There is archeological evidence of a mechanical computer dating back to the ancient Rome, to approximately 60 BCE. According to a Yale professor, Derek J. de Solla Price (1922-1983), this computer was used for calculating time, and it is approximated that it contained around forty gears to calculate lunar, solar, and stellar calendars. In the 19th century, Charles Babbage (1791-1871) invented the Analytical Engine, which could be considered the base of modern digital computers due to the similarity; it has a central processing unit, arithmetic processing unit, and memory storage. It was also possible to input programs and output results into the Analytical Engine. (Swedin and Ferro, 2022, pp.1, 14–18)

The word "robot" was first mentioned in a 1920s play "R.U.R.", written by Karel Čapek (1890-1938). The word stems from the Czech language and means "forced labor" or "serf". The play exhibited a pessimistic and literal image of robots; they worked in factories and were exploited by humanity to a degree so severe that they ended up destroying the world, essentially disobeying the destiny they were created for, rebelling against forced factory labor. (Hans Peter Moravec, 2018)

The behavior of the R.U.R. robots would imply that they possessed some sort of human-like sentience, expressing anger towards their human creators. Today, this is not too far-fetched of an idea since artificial intelligence is used in a very broad range of applications, even to exhibit and sense sapient-like emotions. The need for true artificial intelligence became predominant in the mid-1900s, specifically due to its usefulness in wartime applications, when technological advancement enabled the development of, for example, military radars, digital computers, and

automated control systems for bombs (Appin Knowledge Solutions, 2007, pp.3-4). Even to this day, militaries tend to be interested in the most recent state-of-the-art technology.

After the two world wars of the 1900s, the transistor was first developed in the Bell Telephone Laboratories in 1947 by the physicists John Bardeen (1908-1991) and Walter H. Brattain (1902-1987) (Fig. 1). Transistors work by turning the flow of power on or off, and they are also used as current amplifiers. The transistor was further developed by the project member William B. Shockley (1910-1989), who invented the junction transistor, which in turn became the standard for commercial transistors. Eventually, transistors ended up replacing vacuum tubes in a lot of applications due to their size and reliability, both of which are properties that vacuum tubes are not known for. The invention of the transistor enabled the development of modern digital electronics which humanity relies on today. (Swedin and Ferro, 2022, pp.54-56)

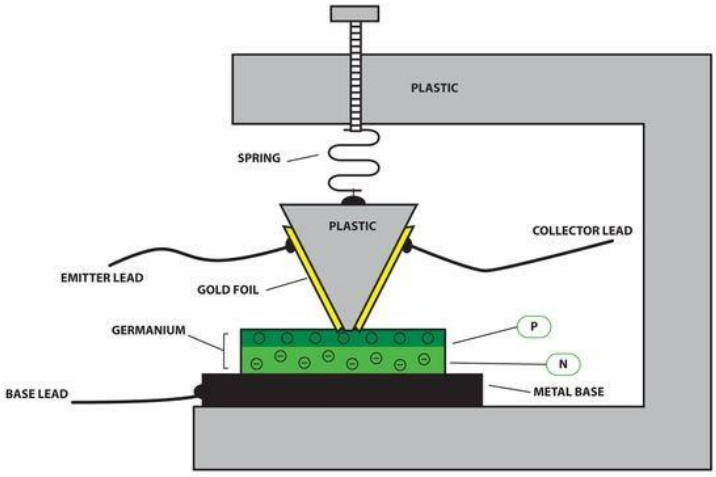

Figure 1. The elements of the first transistor (Computer History Museum, no date)

Around a decade after the invention of the transistor, the first commercial robot arm Unimate was invented by Joseph F. Engelberger (1925-2015), whose name carries the honorary title of "the Father of Robotics". Unimate was based on a mechanical arm that was patented in 1954 by inventor George Devol (1912-2011) (Fig. 2). Engelberger's focus was to make a robot which could perform tasks that can be considered dangerous for humans, inspired by the Three Laws of Robotics by Isaac Asimov. In 1959, the first Unimate #001 prototype was installed at a diecasting plant. Afterwards, approximately 450 other Unimate robot arms would work in

diecasting processes, replacing human beings in this harmful and often dangerous process of manufacturing. (A Tribute to Joseph Engelberger - Father of Robotics, no date)

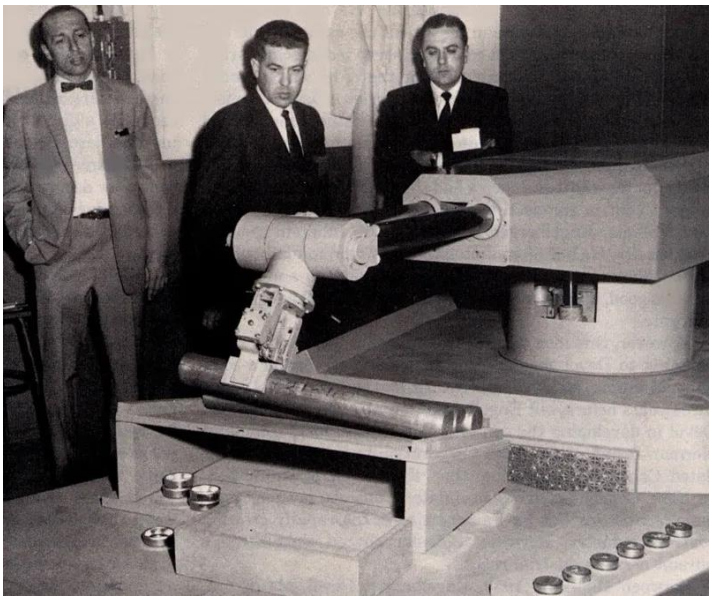

Figure 2. Unimate in action (Malone, no date)

After the invention of Unimate, the number of robots used in factories had increased rapidly with the help of advancing technology. In just one year, from 2018 to 2019, there was an increase of +12% in the number of industrial robots used in factories globally, resulting in total 2.7 million industrial robots in 2019 (IFR, 2020). This is no surprise as there exist factories which rely on robots nearly entirely and require minimal human labor in production.

Despite all the development within the field of robotics within the past 100 years, the development has mostly focused on industrial automation. For example, humanoid robots are not nearly as common as industrial robots due to the complexity of their design; it is, still, very difficult to emulate sapient tasks that are simple to us humans, such as walking over obstacles and jumping, and making it all look natural, effortless, and smooth.

2.2 Industry 4.0

According to Klaus Schwab (2018), the founder of the World Economic Forum, The Fourth Industrial Revolution, also known as Industry 4.0, consists of four different aspects that are predicted to unfold during the 21st century. These four aspects include different social, political, cultural, and economical changes. He believes that Industry 4.0 is powered by technological

innovation and will result in global societal transformation with the goal of improving the lives of all human beings.

In the heart of Industry 4.0 is the Industrial Internet. The Industrial Internet could help a company to be more aware of their assets and operations by utilizing technologies such as software, machine sensors, and cloud computing. This would enhance the company's efficiency. Essentially, sensors could become self-aware and provide the company with predictions or comparisons. Raw data could be harvested from sensors and other devices, and then analyzed. With the installation of such systems, a company could, for example, anticipate equipment failures well in advance. (Gilchrist, 2016, pp.1, 3–5)

2.3 Different Types and Classification of Robots

Since different types of robots have different purposes, there are general ways of classifying robots according to their design. In some cases, a robot could also be a hybrid; combining various aspects of different robot types. Four distinctive and common robot types are introduced with the help of examples: autonomous robots, articulated robots, cobots, and humanoid robots.

2.3.1 Autonomous Robots

An example of an autonomous robot is the Autonomous Spot robot (Fig. 3). The Autonomous Spot robot is based on Boston Dynamics' Spot mobility platform and runs on the NeBula (Networked Belief-aware Perpetual Autonomy) autonomy software architecture developed by NASA's Jet Propulsion Laboratory, which focuses on solving tasks in uncertain settings through computationally tractable methods, probabilities, and risk-assessments. It utilizes the factory sensors of the Boston Dynamics' Spot robot as well as its own sensors.

This robot was put through a DARPA challenge, which measures a robot's ability to autonomously go through extreme underground environments with difficult terrains, exploring and mapping them, as well as searching for artifacts within the environment. Two Autonomous Spot robots were deployed to the DARPA challenge environment, and they were able to find 16 artifacts, such as human survivors, gas-leaks, and cell phones, using their sensors and the NeBula architecture.

(Bouman et al., 2020)

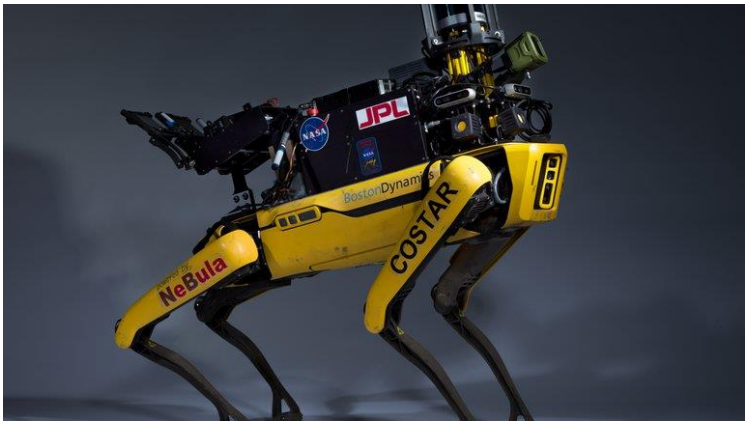

Figure 3. The Autonomous Spot robot (NASA and JPL-Caltech, no date)

2.3.2 Articulated Robots

Articulated robots are useful in applications where human arm mobility and capabilities are needed, such as factory work. One example of an articulated robot would be ABB's IRB 8700 six-axis heavy-duty industrial robot.

The IRB 8700 robot arm is intended for heavy-payload applications such as the automotive, foundry, mining, and metal fabrication industry (Fig. 4). It comes in two variants, IRB 8700-550/4.20 and IRB 8700-800/3.50, and the latter variant can handle payloads up to 800 kilograms with a reach of 3.50 meters. This variant of the robot weighs 4525 kilograms. The IRB 8700-550 variant has a longer reach, 4.20 meters, and is slightly heavier, 4575 kilograms. (ABB, 2022)

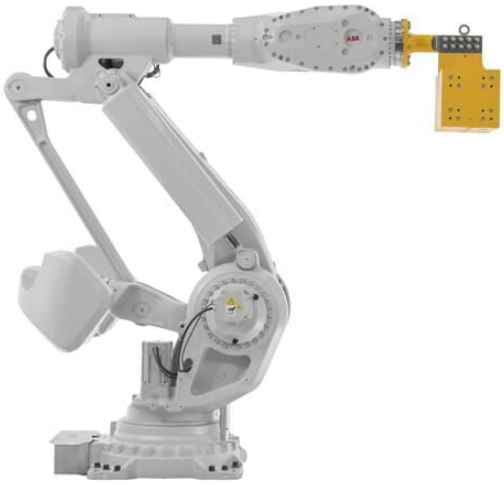

Figure 4. ABB IRB 8700 industrial robot (ABB, no date)

2.3.3 Cobots

Cobots, or collaborative robots, are unique robots needed in applications where humans and robots must work side by side without any safety fencing. For example, cobots are very useful in assembly lines where human dexterity and understanding is needed, but robots are also desired as close-proximity coworkers to perform tasks that are, for example, repetitive and straining for humans. Due to their working environment, cobots are equipped with specialized hardware and software to prevent injuring their human coworkers.

LBR iiwa is a lightweight collaborative robot developed by the German company KUKA (Fig. 5). It comes in two variants which have different load capacities and arm lengths. The LBR iiwa robot can collaborate with humans using its joint torque sensors, which sense human proximity and reduce the level of speed and force of the robot in case a human is nearby. This robot is also able to learn coordinate positions with human guidance. Mentioned applications for the LBR iiwa robot are, for example, assembly, handling, packaging, and measuring. (KUKA AG, 2022a)

Figure 5. KUKA LBR iiwa collaborative industrial robot (KUKA AG, 2022b)

2.3.4 Humanoid Robots

Humanoid robots are robots that closely resemble humans in terms of anatomy, intelligence, and movement.

One of the most well-known and advanced humanoid robots is Honda's ASIMO robot, first introduced in 2000, with the latest version having been released in 2011 (Fig. 6). ASIMO has a body with 57 servo motors, therefore having 57 degrees of freedom. ASIMO weighs only 48 kilograms due to its lightweight 130 centimeters tall body. It can run at 9 km/h, and it is also able to adapt to uneven surfaces while moving. ASIMO is also able to work in an environment with humans; it is able to avoid them by predicting where they're walking, using sensors. This means that the robot can plan its path again in case its path intersects with something unpredictable. ASIMO is equipped with tactile and force sensors in its hands, which enable it to perform tasks such as opening bottles and pouring the liquid inside the bottle into a glass and handling fragile objects without squishing them. It is also able to recognize faces and voices, even if there are multiple people speaking at the same time. Consequently, the robot is also capable of autonomous, continuous behavior without any human intervention. (Honda Motor Co., Ltd., no date)

Figure 6. Two ASIMO robots interacting with a human (Honda Motor Co., Ltd., no date)

2.4 Control of Robots

2.4.1 Control Systems

Control systems are needed in various machines such as robots for several different purposes.

One type of a control system is the open loop control system. An input signal is inserted into a control system, which transforms the input signal into a specific output signal (Fig. 7). The control system could, for example, amplify the input signal and emit the amplified signal. An open-loop control system works in a linear manner with no feedback; the flow of information goes only one way, and there is no communication coming back to the control system after it emits an output signal. Due to many reasons an error can occur which is known as a steady state error, which occurs in all control systems. (Bergren, 2003, pp.22–24)

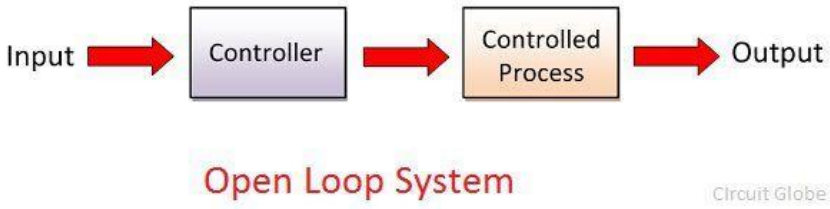

Figure 7. The open loop control system (Circuit Globe, 2022b)

Contrary to an open loop control system, a closed loop control system has feedback control (Fig. 8). Essentially, feedback control is established by comparing the output signal to the input signal, and if they do not match, the actuator will receive a nonzero signal which will indicate

the control system to correct the output of the actuator to match it with the desired output. In many cases, the output signal must be transformed for the comparison to be done; the signal may have to be scaled or converted into another signal type. (Bergren, 2003, pp.26–27)

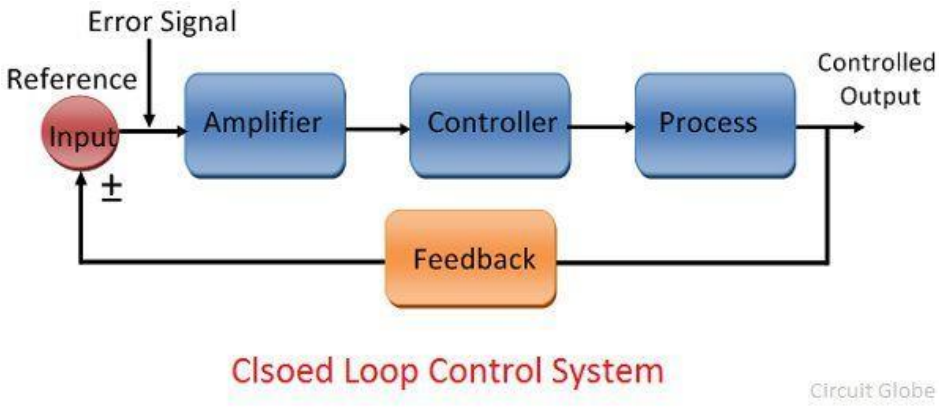

Figure 8. The closed loop control system (Circuit Globe, 2022a)

2.4.2 Programming Languages

Programming languages are essential when it comes to controlling a robot to determine its behavior and actions. According to Pierre Carbonnelle (2022), the Python programming language is the most popular programming language in the year 2022, with Java being the second most popular one.

It is important to understand the definition of machine language before understanding modern programming languages any further. Modern digital computers cannot directly understand programming languages such as Python, and therefore interpreters must be used to translate the raw code into machine language that the computer can read. How machine language looks to humans is very abstract compared to modern intuitive programming languages that are mostly comprehensible and logical. Machine language consists of different lengths and placements of numbers 0 and 1, essentially binary digits, and varies depending on the computer it is executed on (Hemmendinger, 2022a). To write a simple prompt in machine language, such as “Good morning!”, takes a lot of effort and knowledge compared to doing the same task in Python which typically takes a few seconds.

Python was developed in the late 1980’s and is free for everyone to use (Fig. 9). It is user-friendly and more intuitive to learn compared to other programming languages, and available for all common operating systems, such as Windows and Linux. Python is also an interpreted

language, meaning that Python programs are not compiled to machine code directly, but run through an interpreter, leading to the ease of testing, and debugging of Python programs. Using an interpreted programming language comes with a negative side effect, which is the fact that the written programs are not stand-alone and can be only run through an interpreter. (Kiusalaas, 2005, pp.1–2)

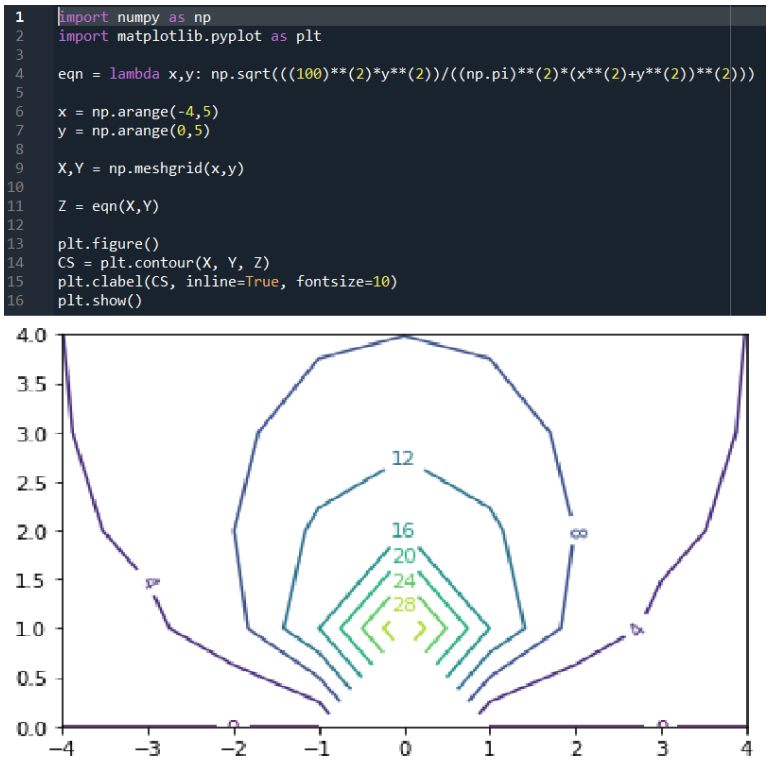

Figure 9. A Python program utilizing the Numpy and Matplotlib libraries to plot an equation

Java, just like Python, is a very popular programming language. Contrary to Python, it can be used on any computer if the computer has an interpreter for Java bytecode, which is the machine language for the virtual computer called Java Virtual Machine (JVM) to compile the written program into machine language (Fig. 10). For a computer to understand and run a Java program, it needs a Java Virtual Machine so that the Java bytecode can interpret the program and compile it into machine language that the computer can read. (Eck, 2022)

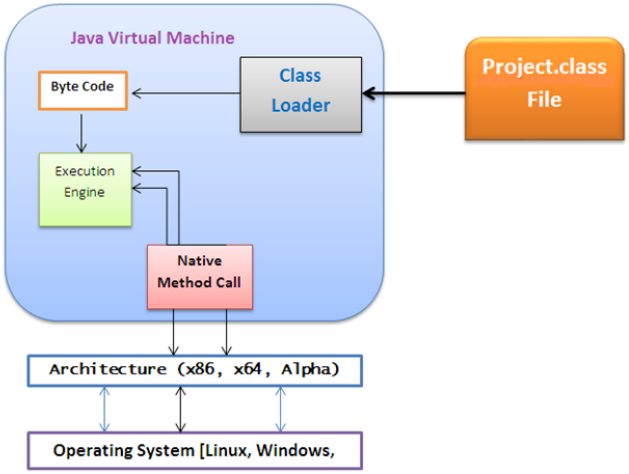

Figure 10. A JVM processes the raw Java code (Infosec Institute, 2014)

Both Python and Java are so-called object-oriented programming languages. Object-oriented programming languages have predefined modular units to ease the process of programming, making larger programs faster to execute and easier to maintain (Hemmendinger, 2022b). All object-oriented programming languages share the following basic components:

- Object: an instance of the class
- Class: creates a new type where objects are instances of a class
- Method: a function used by an object in a class
- Polymorphism: an object can be substituted as another object
- Inheritance: reusing the characteristics of a class and implementing a sub-type of it

(H., no date)

2.5 Hardware

2.5.1 Links and Joints

Just like humans, robots need links and joints to manipulate parts of their body. Consequently, there exist different types of joints in robots, connected by links, a lot like the ones found in the human body. Joints also have different degrees of freedom (of movement). The degree of freedom is expressed with the three-axis coordinate system is known as the 3D Cartesian coordinate system (Fig. 11), and the degree of freedom determines in which axes a joint can translate or rotate in. In a 3D Cartesian coordinate system, it is possible to form three linear axes and three rotational axes.

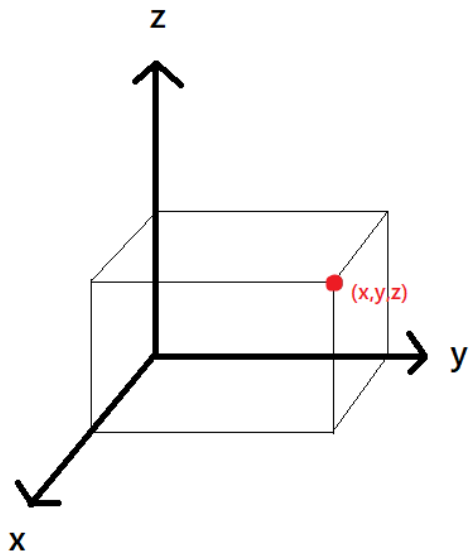

Figure 11. The 3D Cartesian coordinate system

The six types of joints for the mechanisms are the following:

1. Spherical: 3 degrees of freedom
2. Plane: 3 degrees of freedom
3. Cylindrical: 2 degrees of freedom
4. Screw: 1 degree of freedom
5. Revolute: 1 degree of freedom
6. Prismatic: 1 degree of freedom

(Denavit and Hartenberg, 1955)

Focusing on the joint design of 6-axis robot arms; 6-axis robot arms are very similar to human arms in terms of function and joint design, so much that one can easily relate a human arm to a six-axis robot arm by comparing them next to each other (Fig. 12). Essentially, the 1st and 3rd axes of the robot arm are equivalent to the human waist and the arm. The 4th to the 6th axes is equivalent to the human wrist all the way to the human fingertip. (Kawasaki Heavy Industries, Ltd., 2018)

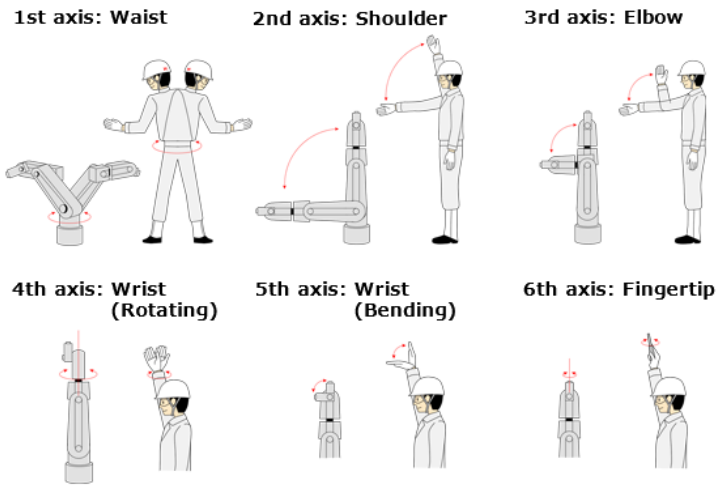

Figure 12. Comparison of human waist and arm mobility to an industrial 6-axis robot arm (Kawasaki Heavy Industries, Ltd., 2018)

2.5.2 Motors and Gears

Actuators such as motors are very common and a fundamental part of robots. Motors work to power the joints of the robot, producing movements such as rotation. It is important that high-precision motors are used for accuracy, for example servo motors. However, motors alone cannot produce the best power output from itself; reduction gears are used to increase the power output of a motor. Essentially, by choosing two gear wheels with a different number of gearteeth, and then reducing the rotational speed of the motor by a factor of 10, one can achieve a power output that is 10 times larger than the original. (Kawasaki Heavy Industries, Ltd., 2018)

The output torque M_o of a gear when connected to a motor can be calculated as:

$$M_o = M_i * r * \mu \quad (1)$$

Equation 1. Output torque (Engineering ToolBox, 2010)

Where

M_i = the input torque that comes from the motor (Nm)

r = the gear transmission ratio

μ = the gear efficiency (%)

There are two common types of conventional motors: alternating (AC) and direct current (DC) motors. Both have their advantages and disadvantages, and different applications depending on the task assigned for the motor.

AC motors are motors that utilize alternating current. An AC motor has two major components: the stator and the rotor. The coils of the motor are built outside the stator and rotor (Fig. 13). Since an AC motor utilizes alternating current, the motor responds to the alternating frequency of current that is fed to it. At one specific frequency, the speed of a AC motor is constant, however, the speed can be varied using frequency control. The speed also depends on the number of windings of the motor. (Bergren, 2003, pp.275-276)

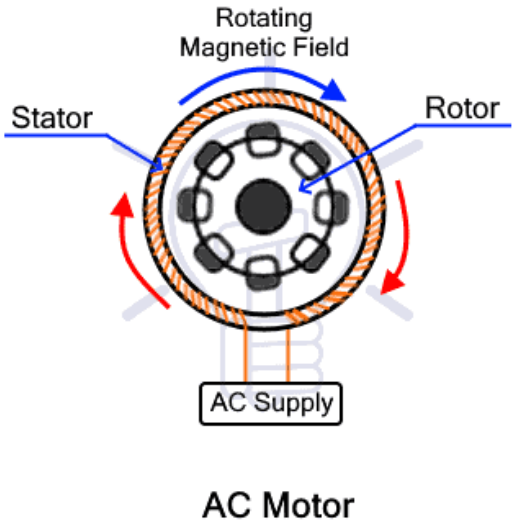

Figure 13. AC Motor (Electrical Technology, no date)

Since AC motors work at a fixed speed and cannot be controlled easily, they may not be the best for controlling a robot since robots often require rapid change of direction and speed in operation.

DC motors are, in a way, the opposite when it comes to their design; a DC motor relies on generated magnetic fields within its motor, and its permanent magnets reside within its stator, while its rotor has the coils (Fig. 14). Although a DC motor operates on a constant voltage, it still necessitates a change in the current direction for rotation. Essentially, the polarity of the DC voltage alternates in the coil due to a commutator, which remains stationary on the rotor bearing. Brushes make contact with the commutator, supplying power to the coils within the rotor and enabling the change in current direction. The problem with conventional DC motors is that the brushes wear out and therefore a DC motor needs constant maintenance, which is often not desirable. Therefore, brushless DC motors were invented to tackle this problem. The design of brushless DC motors is very similar to AC motors. The speed and torque of a DC motor can be controlled with changing the source voltage and current. (Bergren, 2003, pp.276-278)

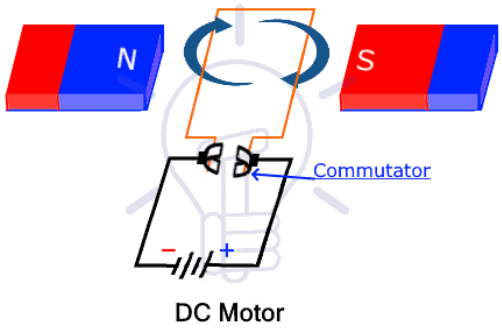

Figure 14. DC Motor (Electrical Technology, no date)

The third type of motors commonly found in robotics is servo motors. Servos are good for applications where rapid acceleration and deceleration is needed, and high levels of torque are also required. In servos, the inertia of the rotor has been minimized to get the highest value of torque possible for the design, and this is achieved by enabling only the conductors over the rotor to move, while the magnet remains stationary. Compared to conventional motors, such as AC or DC motors, servos are a lot better for precision and feedback control. A servo motor has a potentiometer which monitors the shaft's angular position, comparing it to the input signal; if these differ, a position error signal is amplified, and the motor will be rotated to its desired position where the position error equals zero. Conventional servos typically have a power output of 2-3 kW. (Hughes, 2006, pp.159–162)

2.5.3 Sensors

Sensors are very much a necessity for engineering systems such as robots; they provide the system with information about the surrounding world. Sensors can provide information to the system about the shape of the room it is in, or the room temperature. One could say that sensors for robots could be compared to the parts of the human sensory system.

One common type of sensors are proximity sensors which help robots sense their environment in terms of the proximity of an object. Proximity sensors are especially useful in collision-detection and prevention. Hall-effect sensors are commonly used in this application, and they work with a very fundamental principle of physics. This type of sensor can detect changes in magnetic fields, which in turn can be used to detect objects. The closer an object comes to a sensor, the stronger the detected magnetic field is (Fig. 15). In addition to this, the Hall-effect sensor is also able to detect the direction of change of a magnetic field. A hall-effect sensor utilizes a constant base voltage which changes according to the proximity of a magnetic field. (Appin Knowledge Solutions, 2007, p.100)

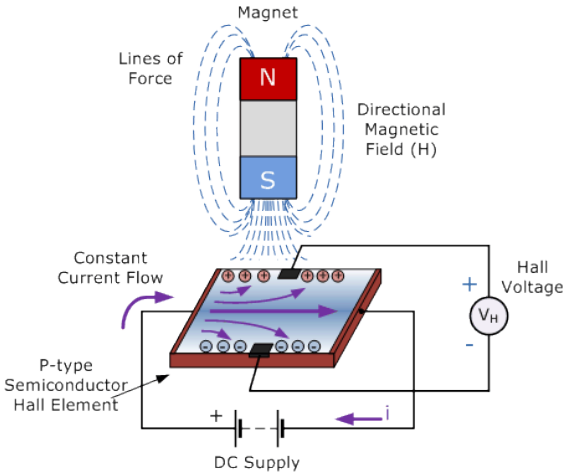

Figure 15. A Hall-effect sensor (AspenCore, no date)

Another type of sensor is an encoder, commonly found in actuators and motors in general. Incremental optical encoders are used as feedback sensors. They work in a way that as the motor rotates a single shaft revolution, a certain number of pulses (square or sine waves) are produced by a light generator (Fig. 16). Between the light generator and a photodetector is a disk with transparent slits that are placed periodically. As the disk rotates, the photodetector reads the pulse as the light passes through one of the slits. In this manner, an incremental optical

encoder can measure the rotational velocity of the motor and its relative position. (Appin Knowledge Solutions, 2007, pp.244-245)

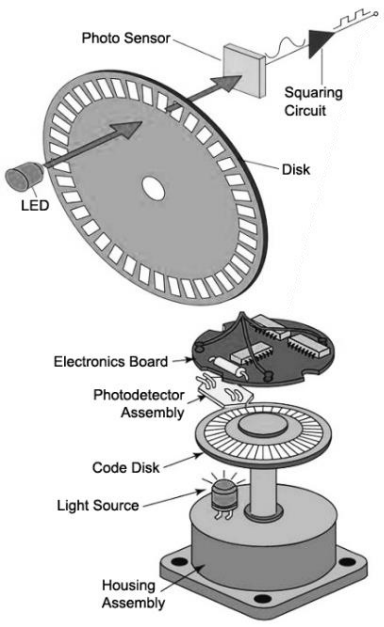

Figure 16. An incremental optical encoder (Anaheim Automation, 2021)

2.6 Mechanics, Kinematics, and Dynamics of Articulated Robots

Robotics revolves around the interpretation of objects in three-dimensional space. These objects include the links of the manipulator (robot arm), and its parts and tools, as well as the objects in its surrounding environment, for example a box. The manipulator is a set of rigid bodies connected by a continuous chain with the use of joints. The joints are connected by links. To be able to interpret the movement of the manipulator, it is needed to attach coordinate systems to these objects. As the manipulator moves, the coordinate systems go through transformations or rotations, which can be represented mathematically. (Craig, 2018, pp.1, 4, 67)

For the robot to be able locate objects in the 3D-world, the 3-axis cartesian coordinate system (Fig. 11) can be presented as a vector in its matrix form as follows, with $\{B\}$ being the coordinate system:

$$^{B}P = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \quad (2)$$

Equation 2. The position vector (Craig, 2018, p.22)

Additionally, it is also relevant to be able to describe how the object is oriented in the 3D-world, which is possible by attaching a coordinate system that keeps track of rotation of the object. This rotation-focused coordinate system will be given a description that is relative to the reference system. We can denote this attached coordinate system as $\{B\}$, and the reference system as $\{A\}$. The attached coordinate system can be described by three unit vectors written in terms of the reference system $\{A\}$: $^{A}\hat{X}_{B}$, $^{A}\hat{Y}_{B}$, $^{A}\hat{Z}_{B}$. (Craig, 2018, p.23)

One can use the three unit vectors to construct a rotation matrix which describes the relation between $\{B\}$ and $\{A\}$:

$$^{A}_{B}R = \begin{bmatrix} ^{A}\hat{X}_{B} & ^{A}\hat{Y}_{B} & ^{A}\hat{Z}_{B} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (3)$$

Equation 3. The rotation matrix (Craig, 2018, p.23)

The rotations about the x, y, and z axes in the form of planar rotations are written as follows:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(4)

Equation 4. Rotations about all three axes in the form of planar rotations (Craig, 2018, p.49)

To find ^{A}P, one can simply multiply ^{B}P and $^{A}_{B}R$ as follows:

$$^{A}P = {}^{B}P \cdot {}^{A}_{B}R \quad (5)$$

Equation 5. Vector ^{B}P expressed as vector ^{A}P in the $\{A\}$ reference system (Craig, 2018, p.28)

2.6.1 Kinematics

To study motion further in detail, robotics makes use of kinematics, which is essentially the science of motion that does not regard forces which cause the motion. Position, velocity, acceleration, and the other higher order derivatives of the position variables are studied by the science of kinematics. The joints of a robot are equipped with sensors that detect motion, and since all joints should have a motion sensor, it is possible to determine the relative position of a certain joint and its link. Rotary or revolute joints, in turn can have displacements that can be interpreted in joint angles. (Craig, 2018, p.5)

It is important to know the joint angles when it comes to picking up a box, for example. The end-effector (fingers of the robot arm) and its success to pick up the box depend on the correct values of joint angles in the robot. This is when the problem of inverse kinematics comes to play. With inverse kinematics, it is possible to move a robot arm much like a human arm, but it's not "automated" in the way it is in the human brain and nervous system, and an algorithm is needed to accomplish the movement. Most modern robots use inverse kinematics and algorithms based on it. To do this, one needs to utilize a Jacobian matrix specific for a robot arm; it can be used to specify the mapping from velocities in joint space to velocities that reside within Cartesian space. (Craig, 2018, pp.5, 7)

The Jacobian matrix for *n*-number of variables (*x*) and equations (*y*) takes the form:

$$\mathbf{J}(x_1, \dots, x_n) = \begin{bmatrix} \frac{\partial \mathrm{y_1}}{\partial \mathrm{x_1}} & \cdots & \frac{\partial \mathrm{y_1}}{\partial \mathrm{x_n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathrm{y_n}}{\partial \mathrm{x_1}} & \cdots & \frac{\partial \mathrm{y_n}}{\partial \mathrm{x_n}} \end{bmatrix} \quad (6)$$

Equation 6. Jacobian matrix (Weisstein, 2022)

Where the **J** is the Jacobian determinant which is defined by:

$$J = \left| \frac{\partial(\mathrm{y_1}, \dots, \mathrm{y_n})}{\partial(\mathrm{x_1}, \dots, \mathrm{x_n})} \right| \quad (7)$$

Equation 7. Jacobian determinant (Weisstein, 2022)

In robotics, the Jacobian is used to determine the relation between joint velocities and end-effector velocities with the following equation (Krishna, no date):

$$\dot{X} = \mathbf{J}\dot{\theta} \quad (8)$$

Equation 8. Equation that connects the Jacobian to joint velocities and end-effector velocities (Krishna, no date)

Where

$\dot{X}$ = a matrix representing end-effector velocities of the manipulator

J = the Jacobian matrix which is a function of the position of the end-effector

$\dot{\theta}$ = a matrix representing joint velocities of the manipulator

The equation (8) is then represented in its matrix form as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} & J_{16} \\ J_{21} & J_{22} & J_{23} & J_{24} & J_{25} & J_{26} \\ J_{31} & J_{32} & J_{33} & J_{34} & J_{35} & J_{36} \\ J_{41} & J_{42} & J_{43} & J_{44} & J_{45} & J_{46} \\ J_{51} & J_{52} & J_{53} & J_{54} & J_{54} & J_{55} \\ J_{61} & J_{62} & J_{63} & J_{64} & J_{65} & J_{66} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{bmatrix} \quad (9)$$

Equation 9. Equation that connects the Jacobian to joint velocities and end-effector velocities in matrix form (Krishna, no date)

One may take a closer look at the equation (9). The columns of the Jacobian matrix ($J_{11}, \dots, J_{1n}$) represent the joints of the robot. For example, if a robot is a six-axis manipulator, the Jacobian matrix will have 6 columns in total. The first three values of the $\dot{X}$ matrix are linear velocities ($\dot{x}, \dot{y}, \dot{z}$), while the three last values are angular velocities in x, y, and z directions ($\omega_x, \omega_y, \omega_z$). (Krishna, no date)

2.6.2 Denavit-Hartenberg Convention

In forward kinematics, the Denavit-Hartenberg convention can be used to represent a robot as a series of connected joints and links. Using this notation, all joints of a robot are considered to have just one degree of freedom, and all of these joints are either prismatic or revolute joints. The base of the robot is referred to as link 0, and the joint that connects the link 0 and link 1 is

called joint 1. This notation uses four parameters: link length a, link twist α, link offset d, and joint angle θ (Fig. 17). (Appin Knowledge Solutions, 2007b, p.222)

Frames are attached to the joints of the robot in order to determine the robot's forward kinematics. The frames are 3-dimensional Cartesian frames and the direction of each component of the frame depend on the properties of the joint, but the direction of the z-axis for revolute joints is along the axis of the joint, as seen on Figure 17. (Appin Knowledge Solutions, 2007b, pp.226-227)

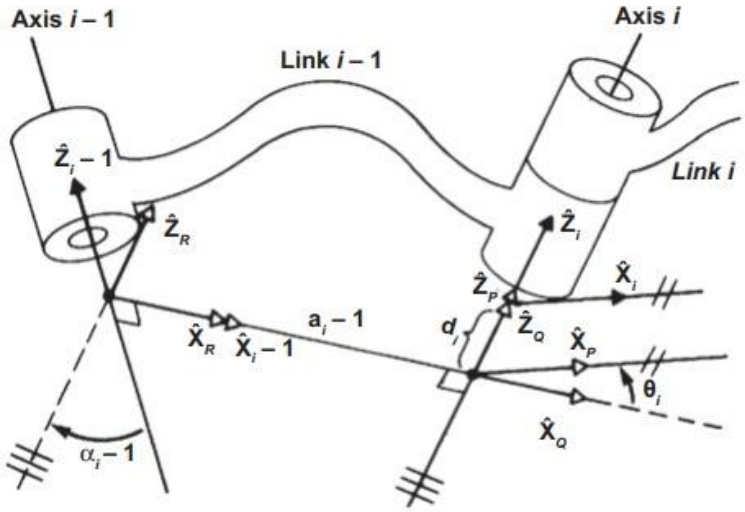

Figure 17. Representation of a link (Appin Knowledge Solutions, 2007a)

For revolute joints, the joint angles are controlled variables while the link offset is fixed. The opposite is true for prismatic joints. For both types, link length and link twist are always controlled parameters. The link length and link twist at the first joint and the last joint are always zero (Appin Knowledge Solutions, 2007b, p.225).

Based on the Denavit-Hartenberg parameters, one can find the homogeneous transformation matrix which includes the rotation 3x3 and the translation 3x1 submatrix:

$$^{n-1}T_n = \begin{bmatrix} \cos\theta_n & -\sin\theta_n \cos\alpha_n & \sin\theta_n \sin\alpha_n & r_n \cos\theta_n \\ \sin\theta_n & \cos\theta_n \cos\alpha_n & -\cos\theta_n \sin\alpha_n & r_n \sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & R & & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (10)$$

Equation 10. The Denavit-Hartenberg homogeneous transformation matrix (Wikipedia Contributors, 2022)

2.6.3 Dynamics

In contrary to kinematics, dynamics is interested in the forces that drive the motion of the robot. The actuators in the robot must generate torques to cause motion, and this is all possible by using dynamic equations of motions. Considering dynamics is important especially in articulated robots which often serve the purpose of lifting and manipulating heavy objects for humans; the robot must use dynamic equations of motion to react to the mass of the object to act in the desired manner defined by the operator.

One may now consider how a rigid object accelerates due to the forces that are subjected onto it. The velocity of an object is essentially the first integral of either angular or linear acceleration over a specific time, $^{A}\dot{\Omega}_{B}$ and $^{A}\dot{V}_{Q}$. The angular and linear accelerations can be calculated as follows:

$$^{A}\dot{\Omega}_{B} = \lim_{\Delta t \to 0} \frac{^{A}\Omega_{B}(t+\Delta t) - {}^{A}\Omega_{B}(t)}{\Delta t} \quad (11)$$

Equation 11. The angular acceleration relative to the frame $\{A\}$ (Craig, 2018, p.178)

$$^{B}\dot{V}_{Q} = \lim_{\Delta t \to 0} \frac{^{A}V_{B}(t+\Delta t) - {}^{A}V_{B}(t)}{\Delta t} \quad (12)$$

Equation 12. The linear acceleration relative to the frame $\{A\}$ (Craig, 2018, p.178)

Where

$t =$ the instantaneous time

$\Delta t =$ the difference between two moments of time

Since velocities and accelerations are now discussed, it is crucial to highlight the importance of the object's mass when it comes to dynamics. In a system with just one degree of freedom, the weight distribution of the object may become negligible for simplification, and can instead

be expressed with just one single value of mass. However, if an object is free to move in all three dimensions, it is useful to consider the inertia tensor, which can considered to be the scalar moment of inertia of the object. In this case, the inertia tensor is relative to the frame $\{A\}$ that is attached to our object, and can be expressed as follows in a matrix from:

$$ {}^{A}I = \begin{bmatrix} \mathrm{I_{xx}} & -\mathrm{I_{xy}} & -\mathrm{I_{xz}} \\ -\mathrm{I_{xy}} & \mathrm{I_{yy}} & -\mathrm{I_{yz}} \\ -\mathrm{I_{xz}} & -\mathrm{I_{yz}} & \mathrm{I_{zz}} \end{bmatrix} \quad (13) $$

Equation 13. The inertia tensor relative to the frame $\{A\}$ expressed in its 3x3 matrix form (Craig, 2018, p.180)

With the scalar elements containing the volume elements dv and density p:

$$ I_{xx} = \iiint_v (y^2 + x^2) p dv $$

$$ I_{yy} = \iiint_v (x^2 + z^2) p dv $$

$$ I_{zz} = \iiint_v (x^2 + y^2) p dv $$

$$ I_{xy} = \iiint_v xyp dv $$

$$ I_{xz} = \iiint_v xzp dv $$

$$ I_{yz} = \iiint_v yzp dv $$

The I_{xx}, I_{yy}, I_{zz} elements are mass moments of inertia, while the other elements are mass products of inertia. Each one of the volume elements dv can be located with a vector ${}^{A}P = [xyz]^T$.

Now, one shall consider two fundamental equations that drive dynamics, namely Euler's and Newton's Equations. Euler's Equation concerns the situation where a rigid object rotates with an angular velocity ω, and with an angular acceleration $\dot{\omega}$, which means that a moment Nm must act on the object to cause this rotational movement. Euler's Equation denotes the inertia

tensor of the object written in the frame $\{C\}$ with its origin located in the center of the mass as ${}^{C}I$, and is written as follows:

$$N = {}^{C}I\dot{\omega} + \omega \times {}^{C}I\omega \quad (14)$$

Equation 14. Euler's Equation, moment N (Craig, 2018, p.184)

The Newton's Equation denotes how a force F causes an acceleration $\dot{v}_C$ at the center of the mass of the object. The object has a total mass m:

$$F = m\dot{v}_C \quad (15)$$

Equation 15. Newton's Equation, force F (Craig, 2018, p.184)

(Craig, 2018, pp.9-10,178-180,184)

3 Method

This section outlines the practical work of this thesis, including the setup, operation, programming, and kinematic analysis of the Ned2 articulated cobot.

3.1 Setting Up the Robot

For this thesis, a Ned2 robot arm manufactured by Niryo will be used to investigate the problem definition. The robot is located in the production laboratory of Arcada University of Applied Sciences and is owned by the same institution.

The Ned2 six-axis articulated robot arm is an entry-level collaborative robot designed for mainly robotics education but also research, especially the education of industrial robotics and their possible applications. It is compatible for many different levels of expertise depending on the user's skill level and knowledge of robotics and programming. In addition to this, the robot can be controlled in several different ways; for those who do not know any programming languages, the robot comes with the intuitive Niryo Studio software. The software is based on Blockly, Google's JavaScript library that represents programming in the form of visual blocks (Google Developers, no date). The more advanced means of controlling the Ned2 robot include PyNiryo and ROS (Robot Operating System).

Ned2 supports two add-ons, the Vision Set as well as the Conveyor Belt Set. The Vision Set includes a camera, workspace, calibration tip, and six objects. This set can be used for image processing and machine learning applications. The Conveyor Belt set includes the same six objects and a mountable conveyor belt workspace intended to simulate production lines with possible Industry 4.0 applications (Fig. 18). (Niryo, 2022)

Figure 18. The Ned2 cobot with the Conveyor Belt Set

3.1.1 Overview of the Software and Hardware of Ned2

The Ned2 robot is based on the Raspberry Pi 4 mini-computer and has 4 GB of RAM (Random Access Memory). It employs the ARM V8 as its processor which has a typical clock speed of 1.5 GHz. Ned2 supports WIFI 5 connectivity and can be connected to a computer via USB 3.0. The software of Ned2 is based on the Ubuntu 18.04 Linux operating system as well as ROS Melodic. The robot weighs 7 kilograms, and its material composition is mostly aluminum and injection-molded plastic. The robot has a set of speakers and a microphone, which means that voice commands and sound output are also possible during its operation. (Niryo, 2023)

Since Ned2 is a collaborative robot, it also supports free motion and teaching by the operator. While pressing the “FreeMotion” button along its arm, one can activate a mode that enables the operator of the robot to manipulate its joints and move it in any possible joint configuration. This position can then be recorded on the software of choice and saved for later use. (Niryo, 2022)

3.1.2 Safe Operation and Programming of Ned2

The operation of all mechatronic machines typically poses various risks for the operator. In addition to this, the operator may also make mistakes that can lead to the machine getting

damaged unless certain precautions are carefully observed and implemented. In this section, the focus will be on the safe operation of the Ned2 articulated robot, particularly in educational facilities such as the production laboratory of Arcada University of Applied Sciences. When assembling the robot, one must refer to Niryo's official assembly guide with no exceptions.

Firstly, it is important that a flat, dedicated clear surface is allocated for the robot. The area should be cleaned thoroughly to make sure that the surface is free from liquids, and everything unrelated and unneeded for the operation of the robot should be removed from vicinity. If possible, the robot should be kept in a closed-off area or room that is accessible only for institutional staff and students who have been trained to operate the robot to reduce the risks such as theft or unauthorized and dangerous operation of the robot.

Safety zones must be defined before attempting to operate the robot through an external terminal, e.g., a computer. By using the FreeMotion function of Ned2, it is easy to define the reach of the robot in 3D space. The objects in 3D space can then be moved according to the reach of the robot, so the danger of getting struck by it can be reduced significantly. Consequently, when the robot is being operated, the operator must make sure that the safety zone is clear of people and objects that do not belong within it.

While running a program with Ned2, it is important to keep an eye on the robot's movements. This also applies for the case when the operator has simulated the robot's movement in a 3D environment. The actual surroundings of the robot may be subjected to changes that can drastically alter the real outcome of the program; one can never be sure how simulations turn out to be in real-life environments. The placement of the STOP button is very relevant, and it should be kept next to the operator's terminal in case something unexpected occurs. It is highly recommended that the operator does not intervene with the robot's movements by entering the safety zone while the program runs, but stops the robot from the terminal instead, or by pressing the STOP button.

While writing programs for Ned2, one must keep in mind the dimensions and contents of the working surface. For example, when different add-ons are used, such as the conveyor belt or objects, the robot can damage itself or the add-ons in case of collision. Therefore, it is prudent to add offset steps for the code. An offset is essentially a step where the gripper part of the robot arm is retracted from the working surface to prevent undesirable collisions with nearly

objects residing over the working surface (Fig. 19). When verifying and testing a program, the motor speeds must be decreased significantly from the desired values for the operator to be able to react quickly if an unexpected event occurs.

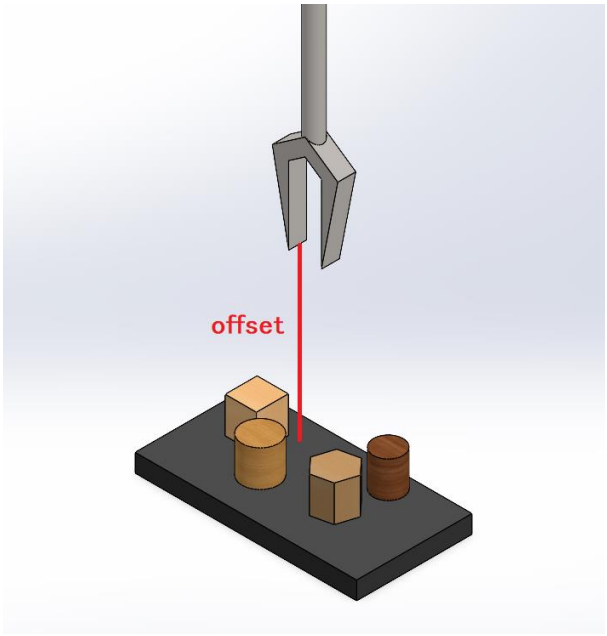

Figure 19. An example of an offset from the gripper to the working surface

3.1.3 Integration of the Vision and Conveyor Belt Sets

The operation of the Ned2 robot can be complemented with a camera that comes with the Vision Set that can be used to enable computer vision; with the camera, the robot is able to detect and classify objects by their color and shape. This feature is especially useful for industrial applications. In addition to this, Ned2 also supports a conveyor belt that can be used with the camera. The conveyor belt also has an infrared (IR) sensor which detects objects when they come close to it, such as objects moving over the conveyor belt.

The camera, IR sensor, and the conveyor belt are all controllable via the computer. The conveyor belt can move either forward or backward. The camera has a specific and fixed location where it is attached to the Ned2 robot, namely right above the tool. The camera was screwed onto the robot and then connected via a USB port that is located behind the base of the robot. Referring to Figure 20., the camera can be connected to one of the four USB ports at numbers one and two. The conveyor belt can be connected to the ports at 10, while the IR sensor can be connected right next to it at port number 12.

Figure 20. The back panel interface of Ned2 (Niryo, 2022b)

3.1.4 PyNiryo

PyNiryo is a Python module that can be used to control Ned2. With PyNiryo, one can create programs to be used with Ned2 and control the robot remotely without the need to connect the robot through a terminal. Before the PyNiryo package can be installed, one needs to make sure that Python is installed. PyNiryo can then be installed through Windows' Command Prompt interpreter. PyNiryo also requires the NumPy package, and to work with the Vision function of Ned2, one must also install the OpenCV package. The "pip install" command can be utilized for the installations. The installation of the needed packages through the Command Prompt can be seen on Figure 21.

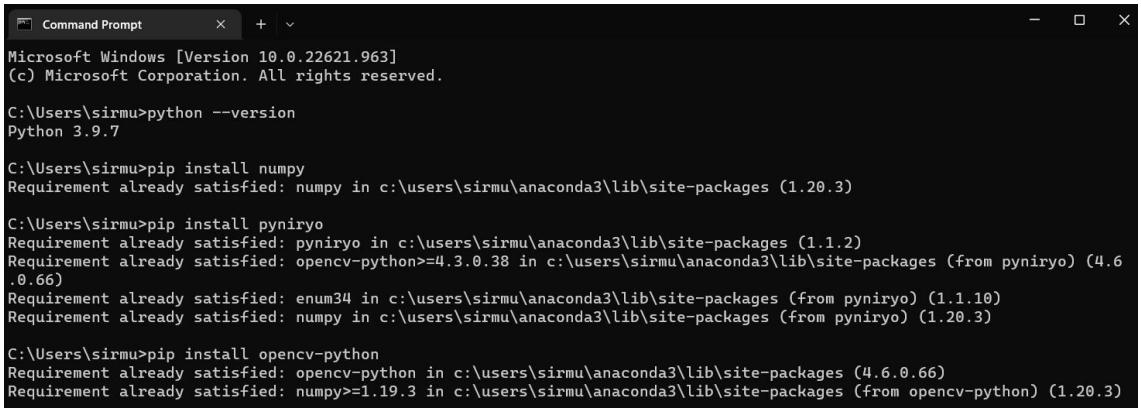

Figure 21. The installation and verification of needed packages with the Command Prompt on Windows

It is highly important that the computer is running the Python version that is 3.5 or higher. The right version of Python can be checked with “python –version” command in the Command Prompt interpreter (Fig. 21). If the version is outdated on Windows-based systems, one needs to update Python to the latest version by downloading it. On Ubuntu 18.04, one can simply open the Terminal and input “sudo apt install python-pip”, which downloads the latest version of Python. The sudo part of the command means that the command is executed with elevated rights. Apt, on the other hand, stands for advanced package tool, which is used to install packages. The other needed packages can be installed in the same way when it comes to Ubuntu 18.04, replacing “python” with the name of the desired package.

To program in Python, an editor is needed. For this thesis, Visual Studio Code is utilized to write, debug, and run programs for Ned2. Visual Studio Code is an open-source code editor developed by Microsoft which is free to download and use without any limitations. It offers its users an intuitive and simple user interface, and it supports a wide range of programming languages. After downloading and installing Visual Studio Code, Ned2 needs to be connected to the computer either wirelessly or via Ethernet. For this thesis, the Ethernet option is used in order to connect the robot. An Ethernet cable was connected from the robot to the computer, and a static IP was set, which is 169.254.200.200 by default.

When creating a new file in Visual Studio Code, one needs to make sure that the chosen interpreter is for Python 3.5 or a more recent version. This is due to the fact that the most recent PyNiryo module supports only Python 3.5 or higher (Niryo, 2023b). In Visual Studio code, one can establish connection between the robot and the editor by first importing the PyNiryo package. The importing is done by using the from-argument, then defining the module where the import is taken from, followed by the import-argument and whatever one wants to import from the module, in this case all the features of the module, which is denoted by the asterisk (*).

To define the IP of the robot, one creates a string object called robot, and then make it equal to NiryoRobot with the IP as its parameter:

```
from pyniryo import *

robot = NiryoRobot(“169.254.200.200”)
```

To close the connection between the computer and the robot, one needs to utilize the “robot” object with a function that is imported from the PyNiryo module called “close_connection()”. This can be implemented in the end of every Python file:

```
robot.close_connection()
```

3.1.5 Robot Operating System

The Robot Operating System (ROS) can be installed on Unix-based operating systems. In this subsection, the focus is put on the installation of the ROS Melodic distribution and how it can be implemented with the Ubuntu version 18.04. ROS can be used to control the real robot, or alternatively for simulation purposes. ROS can be installed either on a computer with the actual Ubuntu 18.04 OS, or as a Windows subsystem. For this thesis, both alternatives were investigated. The computer that is attached to the actual robot uses Ubuntu 18.04 as its operating system, while the possibilities of simulating the robot as a virtual twin are investigated with another computer that runs the operating system as a subsystem on Windows 11.

For this thesis, the ROS Melodic distribution is installed that is compatible with both Ned2 and Ubuntu 18.04. After the installation of ROS, one can install the needed Niryo ROS stack which is necessary to simulate Ned2 in a virtual environment. The installation of ROS and the Niryo ROS stack can be both done via the Terminal application on Ubuntu 18.04. ROS will be first installed, which must be started by installing the “curl” package. The curl package is used to transfer data between servers, and it supports multiple different data transfer protocols (curl.se, no date). This can be done by simply inputting “sudo apt install curl”.

The Ubuntu system must be able to accept packages from the ROS server in order to install the Melodic distribution. To accept packages and software from the server, the following command must be run: “sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'”. This adds ROS to the accepted sources list. The -c command signifies a flag which can be used to run something. The deb command is simply used to signify Debian packages. The echo command can used to display a string that is inputted by the user.

Next, one must define the source where the data is transferred from. To be able to import the ROS package, the following line of code must be inputted into the Terminal: “curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add –“. This command adds the key to the registry and enables the transfer to take place without restrictions. If the system is up to date, the ROS Melodic installation can be started with running “sudo apt install ros-melodic-desktop-full”, which is the full installation that includes all necessary functions to simulate Ned2.

After the installation is complete, one must set up the ROS environment. Assuming that only one ROS distribution is assumed, one can input the following line to the Terminal: “echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc”, and then “source ~/.bashrc”. In order to install dependencies and packages for various ROS workspaces, one must run the following command: “sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential”. Next, the installed package rosdep must be initialized by simply imputing “sudo rosdep init” and then “rosdep update”.

Finally, one can install the needed packages to simulate and control Ned2 via ROS. First, the needed Ubuntu packages need to be installed: sqlite3, and ffmpeg. They can both be installed using the “sudo apt install” command. The required Python environment is then installed with the following command: “pip2 install -r src/requirements_ned2.txt”. A new folder has to be created in order to store all the Niryo files with the command: “mkdir -p catkin_ws_niryo_ned/src”. The mkdir command is short for “make directory”, while -p means “parent”.

Lastly, the ROS dependencies for Ned2 must be installed, and the Ned2 repository must be cloned to the freshly created “catkin_ws_niryo_ned” folder. The latter can be simply done by writing the command “cd catkin_ws_niryo_ned” to move to the folder, and the cloning can be done by inputting “git clone https://github.com/NiryoRobotics/ned_ros src”. ROS dependencies can be then easily installed by the following two lines: “rosdep update” and “rosdep install --from-paths src --ignore-src --default-yes --rosdistro melodic --skip-keys "python-rpi.gpio"” in the same folder.

Remaining in the “catkin_ws_niryo_ned” folder, the final step of the ROS Ned2 installation must be completed by setting up the Ned ROS environment. Creating a new directory by inputting “catkin_make”, and then by using the source command, all Ned packages are added to the ROS environment: “source devel/setup.bash”. The latter command is important to remember as it must run each time a new terminal is launched. However, it is easy to add it to the bashrc file so that it is appended to every new launched terminal with the following commands: “echo "source $(pwd)/devel/setup.bash" >> ~/.bashrc” and “source ~/.bashrc”.

The integrity of the ROS installation and the Ned2 packages can be tested by opening a simple ROS simulation environment with the following command: “roslaunch niryo_robot_bringup niryo_ned2_simulation.launch”. This environment has no visualization nor the physics. To open a simulation environment with visualization and trackbar control for Ned2, one can utilize the RViz tool by inputting the command “roslaunch niryo_robot_description display.launch”. RViz enables for visualization and a user interface (Fig. 22).

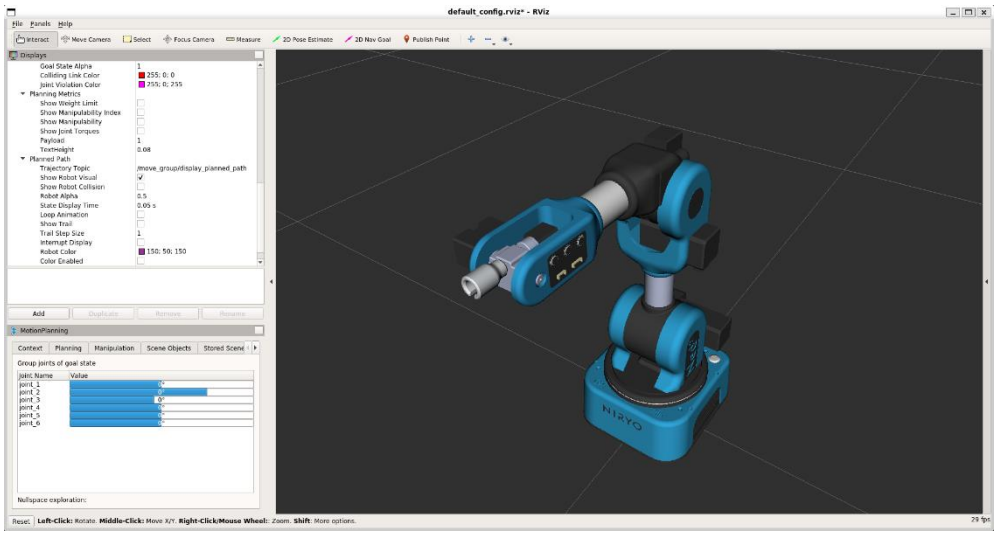

Figure 22. RViz with Ned2

3.2 Case Studies

3.2.1 Manual Pick-and-Place with PyNiryo

The goal of this subsection is to write a simple demonstration with PyNiryo and analyze the movements and actions of the robot with forward kinematics. The program utilizes the most fundamental functions of the PyNiryo Python module, such as joints and positions, but also

introduces the IR sensor and the conveyor belt and how they can be controlled and implemented to work together with the Ned2 cobot.

The idea of the case study is to move the conveyor belt forwards towards the direction of the robot with an object on it. As the object moves forward, it will be detected by the IR sensor. The IR sensor gives Ned2 a high signal, to which the robot responds by moving the tooltip/gripper towards the object with a slight delay so that the object will be under the tooltip. The robot will open its tool, approach the object, and then grasp the object. After this, the robot arm moves to the start of the conveyor belt and places the object back onto it. This action will be repeated by the robot until the program is manually stopped by the user.

Since this program does not utilize vision pick or adapt to the changes such as moving the robot further away from the conveyor belt, its success depends on the precise, pre-defined locations of the robot, objects, and the conveyor belt.

The program starts by importing everything from the PyNiryo package with “from pyniryo import *”. Next, an object is created called “tool_used” and assign it to a parameter “ToolID.GRIPPER_1”, which defines the tool that is used during the program, in this case the custom gripper which is denoted by the number one. After this, one can create the “robot” object and assign it to the robot’s IP address, the same way it was done while setting up PyNiryo. In order to connect the conveyor belt, one needs to create an object called “conveyor_id” and assign in to the parameter “robot.set_conveyor()”, while the parameter of the IR sensor depends on the digital input. In this case, the IR sensor is set to the digital input terminal number five, which is the designated port for the sensor (Fig. 20).

The robot is then calibrated with the “robot.calibrate_auto()” function and the tool is updated with “robot.update_tool()”. The first part of the program appears as follows:

```
from pyniryo import *

tool_used = ToolID.GRIPPER_1

robot = NiryoRobot("169.254.200.200")
```

```
conveyor_id = robot.set_conveyor()

sensor_pin_id = PinID.DI5

robot.calibrate_auto()
robot.update_tool()
```

The backbone of the program is the try-except structure. Under try resides the functions that define the actions of the robot. Under except there is a short, simple program that stops the actions of the robot if the user presses the keyboard letters CTRL and C simultaneously while being in the terminal. This is written as follows:

```
except KeyboardInterrupt:
            #CTRL+C to trigger keyboard interrupt
            robot.stop_conveyor(conveyor_id)
            pass
```

Being able to manually stop the program from running is very useful due to the fact that the program under try is an endless loop that will not stop automatically at any moment. If the program is stopped, the “robot.close_connection()” function is executed, and the connection between the computer and robot is closed.

Inside try, a set of functions are defined that are all part of the loop execution cycle. The first function that is defined is called “previous_position()” which prints a string “Returning back to the previous position” and manipulates the joints of the robot to the joint position “-0.039, 0.599, -1.102, 0.0139, -0.985, -0.0536” with a PyNiryo function using the “robot” as its variable. This joint position can be considered as the observation position (pose) where the camera is overlooking the conveyor belt, in which the gripper is ready to move towards the conveyor belt to grasp objects. The “previous_position()” function is written as follows:

```
def previous_position():
            print("Returning back to the previous position.")
            robot.move_joints(-0.039, 0.599, -1.102, 0.0139, -0.985, -0.0536)
```

The next function is called “checking_objects()” which includes a while-else structure for the IR sensor and conveyor belt. This part of the program starts with defining the function itself, and then calls the “previous_position()” function where the robot arm moves back to its observation pose. The while-else structure comes after this, where the condition for while is that the digital signal from the IR sensor is high. This means that the IR sensor does not detect anything on the conveyor belt. A print statement “No objects.” is shown in the terminal and the conveyor belt is run at speed 50% with the forwards direction. However, if the digital signal from the IR sensor is low, the else-condition gets triggered, which calls for a function called “ready_pick()”. This part of the program looks like this:

```
def checking_objects():
            previous_position()
            while robot.digital_read(sensor_pin_id) == PinState.HIGH:
                print("No objects.")
                robot.run_conveyor(conveyor_id, speed=50,
direction=ConveyorDirection.FORWARD)
            else:
                ready_pick()
```

Now, it is logical to consider the contents of the “ready_pick()” function, which is the next step of the program where the conveyor belt stops and the robot moves on top of the object for the grasping. As before, first it is needed to define the function with “def ready_pick():”, and then the terminal prints the string “Objects found!”. The robot object is used with the “wait” argument, and the robot stops for four seconds before the conveyor belt is stopped. It was found that four seconds is the time when the object is approximately in front of the robot. During these four seconds, the object that is detected on the conveyor belt will move directly under the robot. Then the “move_joints” function is used again with the variable “robot”. The position is in the joint position format. After the robot has moved to the grasping location, it is set to wait for two and half seconds before the “grasp_tool()” function is called. This piece of code is written as follows:

```
def ready_pick():
            print("Objects found!")
            robot.wait(4.0)
            robot.stop_conveyor(conveyor_id)
```

```
            robot.move_joints(-0.055, -0.2914, -0.414, -0.0674, -0.902, -0.064)
            robot.wait(2.5)
            grasp_tool()
```

The next function is the “grasp_tool()” function which includes the part where the robot opens the tool in order to grasp the object from the conveyor belt, lowers itself further towards the object, and closes the tool/gripper around the object, retracting from the conveyor belt while holding onto the object. The terminal first prints the string “Grasping!”, and then the tool opens by calling the “robot.release_with_tool()” function. There is a little wait time of 2.5 seconds before the robot arm lowers itself to the object with the function, after which the object is grabbed using the “robot.grasp_with_tool()” function, and then the robot retracts the tool from the conveyor belt with the object after another wait period of 2.5 seconds, moving to a different position. After this, the “placing_back()” function is called. This function is defined as follows:

```
def grasp_tool():
            print("Grasping!")
            robot.release_with_tool()
            robot.wait(2.5)
            robot.move_joints(-0.0546, -0.3065, -0.4462, -0.064334, -0.84838,
-0.064334539)
            print("Releasing!")
            robot.grasp_with_tool()
            robot.wait(2.5)
            robot.move_pose(0.2746, -0.0075, 0.2269, -1.790, 1.500, -1.800)
            robot.wait(2.5)
            placing_back()
```

The last function of this program is the “placing_back()” function, in which the robot arm moves the object to the end of the conveyor belt, and the try-except structure will be repeated unless the conditions under except are met. This function starts by printing “Placing the object back on the conveyor belt” in the terminal, and then the arm begins to move towards the end of the conveyor belt with the grasped object. There is a waiting time of 2.5 seconds before the arm would begin to lower itself towards the face of the conveyor belt, close enough so that the object would not bounce as it’s eventually released. The robot waits another 2.5 seconds before releasing the object onto the conveyor belt. It then retracts itself away from the object, and the

tool is closed again. The “checking_objects()” function is called, and the program repeats again unless stopped. This function is written as follows:

```
def placing_back():
            print("Placing the object back on the conveyor belt.")
            robot.move_pose(0.2746, -0.2468, 0.2269, -1.790, 1.500, -1.800)
            robot.wait(2.5)
            robot.move_pose(0.2746, -0.2468, 0.1969, -1.793, 1.500, -1.805)
            robot.wait(2.5)
            robot.release_with_tool()
            robot.wait(2.5)
            robot.move_joints(-0.7136,   -0.49288,   0.111318,   -0.0101645,   -
1.1981216, -0.650315)
            robot.grasp_with_tool()
            checking_objects()
```

The full program goes as follows:

```
from pyniryo import *

tool_used = ToolID.GRIPPER_1

robot = NiryoRobot("169.254.200.200")

conveyor_id = robot.set_conveyor()

sensor_pin_id = PinID.DI5

robot.calibrate_auto()
robot.update_tool()

try:
    while True:
        def previous_position():
            print("Returning back to the previous position.")
            robot.move_joints(-0.039, 0.599, -1.102, 0.0139, -0.985, -0.0536)
```

```
        def checking_objects():
            previous_position()
            while robot.digital_read(sensor_pin_id) == PinState.HIGH:
                print("No objects.")
                                    robot.run_conveyor(conveyor_id,    speed=50,
direction=ConveyorDirection.FORWARD)
            else:
                ready_pick()

        def ready_pick():
            print("Objects found!")
            robot.wait(4.0)
            robot.stop_conveyor(conveyor_id)
            robot.move_joints(-0.055, -0.2914, -0.414, -0.0674, -0.902, -0.064)
            robot.wait(2.5)
            grasp_tool()

        def grasp_tool():
            print("Grasping!")
            robot.release_with_tool()
            robot.wait(2.5)
             robot.move_joints(-0.0546, -0.3065, -0.4462, -0.064334, -0.84838,
-0.064334539)
            print("Releasing!")
            robot.grasp_with_tool()
            robot.wait(2.5)
            robot.move_pose(0.2746, -0.0075, 0.2269, -1.790, 1.500, -1.800)
            robot.wait(2.5)
            placing_back()

        def placing_back():
            print("Placing the object back on the conveyor belt.")
            robot.move_pose(0.2746, -0.2468, 0.2269, -1.790, 1.500, -1.800)
            robot.wait(2.5)
            robot.move_pose(0.2746, -0.2468, 0.1969, -1.793, 1.500, -1.805)
            robot.wait(2.5)
            robot.release_with_tool()
            robot.wait(2.5)
```

```
                robot.move_joints(-0.7136,  -0.49288,  0.111318,  -0.0101645,  -
1.1981216, -0.650315)
            robot.grasp_with_tool()
            checking_objects()

        checking_objects()

except KeyboardInterrupt:
    #CTRL+C to trigger keyboard interrupt
    robot.stop_conveyor(conveyor_id)
    pass

robot.close_connection()
```

3.2.2 Vision Pick-and-Place with PyNiryo

This case study includes the usage of the vision module of Ned2. Instead of manually programming the robot to pick up objects, a camera and computer vision is used instead. The paper workspace, as seen on Figure 23., is going to be used to define the landmarks, the four circular shapes in the four corners of the paper workspace, followed by the initialization of the workspace with Blockly by manually defining the locations of each landmark of the workspace with the robot's tooltip.

The program uses a loop which includes a continuous search for blue objects. With the use of vision, the robot can judge the objects placed over the workspace. In this case, there are two different shapes of objects and three different colors of objects available. In the program, it is chosen that the robot tries to pick up all blue objects and place them back on the workspace after grasping and retracting from the pick location. Since the robot will rely on vision to judge the object's color, there are uncertainties which depend on the object recognition algorithm or the environment surrounding the robot.

For the best results, one can adjust the camera settings such as brightness and contrast, so that the colors look as vibrant as possible, and the contours that the objects create are as sharp as possible. One can change these parameters either on PyNiryo or via Niryo Studio. In this case study, the settings of the camera are adjusted on Niryo Studio.

Contrast can be used in order to make the colors of the objects stand out more. As seen on Figure 23., “contrast” brings more color to objects which aids the vision module to better judge the objects’ color and consequently pick the right one.

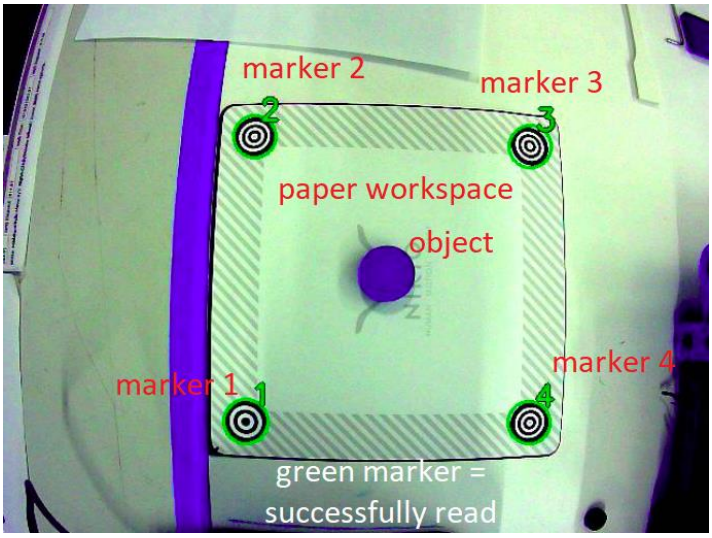

Figure 23. Camera view with maximum contrast

If the environment of the robot is dim and light intensity cannot be increased, the camera settings can be modified to compensate for the lack of brightness with the “brightness” slider (Fig. 24).

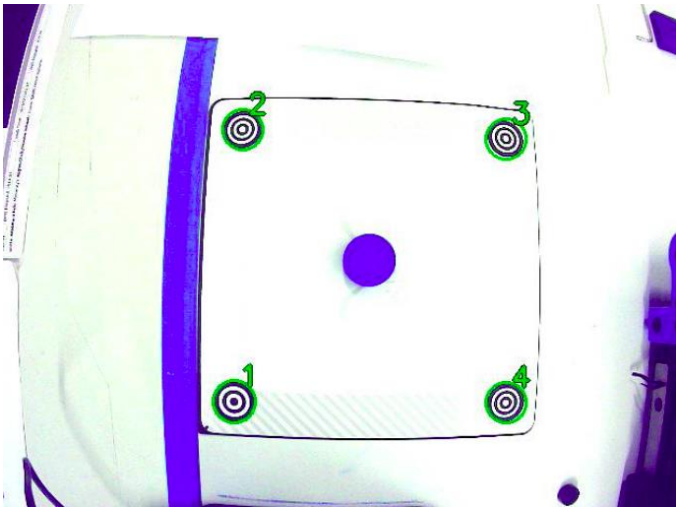

Figure 24. Camera view with maximum brightness

The 2D camera calculates the positions of the objects within the workspace using the four markers, where the markers are the origin in the 3D space. Essentially, the camera

communicates with the robot and gives it a relative position in its own reference frame, which the robot translates to its own interpretation of position in its own reference frame. (Niryo, 2021)

It must be noted that the objects have to be already within the workspace before running the program with the robot. It is not possible to add more objects during vision pick-and-place.

One starts writing the program by importing the PyNiryo package and defining the IP of the robot as previously:

```
from pyniryo import *

robot = NiryoRobot("169.254.200.200")
```

Now, one needs to create a workspace on Niryo Studio. In our case, the physical workspace is the paper workspace that came with Ned2, and it is on the robot's left-hand side over the table (Fig. 25).

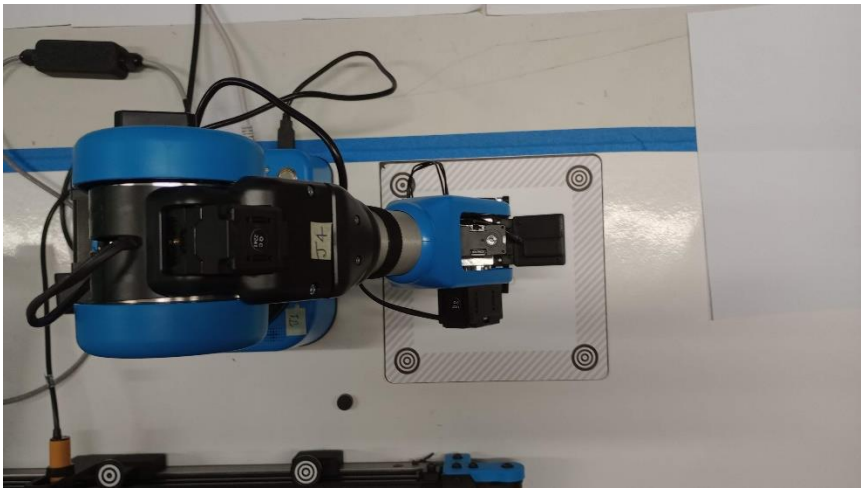

Figure 25. Ned2 with the paper workspace

The workspace has to be defined by adding a new workspace in the vision tab by clicking the plus button, and then following the markers in the corner of the paper clockwise. By following the instructions given by the software, the workspace is then ready to be used with PyNiryo (Fig. 26).

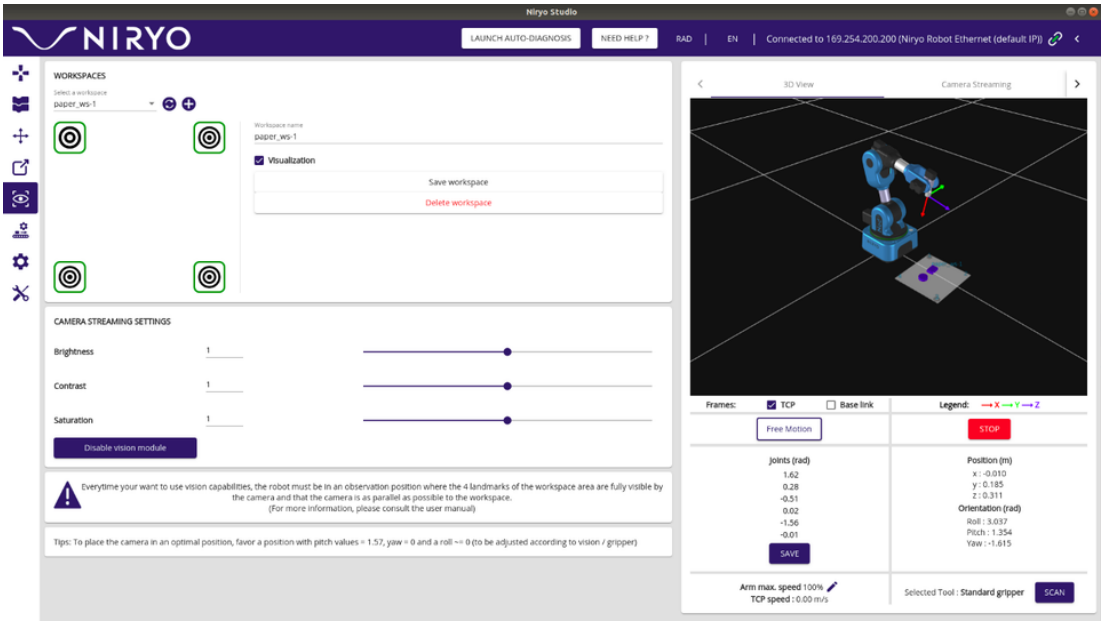

Figure 26. A defined workspace in Niryo Studio

This workspace is then imported to PyNiryo by creating an object called “workspace_name”, and by using the name of the workspace defined on Niryo Studio, which in this case is “paper_ws-1”:

```
workspace_name = "paper_ws-1"
```

Then one calibrates the robot and update the tool just like in the previous case study before writing the program itself:

```
robot.calibrate_auto()
robot.update_tool()
```

To use the vision module, two new objects have to be defined that denote the pick-and-place locations. They are called “observation_pose” and “place_pose”. The observation pose defines the location where the robot is observing the workspace and detecting and analyzing objects by their shape and color. The user can input the x, y, and z coordinates and the roll, pitch, and yaw of the end-effector. The place pose defines where the robot places the picked-up object. The objects are written as follows:

```
observation_pose = PoseObject(
```

```
    x=-0.010, y=0.185, z=0.311,
    roll=3.032, pitch=1.352, yaw=-1.615)

place_pose = PoseObject(
    x=-0.012, y=0.252, z=0.086,
    roll=2.772, pitch=1.559, yaw=-1.913)
```

After defining the locations for observation and place, before the start of the try-except structure, the robot is moved to the observation pose:

```
robot.move_pose(observation_pose)
```

One uses the try-except structure again to make a looping program unless the user interrupts it with the key combination. After this, the function “robot.close_connection()” is used again to close to connection between the computer and the robot. The except structure is written as follows:

```
except KeyboardInterrupt:
    #CTRL+C to trigger keyboard interrupt
    Pass
```

Under “try”, a new indent is created and a code section called “while True” is included, which is the section where the observation, picking, and placing happens unless it becomes false with the keyboard interrupt. The first part that is needed to be defined by the user is to define an object called “obj_found, shape_ret, color_ret”:

```
while True:
        obj_found, shape_ret, color_ret = robot.vision_pick(workspace_name,
                                                    height_offset=0.01,
                                                    shape=ObjectShape.ANY,
                                                    color=ObjectColor.BLUE)
```

This object includes the “robot.vision_pick” function which is a preset function in PyNiryo which is used to define the workspace, the height offset from the workspace, the shape of the object, and the color of the object that is to be picked. A $0.01\,m$ offset is chosen with

“height_offset” and the shape of the object can be anything with “shape=ObjectShape.ANY”, with the desirable color being blue by writing “color=ObjectColor.BLUE”.

Then another indentation is made, and it is defined what happens if a suitable object is found with “if obj_found:”. What was defined previously plays an important role here to initiate this if-statement since this statement will only get triggered if the camera detects a suitable object. In case the prescribed conditions are met, the robot arm will approach the object according to the height of the offset, grasp it, and then place it to the location defined in the “place_pose” object. After this, the arm goes back to the “observation_pose” position and the program repeats unless stopped.

```
        if obj_found:
            robot.place_from_pose(place_pose)
            robot.move_pose(observation_pose)
```

All these parts combined; the program is written as follows:

```
from pyniryo import *

robot = NiryoRobot("169.254.200.200")
workspace_name = "paper_ws-1"

robot.calibrate_auto()
robot.update_tool()

observation_pose = PoseObject(
    x=-0.010, y=0.185, z=0.311,
    roll=3.032, pitch=1.352, yaw=-1.615)

place_pose = PoseObject(
    x=-0.012, y=0.252, z=0.086,
    roll=2.772, pitch=1.559, yaw=-1.913)

robot.move_pose(observation_pose)

try:
```

```
    while True:
        obj_found, shape_ret, color_ret = robot.vision_pick(workspace_name,
                                                            height_offset=0.01,
                                                            shape=ObjectShape.ANY,
                                                            color=ObjectColor.BLUE)

        if obj_found:
            robot.place_from_pose(place_pose)
            robot.move_pose(observation_pose)

except KeyboardInterrupt:
    #CTRL+C to trigger keyboard interrupt
    pass

robot.close_connection()
```

3.2.3 RViz and Gazebo (ROS)

This case study is a brief example of how Ned2 can be controlled in RViz and Gazebo simulation and also introduces the basic capabilities of both of the software and their possibilities in robotics research. The goal is to show how the robot's motion can be planned in RViz and then executed, and how objects and virtual environments can be created in Gazebo.

To open RViz with ROS control for Ned2, one needs to execute the following command in the Ubuntu Terminal:

```
roslaunch niryo_robot_bringup desktop_rviz_simulation.launch
```

To open Gazebo with the Ned2 robot, a new terminal has to be opened and the following has to be run:

```
roslaunch niryo_robot_bringup desktop_gazebo_simulation.launch
```

One can see that the position of the end-effector of Ned2 in RViz can be defined by the three Cartesian axes and the three different types of rotations: roll, pitch, and yaw (Fig. 27).

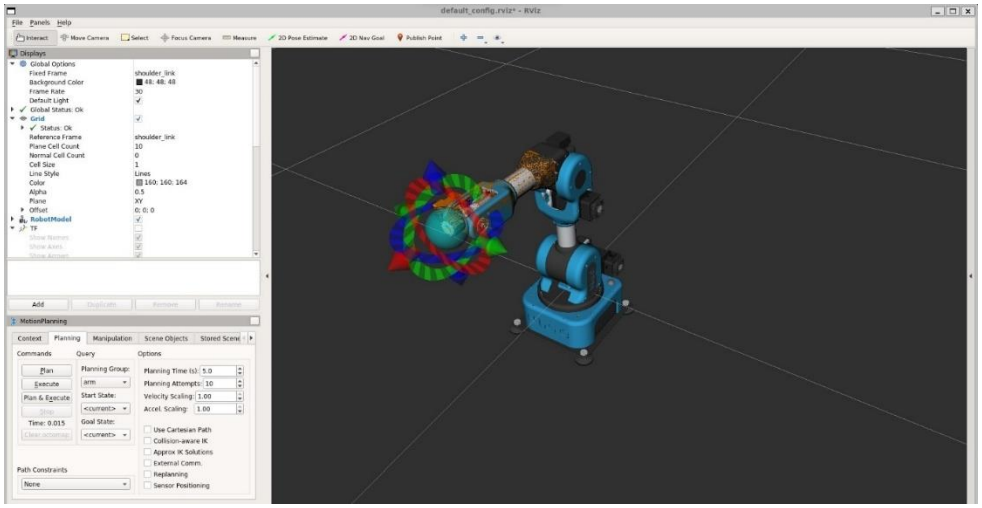

Figure 27. Ned2 in RViz

Motion planning can be done easily by manipulating the end-effector of the robot with the given sliders. After the desired position is defined by the user by interacting with the end-effector, one can go to the “Planning” tab under “MotionPlanning” and click “Plan”. It is useful to check the “Collision-aware IK” box under “Options”, which will show if the joints or links of the robot collide with one another. To add a start position manually, one can click “<current>” under “Start State” under “Query” and choose “Straight Forward”. After this, one clicks “Execute” in order to make the robot move to the desired position (Fig. 28).

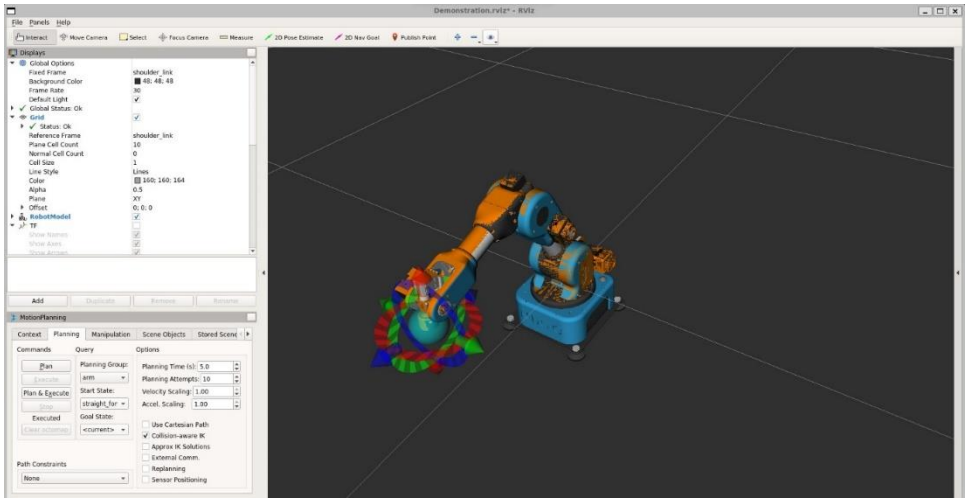

Figure 28. Executing the desired planned motion in RViz

It is possible to add objects such as a cylinder into the simulation by going to the “Scene Objects” tab under “MotionPlanning”. Under “Add/Remove scene object(s)”, a cylinder can

be added by clicking on the “Box” dropdown menu and then choosing “Cylinder”, after which one can press the plus sign. It is also possible to change the size of the cylinder under “Add/Remove object(s)” by changing the three values, two of which are available for cylinders due to the object’s geometry (Fig. 29). The cylinder will appear to the 3D view, and it can be moved with the arrows.

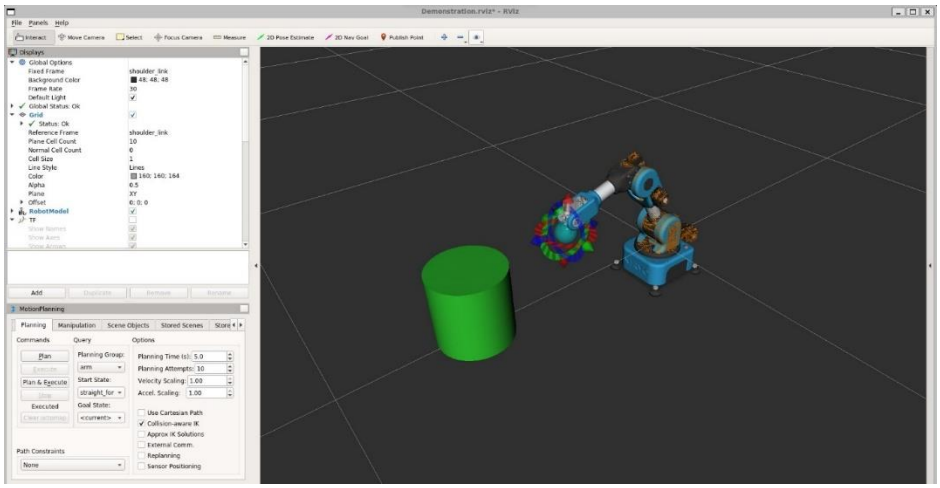

Figure 29. Ned2 with a cylinder in RViz

RViz can also display the joint angles of the robot in degrees in the “Joints” tab under “MotionPlanning” (Fig. 30).

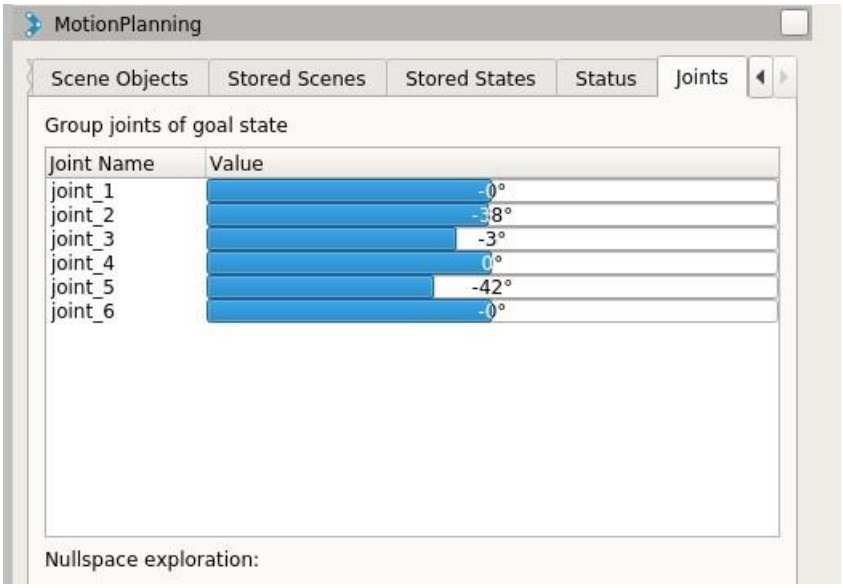

Figure 30. Ned2 joint angles in RViz

Now, the individual capabilities of Gazebo are investigated. Gazebo offers more versatile tools compared to RViz when it comes to simulating a real environment. It is possible to add light, wind, change the temperature of the environment, and even edit the values of gravity, which can be useful for applications such as space robotics. It is also possible to add objects and build an environment with, for example, walls and even stairs and floors. The possibility to create a whole building in Gazebo is useful especially for mobile robotics research.

First it is demonstrated how to create an object in Gazebo. By clicking “Edit” in the left-hand corner of the Gazebo window and then clicking “Model Editor”, the Model Editor window pops up where one can create and edit objects (Fig. 31).

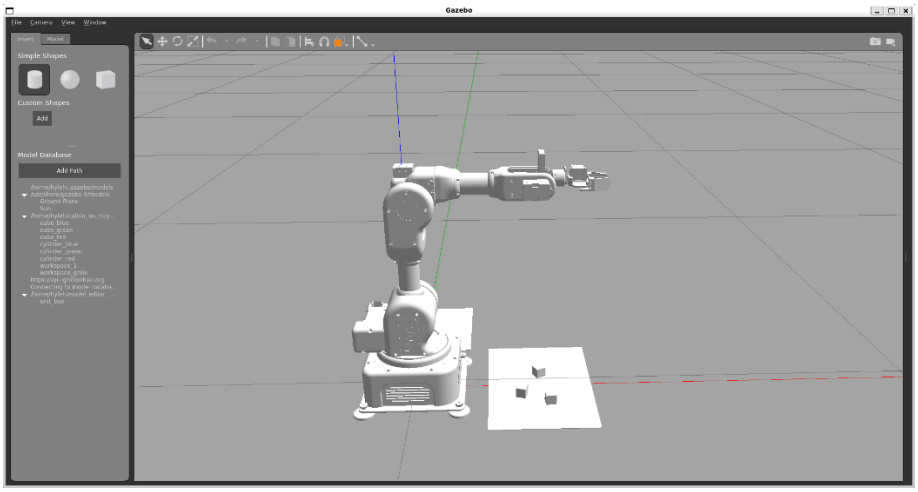

Figure 31. The Model Editor of Gazebo

It is possible to either choose a simple shape or import a custom 3D mesh. Now, the cylinder is chosen under “Simple Shapes” for simplicity. After choosing the desired shape, one must drag it to the simulation window and then left click to place it in the 3D view of Gazebo. The objects can be easily scaled by entering the Scale Mode by pressing “S” on the keyboard. Consequently, after resizing the object if needed, one can go back to the Selection Mode, one can press “ESC”. To move the resized object, one presses “T” to enter the Translation Mode. Exiting the model editor can be done by clicking “Edit” and then “Exit Model Editor”, after which the model can be saved.

Gazebo enables the user to add different kinds of lights. On the top toolbar of the simulation window, three kinds of different lights can be added. In this demonstration, we add a “spot light” and drag it over the robot and the workspaces to highlight the area better (Fig. 32).

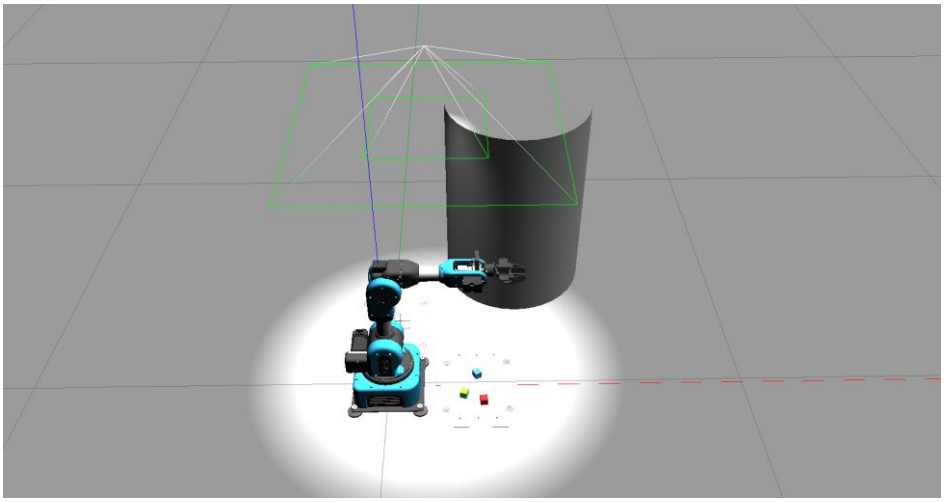

Figure 32. The simulation environment with a cylinder and light

To create an actual virtual environment around the robot, the “Building Editor” can be opened under “Edit”. In this demonstration, a simple rectangular room is added to the simulation with a window and door. Under “Create Walls”, “Wall” is chosen and then the walls are drawn onto the gridded area on the right. Adding windows and doors is similar and can be done by clicking under “Add Features”. Colors and textures can also be added to the various building objects under “Colors” and “Add Texture”. The completed room can be seen on Figure 33.

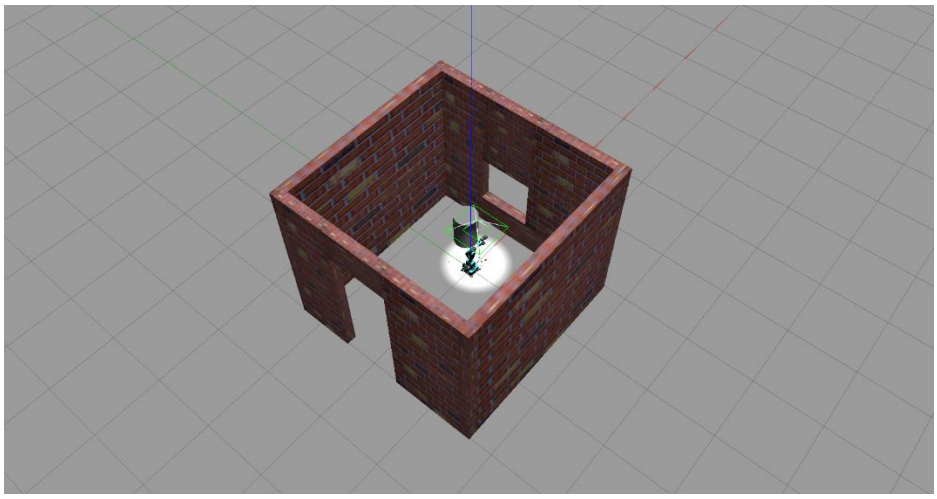

Figure 33. Ned2 inside a virtual room created with Gazebo

3.3 Kinematic Studies

3.3.1 Forward Kinematics of Ned2

In this demonstration, the focus is shifted from a more hands-on example to the mathematics behind robot control. First, the movement during the "grasp_tool()" and "placing_back()" functions of the manual pick-and-place PyNiryo case study are inspected. During this movement, the robot moves from the grasping point to the end of the conveyor belt. Instead of using joints, positions are used when it comes to the "robot.move_pose()" function of the case study in order to analyze the translation matrix from point A to B.

The origin of this Cartesian coordinate system is the surface under the robot. The first three values inside this function are the x, y, and z coordinates, and the latter three are roll, pitch, and yaw; roll is related to the x-axis, pitch is related to the y-axis, and yaw is related to the z-axis. One can first create a transformation translation matrix $T_{A,B}$ with A being the position (0.2746, -0.0075, 0.2269, -1.790, 1.500, -1.800) and B being the position (0.2746, -0.2468, 0.2269, -1.790, 1.500, -1.800) denoted by P_B.

The translation matrix is found by implementing the equation (2):

$$P_B = \begin{bmatrix} 0.2746 \\ -0.2468 \\ 0.2269 \end{bmatrix} = \begin{bmatrix} 0.2746 \\ -0.0075 \\ 0.2269 \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} x \\ y \\ z \end{bmatrix} = T_{A,B} = \begin{bmatrix} 0.2746 \\ -0.2468 \\ 0.2269 \end{bmatrix} - \begin{bmatrix} 0.2746 \\ -0.0075 \\ 0.2269 \end{bmatrix} = \begin{bmatrix} 0.0000 \\ -0.2393 \\ 0.0000 \end{bmatrix}$$

This means that the end-effector of the robot does not translate in the x or z axes, but only in the y-axis. The translation matrix can be expressed in its vector form as follows $T_{A,B} = [0.0000, -0.2393, 0.0000]$.

In order to calculate the transformation matrix with the use of joints, one must consider all the six joints of the robot to find the final position of the end-effector. Let's take the joint movement from the function "ready_pick()" from the manual pick-and-place case study. The joint values in radians are (-0.055, -0.2914, -0.414, -0.0674, -0.902, -0.064). These joint

values shall be denoted as $J_A =$
$[-0.0550, -0.2914, -0.1410, -0.0674, -0.9020, -0.0640]$.

In order to calculate transformation matrix for the joints, one must know the lengths of each link in the robot arm. Using the "Measure" function in RViz, the link lengths were measured:

Table 1. Link lengths measured with RViz

Link	Length (m)
1	0.079
2	0.220
3	0.060
4	0.180
5	0.030
6	0.020

The transformations are calculated for each joint separately using the Denavit-Hartenberg convention, and the total transformation at the end-effector of a six-axis robot can be calculated with the following equation:

$$^{0}T_{6} = {}^{0}T_{1} \cdot {}^{1}T_{2} \cdot {}^{2}T_{3} \cdot {}^{3}T_{4} \cdot {}^{4}T_{5} \cdot {}^{5}T_{6}$$

Before being able to determine the transformation matrix for each joint, the Denavit-Hartenberg method must be used in order to determine the link and joint parameters. One can determine the default pose for Ned2 when all joint angles are zero with RViz (ROS) (Fig. 34), which will help with constructing the diagram.

Figure 34. Zero joint angles of Ned2 in RViz

The Denavit-Hartenberg diagram for Ned2 is shown in Figure 35. with the rotational direction of the angles denoting the positive direction. In the diagram, it is to be noticed that the geometry of the robot has been simplified, and one geometric property of Ned2 have been omitted, namely the offset from joint 3 to joint 4; this may lead to an imperfect result that does not correspond to the position of the real robot. As can be seen in the diagram, the fifth frame had to be moved over to the fourth frame since the previous z-axis and the current x-axis must intersect each other according to the rules of the Denavit-Hartenberg convention.

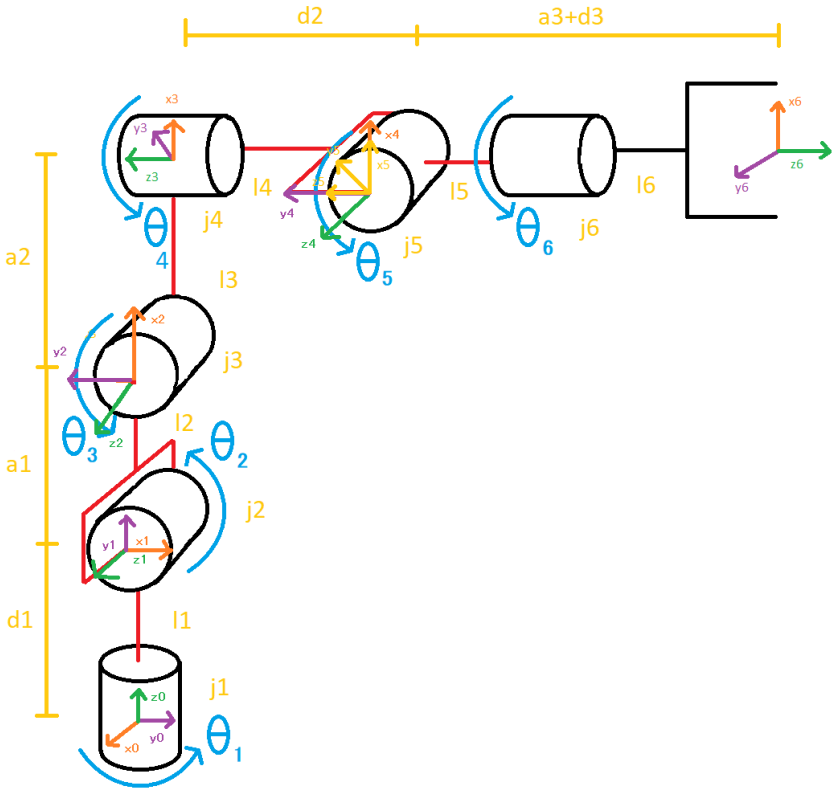

Figure 35. The Denavit-Hartenberg notation for Ned2

Based on Figure 35., one can create a Denavit-Hartenberg parameter table (Table 2.).

Table 2. The Denavit-Hartenberg parameter table for Ned2

Joint	**Joint angle** $\boldsymbol{\theta}$ **(rad)**	**Link twist** $\boldsymbol{\alpha}$ **(rad)**	**Link length** $\boldsymbol{a/r}$ **(m)**	**Link offset** $\boldsymbol{d}$ **(m)**
1	θ_1	1.571	0	d_1
2	θ_2	0	a_1	0
3	θ_3	-1.571	a_2	0
4	θ_4	1.571	0	$-d_2$
5	θ_5	-1.571	0	0
6	θ_6	3.141	0	$-(a_3 + d_3)$

Now, with values of a/r and d measured with RViz:

Table 3. The Denavit-Hartenberg parameter table for Ned2

Joint	**Joint angle $\boldsymbol{\theta}$ (rad)**	**Link twist $\boldsymbol{\alpha}$ (rad)**	**Link length $\boldsymbol{a/r}$ (m)**	**Link offset $\boldsymbol{d}$ (m)**
1	θ_1	1.571	0	0.079
2	θ_2	0	0.220	0
3	θ_3	-1.571	0.060	0
4	θ_4	1.571	0	-0.180
5	θ_5	-1.571	0	0
6	θ_6	3.141	0	-(0.030 + 0.020)

Finally, inputting the joint angles J_A:

Table 4. The Denavit-Hartenberg parameters for joint angles J_A

Joint	**Joint angle $\boldsymbol{\theta}$ (rad)**	**Link twist $\boldsymbol{\alpha}$ (rad)**	**Link length $\boldsymbol{a/r}$ (m)**	**Link offset $\boldsymbol{d}$ (m)**
1	-0.055	1.571	0	0.079
2	-0.291	0	0.220	0
3	-0.141	1.571	0.060	0
4	-0.067	-1.571	0	-0.180
5	-0.902	1.571	0	0
6	-0.064	3.141	0	-0.050

These values can then be inputted into the Denavit-Hartenberg homogeneous transformation matrix using the equation (10) for each joint, starting with the first joint:

$$ {}^{0}T_{1} = \begin{bmatrix} \cos(-0.055) & -\sin(-0.0550)\cos(1.571) & \sin(-0.055)\sin(1.571) & 0\cos(-0.055) \\ \sin(-0.055) & \cos(-0.055)\cos(1.571) & -\cos(-0.055)\sin(1.571) & 0\sin(-0.055) \\ 0 & \sin(1.571) & \cos(1.571) & 0.079 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

For the second joint:

$$
{}^{1}T_{2} = \begin{bmatrix} \cos(-0.291) & -\sin(-0.291)\cos(0) & \sin(-0.291)\sin(0) & 0.220\cos(-0.291) \\ \sin(-0.291) & \cos(-0.291)\cos(0) & -\cos(-0.291)\sin(0) & 0.220\sin(-0.291) \\ 0 & \sin(0) & \cos(0) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

For the third joint:

$$
{}^{2}T_{3} = \begin{bmatrix} \cos(-0.141) & -\sin(-0.141)\cos(1.571) & \sin(-0.141)\sin(1.571) & 0.060\cos(-0.141) \\ \sin(-0.141) & \cos(-0.141)\cos(1.571) & -\cos(-0.141)\sin(1.571) & 0.060\sin(-0.141) \\ 0 & \sin(1.571) & \cos(1.571) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

For the fourth joint:

$$
{}^{3}T_{4} = \begin{bmatrix} \cos(-0.067) & -\sin(-0.067)\cos(-1.571) & \sin(-0.067)\sin(-1.571) & 0\cos(-0.067) \\ \sin(-0.067) & \cos(-0.067)\cos(-1.571) & -\cos(-0.067)\sin(-1.571) & 0\sin(-0.067) \\ 0 & \sin(-1.571) & \cos(-1.571) & -0.180 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

For the fifth joint:

$$
{}^{4}T_{5} = \begin{bmatrix} \cos(-0.902) & -\sin(-0.902)\cos(1.571) & \sin(-0.902)\sin(1.571) & 0\cos(-0.902) \\ \sin(-0.902) & \cos(-0.902)\cos(1.571) & -\cos(-0.902)\sin(1.571) & 0\sin(-0.902) \\ 0 & \sin(1.571) & \cos(1.571) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

And for the last joint, the sixth:

$$
{}^{5}T_{6} = \begin{bmatrix} \cos(-0.064) & -\sin(-0.064)\cos(-3.140) & \sin(-0.064)\sin(-3.140) & 0\cos(-0.064) \\ \sin(-0.064) & \cos(-0.064)\cos(-3.140) & -\cos(-0.064)\sin(-3.140) & 0\sin(-0.064) \\ 0 & \sin(-3.140) & \cos(-3.140) & -0.050 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Now, using the equation (14):

$$
{}^{0}T_{6} = \begin{bmatrix} 0.889 & -0.066 & -0.453 & 0.317 \\ 0.059 & 0.998 & -0.032 & -0.019 \\ 0.455 & 0.002 & 0.891 & 0.200 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

This yields a rotation matrix R_A:

$$R_A = \begin{bmatrix} 0.889 & -0.066 & -0.453 \\ 0.059 & 0.998 & -0.032 \\ 0.455 & 0.002 & 0.891 \end{bmatrix}$$

And a position matrix P_A:

$$P_A = \begin{bmatrix} 0.317 \\ -0.019 \\ 0.200 \end{bmatrix} \quad (16)$$

3.3.2 Jacobian Matrix of Ned2

A Jacobian matrix is used when the joint velocities of the robot must be related to the velocity of the end-effector. Using the equation (9) and inputting it into the equation (8), the latter for a six-axis robot arm is denoted as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} & J_{16} \\ J_{21} & J_{22} & J_{23} & J_{24} & J_{25} & J_{26} \\ J_{31} & J_{32} & J_{33} & J_{34} & J_{35} & J_{36} \\ J_{41} & J_{42} & J_{43} & J_{44} & J_{45} & J_{46} \\ J_{51} & J_{52} & J_{53} & J_{54} & J_{55} & J_{56} \\ J_{61} & J_{62} & J_{63} & J_{64} & J_{65} & J_{66} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \end{bmatrix}$$

To find the Jacobian matrix, one needs to first write down the transformation matrices for each link in the robot. In total, six transformation matrices are required, one for each of the six links. In this case, MATLAB can used when multiplying the matrices to find the transformation matrix from frame 0 to 6, ${}^0T_6$ to aid the calculation, which is expected to be tedious since six variables (joint angles) have to be considered. The same transformation matrices as in the previous demonstration can be used with the same values of α, a, and d, but without defined joint angles:

$${}^0T_1 = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1)\cos(1.571) & \sin(\theta_1)\sin(1.571) & 0\cos(\theta_1) \\ \sin(\theta_1) & \cos(-0.055)\cos(1.571) & -\cos(\theta_1)\sin(1.571) & 0\sin(\theta_1) \\ 0 & \sin(1.571) & \cos(1.571) & 0.079 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For the second joint:

$$
{}^{1}T_{2} = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2)\cos(0) & \sin(\theta_2)\sin(0) & 0.220\cos(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2)\cos(0) & -\cos(\theta_2)\sin(0) & 0.220\sin(\theta_2) \\ 0 & \sin(0) & \cos(0) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

For the third joint:

$$
{}^{2}T_{3} = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3)\cos(-1.571) & \sin(\theta_3)\sin(-1.571) & 0.060\cos(\theta_3) \\ \sin(\theta_3) & \cos(\theta_3)\cos(-1.571) & -\cos(\theta_3)\sin(-1.571) & 0.060\sin(\theta_3) \\ 0 & \sin(-1.571) & \cos(-1.571) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

For the fourth joint:

$$
{}^{3}T_{4} = \begin{bmatrix} \cos(\theta_4) & -\sin(\theta_4)\cos(1.571) & \sin(\theta_4)\sin(1.571) & 0\cos(\theta_4) \\ \sin(\theta_4) & \cos(\theta_4)\cos(1.571) & -\cos(\theta_4)\sin(1.571) & 0\sin(\theta_4) \\ 0 & \sin(1.571) & \cos(1.571) & -0.180 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

For the fifth joint:

$$
{}^{4}T_{5} = \begin{bmatrix} \cos(\theta_5) & -\sin(\theta_5)\cos(1.571) & \sin(\theta_5)\sin(1.571) & 0\cos(\theta_5) \\ \sin(\theta_5) & \cos(\theta_5)\cos(1.571) & -\cos(\theta_5)\sin(1.571) & 0\sin(\theta_5) \\ 0 & \sin(1.571) & \cos(1.571) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

And for the last joint, the sixth:

$$
{}^{5}T_{6} = \begin{bmatrix} \cos(\theta_6) & -\sin(\theta_6)\cos(-3.140) & \sin(\theta_6)\sin(-3.140) & 0\cos(\theta_6) \\ \sin(\theta_6) & \cos(\theta_6)\cos(-3.140) & -\cos(\theta_6)\sin(-3.140) & 0\sin(\theta_6) \\ 0 & \sin(-3.140) & \cos(-3.140) & -0.050 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Now, using the equation (14) with MATLAB:

$$
{}^{0}T_{6} = \begin{bmatrix} \cdots & \cdots & \cdots & \mathrm{x} \\ \cdots & \cdots & \cdots & \mathrm{y} \\ \cdots & \cdots & \cdots & \mathrm{z} \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Where

$$x = (676279337148533 * sin(t1))/18446744073709551616 + (11 * cos(t1) * cos(t2))/50 - (1801439813583837 * cos(t5) * ((4503599440548691 * cos(t3) * (cos(t1) * sin(t2) - (7514214926474921 * cos(t2) * sin(t1))/36893488147419103232))/4503599627370496 - (3757107307382221 * sin(t1))/18446744073709551616 + (7514214926474921 * sin(t4) * (cos(t3) * (cos(t1) * cos(t2) + (7514214926474921 * sin(t1) * sin(t2))/36893488147419103232) - sin(t3) * (cos(t1) * sin(t2) - (7514214926474921 * cos(t2) * sin(t1))/36893488147419103232)))/36893488147419103232 + (4503599440548691 * sin(t3) * (cos(t1) * cos(t2) + (7514214926474921 * sin(t1) * sin(t2))/36893488147419103232))/4503599627370496 - (7514214926474921 * cos(t4) * ((4503599440548691 * sin(t1))/4503599627370496 + (7514214926474921 * cos(t3) * (cos(t1) * sin(t2) - (7514214926474921 * cos(t2) * sin(t1))/36893488147419103232))/36893488147419103232 + (7514214926474921 * sin(t3) * (cos(t1) * cos(t2) + (7514214926474921 * sin(t1) * sin(t2))/36893488147419103232))/36893488147419103232))/36028797018963968 + (1291505690487877 * sin(t1) * sin(t2))/28823037615171174400 + (3 * cos(t3) * (cos(t1) * cos(t2) + (7514214926474921 * sin(t1) * sin(t2))/36893488147419103232))/50 - (8106479254538167 * cos(t3) * (cos(t1) * sin(t2) - (7514214926474921 * cos(t2) * sin(t1))/36893488147419103232))/45035996273704960 + (46963842316373 * sin(t4) * (cos(t3) * (cos(t1) * cos(t2) + (7514214926474921 * sin(t1) * sin(t2))/36893488147419103232) - sin(t3) * (cos(t1) * sin(t2) - (7514214926474921 * cos(t2) * sin(t1))/36893488147419103232)))/4611686018427387904 - (8106479254538167 * sin(t3) * (cos(t1) * cos(t2) + (7514214926474921 * sin(t1) * sin(t2))/36893488147419103232))/45035996273704960 - (3 * sin(t3) * (cos(t1) * sin(t2) - (7514214926474921 * cos(t2) * sin(t1))/36893488147419103232))/50 - (46963842316373 * cos(t4) * ((4503599440548691 * sin(t1))/4503599627370496 + (7514214926474921 * cos(t3) * (cos(t1) * sin(t2) - (7514214926474921 * cos(t2) * sin(t1))/36893488147419103232))/36893488147419103232 + (7514214926474921 * sin(t3) * (cos(t1) * cos(t2) + (7514214926474921 * sin(t1) * sin(t2))/36893488147419103232))/4611686018427387904 + (1801439813583837 * sin(t5) * (cos(t4) * (cos(t3) * (cos(t1) * cos(t2) + (7514214926474921 * sin(t1) * sin(t2))/36893488147419103232) - sin(t3) * (cos(t1) * sin(t2) - (7514214926474921 * cos(t2) * sin(t1))/36893488147419103232)) + sin(t4) * ((4503599440548691 * sin(t1))/4503599627370496 + (7514214926474921 * cos(t3) * (cos(t1) * sin(t2) - (7514214926474921 * cos(t2) * sin(t1))/36893488147419103232))/36893488147419103232 + (7514214926474921 * sin(t3) * (cos(t1) * cos(t2) + (7514214926474921 * sin(t1) * sin(t2))/36893488147419103232))/36893488147419103232)))/36028797018963968$$

$$y = (11 * cos(t2) * sin(t1))/50 - (1291505690487877 * cos(t1) * sin(t2))/28823037615171174400 - (676279337148533 * cos(t1))/18446744073709551616 - (8106479254538167 * cos(t3) * ((7514214926474921 * cos(t1) * cos(t2))/36893488147419103232 + sin(t1) * sin(t2)))/45035996273704960 - (3 * cos(t3) * ((7514214926474921 * cos(t1) * sin(t2))/36893488147419103232 - cos(t2) * sin(t1)))/50 - (46963842316373 * sin(t4) * (cos(t3) * ((7514214926474921 * cos(t1) * sin(t2))/36893488147419103232 - cos(t2) * sin(t1)) + sin(t3) * ((7514214926474921 * cos(t1) * cos(t2))/36893488147419103232 + sin(t1) * sin(t2))))/4611686018427387904 - (3 * sin(t3) * ((7514214926474921 * cos(t1) * cos(t2))/36893488147419103232 + sin(t1) * sin(t2)))/50 + (8106479254538167 * sin(t3) * ((7514214926474921 * cos(t1) * sin(t2))/36893488147419103232 - cos(t2) * sin(t1)))/45035996273704960 + (46963842316373 * cos(t4) * ((4503599440548691 * cos(t1))/4503599627370496 - (7514214926474921 * cos(t3) * ((7514214926474921 * cos(t1) * cos(t2))/36893488147419103232 + sin(t1) * sin(t2)))/36893488147419103232 + (7514214926474921 * sin(t3) * ((7514214926474921 * cos(t1) * sin(t2))/36893488147419103232 - cos(t2) * sin(t1)))/36893488147419103232))/4611686018427387904 - (1801439813583837 * sin(t5) * (cos(t4) * (cos(t3) * ((7514214926474921 * cos(t1) * sin(t2))/36893488147419103232 - cos(t2) * sin(t1)) + sin(t3) * ((7514214926474921 * cos(t1) * cos(t2))/36893488147419103232 + sin(t1) * sin(t2))) + sin(t4) * ((4503599440548691 * cos(t1))/4503599627370496 - (7514214926474921 * cos(t3) * ((7514214926474921 * cos(t1) * cos(t2))/36893488147419103232 + sin(t1) * sin(t2)))/36893488147419103232 + (7514214926474921 * sin(t3) * ((7514214926474921 * cos(t1) * sin(t2))/36893488147419103232 - cos(t2) * sin(t1)))/36893488147419103232))/36028797018963968 - (1801439813583837 * cos(t5) * ((3757107307382221 * cos(t1))/18446744073709551616 + (4503599440548691 * cos(t3) * ((7514214926474921 * cos(t1) * cos(t2))/36893488147419103232 + sin(t1) * sin(t2)))/4503599627370496 - (7514214926474921 * sin(t4) * (cos(t3) * ((7514214926474921 * cos(t1) * sin(t2))/36893488147419103232 - cos(t2) * sin(t1)) + sin(t3) * ((7514214926474921 * cos(t1) * cos(t2))/36893488147419103232 + sin(t1) * sin(t2))))/36893488147419103232 - (4503599440548691 * sin(t3) * ((7514214926474921 * cos(t1) * sin(t2))/36893488147419103232 - cos(t2) * sin(t1)))/4503599627370496 + (7514214926474921 * cos(t4) * ((4503599440548691 * cos(t1))/4503599627370496 - (7514214926474921 * cos(t3) * ((7514214926474921 * cos(t1) * cos(t2))/36893488147419103232 + sin(t1) * sin(t2)))/36893488147419103232 + (7514214926474921 * sin(t3) * ((7514214926474921 * cos(t1) * sin(t2))/36893488147419103232 - cos(t2) * sin(t1)))/36893488147419103232))/36028797018963968$$

$$
\begin{aligned}
z = {} & (154811233979861 * sin(t2))/703687441776640 + (3242591634559417 * cos(t2) \\
& * cos(t3))/18014398509481984 + (5404319440751511 * cos(t2) \\
& * sin(t3))/90071992547409920 + (5404319440751511 * cos(t3) \\
& * sin(t2))/90071992547409920 + (46963842316373 * cos(t4) * ((234819211581865 \\
& * cos(t2) * cos(t3))/1152921504606846976 - (234819211581865 * sin(t2) \\
& * sin(t3))/1152921504606846976 \\
& + 234819211581865/1152921504606846976))/4611686018427387904 \\
& - (3242591634559417 * sin(t2) * sin(t3))/18014398509481984 - (1801439813583837 \\
& * sin(t5) * (sin(t4) * ((234819211581865 * cos(t2) * cos(t3))/1152921504606846976 \\
& - (234819211581865 * sin(t2) * sin(t3))/1152921504606846976 \\
& + 234819211581865/1152921504606846976) - cos(t4) * ((9007199067919185 * cos(t2) \\
& * sin(t3))/9007199254740992 + (9007199067919185 * cos(t3) \\
& * sin(t2))/9007199254740992)))/36028797018963968 - (1801439813583837 * cos(t5) \\
& * ((7514214926474921 * cos(t4) * ((234819211581865 * cos(t2) \\
& * cos(t3))/1152921504606846976 - (234819211581865 * sin(t2) \\
& * sin(t3))/1152921504606846976 \\
& + 234819211581865/1152921504606846976))/36893488147419103232 \\
& - (9007198694275583 * cos(t2) * cos(t3))/9007199254740992 + (9007198694275583 \\
& * sin(t2) * sin(t3))/9007199254740992 + (7514214926474921 * sin(t4) \\
& * ((9007199067919185 * cos(t2) * sin(t3))/9007199254740992 + (9007199067919185 \\
& * cos(t3) * sin(t2))/9007199254740992))/36893488147419103232 \\
& + 783587428139897/18889465931478580854784))/36028797018963968 \\
& + (46963842316373 * sin(t4) * ((9007199067919185 * cos(t2) \\
& * sin(t3))/9007199254740992 + (9007199067919185 * cos(t3) \\
& * sin(t2))/9007199254740992))/4611686018427387904 \\
& + 2846274695474751/36028797018963968
\end{aligned}
$$

The variables $t1$, $t2$, $t3$, $t4$, $t5$, and $t6$ denote joint angles $\theta_1, \ldots, \theta_6$ respectively.

Now, one must take partial derivatives of the right-hand side column of the resulting transformation matrix ${}^0T_6$, with each of the rows denoting x, y, and z positions with respect to each of the six joint angles. This operation results in the needed J_v matrix which relates to the linear velocities of the end-effector. As per equation (6), the J_v matrix is defined as:

$$
J_v = \begin{bmatrix}
\frac{\partial \mathrm{x}}{\partial \theta_1} & \frac{\partial \mathrm{x}}{\partial \theta_2} & \frac{\partial \mathrm{x}}{\partial \theta_3} & \frac{\partial \mathrm{x}}{\partial \theta_4} & \frac{\partial \mathrm{x}}{\partial \theta_5} & \frac{\partial \mathrm{x}}{\partial \theta_6} \\
\frac{\partial \mathrm{y}}{\partial \theta_1} & \frac{\partial \mathrm{y}}{\partial \theta_2} & \frac{\partial \mathrm{y}}{\partial \theta_3} & \frac{\partial \mathrm{y}}{\partial \theta_4} & \frac{\partial \mathrm{y}}{\partial \theta_5} & \frac{\partial \mathrm{y}}{\partial \theta_6} \\
\frac{\partial \mathrm{z}}{\partial \theta_1} & \frac{\partial \mathrm{z}}{\partial \theta_2} & \frac{\partial \mathrm{z}}{\partial \theta_3} & \frac{\partial \mathrm{z}}{\partial \theta_4} & \frac{\partial \mathrm{z}}{\partial \theta_5} & \frac{\partial \mathrm{z}}{\partial \theta_6}
\end{bmatrix}
$$

To proceed with constructing the Jacobian, next one must find the J_ω matrix that relates the angular velocities to the end-effector. The values $\widehat{\omega}_1, \ldots, \widehat{\omega}_6$ can be found easily from the transformation matrices defined earlier. The third column of these matrices is the corresponding value for $\widehat{\omega}_\mathrm{n}$ since the axis of rotation for each joint is around the z-axis in our Denavit-Hartenberg diagram (Fig. 35), as per the rules of the Denavit-Hartenberg convention:

$$\widehat{\omega}_1 = \begin{bmatrix} \sin(\theta_1)\sin(1.571) \\ -\cos(\theta_1)\sin(1.571) \\ \cos(1.571) \end{bmatrix} = \begin{bmatrix} \sin(\theta_1) \\ -\cos(\theta_1) \\ 0 \end{bmatrix}$$

$$\widehat{\omega}_2 = \begin{bmatrix} \sin(\theta_2)\sin(0) \\ -\cos(\theta_2)\sin(0) \\ \cos(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\widehat{\omega}_3 = \begin{bmatrix} \sin(\theta_3)\sin(-1.571) \\ -\cos(\theta_3)\sin(-1.571) \\ \cos(-1.571) \end{bmatrix} = \begin{bmatrix} -\sin(\theta_3) \\ \cos(\theta_3) \\ 0 \end{bmatrix}$$

$$\widehat{\omega}_4 = \begin{bmatrix} \sin(\theta_4)\sin(1.571) \\ -\cos(\theta_4)\sin(1.571) \\ \cos(1.571) \end{bmatrix} = \begin{bmatrix} \sin(\theta_4) \\ -\cos(\theta_4) \\ 0 \end{bmatrix}$$

$$\widehat{\omega}_5 = \begin{bmatrix} \sin(\theta_5)\sin(-1.571) \\ -\cos(\theta_5)\sin(-1.571) \\ \cos(-1.571) \end{bmatrix} = \begin{bmatrix} -\sin(\theta_5) \\ \cos(\theta_5) \\ 0 \end{bmatrix}$$

$$\widehat{\omega}_6 = \begin{bmatrix} \sin(\theta_6)\sin(3.140) \\ -\cos(\theta_6)\sin(3.140) \\ \cos(3.140) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

Therefore:

$$J_\omega = \begin{bmatrix} \sin(\theta_1) & 0 & -\sin(\theta_3) & \sin(\theta_4) & -\sin(\theta_5) & 0 \\ -\cos(\theta_1) & 0 & \cos(\theta_3) & -\cos(\theta_4) & \cos(\theta_5) & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

Finally, one can write the Jacobian matrix by combining J_v and J_ω into a single matrix:

$$J = \begin{bmatrix} \mathrm{J_v} \\ \mathrm{J_\omega} \end{bmatrix}$$

Plugging in J_v and J_ω:

$$J = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} & \frac{\partial x}{\partial \theta_4} & \frac{\partial x}{\partial \theta_5} & \frac{\partial x}{\partial \theta_6} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} & \frac{\partial y}{\partial \theta_4} & \frac{\partial y}{\partial \theta_5} & \frac{\partial y}{\partial \theta_6} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_3} & \frac{\partial z}{\partial \theta_4} & \frac{\partial z}{\partial \theta_5} & \frac{\partial z}{\partial \theta_6} \\ \sin(\theta_1) & 0 & -\sin(\theta_3) & \sin(\theta_4) & -\sin(\theta_5) & 0 \\ -\cos(\theta_1) & 0 & \cos(\theta_3) & -\cos(\theta_4) & \cos(\theta_5) & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

Plugging in the Jacobian matrix to the equation (8):

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} & \frac{\partial x}{\partial \theta_4} & \frac{\partial x}{\partial \theta_5} & \frac{\partial x}{\partial \theta_6} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} & \frac{\partial y}{\partial \theta_4} & \frac{\partial y}{\partial \theta_5} & \frac{\partial y}{\partial \theta_6} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_3} & \frac{\partial z}{\partial \theta_4} & \frac{\partial z}{\partial \theta_5} & \frac{\partial z}{\partial \theta_6} \\ \sin(\theta_1) & 0 & -\sin(\theta_3) & \sin(\theta_4) & -\sin(\theta_5) & 0 \\ -\cos(\theta_1) & 0 & \cos(\theta_3) & -\cos(\theta_4) & \cos(\theta_5) & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \end{bmatrix}$$

4 Results

4.1 Implementation of Ned2

The robot was successfully installed and implemented into the laboratory of Arcada University of Applied Sciences. A designated working area has been allocated for the robot at the laboratory, and the robot can only be used by authorized personnel due to the fact that the computer that is used as the terminal for controlling the robot is protected by a password. This ensures that only trained staff and students are able to operate the robot. In addition to this, an additional safety measure is the fact that the laboratory can be only accessed by academic staff and students with permission to enter.

There is a sufficient safety zone around the robot so that it will not come in contact with unrelated objects or people while in operation. The desk on which the robot is placed on is located in the corner of the laboratory, away from other working areas and the main corridor. The Conveyor Belt Set has enough space within the working area despite its size, and there is enough overhead light for the camera of the Vision Set to work properly in order to detect objects, their colors, and to find the landmarks of the workspace. The maximum reach of the robot is demonstrated in Figure 36., which shows that there is enough space between the operator and Ned2.

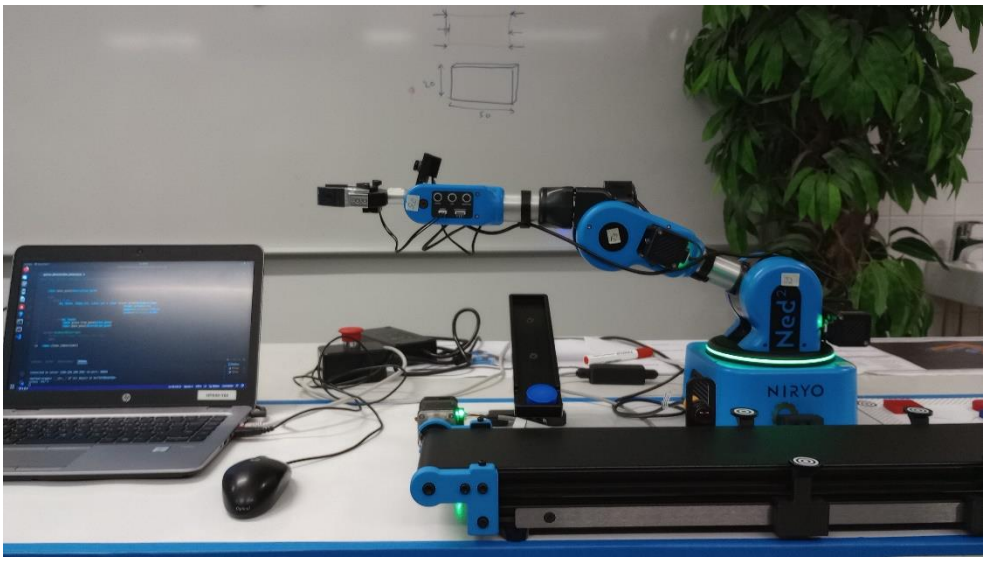

Figure 36. Ned2 and its maximum reach

4.2 Case Studies

The manual pick-and-place program yielded differing results depending on the number of repeats and if the defined parameters were followed correctly by the operator. Since this program cannot adjust to unexpected changes while it is being run, such as the position of the object to be picked, or the distance of the robot from the conveyor belt, it is highly unreliable in a real-life setting if the location of the object is not strictly controlled and kept constant. The IR sensor worked as expected when detecting approaching objects on the conveyor belt, and the only problems were caused by objects or people moving or staying close to the sensor while in operation, which led to a number of false detections.

The success of this program depended highly on the accuracy of the operator. Since the object had to be placed in a specific location on the conveyor belt, sometimes it would not be able to grasp the object, especially if it was rectangular. The tooltip is slightly wider than the rectangular object when it is opened for picking up objects, and therefore, sometimes the robot would collide with the edges of the object over the conveyor belt. A problem also arose if the conveyor belt was not completely parallel with the robot, which causes an offset every time the robot places the object back on the conveyor belt. This also led to the same scenario of the tooltip of the robot colliding with the object instead of successfully grasping it.

Since the robot used object recognition to pick-and-place objects from the workspace for the vision pick-and-place, this program proved to be more dynamic and applicable for real-life processes with an uncertain and changing environment than the manual pick-and-place program. It was found that the parameters of the camera were a large factor in the success of the vision pick function, as well as the lighting in the surrounding environment. Having sufficient lighting around the workspace was needed for more accurate picks so that the colors and contours of the objects were well-shown for the camera. The lighting and camera parameters also affected how the landmarks of the workspace were perceived.

The simulation with RViz and Gazebo showed great promise for the development of more advanced simulations without having to use the actual robot itself, which in turn can be an attractive alternative for the research of robots. Gazebo turned out to have a number of applications outside of the simulation of articulated robots, especially for mobile and even

space robots research, since it is possible to create a complex virtual environment, such as a building with multiple floors and objects scattered about.

4.3 Kinematic Studies

The Denavit-Hartenberg convention for selected joint angles J_A was implemented for the Ned2 six-axis robot. The real position of the end-effector according to the given joint angles J_A can be found by using Niryo Studio, as seen on Figure 37.

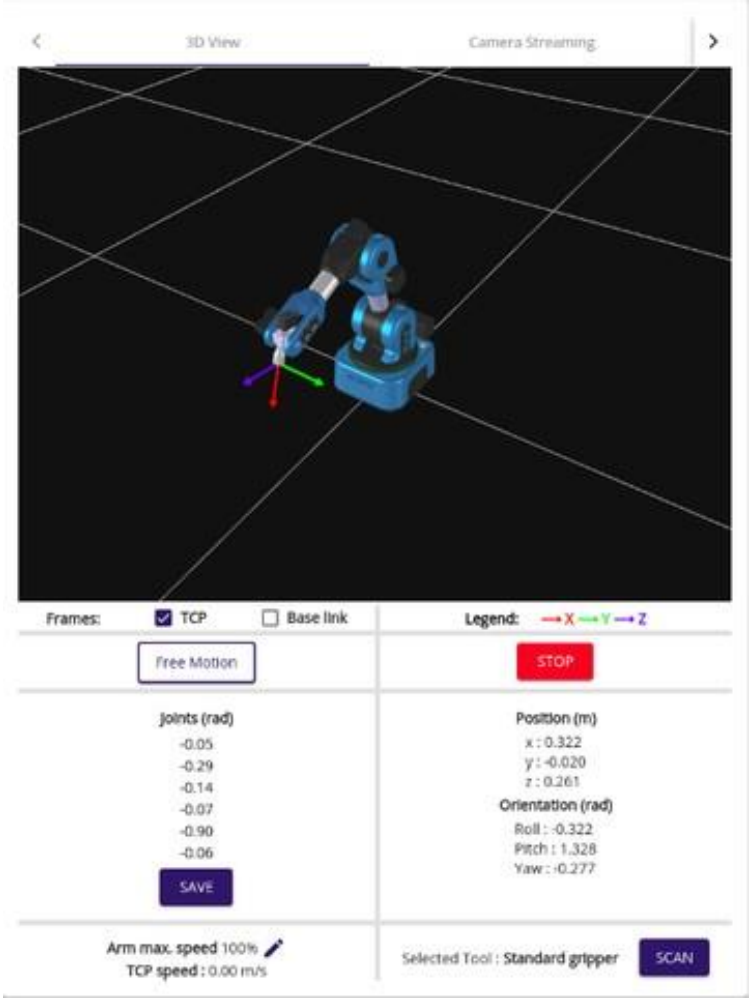

Figure 37. Using Niryo Studio to verify the values obtained from the Denavit-Hartenberg matrix

Niryo Studio regards the flat surface the robot is placed on as the $z = 0\ m$ position, which means that it is required to subtract the height from the base of the robot up to the first joint in order to get the z-value with the first joint as the origin. The height is measured to be around $0.095\ m$, so the z-position of the end-effector with the value from Niryo Studio is as follows:

$$z_{end-effector} = 0.261 - 0.095 = 0.166\, m$$

The values of Niryo Studio and the result obtained with Denavit-Hartenberg convention, namely the values of equation (16), are compared in Table 5. The error is a percent error where the values from Niryo Studio are regarded as true, while the Denavit-Hartenberg convention values are regarded as experimental, using the following formula:

$$\%\, Error = \frac{|Experimental\ Value - True\ Value|}{|True\ Value|} * 100\%$$

Table 5. Comparison of position values between Niryo Studio and the values obtained with the Denavit-Hartenberg convention

	$x_{end-effector}\ (m)$	$y_{end-effector}\ (m)$	$z_{end-effector}\ (m)$
Niryo Studio	0.322	-0.020	0.166
Denavit-Hartenberg method	0.320	-0.019	0.200
Error (%)	0.621	5.000	20.482

The speculated cause for the slightly different results is the fact that the actual link lengths of Ned2 were not used, and the Denavit-Hartenberg diagram was largely simplified compared to the anatomy of the real robot.

The values of the Jacobian matrix J_v for the linear velocities of the end-effector could not be shown on this thesis due to the values being incredibly long after obtaining the partial derivates for the Jacobian; the six joint angles were variables. Using this method for 6-axis robots demands the use of a numeric computing environment such as MATLAB, which was the primary tool in order to determine the Jacobian. It is also possible to use software such as Microsoft Excel or other spreadsheet software.

5 Discussion

Since the Ned2 robot is small in size and relatively light, it does not pose a particular risk to its environment or people while in operation. In addition to this, the robot is designed to be a cobot, which means that the sensors it is equipped with should provide adequate protection to itself and the surroundings. Despite this, it is still important to follow the safety measures at all times which were defined in the method of this thesis. Additional protection would include a physical barrier between the environment and the robot, but this is not relevant in the case of Ned2.

Some of the features of Gazebo deemed to be more geared towards mobile robots compared to articulated robots such as Ned2. Most of the features of Gazebo can still be of great use for different types of robots in general, and it is possible to simulate and create a wide range of scenarios relevant for different purposes, which makes the tool very versatile and useful for robotics.

The results of the Denavit-Hartenberg method for the determination of the end-effector location could be further improved and perfected by using the real values of the link lengths instead. However, the measured link lengths seem sufficient as the error is not very significant compared to the actual position of the end-effector displayed by Niryo Studio. Errors are probable with different joint angle values due to the simplified geometry of the Denavit-Hartenberg diagram. The link lengths were determined using the RViz simulation by using the "Measure" function of the software to measure them manually and approximating the obtained values. In addition to this, it is good to note that the length of the tool was disregarded and not added to the transformation matrix ${}^{5}T_{6}$ since Ned2 does not use it to determine the position of the end-effector.

6 Conclusion

This thesis demonstrated how a small articulated robot can be implemented to a university laboratory environment, as well as a number of ways that can be utilized in order to control and analyze modern robots. The objectives of the thesis were successfully investigated through practical and computational methods.

The implementation of Ned2 to an educational institution was easily accomplished since the needed resources were already available to be used. For example, the laboratory of Arcada University of Applied Sciences had already enough space for the workspace of the robot, as well as a suitable working surface with enough space for the robot, and the goals regarding safety were easily implemented. This process may not be so easy when resources and space are limited.

Programming the Ned2 robot through PyNiryo was intuitive due to the nature of the Python programming language, as well as the useful functions provided by the PyNiryo package. The first pick-and-place demonstration which was manually programmed proved to need some improvement and its weakness was found to be the lack of adaptability to uncertainties and changes. The vision pick-and-place demonstration was effortless to program with the use of PyNiryo, and it worked very well as long as the parameters of the camera were adjusted to improve the object detection and the environment of the robot was well-lit.

The Denavit-Hartenberg transformation matrices were obtained from forward kinematics for each six joints, and then they were used to calculate the position of the end-effector. It was noted that the utilization of the Denavit-Hartenberg convention could use some improvement and, in some cases with different joint angles, it may fail to represent the actual position of the end-effector. The reason for this was speculated to be the fact that the diagram was oversimplified and the measurements of the link lengths of Ned2 were inaccurate. The Jacobian matrix, using the Denavit-Hartenberg diagram and consequently the transformation matrices, was also determined successfully.

7 References

A Tribute to Joseph Engelberger - Father of Robotics. (no date). *Unimate - The First Industrial Robot*. [online] Available at: https://www.automate.org/a3-content/joseph-engelberger-unimate [Accessed 15 Oct. 2022].

ABB (no date). *ABB IRB 8700*. [Online image] *ABB*. Available at: https://new.abb.com/products/3HAC020536-024/irb-8700 [Accessed 26 Oct. 2022].

ABB (2022). *Product specification IRB 8700*. *ABB Library*. Available at: https://search.abb.com/library/Download.aspx?DocumentID=3HAC052852-001&LanguageCode=en&DocumentPartId=&Action=Launch [Accessed 26 Oct. 2022].

Anaheim Automation (2021). *Incremental Optical Encoder*. [Online image] *Anaheim Automation*. Available at: https://www.anaheimautomation.com/manuals/forms/encoder-guide.php [Accessed 23 Oct. 2022].

Appin Knowledge Solutions (2007a). *Representation of joint angle.* [Book print] *Robotics*.

Appin Knowledge Solutions (2007b). *Robotics*. Massachusetts, USA: Jones & Bartlett Learning, pp.3–4, 100, 222, 225–227, 244–245.

AspenCore (no date). *Hall Effect Sensor Principles*. *Electronics Tutorials*. Available at: https://www.electronics-tutorials.ws/electromagnetism/hall-effect.html [Accessed 18 Oct. 2022].

Bergren, C.M. (2003). *Anatomy of a Robot*. New York: Mcgraw-Hill, pp.22–24, 26–27, 275–278.

Bouman, A., Ginting, M.F., Alatur, N., Palieri, M., Fan, D.D., Touma, T., Pailevanian, T., Kim, S.-K., Otsu, K., Burdick, J. and Agha-Mohammadi, A. (2020). Autonomous Spot: Long-Range Autonomous Exploration of Extreme Environments with Legged Locomotion. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [online] doi:https://doi.org/10.1109/iros45743.2020.9341361.

Carbonnelle, P. (2022). *PYPL Popularity of Programming Language index*. [online] PYPL PopularitY of Programming Language. Available at: https://pypl.github.io/PYPL.html [Accessed 9 Nov. 2022].

Circuit Globe (2022a). *Closed loop control system*. [Online image] *Circuit Globe*. Available at: https://circuitglobe.com/difference-between-open-loop-and-closed-loop-system.html [Accessed 3 Nov. 2022].

Circuit Globe (2022b). *Open Loop Control System*. [Online image] *Circuit Globe*. Available at: https://circuitglobe.com/difference-between-open-loop-and-closed-loop-system.html [Accessed 31 Oct. 2022].

Kawasaki Heavy Industries, Ltd. (2018a). *Comparison between robot and human movement*. [Online image] *XYZ*. Available at: https://robotics.kawasaki.com/ja1/xyz/en/1804-03/ [Accessed 20 Oct. 2022].

Computer History Museum (no date). *Elements of Bardeen and Brattain's Transistor*. [Online image] *Computer History Museum*. Available at: https://www.computerhistory.org/revolution/digital-logic/12/273/2237 [Accessed 23 Oct. 2022].

Craig, J. (2018). *Introduction to robotics : mechanics and control*. 4th ed. New York, New York: Pearson, pp.1, 4, 5, 7, 9–10, 22–23, 28, 49, 67, 178–180, 184.

curl.se. (no date). *curl - How To Use*. [online] Available at: https://curl.se/docs/manpage.html [Accessed 31 Jan. 2023].

Denavit, J. and Hartenberg, R.S. (1955). A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. *Journal of Applied Mechanics*, [online] 22(2), p.215. doi:https://doi.org/10.1115/1.4011045.

Eck, D.J. (2022). *Javanotes 9, Section 1.3 -- The Java Virtual Machine*. [online] Introduction to Programming Using Java. Available at: https://math.hws.edu/javanotes/c1/s3.html [Accessed 9 Nov. 2022].

Electrical Technology (no date). *AC Motor*. *Electrical Technology*. Available at: https://www.electricaltechnology.org/2020/06/difference-between-ac-dc-motor.html [Accessed 15 Oct. 2022].

Engineering ToolBox (2010). *Gear Reducing Formulas*. [online] Engineering ToolBox. Available at: https://www.engineeringtoolbox.com/gear-output-torque-speed-horsepower-d_1691.html [Accessed 30 Oct. 2022].

Gilchrist, A. (2016). *Industry 4.0*. California, USA: Apress Media, pp.1, 3–5.

Google Developers (no date). *Blockly | Google Developers*. [online] Google Developers. Available at: https://developers.google.com/blockly [Accessed 8 Jan. 2023].

H., S.C. (no date). *Object Oriented Programming · HonKit*. [online] A Byte of Python. Available at: https://python.swaroopch.com/oop.html [Accessed 9 Nov. 2022].

Hans Peter Moravec (2018). Robot | technology. In: *Encyclopædia Britannica*. [online] Available at: https://www.britannica.com/technology/robot-technology [Accessed 13 Oct. 2022].

Hemmendinger, D. (2022a). *Machine language | computing*. [online] Encyclopedia Britannica. Available at: https://www.britannica.com/technology/machine-language [Accessed 9 Nov. 2022].

Hemmendinger, D. (2022b). Object-oriented programming | computer science. In: *Encyclopædia Britannica*. [online] Available at:

https://www.britannica.com/technology/object-oriented-programming [Accessed 9 Nov. 2022].

Honda Motor Co., Ltd. (no date). *Honda Global | ASIMO*. [online] Honda. Available at: https://global.honda/innovation/robotics/ASIMO.html [Accessed 30 Oct. 2022].

Honda Motor Co., Ltd. (no date). *Two ASIMO robots*. [Online image] *Honda*. Available at: https://global.honda/innovation/robotics/ASIMO.html [Accessed 30 Oct. 2022].

Hughes, A. (2006). *Electric Motors and Drives: Fundamentals, Types, and Applications*. Amsterdam; Boston: Elsevier/Newnes, pp.159–162.

IFR (2020). *IFR presents World Robotics Report 2020*. [online] IFR International Federation of Robotics. Available at: https://ifr.org/ifr-press-releases/news/record-2.7-million-robots-work-in-factories-around-the-globe [Accessed 15 Oct. 2022].

Infosec Institute (2014). *How Java code is interpreted and transformed to machine code*. [Online image] *Infosec Institute*. Available at: https://resources.infosecinstitute.com/topic/java-bytecode-reverse-engineering/ [Accessed 9 Nov. 2022].

Kawasaki Heavy Industries, Ltd. (2018a). *Comparison between robot and human movement*. [Online image] *XYZ*. Available at: https://robotics.kawasaki.com/ja1/xyz/en/1804-03/ [Accessed 20 Oct. 2022].

Kawasaki Heavy Industries, Ltd. (2018b). *How Are Industrial Robots Built? A Guide on the Components and the Movement of Robot Arms| XYZ | Kawasaki Heavy Industries, Ltd*. [online] XYZ. Available at: https://robotics.kawasaki.com/ja1/xyz/en/1804-03/ [Accessed 20 Oct. 2022].

Kiusalaas, J. (2005). *Numerical methods in engineering with Python*. New York: Cambridge [Etc.]: Cambridge University Press, pp.1–2.

Krishna, S. (no date). *Jacobian | ROS Robotics*. [online] Home. Available at: https://www.rosroboticslearning.com/jacobian [Accessed 27 Feb. 2023].

KUKA AG (2022a). *LBR iiwa*. [online] KUKA AG. Available at: https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa [Accessed 30 Oct. 2022].

KUKA AG (2022b). *LBR iiwa collaborative industrial robot*. [Online image] *KUKA*. Available at: https://www.kuka.com/en-us/products/robotics-systems/industrial-robots/lbr-iiwa [Accessed 30 Oct. 2022].

Malone, B. (no date). *Unimate Industrial Robot in Action*. [Online image] *IEEE Spectrum*. Available at: https://spectrum.ieee.org/george-devol-a-life-devoted-to-invention-and-robots [Accessed 23 Oct. 2022].

NASA and JPL-Caltech (no date). *The Autonomous Spot Robot*. [Online image] *JPL-Caltech*. Available at: https://www.jpl.nasa.gov/robotics-at-jpl/nebula-spot [Accessed 23 Oct. 2022].

Niryo (2021). *How the Vision Set works — Vision Set User Manual v1.0.0 documentation*. [online] Niryo Docs. Available at: https://docs.niryo.com/product/vision-set/v1.0.0/en/source/how_it_works.html [Accessed 21 Feb. 2023].

Niryo (2022a). *Robot operating mode — Ned2 User Manual v1.0.0 documentation*. [online] Niryo Docs. Available at: https://docs.niryo.com/product/ned2/v1.0.0/en/source/ [Accessed 8 Jan. 2023].

Niryo (2022b). *The back panel interface*. [Online image] *Niryo Docs*. Available at: https://docs.niryo.com/product/ned2/v1.0.0/en/source/hardware/electrical_interface.html [Accessed 2 Feb. 2023].

Niryo (2023a). *Ned2 : robotic arm for education*. [online] Niryo. Available at: https://niryo.com/product/ned-2-education-robotics-arm/ [Accessed 8 Jan. 2023].

Niryo (2023b). *pyniryo: Package to control Niryo Robot 'Ned' through TCP*. [online] PyPI. Available at: https://pypi.org/project/pyniryo/ [Accessed 30 Jan. 2023].

Schwab, K. (2018). The Fourth Industrial Revolution | Special Feature. In: *Encyclopædia Britannica*. [online] Available at: https://www.britannica.com/topic/The-Fourth-Industrial-Revolution-2119734 [Accessed 18 Oct. 2022].

Swedin, E.G. and Ferro, D.L. (2022). *The Computer: A Brief History of the Machine That Changed the World*. California, USA: Greenwood, pp.1, 14–18, 54–56.

Weisstein, E.W. (2022). *Jacobian*. [online] Wolfram MathWorld. Available at: https://mathworld.wolfram.com/Jacobian.html [Accessed 2 Dec. 2022].

Wikipedia Contributors (2022). *Denavit–Hartenberg parameters*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters#Denavit%E2%80%93Hartenberg_matrix [Accessed 17 Feb. 2023].