BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

ROBOTIC ARM WITH RC COMPONENTS AND SERVOS

ROBOTICKÁ RUKA S VYUŽITÍM RC KOMPONENTŮ A SERV

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. PETR BOBČÍK**
AUTOR PRÁCE

SUPERVISOR **prof. Dr. Ing. PAVEL ZEMČÍK**
VEDOUCÍ PRÁCE

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

147199

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Bobčík Petr, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Vestavěné systémy
Název:	**Robotická ruka s využitím RC komponentů a serv**
Kategorie:	Vestavěné systémy
Akademický rok:	2022/23

Zadání:

1. Prostudujte způsob řízení modelářských serv a možnosti jejich ovládání počítačem. Dále prostudujte existující konstrukce "robotických" zařízení s využitím RC komponentů.
2. Navrhněte jednoduchý model robotické ruky sestrojený s využitím modelářských RC komponentů a případně dalších dostupných "off the shelf" komponentů, případně 3D tisku tak, aby byl snadno opakovatelně sestrojitelný.
3. Navrhněte konstrukci/implementaci ruky a způsob ovládání s ohledem na to, aby získala využitím počítače nové lepší vlastnosti.
4. Popište možnosti konstrukce, ovládání i automatizace provozu a diskutujte dosažitelné vlastnosti.
5. Navržený model implementujte a demonstrujte jeho vlastnosti na vhodné úloze.
6. Diskutujte dosažené výsledky a možnosti pokračování práce.

Literatura:

- Dle pokynů vedoucího

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Zemčík Pavel, prof. Dr. Ing., dr. h. c.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	17.5.2023
Datum schválení:	31.10.2022

Assignment:

1.) Study the way of controlling modeling servos and the possibilities of their computer control. Next, study existing designs of "robotic" devices using RC components.

2.) Design a simple model of a robotic hand built using modeling RC components and possibly other available "off the shelf" components, or 3D printing so that it can be easily built repeatedly.

3.) Design the design/implementation of the hand and the way of control, considering that it will acquire new and better properties through the use of the computer.

4.) Describe design, control, and automation options and discuss achievable features / abilities.

5.) Implement the designed model and demonstrate its abilities on an appropriate task.

6.) Discuss the results achieved and options for continuing the work.

Abstract

The goal of this work was to create own robotic arm using RC components, servos and provide own graphical user interface to control it. I decided that the solution should use sensors that provides some kind of autonomous beahviour. For my solution I modified existing stepper motor based robotic arm with five degrees of freedom. As a sensors, the accelerometer, encoders, current sensor, laser distance sensor and camera were used. Thanks to these sensors, the robotic arm is able to detect stall, position of disconnected stepper motors, grabbing of an object or measure distance to the object to compute its position in a space. My solution offers own graphical user interface that allows to control each joint separately, autonomous controlling using camera or hand driven controlling.

Abstrakt

Cílem této práce bylo postavit vlastní robotické rameno, s využitím RC komponent, serv a dodat k němu i vlastní uživatelské rozhraní, které umožní jeho řízení. Součástí řešení bylo také opatřit robotické rameno potřebnou senzoriku, která by umožnila jistou míru autonomnosti. Pro mé řešení jsem se rozhodl upravil již existující design robotického ramene, s pěti stupni volnosti, založeném na krokových motorech. Přidal jsem senzory, jako je akcelerometr, enkodér, měřič proudu, laserové měření vzdálenosti a kameru. Na základě těchto senzorů je robotické rameno schopné detekovat náraz, pozici odpojených motorů, uchopení předmětu uchopovacím mechanizmem nebo měřit vzdálenost předmětu v prostoru za účelem inverzní kinematiky. Vytvořil jsem také jednoduché uživatelské rozhraní, které umožňuje tři typy ovládání, jako je ovládání jednotlivých kloubů samostatně, autonomně s využitím kamery nebo ručním napozicováním.

Keywords

diploma work, robotic arm, sensors, camera, object recognition, stall detection, 3D print, hand controlled, inverse kinematics, RC componets

Klíčová slova

diplomová práce, robotické rameno, robotický manipulátor, senzory, kamera, rozpoznávání predmětů, detekce nárazu, 3D tisk, ručně řízený, inverzní kinematika, RC komponenty

Reference

BOBČÍK, Petr. *Robotic arm with RC components and servos.* Brno, 2023. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Dr. Ing. Pavel Zemčík

Robotic arm with RC components and servos

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Mr. prof. Dr. Ing. Pavel Zemčík. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Petr Bobčík
May 18, 2023

Acknowledgements

I would like to thank my supervisor Mr. prof. Dr. Ing. Pavel Zemčík for his support, willingness and for patience during this diploma work that he offered. I would like to also thank my family for their support during my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robotic arms are nowadays well-known devices that can be used in many ways across the industry. The main goal of the robotic arm is to manipulate objects that can be heavy for humans, or as independent manipulators on an automated production line. In general, robotic arm aid the human workforce or completely substitute the human workforce in areas where they are needed. Possibilities of usage are vast, from home robots, such as kitchen robots, through industrial robots, such as automated car production to surgery robotic arms.

The main goal of this thesis is to make a robotic arm on my own by using RC components. The robot should manipulate objects, check surroundings and detect objects that are placed around them. Another important feature that this robotic arm should have is collision detection, where the robotic arm should stop when hitting something to prevent injury or to protect itself.

I decided to make my own robotic arm because of my interest in them. Another reason is to make something more complex than what was made in my bachelor thesis. It was a very simple device that was able to move and pick objects, but it had no sensors, no detection of surroundings and the interaction level with the user was very low. That led me to make something more complex.

The structure of this thesis is divided into four main chapters. The Robotic Arm introduction chapter, where the user gains awareness of what the robotic arm is and its motion study. The Components of robotic arm chapter describes existing components, useful for the robotic arm, such as actuators, control boards, etc. Not all components are mentioned, only the components, that are related to this thesis. The Concept of my robotic arm solution chapter forms the concept of my own robotic arm and specifies the requirements for design, hardware, software, user interface and the tests to make sure all goals were fulfilled. The last chapter is the implementation of what was mentioned in the Concept of my robotic arm solution section. The reader will understand how the solution works, how were all goals implemented and if all goals were fulfilled by provided tests.

Chapter 2

Robotic arm introduction

This chapter focuses on the basic theory behind robotic arms, such as the basic description of what the robotic arm is, the motion study behind the robotic arm like forward and inverse kinematics and in the end, it enumerates some existing solutions. This chapter does not serve as an encyclopedic enumeration of all solutions but focuses on solutions related to this thesis.

2.1 What is the robotic arm

The robotic arm is an electro-mechanical device, similar to the human arm, that is very popular, and shows a high sell rate all over the world [31]. It is a very important device used in development. It helps the human or completely substitutes the human to increase efficiency and productivity. It can be used in many different ways, such as assembly lines for building cars, packing boxes, material handling, welding, painting and many other applications.

There are many reasons to use robotic arms instead of humans [31]. For example, the robotic arm can handle heavier objects, move faster, and be more precise and more consistent than humans (it can apply the same forces in every repetition, etc.). Another reason can be price because robotic arms are cheaper to operate than humans and the number of injuries is reduced as well. It can also work in conditions that are not suitable for humans, such as environments with high temperatures.

The robotic arm consists of a set of bodies that are rigidly connected (it is called links), which specifies the robotic arm configurations (position of each joint) [31]. Each robotic arm can be described by a set of different parameters, such as **number of axes**, **degree of freedom**, **working envelop** and **working space** where the robotic arm operates. Other parameters can be kinematics, **payload**, speed, etc.

To further describe the abovementioned [31]:

- **Axis** - Determines how many axes the joint can move- it can be described by the roll, pitch and yaw factors.
- **Degrees of freedom (DOF)** - Determines how many joints the robotic arm has. For example, when a robotic arm has 5 joints, it has 5 degrees of freedom (each joint has one degree of freedom).
- **Working envelop** - The range that can be covered by a robotic arm or by a range of motion.

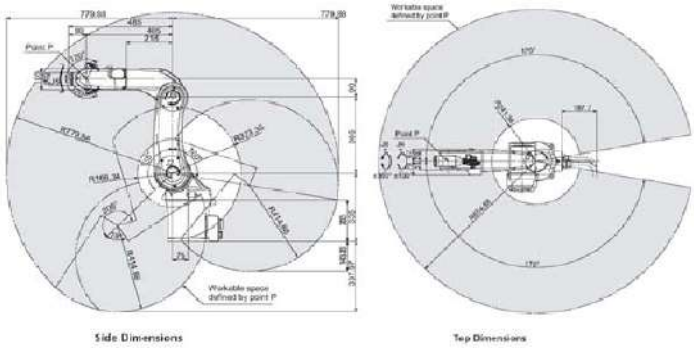

Figure 2.1: Example of the robotic arm envelop[1]

- **Working space** - The area where the robotic arm can participate (the position which can be reached by endpoint effector with gripper), so each configuration of the given robotic arm will stay in this area.
- **Payload** - The weight that the arm can lift and manipulate.

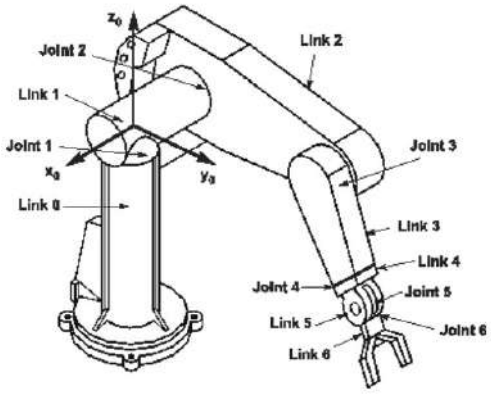

Figure 2.2: Example of robotic arm[2]

Figure 2.2 demonstrates how the robotic arm should look and contains labels to better understand the idea mentioned in this chapter.

Each robotic arm needs at least the following components (the construction itself is not included in this list). **The actuators** that realize the motion of the robotic arm; **a controller** which is a device with a microcontroller that coordinates the motion of the robotic arm and gets data from the environment via sensors; **sensors** provide information from surrounding (environment) in real time - this is almost the same as receptors of a human; the last component is **a power supply** that energize the whole robotic arm's electronics [31].

[1]Downloaded from: https://www.researchgate.net/publication/353478116

[2]Downloaded from: https://medium.com/@sarvagya.vaish/forward-kinematics-using-orocos-kdl-da7035f9c8e

2.2 Motion study of robotic arm

In the beginning, it is necessary to define important keywords used in this section [6]:

- **Link** - right piece of the robot arm that connects two joint
- **Joint** - the connection between links and it allows to rotate or translate with another link
- **Joint Axis** - the axis around which the revolute joint turns or along which it translated
- **Degrees of freedom** - defines the number of dimensions also known as mobility, where the joint is in [34]

Kinematics is the study of motion which does not count on the cause of the motion such as force or torque [34]. The kinematics model consists of segments, connected with joints. These segments and joints are connected in a hierarchic structure. The joints have the ability to make either rotation or translation movement. There are two main types of kinematics. **The inverse kinematics** and **the forward kinematics**.

The inverse kinematics knows the desired endpoint for the end-effector and determines an appropriate joint configuration [34]. When the user calculates the joint angles, it is possible to use the Jacobian matrix to move the end effector from the beginning position to the end target position. Inverse kinematics is more complicated than forward kinematics because when a robotics arm has multiple revolute joints, it generates multiple solutions, not just one [30].

Without inverse kinematics, the programming of the robotic arm (generally a robot) will be very complicated. Two basic options can be used to find inverse kinematics. The first is to do it by yourself and the second is to use an existing solver [30].

The forward kinematics is the right opposite of the inverse kinematics. The inverse kinematics tries to calculate the configuration of all joints according to the end effector. The goal of the forward kinematics is to calculate the position of the target joint (end effector) according to a given joint configuration [34]. Compared to inverse kinematics, forward kinematics has only one possible solution because when forward kinematics gets one joint configuration, it must lead to the same solution [30].

Inverse kinematics

As was mentioned before, inverse kinematics is useful in situations when the target position of the end-effector is known and the goal is to calculate how each joint should be configured (rotated). It can, for example, be solved by numerical methods, such as a **Jacobian inverse method** or by analytical method.

Jacobian - For the numerical method the Jacobian inverse method can be used. The Jacobian is a matrix containing partial derivations of the whole chain system (e.g. all robotic arm joints), relative to the end-effector [21]. All it does is map vector-valued change from joint space $\Delta\theta$ to real physical space (e.g. space of the robotics arm), using the Jacobian matrix. Figure 2.3 illustrates what the Jacobian matrix looks like [3].

[3]Described in detaile: https://nrsyed.com/2017/12/10/inverse-kinematics-using-the-jacobian-inverse-part-2/

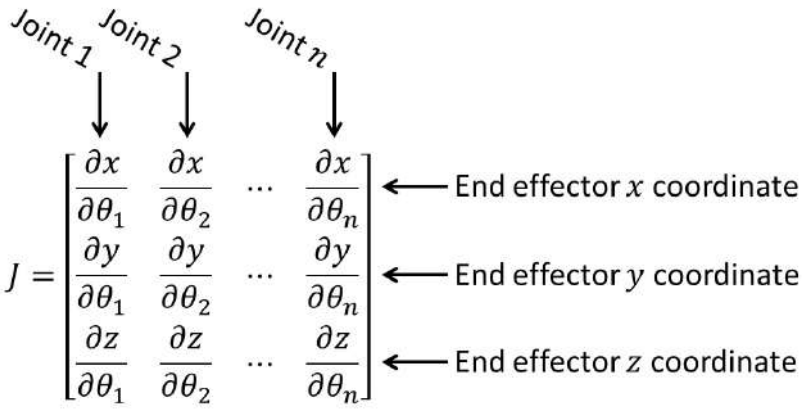

Figure 2.3: Jacobian matrix for n joints[4]

The goal is to calculate how the end-effector moves if only one of the joint angles changes, so the goal is to take a partial derivate of each component. The result says that the final change in each coordinate of the end-effector can be calculated as the following equation shows:

$$\sum_{i=1}^{n}(\frac{\partial coord}{\partial \theta_i} \times \Delta\theta_i)$$

where i represents given joint number, n represents a total number of joints (so we sum through all joints), *coord* stats for change in given coordinate (e.g. the x coordinate) and the θ_i represents the change of position of the joint [39]. This is counted for all three coordinates. Figure 2.4 shows the resulting matrix.

$$\Delta\vec{r} = \begin{Bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{Bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \cdots & \frac{\partial x}{\partial \theta_n} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \cdots & \frac{\partial y}{\partial \theta_n} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \cdots & \frac{\partial z}{\partial \theta_n} \end{bmatrix} \begin{Bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \\ \vdots \\ \Delta\theta_n \end{Bmatrix} = \begin{Bmatrix} \frac{\partial x}{\partial \theta_1}\Delta\theta_1 + \frac{\partial x}{\partial \theta_2}\Delta\theta_2 + \cdots + \frac{\partial x}{\partial \theta_n}\Delta\theta_n \\ \frac{\partial y}{\partial \theta_1}\Delta\theta_1 + \frac{\partial y}{\partial \theta_2}\Delta\theta_2 + \cdots + \frac{\partial y}{\partial \theta_n}\Delta\theta_n \\ \frac{\partial z}{\partial \theta_1}\Delta\theta_1 + \frac{\partial z}{\partial \theta_2}\Delta\theta_2 + \cdots + \frac{\partial z}{\partial \theta_n}\Delta\theta_n \end{Bmatrix}$$

Figure 2.4: Final jacobian matrix for 3D space with n joints[5]

The solution is **linear approximation** of the inverse kinematics problem (linearly model the end-effectors motion).

- **Jacobian Pseudo-inverse** - this method is also known as the Moore-Penrose inverse of the Jacobian. The joint position difference $\Delta\theta$ is counted as

  $$\Delta\theta = J^{\dagger}\overrightarrow{e}$$

  where $J^{\dagger}$ is called pseudo-inverse of J. It provides the best possible solution to the equation $J\Delta\theta = \overrightarrow{e}$ in the least square sense [21].

[4]Downloaded from: https://nrsyed.com/2017/12/10/inverse-kinematics-using-the-jacobian-inverse-part-2/

[5]Downloaded from: https://nrsyed.com/2017/12/10/inverse-kinematics-using-the-jacobian-inverse-part-2/

- **Jacobian Transpose** - first used method for inverse kinematics [21]. The main difference between the Jacobian inverse method and the transpose method is what the Jacobian uses. The Jacobian transpose uses the transpose Jacobian [21] instead of inverse Jacobian in the case of the Jacobian inverse method. So the joint position change $\Delta\theta$ is compute as

$$\Delta\theta = \alpha J^{\top} \overrightarrow{e}$$

for some appropriate scalar α. The α value is chosen to get as precise values as possible to $\overrightarrow{e}$.

The analytical method solves the system at once [34]. is suitable for a certain amount of degree of freedom. When the amount of degrees is high, the analytical solution can produce infinite solutions. On the other hand, when the amount of joints is low, there will be no solution. When the joint parameters and end-effector poses are given, the IK can find all possible solutions. On the other hand, there is **the iterative** method, which does not give a solution at one as the analytical method does but solves the system by approximation in iterations.

Forward kinematics

As was mentioned above the forward kinematics problem is the opposite problem to the inverse kinematics. It means the position (configuration) of each joint is known, but the position of the end-effector is unknown [6].

Each joint gets a coordinate frame to determine Denavit-Hartenberg (DH) parameter. The DH method uses four parameters which fully determine the link itself [25]. The parameters are link length a_{i-1}, link twist α_{i-1}, link offset d_{i-1} and joint angle Θ_{i-1}. The transformation matrix ${}^{i-1}_{i}T$ for one single joint, looks like this:

$${}^{i-1}_{i}T = R_x(\alpha_{i-1})D_x(a_{i-1})R_z(\Theta_i)Q_i(d_i)$$

where

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_{i-1} & -\sin\alpha_{i-1} & 0 \\ 0 & \sin\alpha_{i-1} & \cos\alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} D_x = \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos\Theta_i & -\sin\Theta_i & 0 & 0 \\ \sin\Theta_i & \cos\Theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} Q_i = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The $R_x and R_z$ matrices stand for rotation transformation around the x-axis and z-axis. The D_x and Q_i stand for translation transformation [25]. The final transformation matrix looks like the following matrix shows

$$\begin{bmatrix} \cos\Theta_i & -\sin\Theta_i & 0 & a_{i-1} \\ \sin\Theta_i\cos\alpha_{i-1} & \cos\Theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha i-1 d_1 \\ \sin\Theta_i\sin\alpha_{i-1} & \cos\Theta\sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To determine whole forward kinematics, it is necessary to take all transformation matrices and simply multiply them together.

2.3 Existing robotic arm solutions

This section describes existing robotic arms that are related to this thesis. It focuses on mainly 3D printed robotic arms, so arms can be easily reproduced without any advanced technology. It is not the encyclopedic enumeration of all existing robotic arms, but robotic arms related to this thesis.

Niryo One and Ned2 robotic arms

A Niryo One is a 3D printed robotic arm (not only 3D printed), used for learning purposes, with 6 axes [11]. The Niryo Education ecosystem is developing for the purpose of improving skills in programming, mechanics etc. It allows the programmer to program the arm from the most intuitive level (beginner) to the most advanced level.

Figure 2.5: The Niryo One robotic arm[6]

The Niryo One can be programmed via **learning mode**, which allows one to move with a robotic arm with hand or Xbox controller and position into the desired position the user wants. This way of programming is simple for beginners with zero or low knowledge of programming. For more advanced programmers (but still beginners) the **Python API** can be used together with **easy-to-use programming interface** that Niryo provides. For really advanced programmers **the ROS** can be used to directly drive the robotic arm, using Python and C++ programming language [11].

The robotic arm comes with up to 5 different end-effector tools. Three types of grippers, vacuum pump and electromagnet. As a gripper one of three following types can be used. **A standard gripper** for picking small or thin objects with precision, **a large gripper** for larger objects and **a adaptive gripper** for fragile objects with uncommon shapes [11].

[6]Downloaded from: https://niryo.com/products-cobots/niryo-one/

- **Number of axes** - 6
- **Total weight** - 3200g
- **Payload** - 300g
- **Repeatability** - approximately 1mm
- **Collision detection** - Magnetic sensor on motor
- **Used materials** - Aluminium and PLA (3D printing)
- **Actuators** - Stepper and servo motor
- **control board** - Arduino

Robotic arms similar to Niryo One are Niryo Ned or Ned2 robotic arms. Due to Ned2 being newer than Ned, only the differences between Niryo One and Niryo Ned2 are described[11].

Figure 2.6: The Niryo Ned2 robotic arm[7]

It has 6 axes, just like Niryo One. It also combines stepper motors and servomotors (for gripper) and the maximal payload of 300g is identical to Niryo One as well. The construction is very similar as well. The main differences between Niryo One and Niryo Ned2 are [16]:

- New servo motors with Silent Stepper Technology feature (reduces the noise level of the robot)

[7]Downloaded from: https://docs.niryo.com/product/ned2/v1.0.0/generated_pdfs/pdf_en.pdf

- New version of Raspberry Pi4 instead of Pi3
- The standard gripper is removed, so only 4 grippers can be used instead of the 5 types that Niryo One has
- Added the Vision Set, which uses a camera, so the user can use it to find a correct object or use it for machine learning
- Precision of 0.5mm
- Improved the repeatability (from +/- 1mm to +/- 0.5mm)
- Niryo Ned2 is not 3D printed, but it is made of aluminium

The Niryo One, Niryo Ned2 and other Niryo robotic arms can be programmed via Niryo Studio, which allows fast and direct control of a given robotic arm [16]. The goal of this studio is to provide a simple interface for users to program the Niryo arm, and control its status and motions. The programming is done by placing one block of code by one.

AR2 and AR4 robotic arms

The AR2 is a small desktop, low-cost, Arduino Mega-powered, 6-axis robotic arm using stepper motors as actuators [14]. The author of this arm is Chris Annin. The robotic arm is designed to be made of 3D printed material or from aluminium. The robotic arm comes with two types of grippers, which are standard grippers using servo or grippers using pneumatics.

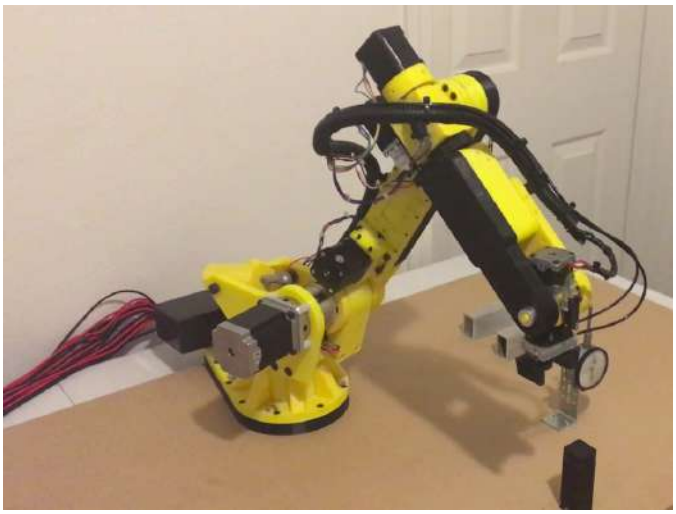

Figure 2.7: The AR2 robotic arm[8]

As mentioned before, the AR2 robotic arm is controlled by Arduino Mega to control servo motors, but Raspberry Pi can be used as well. This version is an open-source project, on which base the AR4 robotic arm was founded. The Ar4 Robotic arm is not an open-source project, but a commercial project.

[8]Downloaded from: https://www.wevolver.com/specs/ar2.robotic.arm

Figure 2.8: The AR4 robotic arm[9]

feature/arm	AR2	AR4
Number of axes	6	6
Total weight	Not mentioned	10000g
Payload	1900g	1000g
Repeatability	0.75mm	0.2mm
Used materials	3D printed or aluminium	3D printed or aluminium
Actuators	Stepper motors	Stepper motors
control board	Arduino Mega and Raspberry Pi	Not mentioned
Category	Open source	Commercial

Table 2.1: Features of AR2 and AR4 stepper motor

Table 2.1 shows the main features of the AR4 robotic arm. The AR4 [22] has improved repeatability, but lower payload than the original AR2 robotic arm provided as an open design.

BCN3D MOVEO robotic arm

The BCN3D MOVEO robotic arm is an open-source project with five axes and 3D printed construction [5]. The arm was developed in collaboration with the Ministry of Education in Barcelona. The goal is to provide quite a low-cost solution for students and other robotic enthusiasts. The MOVEO robotic arm is controlled by Arduino. Due to 3D printed parts, it is possible to make it at home with only a 3D printer.

[9]Downloaded from: https://www.anninrobotics.com/product-page/ar3-complete-solidworks-assembly-step-files

Figure 2.9: The BCN3D MOVEO robotic arm[10]

It is powered by Arduino Mega2560 and as actuators, it uses six stepper motors and one servo motor as a gripper [42]. The robotic arm uses a belt and a pully to transfer motion to manipulate each joint. For better robot control in the cartesian coordinate system, the robotic arm uses a RAMPS, which was developed by Rep Rep. It is the main board for controlling stepper motors that allow connecting DRV8825 stepper motor drivers.

[10]Downloaded from: https://www.bcn3d.com/bcn3d-moveo-the-future-of-learning-robotic-arm/

Chapter 3

Components of robotic arm

This section focuses on existing technology, devices, algorithms and software that can be used for robotics arms, such as actuators, control boards, motion techniques, tools for graphical user interface etc. This chapter is not the encyclopedic enumeration of all existing devices; only devices with a close connection to this thesis are mentioned.

3.1 Robotic arm actuators

Stepper motors

A stepper motor is a brushless DC motor which divides a full rotation into several steps [26]. It converts digital pulses into mechanical rotation. These steps have the same angle of rotation, so to determine the angular position of a motor, external position sensors, such as encoders, are not needed. The angular position of the motor can be computed as the multiplication of the number of steps that the motor performed and the single-step rotation angle [27].

Design of the stepper motor

This stepper motor consists of two main parts, which are the stationary part (**stator**) and the moving part (**rotor**) [17]. The stator is a part that is responsible for creating a magnetic field, which will be changed in time and the rotor will be aligned to. The stator has teeth that will be magnetized by coils attached to them. It is characterised by several phases (number of independent coils attached to the pair of teeth) and pole pairs (determines how many teeth pairs are used for each phase) [27]. Figure 3.1 below shows a two-phase stepper motor on the left and a three-phase motor on the right. Both steppers have one pole pair because each coil is attached to one pair of teeth.

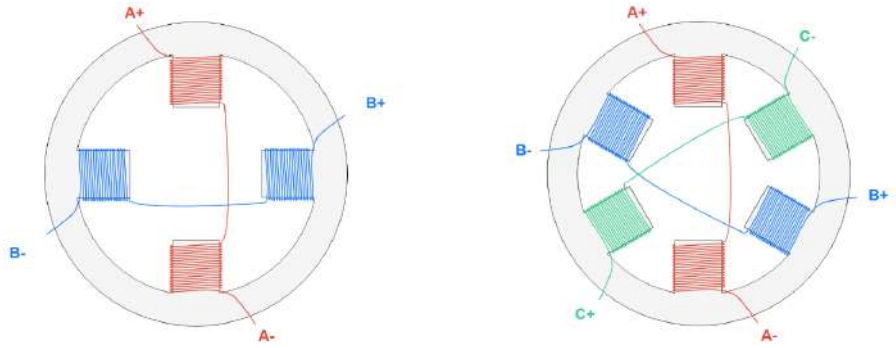

Figure 3.1: Ilustration of two phases and three phases stepper motor[1]

Figure 3.2 shows the two phases stepper motor where the left one has one pole pair and the right one has two pole pairs because two teeth pairs are used for one pole.

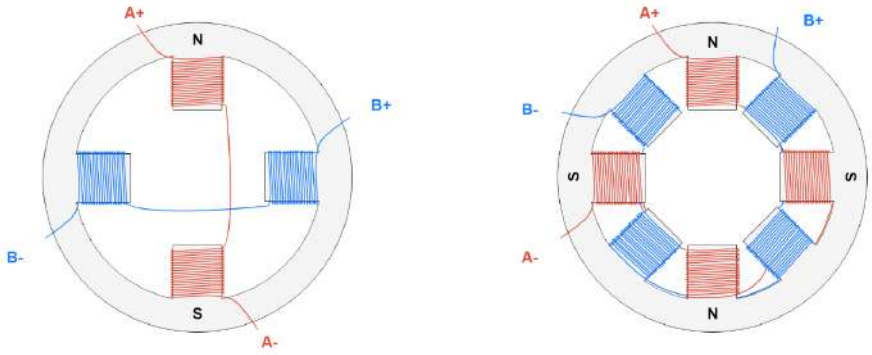

Figure 3.2: Ilustration of two phases, single pole pair and dipole pair stepper motor[2]

The rotor is a moving part that is made of **permanent magnet**, **variable reluctance iron core** or a combination of both mentioned, called **hybrid rotor** [27]. The stepper motor with **permanent magnet** rotor can be used in the project, where the high torque is important, but the resolution is not that important [17]. The stepper motors with **variable reluctance iron core** are suitable for solutions with lower torque, but they can achieve high resolution. This type of rotor has teeth and they are similar to the rotor of an inductor alternator. There is also the third option, which is a stepper motor with **hybrid rotor**. It is a combination of a stepper motor with a permanent magnet and a stepper motor with a variable reluctance structure, so it combines advantages from both of them [17]. This type of stepper motor has a permanent magnet-toothed rotor as well as a toothed stator. Figure 3.3 shows that the rotor is divided into two offset parts, where each part is opposite in polarity.

[1]Downlaoded from: https://media.monolithicpower.com/mps_cms_document/2/0/2020-stepper-motors-basics-types-uses-and-working-principles_r1.0.pdf

[2]Downlaoded from: https://media.monolithicpower.com/mps_cms_document/2/0/2020-stepper-motors-basics-types-uses-and-working-principles_r1.0.pdf

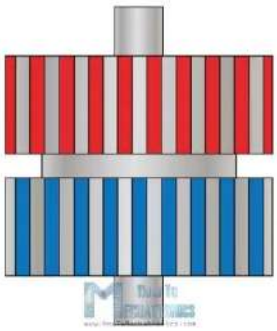

Figure 3.3: The hybrid rotor of stepper motor[3]

How does stepper motor work

As was mentioned the stepper motor rotates in steps, which means the motor does not need any feedback sensor or external sensor such as an encoder to get the current position [17]. For example, when the stepper motor has 200 teeth on the rotor, it can make 200 steps (ticks) to reach a full 360° rotation [15]. This means that each step will rotate the motor shaft for 1.8°, which is the standard step mode called **full step mode**. There are four different step modes, such as **wave step mode**, **full step mode**, **half step mode** and **micro step mode**. These modes are described below, on a two-phase stepper motor.

The wave mode is a mode where only one phase at a time is energized. For example, when two phases (two coils) are used, the first will be called phase A and the second phase B. In the beginning, only phase A is energized and B is off, which turns the rotor to 90°, then only B is energized and phase A is off etc. [27]. Figure 3.4 shows how the wave mode works.

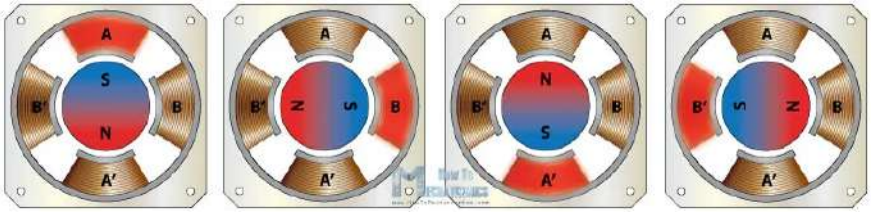

Figure 3.4: Ilustration of how the wave mode works[4]

The full step mode is a mode where two coils are energized at the time [27]. It can provide the highest torque of the mentioned three modes. The disadvantage of this mode is that the motion is not so smooth [15]. Figure 3.5 shows how the full-step mode works.

[3]Downloaded from: https://howtomechatronics.com/how-it-works/electrical-engineering/stepper-motor/

[4]Downloaded from: https://howtomechatronics.com/how-it-works/electrical-engineering/stepper-motor/

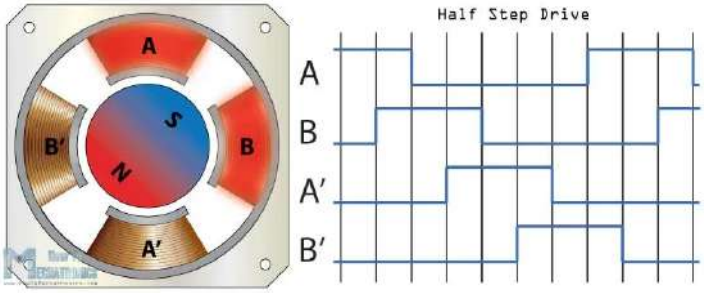

Figure 3.5: Ilustration of the full step mode[5]

The half-step mode can, as the name suggests, perform double the number of steps for a full rotation. It is a combination of the **wave mode** and the **full stem pode**, described above. So in the beginning, only one phase is energized. In the next step, the neighbour phase will be energised too (so two phases are energized at the time), so up to 2 coils are energised at the time [27]. The big disadvantage of this mode is a drop of torque of approximately 30%. The big advantage is smoother rotation [15].

The micro stepping mode is the enhancement of half step mode because we can make even smaller steps than in the case of half step mode. This can be reached by controlling the intensity of the current that goes through each phase [27]. The main goal is to create a rotating magnetic field. This can be done in a few steps described below.

1. At the beginning, phase A is energized for the maximal possible current and phase B is off, so the current is 0A.

2. Current through phase A is controlled to reach 0.92 of the maximal current of phase A and the current through phase B is controlled to reach 0.38 of the maximal current of phase B.

3. Both phases are controlled to reach 0.71 of the maximal current

4. Same as point 2, but phases have opposite values

5. Same as point 1, but phases have opposite values, so phase A is off (0A) and phase B is fully energized

This mode can perform very smooth motion, which is useful for accurate positioning. The disadvantage is the same as for the **half step mode**, which is torque drop for approximately 30% [15].

[5]Downloaded from: https://howtomechatronics.com/how-it-works/electrical-engineering/stepper-motor/

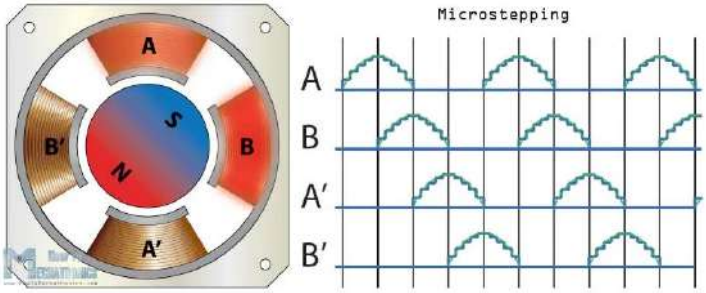

Figure 3.6: Ilustration of stepper motor with micro-stepping mode[6]

Figure 3.6 describes the micro-stepping mode.

Controlling the stepper motor

As mentioned before, to control the stepper motor, it is necessary to control the current flow through each coil [27]. For that purpose, we need a device that consists of two following devices:

- **A transistor bridge** - set of transistors that take care of current flow through the coil by turning appropriate transistors on and of (this is called H bridge and we need one for each motor phase [27])
- **A pre-driver** - [27] a device for controlling the transistor bridge (it is controlled by MCU)

The aforementioned devices can be used separately or together, which is called a driver [27]. It is necessary to have some MCU, which is programmed by the motor user, that will control that pre-driver or driver to do the desired job. Figure 3.7 shows the basic scheme of such a driver.

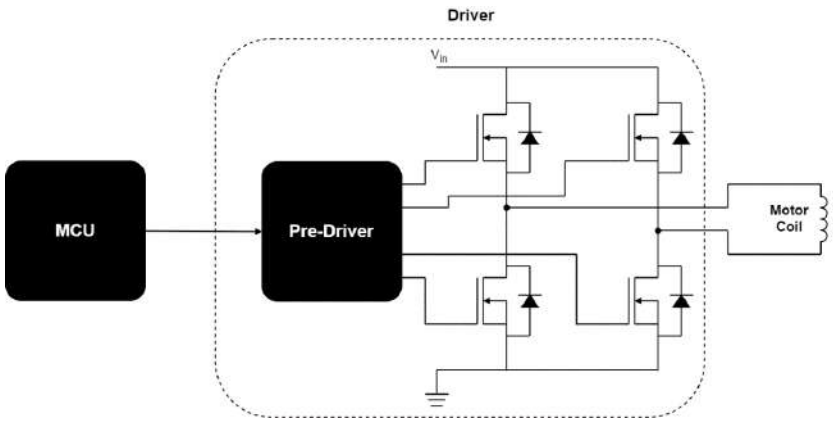

Figure 3.7: Ilustration of stepper motor control[7]

[6]Downloaded from: https://howtomechatronics.com/how-it-works/electrical-engineering/stepper-motor/

[7]Downloaded from: https://media.monolithicpower.com/mps_cms_document/2/0/2020-stepper-motors-basics-types-uses-and-working-principles_r1.0.pdf

Examples of suitable drivers that are available on the market are **a4988**, **TMC2208**, **TMC2130** or **TOSHIBA TB6600**[8].

Advantages and disadvantages

Now, it is time to make a conclusion and summarise the advantages and disadvantages of the stepper motor.

The **adventages** are [27]:

- **No external sensors needed** to determine precise position (due to the internal structure of the stepper motor)
- **Simple controlling** - only a driver is needed, but it is not necessary to calibrate it
- **High accuracy** - with micro stepping it is possible to reach accuracy up to 0.007°
- Good torque at low speed
- Long lifespan

the **disadventages** are [27]:

- If the load torque is high **the chance to miss a step is high** - the user must deal with the wrong position when the step is missed
- The motors **consume the maximal amount of current** even if it stays still - not efficient and it causes overheating
- **Noisy at higher speed**
- **Low power density**
- **Low torque to inertia ration**

In conclusion, the stepper motors are a good solution when the user needs an inexpensive, easily controlled (by using drivers) option and does not care about efficiency or high torque in combination with high speed [27].

Servo motor

The Servo motor is a rotary or linear actuator, suitable for reaching **precise angular or liner position** [37]. It is part of **closed-loop systems** which means that the servo motor uses feedback to control its motion and position. It is a simple motor that runs through a servo mechanism [23]. Servo motors are rated in kg/cm. This indicates that a 10kg/cm servo can lift up to 10 kg when the load is placed 1 cm from the motor shaft. When the load is placed farther from the motor shaft, the final weight that the motor can lift decreases.

[8]Drivers were take from: https://all3dp.com/2/best-stepper-motor-driver/

Design of the servo motor

The servo motor consists of two main parts. The motor that is responsible for movement and the position sensor, which can be encoder for example [7]. There are some other parts, such as an amplifier, a drive gear, an output shaft and a position sensor such as an encoder or a resolver [37]. Figure 3.8 shows how does the servo motor look like inside. The advantage of the servo motor is that it excels at speed and position control, and its precision. Due to position sensors, the servo motor can not be stalled, because of the position sensor. This sensor is responsible for checking the position and if the external force pushes the servo motor back, it can be corrected [7].

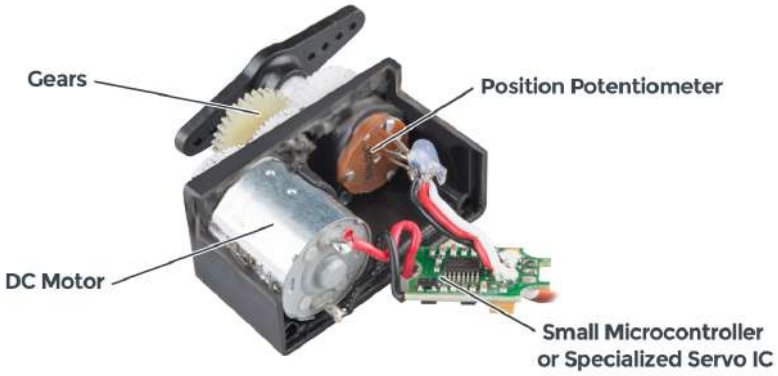

Figure 3.8: Illustration of how the servo motor looks inside[9]

How does servo motor work

As was mentioned before, the servo motor is a closed-loop system, which uses a positive feedback system to determine the position of the output shaft. This feedback signal is compared with a reference input signal [23]. This comparison generates the signal, which is the input signal for controlling the servo motor. The feedback signal is generated via a potentiometer, attached to the output motor shaft. So when the output shaft rotates, the potentiometer rotates too, until the value of the signal generated by the potentiometer and the external signal value is equal. This will stop the servo motor, due to the shaft reaching the desired angular position [23].

Controlling the servo motor

To control this type of motor, the servo offers three types of wires. Two of them are used for supply (GND and VCC) and the third one is used for an external input signal called PWM signal, which is provided by the microcontroller [23].

[9]Downloaded from: https://www.sparkfun.com/servos

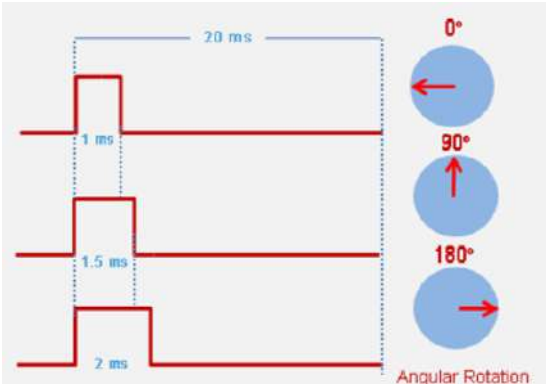

Figure 3.9: Illustration of servo motor controlling[10]

The servo motor is controlled via an external signal - the PWM signal (the PWM signal is described below). The servo expects the input signal (external signal) every 20 milliseconds. The length of the pulse determines how much the servo rotates [23]. The signal can be set from 1 ms, which is 0° up to 2 ms, which is the maximal angle, for example, 180° [23]. Figure 3.9 above illustrates how the servo motor is controlled by a pulsing signal.

What is PWM signal

Pulse Width Modulation or **PWM** is a powerful way to represent analogue value by digital value [29]. Therefore, when the task is just to read the analogue value by MCU, the user can use the ADC (Analog to Digital Converter). The problem comes in a situation when the goal is to control the analogue device according to the read value, which is digital. This can be solved by using DAC (Digital to Analog Converter), but they are expensive to produce in terms of cost and they consume a lot of silicon space [2]. So the better solution for this is to use mentioned PWM.

It controls the amount of power that is given to a device by quickly switching on and off. The amount of time, when the signal is high or On (width of signal in high) in a given period, is called **duty cycle** [36]. For example, duty cycle 80% means, the signal is 80% time in high and 20% in low, so the output value is 80% of input voltage. Figure 3.10 illustrates how the duty cycle works.

[10]Downloaded from: https://circuitdigest.com/article/servo-motor-working-and-basics

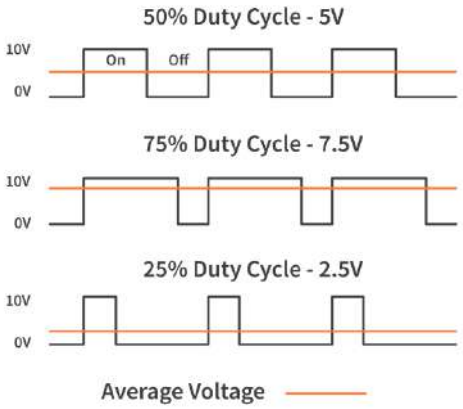

Figure 3.10: Illustration of the PWM duty cycle[11]

The device will not recognize it as switching but as the average voltage value, which is counted as the on-time (the time when the signal is high). The advantage of the PWM technique is that the **power loss is very low** in comparison with the potentiometer, which generates power loss and heat [36].

Advantages and disadvantages

As a conclusion for the stepper motor, I'd like to mention some advantages and disadvantages.

The **advantages** are [37]:

- **High output power** in comparsion with servo motor size and weight
- **High efficiency** (90% at light loads)
- **High torque to inertia ratio**
- **High speed at high torque** - good for position holding
- **Quiet and speed**
- **Higher accuracy** wiht encoder utilization

the **disadvantages** are [37]:

- **Need tuning** to stabilize the feedback loop
- **Unpredictable** when something breaks
- **Peak torque is limited** to a 1% duty cycle
- **Higher system cost**
- **Gearboxes are ofter required** to deliver power at higher speeds

[11] Downloaded from: https://www.circuitbread.com/ee-faq/what-is-a-pwm-signal

3.2 Control boards

As mentioned in previous sections, the control board is necessary for the controlling of motors or the behaviour of the robotic arm itself. This section describes control boards that are often used.

Raspberry Pi

Raspberry Pi is a low-cost computer the size of a credit card made by the Raspberry Pi Foundation. The user can connect the monitor to it and use a standard keyboard and mouse. The user can use it for anything an ok desktop computer is capable of, such as browsing the web, programming, watching videos etc. [28].

It is a programmable device with all the motherboard's critical features, which can be found in the average computer, but it has no peripherals or internal storage. To set up Raspberry Pi, the user must use an SD card for the operating system. It is compatible with Linux OS which is handy because this system does not need too much memory space [28].

There are many generations of Raspberry Pi computers, such as Raspberry Pi Zero, Raspberry Pi 1, Raspberry Pi 2 B, Raspberry Pi 3, Raspberry Pi 4B, Raspberry Pi Pico and Raspberry Pi 400 [28]. Due to the concept of this work only the Raspberry Pi 3, 4 B and Raspberry Pi Pico are described.

Figure 3.11: The Raspberry Pi 3[12]

Both computers have fast processing units, HDMI ports, USB ports, 40 programable general-purpose pins and support Ethernet and Wi-Fi. Raspberry Pi 3 is the predecessor of the Raspberry Pi 4 B, so the Raspberry Pi 4 B offers higher performance, such as 2GB to 8GB of RAM, instead of 1GB of RAM, the Raspberry Pi 3 has and it has a faster processor with frequency 1.5GHz [28].

[12]Downloaded from: https://www.raspberrypi-spy.co.uk/2016/02/introducing-the-raspberry-pi-3-model-b/

Raspberry Pi Pico

The low-cost, high-performance microcontroller board, is designed to interface with and control physical, real-world projects. In comparison to all Raspberry Pi computers, the Raspberry Pi Pico is not capable to run an operating system. It is designed to interface with and control physical, real-world projects.

Figure 3.12: The Raspberry Pi Pico[13]

The Raspberry Pi Pico has 26 GPIO pins, instead of the 40 pins that modern Raspberry Pi boards normally have and the 133 MHz Dual-core Arm Cortex M0+, but it has no Wi-Fi or Bluetooth. [12]. Another advantage is that it is suitable for beginners too because it can be programmed in MicroPython, CircuitPython or even C or C++ for advanced programmers [41].

Arduino Uno

Good development board for beginners, with a 16 MHz ATmega328P microcontroller. It has 14 GPIO pins, an ICSP header and a USB connection [18]. Beginners can begin their programming in Arduino IDE, developed by Arduino developers.

The Arduino IDE is a **versatile editor**, which offers the possibility of installing all libraries directly from the IDE, so users do not need to download it from the internet. It offers a cloud system, and debugging sketches (sheet with code). The user can use inbuild **serial monitor** or **serial plotter** too, which allows you to view data streaming from your board. An example of one of the big features of Arduino IDE is **autocompletition**, which was added in the last (second) version of IDE [40].

ESP32

A powerful, generic Wi-Fi + BT + BLE MCU, that can be used for anything from low-power applications to more complex tasks, such as voice encoding etc. The core of the ESP-WROOM-32 module is an ESP32-D0WDQ6 chip with two cores (that can be individually controlled) with frequency from 80MHz up to 240MHz, 4MB of internal flash memory and 520KB of on-chip SRAM. It comes with 38 pins, where 32 of which are used as general-purpose pins (GPIO pins) [10].

Due to Bluetooth, it is possible to communicate with another device such as a smartphone. The ESP32 chip allows low-power application as well, for example in sleep mode

[13]Downloaded from: https://cz.farnell.com/raspberry-pi/raspberry-pi-pico/raspberry-pi-32bit-arm-cortex/dp/3643332

the current consumption is less than 5μ. This feature also makes this device suitable for wearable applications. The user can use freeRTOS as an operating system and for security the secure (encrypted) over-the-air upgrade can be used [10].

3.3 Stall detection

In many applications, the user can ascertain the position of the motor via a sensor, or by the motor itself, e.g. stepper motors. In some projects, the user needs to know not only the exact position but the state of the motor. It is useful for diagnostics purposes, overloaded detection or checking if the motor hit any physical obstacle [33].

Without stall detection, the motor would hit an obstacle, e.g. a human, and continue to drive through it. It can cause audible noise or cause damage to the motor itself and it can be very dangerous for the human as well [33].

Measuring BEMF - stepper motors

It is necessary to specify one term, necessary to understand stall detection which is **Back EMF**. When the current flows through the motor coils, it generates a rotating magnetic field that interacts with the magnetic field of the rotor magnets. This will cause the rotor will move, but this effect works in the opposite too, so when the rotor moves it induces current in the stator. This is known as back **EMF** (**B**ack **E**lectro**M**otive **F**orce) [35] .

The EMF is used to detect stall detection. All that is needed is to simply measure the voltage on a coil. When the motor is moving, the voltage will be presented over the terminals of the coil, but when the motor is stalled, no voltage is presented [35]. The process of detecting that stall detection is very simple and does not need any other wiring. All that is necessary is to connect both coil pins to the measuring circuit and wait till the coil is not driven (the coil will be turned off and no current will flow through it). At this moment, where the coil is floated, the Back EMF can be detected (measured). When voltage is measured on the floated coil it means the voltage is generated from the moving rotor and the motor is not stalled. When no voltage is measured on the floated coil it means the rotor is not moving, meaning it is stalled [35].

Figure 3.13 shows the moment when the phases are not driven and can be used for measuring back EMF in full-step mode.

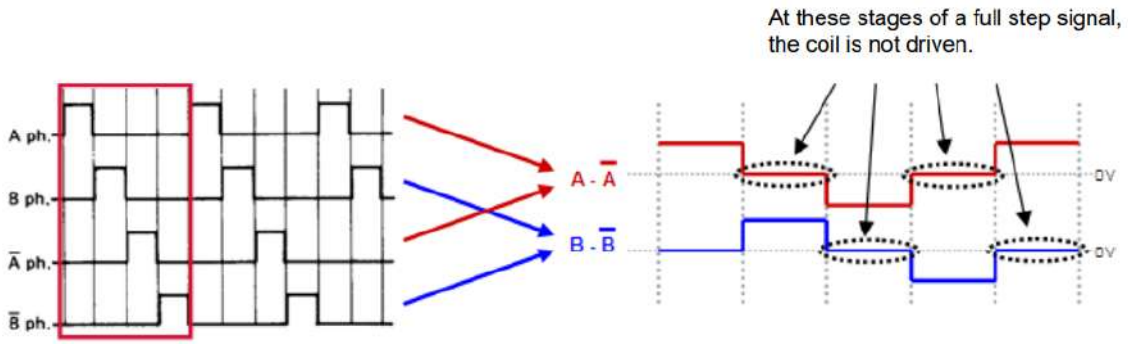

Figure 3.13: Moments to detect back EMF in full-stepping mode[14]

[14]Downloaded from: https://www.nxp.com/docs/en/application-note/AN4024.pdf

Full step mode of the stepper motor, described in section 3.1, is good for BEMF detection, but at a moment, when the coil is floated, the voltage on it can oscillate, which can cause problems for the accuracy of stall detection. Of course, it can be an advantage, because when the peak is high, it is clear that the motor is not stalled. The problem is that it can dictate the speed of stepper motor stepping [35]. The half-stepping and micro-stepping modes can be used to eliminate problems with oscillations, but the problem is that the time when the coil can be opened is so small, so it is difficult to measure the voltage on it.

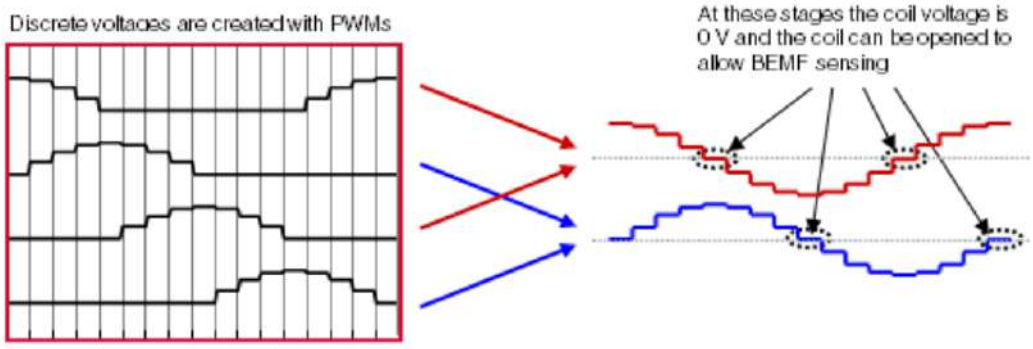

Figure 3.14: Moments to detect back EMF in micro-stepping mode[15]

Figure 3.14 shows when it is suitable to measure the BEMF and it is visible, that the value when the coil voltage is 0v (suitable for opening the coil) is a very short time. A better solution is to measure the flyback time [35].

The BEMF method should be measured after the flyback energy has disappeared. The problem is that due to the motor consisting of coils, the value cannot change immediately. So when the current through the coil is cut off, the coil generates an opposite voltage that will slowly decrease its value. This is known as flyback voltage and the time that takes for the flyback energy to dissipate is called the flyback time [35].

It is not necessary to wait until the flyback energy has dissipated, because the flyback time (time till the flyback energy dissipates) can be easily measured by a simple timer. When the coil is disconnected, the current through the coil starts decreasing and the coil generates an opposite voltage spike (flyback voltage) which spike goes up to the Back EMF voltage. This spike goes back while the coil current is dissipating and reaches the Back EMF voltage when the coil current reaches 0A. The reaching Back EMF voltage creates a rising edge that starts the timer [35].

When a motor is stalled, it is possible the voltage across the coil may reach 0V except for the BEMF value as in the case of a moving motor. The rising edge should not be high enough to trigger the timer. To solve this problem it is necessary to cause the end of the coil to be driven all the way to VDD [35]. That will guarantee that the rising edge will hit the threshold.

[15]Downloaded from: https://www.nxp.com/docs/en/application-note/AN4024.pdf

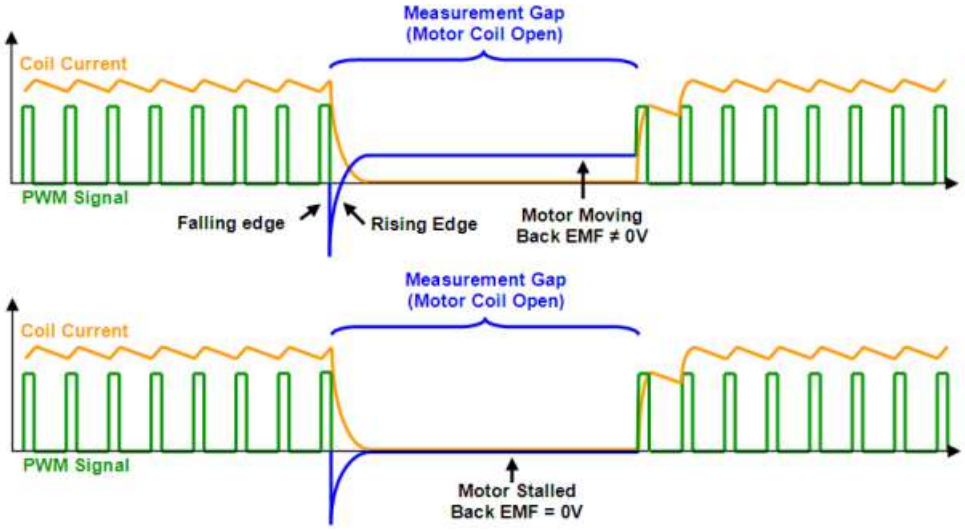

Figure 3.15: Measuring Back EMF voltage with flyback time[16]

Figure 3.15 above shows how the flyback voltage and back EMF voltage behave for the moving motor (upper figure) and for stalled motor (bottom figure). The blue colour represents the measuring gap, where the motor stall can be detected. The orange graph represents the current that flows through the motor coil. The green colour represents the PWM signal to control that stepper motor.

Using encoders

The encoder is a device that converts motion into an electrical signal [1]. According to the type of movement, there are rotary and linear encoders. It can be used as a sensor for detecting movement, speed and position. They can be divided into two types, which are the absolute and incremental encoders. The difference between these two types is that the absolute encoders generate a signal that reflects the movement and knows its position after resetting. The incremental encoder generates a multi-bit digital word that directly indicates the position, but after reset, it does not know its position. groups according to measuring principles. The optical encoders and the magnetic encoders.

The magnetic encoder consists of a magnetized disk (with several poles around) and a sensor (to detect a change in the magnetic field while the magnetic disk is rotating) to detect that field, for example, the Hall sensor [32]. The sensor detects the change in the magnetic polarity of the disc. Figure 3.17 shows what does magnetic encoder looks like.

[16]Downloaded from: https://www.nxp.com/docs/en/application-note/AN4024.pdf

Figure 3.16: Magnetic encoder[17]

On the other hand, the optical encoders consist of an emitter, code disc and optical sensor [32]. The emitter is placed in front of the optical sensor and the code disc is placed between them. When the motor shaft rotates the code disc rotates as well. The output signal is based on actual code that is between the emitter and the receiver, which can be either a transparent region or an opaque region. When the region is transparent, the beam can easily reach the receiver, but when the region is opaque, the beam is blocked. Figure 3.17 shows how the basic concept of the optical rotary encoder.

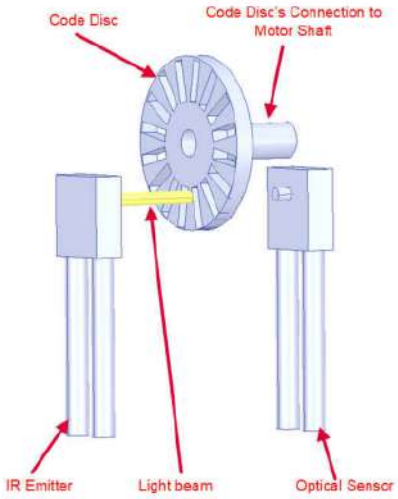

Figure 3.17: Optical encoder[18]

The encoders can be used for stall detection, where the generated step from the microcontroller should match the feedback signal that the encoder produced. When both signals match, the motor is not stalled. The problem occurs when the number of encoder steps does not match the step count. This situation is problematic because it can be caused by one of three events: The motor is stalled, the motor lost step or steps and the resonance situation [9].

[17] Downloadedfrom:https://www.motioncontroltips.com/faq-how-do-magnetic-encoders-work/
[18] https://www.ti.com/lit/an/slya061/slya061.pdf

3.4 Graphical user interface

For interaction with a robotic arm (robot in general) where the position of each joint can be specified or where the robot status is presented, **the graphical user interface** (GUI) is suitable.

wxPyhton

The cross-platform toolkit (supported on Microsoft Windows, Mac OS X or MacOS, Linux and Unix-like systems) for making graphical user interfaces in Python programming language simply and easily. It is in the form of a Python package that wraps the wxWidget library written in C++ programming language. It is in the form of Open Source, which allows one to view and modify the entire code and improve their project [4].

```
1  import wx # import the wxPython package.
2
3  app = wx.App() # create an application object
4  frm = wx.Frame(None, title="Hello World") # create a frame object
5  frm.Show() # show the entire GUI
6
7  app.MainLoop() # start the event loop
```

Figure 3.18: The result of simple wxPython application (Windows OS)[19]

The previous code snippet is a simple code example of the Hello World application in wxPython [4]. Figure 3.18 visualizes the result of that code snippet in the Microsoft Windows operating system.

The wxPython has a project called Phoenix, which is the new implementation of wxPython, which goal is to improve performance, maintainability, and extensibility and clean the entire code. This big cleanup and improvements lead to a problem with backward compatibility with classic wxPython.

For creating GUI with the wxWidget framework, the wxFormBuilder can be used. The user can make the GUI by selecting desired widgets and putting them in a suitable place.

[19]Downloaded from: https://www.tutorialspoint.com/wxpython/wxpython_tutorial.pdf

The code generation behind it is in the role of this builder. It can generate it in five different programming languages such as C++, Python, XRC, LUA and PHP[20].

Tkinter

Python interface The Tkinter is one of the most commonly used modules for making graphical user interface applications in Python programming language. The advantage is that this module comes with Python during installation[21].

The Tkinter is an abbreviation to **Tk** (or newer family Ttk) **interface**, which is a „standard Python interface to the Tcl/Tk GUI toolkit“. It is supported by the Windows operating system and Unix platforms such as macOS [38].

The **Tcl** is a programming language commonly embedded in C languages. It behaves like a scripting engine or an interface to the Tk toolkit. A **Tk** is a package written in C language used to create graphical user interface widgets. Although the Tk is written in C programming language, the Tkinter allows the user to program that GUI in Python programming language [38].

```
1  import tkinter as tk # Python 3.x Version
2  #import Tkinter as tk # Python 2.x Version
3  root = tk.Tk()
4  label = tk.Label(root, text="Hello World!") # Create a text label
5  label.pack(padx=20, pady=20) # Pack it into the window
6  root.mainloop()
```

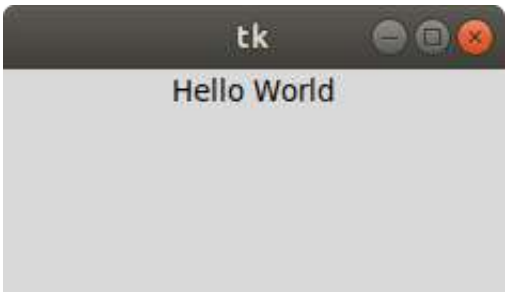

Figure 3.19: The result of simple Tkinter application (Windows OS)[22]

The previous code snippet shows how to create a simple graphical user interface using the Tkinter package. Figure 3.19 shows GUI represented by the previous code snippet [23].

PyQt

It is a module for Python that connects Qt C++ framework, which is cross-platform (supported for Microsoft Windows, macOS X and Linux), with Python language. It is not only a toolkit for making GUI, but it contains aslo libraries for XML, SVG, SQL and other libraries. [3].

[20]Desribed in more detail: https://umar-yusuf.blogspot.com/2015/12/wxformbuilder-tutorial-on-gui-for.html
[21]Desribed in more detail: https://riptutorial.com/Download/tkinter.pdf
[22]Downloaded from: https://riptutorial.com/Download/tkinter.pdf
[23]Code snippet and figure find here: https://riptutorial.com/Download/tkinter.pdf

```
import sys
from PyQt4 import QtGui
def window():
    app = QtGui.QApplication(sys.argv)
    w = QtGui.QWidget()
    b= QtGui.QLabel(w)
    b.setText("Hello World!")
    w.setGeometry(100,100,200,50)
    b.move(50,20)
    w.setWindowTitle('PyQt')
    w.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    window()
```

The code snippet above shows the basic hello world application using PyQt [3]. Instead of writing GUI in code, Qt comes with a Qt designer. It is a tool which acts as a graphical user interface, where the controls (widgets) can be placed instead of writing code [3].

Chapter 4

Concept of the proposed arm solution

This chapter focuses on the design goals of my robotic arm. It describes all features, necessary for the final implementation, and all necessary steps to construct the robotic arm. It also analizes the existing state of the robotic arms, such as existing robotic arms, control board, etc. At the end of this chapter, the tests that help in the process of ensuring the robotic arm works correctly according to specification, are defined.

4.1 Comparison of existing sulution

In previous sections 2.3 and 3.2, different types of control boards and existing robotic arms are mentioned. In this section, all of mentioned robotic arms and control boards are compared and summarised. This section helps to choose the appropriate component. The summarization for both previously mentioned sections is visualised in the table for improved readability.

Comaprsion of robotic arms

All of them are mainly 3D printed or aluminium is used as well. The stepper motors are the most common actuator for each joint and the servo motors are well used for the gripper mechanism. As a control boards, the Raspberry Pi and Arduino board are highly used. The following table visualizes what are the differences.

Robotic arm	Axes	Payload	Repeatability	Features
BCN3D MOVEO	5	/	/	/
AR2	6	1900g	0.75mm	/
AR4	6	1900g	0.2mm	/
Nyrio One	6	300g	1mm	collision detection, gripper can be changed
Nyrio Ned2	6	300g	0.5mm	gripper can be changed, attachable camera

Table 4.1: Sumarization of existing robotic arms

Table 4.1 shows all mentioned robotic arms have the same amount of axes (except for BCN3D MOVEO), but only AR2 and AR4 robotic arms have a big payload. The repeatability is quite similar for all of them except for AR4 with 0.2mm. The biggest difference is the other features, such as collision detection, changeable gripper and attachable camera. These features only Nyrio One and Nyrui Ned2 have.

Comparsion of the control boards

In section 3.2 some of the main control boards that can be used for similar projects as well as my thesis are described. It was not the encyclopedia enumeration of all possible control boards, but only boards, that have a close connection to my thesis are mentioned. Table 4.2 shows the comparison.

Control board	Frequency	RAM size	Pins	Ethernet	Wi-Fi
Raspberry Pi 3	1.2 GHz	1 GB	40	Yes	Yes
Raspberry Pi 4B	1.5 GHz	2 - 8 GB	40	Yes	Yes
Raspberry Pi Pico B	133 MHz	264 kB	40	No	No
Arduino Uno	16 MHz	2 KB	16	No	No
ESP32 WROOM	80 - 240 MHz	520 KB	32	Yes	Yes

Table 4.2: Control boards Features

From the previous table, it is clear that the most powerful commonly used control board is the Raspberry Pi 4B control board with 1.5GHz of processor frequency. The ESP32 WROOM except for the Raspberry Pi family has big internal RAM memory. The ESP32 has Ethernet and Wi-Fi too, which the Arduino does not have, for example. From my comparison, the most powerful control boards are definitely the Raspberry Pi family and the ESP32 WROOM control board.

Comparsion of stepper motor drivers

The drivers that I found are mentioned in section 3.1 and are compared in this section. As well as other comparisons made in this thesis, this is not a comparison based on encyclopedical enumeration.

Name	Max resolution	I_{max}	Features
A4988 [13]	1/16	2A	Nothing
TMC2208 [20]	1/256	2A	Stall detection
TMC2130 [19]	1/256	2A (2.5A peak)	Stall detection
tb6600 [8]	1/16	4.5A	Nothing

Table 4.3: Comparsion of reliable stepper motor drivers

Table 4.3 shows the main features of the stepper motor drivers, such as maximal resolution, the maximal current through the coil and special features if it has any [1]. The aforementioned Toshiba tb6600 is the HY-DIV268N-5A variant. This stepper motor driver is suitable for bigger stepper motors such as NEMA23, due to its current limit, which can be seen in the table above.

[1]Prices of drivers were taken from: https://all3dp.com/2/best-stepper-motor-driver/

4.2 Desired goals for robotic arm

Evaluation of my investigation, made in chapters Robotic arm introduction and Components of the robotic arm, led me to the conclusion on how to create my own robotic arm and what features it should have. The reason I made my own robotic arm solution was because I was curious if I can make a better solution compared to the one I made in my Bachelor's thesis.

This section specifies all the features and goals, that are interesting or important and were missing in many solutions found during research. The following list shows the features implemented in my robotic arm.

- **Easy to reproduce in home conditions** - The robotic arm should be easy to make in home conditions, which means it should be made out of a 3D printer, so no sophisticated technology is needed. This makes my robotic arm suitable for almost everybody.
- **Autonomous** - Almost all robotic arms mentioned in 4.1 do not have any sensors, cameras or any other features which would allow the robotic arm to be autonomous. The robotic arm should have sensors to detect robotic arm position, collision with obstacles around and detect objects with a camera.
- **Accuracy** - The robotic arm should be as accurate as possible. Better accuracy means that it can be used in many different ways, e.g. it can be used as a toy (where the accuracy does not have to be high), but it can be used as a manipulator in small companies (where high precision is necessary)
- **Own and intuitive GUI** - The GUI should provide the user to control the robotic arm in different ways - to control one motor by one, control the whole robotic arm with inversion kinematics, autonomous control with a camera etc. The GUI should be intuitive and customizable (choose units, specify ports...)
- **Documentation** - The final solution should provide a good level of documentation, where the user can find all necessary information.
- **Open Source** - The whole solution must be provided as open source, so everybody can download all my files and reproduce them or edit them according to users' needs.

According to the goals above, my solution should have the features mentioned below, and all of them should be tested at the end to ensure the goals have been fulfilled.

- **Five degrees of freedom**
- The **stepper motors** as the main actuator and **the servo motor** for gripper mechanism
- Intuitive **Graphical User Interface** as application with no web
- **Three types of controlling** - individual motor controlling, semi-autonomous controlling with inversion kinematics and camera and the user hand drive
- Sensors for **actuator position** capturing, **distance measuring** sensor and **current sensor**

- Camera with support of object recognition
- **Payload** at least 300g
- Separated solution into two processors (controlling and computing processors)
- The emergency **stop button** to stop robot if needed

Figure 4.1 shows in an easy way the design goals of the robotic arm with the features listed above.

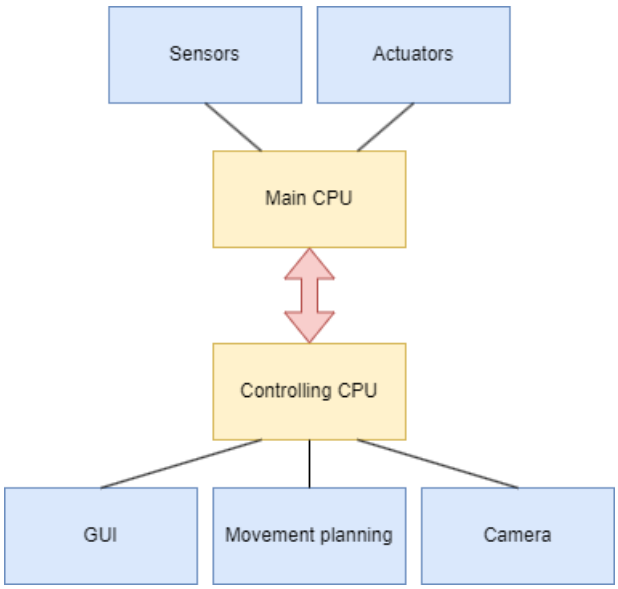

Figure 4.1: The goals of robotic arm goals

The main goal is to separate the computation power into two processors. The reason for that is to separate the load so that one processor can be strong enough and used for computations and a camera, while the second processor is only used for controlling. The control processor should cooperate with all sensors, and actuators (stepper motors and servo).

Both processors should communicate together to send information to each other about the actual state. For example, the control processor should send the positions of actuators and the state of the whole robotic arm. The powerful control board should send information on what the control board should do and aggregate data from it.

4.3 Design goals of implement of my robotic arm

This section focuses on the necessary steps to reach the mentioned goals in the previous section 4.1. There are three main parts that must be done:

- **Construction** - Creates new robotic arm, or modify existing robotic arm to attach all sensors. 4.1.
- **Electronics** - Design a main control board that connects all peripherals to one place. In this step, the following sub-steps must be done:

  - **Powering** - provides power for all components, such as actuators, processors or sensors
  - **Sensors** - makes printed circuit boards for all sensors, that can be easily attached to the robotic arm
  - **Communication** - manages communication between sensors and main control processor or between both processors

- **Software** - There are two main parts that must be done:
  - **Graphical User Interface (GUI)** - Create an intuitive Graphical User Interface, without a web server, providing three different types of controlling.
    * **Each motor separately** - user should be able to control each motor separately without affecting the position of other stepper motors
    * **Whole robotic arm with camera** - this should allow the user to detect objects around via camera and by using inverse kinematics, the robotic arm should grab the object
    * **Hand driven mode** - user can move the robotic arm to a certain position, capture the position of all joints and store it for further replay
  - **Firmware** - This part covers firmware for both the main control processor and the main compute processor. For the main control processor, the firmware includes reading data from sensors and communication with the main computing processor. Firmware for the main compute processor covers the full backend for GUI, so all widgets should be fully functional, including the computing of inverse and forward kinematics, camera recognition etc.

This section describes only the design goals of the implementation. Chapter 5, describes the physical solution itself in a detailed way.

4.4 Design of the test procedure of robotic arm

The section 4.1 describes features which must be implemented in my robotic arm. To ensure all my features are implemented correctly, it is necessary to specify the test set. It helps to determine if the goals were fulfilled. The following tests are designed to test goals and features from section 4.1. The test set and the design of testing use cases are similar to my bachelor thesis [24]. There are other test cases that can be tested, but they are not mentioned there because of the time limitation of the thesis.

- **Full weight test** - This test cannot fail or pass. The output from this test will be used only for information on how heavy is the robotic arm with all its components.
- **Payload test** - The goal is to lift at least 300g object. The test will test objects from 100g to at least 300g. The robotic arm lifts one object by one, according to its weight and if the arm lifts all of them, the test is successful. The reason why the robot starts at 100g object and not 300g object is only due to safety reasons. The robotic arm can crash if the weight is too high, however, if it starts with a low weight, the behaviour of the arm can be observed and it helps to prevent accidents. The load can be considered as successfully lifted if at least half of the robotic arm stepper motors do not lose any step.

- **Accuracy** - The robotic arm will be set to default position, e.g. straight, where all joints point upwards. Afterwards, if the robotic arm is set to a different position, we measure the current position. From the coordinates which the robotic arm should have reached and those coordinates that were actually reached, the accuracy/precision can be measured.
- **Repeatability** - The goal is to replicate the accuracy test many times, e.g. 10 times and 100 times. The difference between the accuracy test and the repeatability test is that the precision will not be measured after each repetition but after all repetitions in the given set. For example, when the repeatability will be tested on 10 repetitions, the precision will be measured after 10. repetition.
- **Stall detection** - Robotic arm is in one default position and a different position is specified. Meanwhile, if the robotic arm reached the target position, it will be blocked by an object. If the object will cause the robotic arm will detect it and stop it, the test is successful.
- **Gripper object detection** - The goal is to test if the gripper is able to stop closing when detects an object in its claws space.
- **GUI test** - Check if the user interface is still correctly visible, all widgets work as they should and are stable when it is used longer time.

The list of tests mentioned above does not cover everything that can be tested, but due to time limitations for this thesis, it is good enough. Another test that can be tested by the future user is checking the behaviour of the robotic arm when it is used longer time period, e.g. a few days without reset.

Chapter 5

Implementation of the robotic arm

This chapter focuses on the physical implementation of what was described in the previous chapter. The reader will understand how this robotic arm was designed. It covers everything necessary for this thesis, such as the design of a robotic arm, firmware, graphical user interface, hardware (electronics) and testing. The final robotic arm can be seen in appendix D.

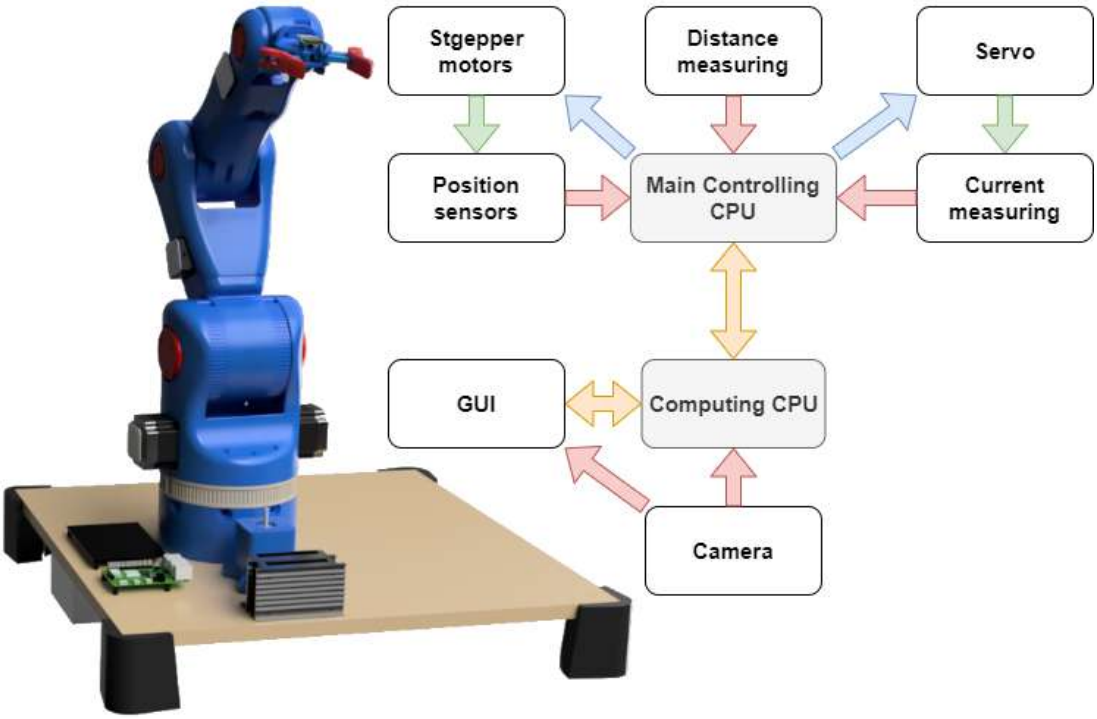

Figure 5.1: The robotic arm realization overview

Figure 5.1 shows the robotic arm realization overview to better understand what was designed. It can be seen that the main control board contains ESP32 and the Raspberry Pi4 is used as well. This solution caused the load to be split between these two processors and the final load will be lower.

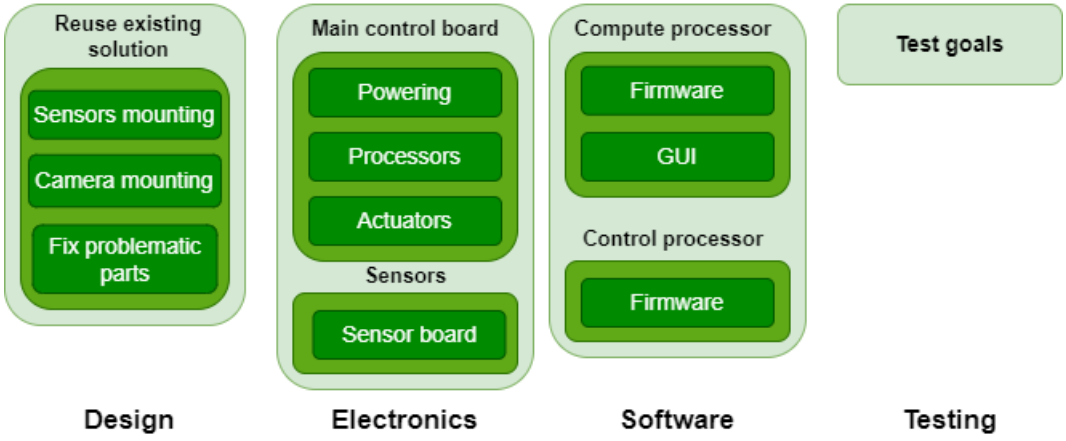

Figure 5.2: breakdown of all tasks to implement

Figure 5.2 visualizes the breakdown of the tasks that were necessary to implement. The blue box represents the final product, which is the robotic arm itself. The light green colour boxes are top-level tasks to implement. All tasks can be divided into many subtasks. In the following section, the reader will understand how the robotic arm was designed.

Design of robotic arm

For the robotic arm, I reused the existing solution, which was modified according to specified goals. The following list enumerates all subtasks that were important for this solution.

- **Sensors mounting** - The sensor board was made, but the original design of the robotic arm does not count with such a board. For this task, the mounting mechanism was designed for each stepper motor type in this thesis, so the sensor board can be attached to it.
- **Camera mounting** - The original robotic arm did not count with a camera, so this subtask fix it. The camera was mounted to the gripper where the servo motor is placed.
- **Fix problematic parts** - Some parts were not usable for my solution due to wrong diameters or design itself and some parts were designed for different stepper motor types. This task fixed the parts according to my goals.

After this task was finished, the robotic arm was completely modified according to my goals. The camera and sensor boards were able to mount to this new solution.

Electronics for robotic arm

The whole electronics part of the robotic arm solution should be divided into two main subtasks. The first is the main control board and the second is the sensor board.

- **Main control board** - responsible for connecting all sensors together, providing powering and communication. The subtasks for the main controlling boar are the following:

  - **Powering** - All components need power. This subtask was responsible for providing 3.3V for communication, 5V for powering the sensors and the stepper motor drivers' logic, 12V for small stepper motor drivers and 35V for big stepper motors.
  - **Communication** - The main controlling board provides two communication types. The UART is for communication between both processors (control and compute board) and I2C is for communication between the control board and sensor boards.
  - **Acutator** - The goal of this subtask was to connect stepper motors with the control board. The steps of the stepper motor can be controlled directly through the control board (ESP32), but due to the lack of a pin, the direction and resolution are controlled by three shift registers.

- **Sensor board** - For each stepper motor the encoder and accelerometer are used. To unify that the sensor board was designed. It connects both the encoder and accelerometer and provides a uniform way

After this whole task was finished, the robotic arm was completely prepared for the software.

Software

This task is focused on the whole program part of the robotic arm. It focuses on firmware for both processors, Graphical User Interface and communication.

- **compute processor** - this task was divided into firmware part and GUI:
  - **Firmware** - The compute processor is the Raspberry Pi. As a firmware the camera recognition, inverse kinematics and forward kinematics were made. Forward kinematics are used in combination with distance measuring for detecting objects in space.
  - **Graphical User Interface** - In this task, the intuitive user interface was made. It offers 3 ways of controlling each joint separately, semi-autonomous with a camera and the hand-driven mode. Everything was done by using the wx Python library.

- **Control processor firmware** - The goal of this subtask was to provide firmware for the control processor. The firmware covers the actuator controlling (stepper motors via timers), reading values from sensors, communication with the compute processor, stall detecting etc.

After this task, the robotic arm was fully functional and ready for the testing phase.

Test goals

In this task, I tested the previous three parts. All tests that were covered are mentioned in section 4.4.

5.1 Design of the robotic arm

The design part is one of the most important parts of the whole project. The reason is that it specifies the limitations of robotic arms, such as mobility, maximal payload, precision, price, etc. Because the design of the robotic arm is important and it should be functional, I decided to reuse the existing solution.

The reused robotic arm is the BCN3D MOVEO, described in section 2.3. There were many reasons to reuse exactly this robotic arm. The first reason is that this is a common robotic arm, offered as open source from company BCN3D and it is commonly used and tested by many users. The second reason is that it uses only stepper motors as actuators and servo motors as grippers.

Almost the whole robotic arm was usable for my purpose, except for small four parts that must be changed (except for the table used for robotic arm mounting and playground). Figure 5.3 shows these four changed parts. Part of this thesis was the covering box for the main control board, which can be seen in appendix C.

Figure 5.3: Visualization of the arm with an emphasis on changed parts

The changes were made, due to different stepper motor types, small space for other sensors or missing solutions for my goals, such as camera mounting and sensor mounting.

Sensor mounting

Figure A.1 shows the accelerometer, gyroscope and encoder sensor board, which are the two components that were designed completely by myself. All stepper motors have the same sensor board, due to a cheaper manufacturing process, so the mounting mechanisms,

specific for each stepper motor, must be designed. Finally, I made two holders, where the first was used for the biggest stepper motor NEMA23, and the second is used for the smallest stepper motor NEMA14 the rest NEMA17 stepper motors use only distances, which is good enough. The reason is the sensor board was designed primarily for NEMA17, so all holes in this sensor board match this type of motor. Except for the dimensions, both holders are technically the same. The main concept of it is to put it on the stepper motor and screw it.

Figure 5.4: Sensor boards for the Nema14 (the first) and the Nema23 (the rest)

Figure 5.4 shows how both holders look (in a cut) and how it looks on a stepper motor with a sensor board on it for better illustration. All parts were 3D printed with PLA material.

Camera mounting

The best place to place the camera was the gripper. It was well-designed, but the mounting mechanism for the camera was missing. Another improvement was made due to the different servo motors. The servo motor, used in my thesis has a different size so it was redesigned according to my servo motor.

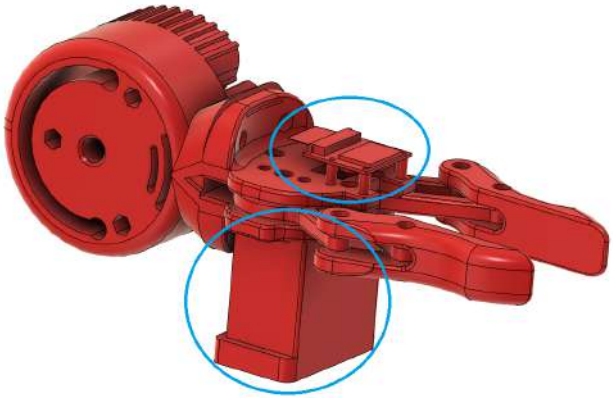

Figure 5.5: The modified gripper - changed servo motor and camera was mounted on it

The rest of the gripper itself remained unchanged. Figure 5.5 shows both changes on the gripper.

Problematic parts fix

The first problematic part was mounting the base stepper motor that allows the whole robotic arm to rotate. This part was designed exactly to one certain type of NEMA17 stepper motor and the hole for the connector was too small. The solution was increasing the hole size. Figure 5.6 shows the difference between the original part and my changed part.

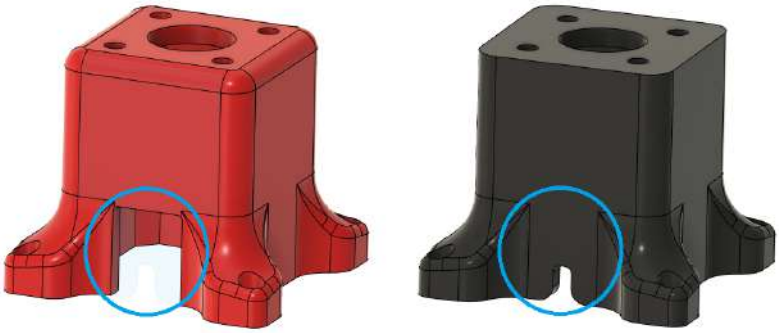

Figure 5.6: Modified base motor holder part on the left and original part on the right

Another part, that needed to be changed was link number 3, where three changes were made. The problems were only the space for the stepper motor and for the sensor board. So I made a bigger space inside for the stepper motor and for the sensor board as well. The hole for the connector was increased, due to the same problem that the first base link had.

Figure 5.7: The modified link 3

The last changed part was link number 4. This part had many problems to solve. The first is big friction between link 3 and link 4, where the connection was too tight and with no space between joints. The problem was solved by adding the distance column between joints. The connection itself was problematic too because when it was tightened together,

it was impossible to disassemble that. The solution was made of a hole that goes from the top to the bottom of joint 4 which allows me to put a screw instead of a regular instead of the previous solution which was the threaded rod.

Figure 5.8: The modified link 4 (red) and the original part (blue)

Figure 5.8 visualizes both mentioned changes on link 4. These changes make this component available to disassembly. The distance column reduced the friction between links.

5.2 Electronics for the robotic arm

The concept in chapter led me to decide to make my own main control board that will connect all necessary parts such as the main control processor, sensors, actuators, etc. Figure 5.9 below shows what features the main board should have.

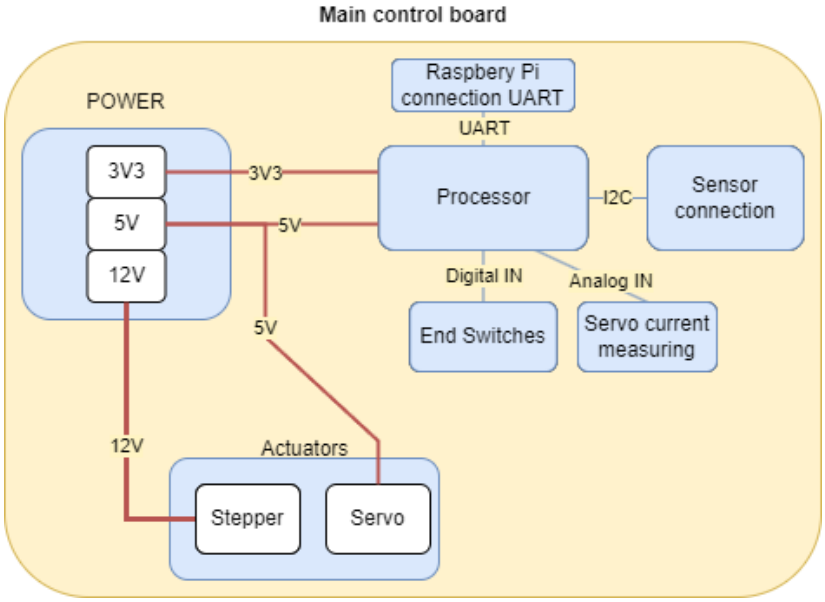

Figure 5.9: The main board implementation overview

Each block from the implementation overview above, such as power, sensors, processors and actuators is described in a more detailed way in the following sections. All final printed circuit boards, related to this section, can be found in appendix A.

Processors for robotic arm

As was mentioned in chapter 4.1, the concept was to divide tasks between two processors. The first control processor is for controlling all peripherals and devices and the second processor is for the camera, computing, AI, etc. The ESP32 WROOM32 was chosen as the main control processor, which is my personally preferred choice.

The second processor, used for computations and the camera, was the Raspberry Pi 4, which is a powerful solution for AI, and computations and offers me an easy solution to work with the camera. It is again my personal experience. Both processors are described in section 3.2.

The communication between both processors is realized by UART. It connects default pins (GPIO 3, GPIO 1 for ESP and GPIO 14 and GPIO 15 for the Raspberry Pi 4). The commands that are sent via UART are described in section 5.3. The pin mapping for ESP32 can be found in appendix B.

Sensors for robotic arm

Figure 5.9 shows, that the processor (ESP32) should communicate with sensors via I2C. The reason to choose I2C instead of SPI was an issue with the lack of free pins. The SPI needs at least 4 wires (MISO, MOSI, CLK and chip select), but I2C needs only 2 (SDA and SCL), so it was a cheaper solution.

Due to my decision to use one accelerometer, gyroscope and encoder for each actuator, the custom PCB, that connects all these sensors together, had to be done. This board contains MPU-6500, which covers the accelerometer and gyroscope, so one module covers two features at once and an AS5600 encoder module with 12b resolution.

Each sensor board contains the same set of sensors. The problem was with the same addresses, which should cause the circuit to short when at least two modules will communicate at the same time. Due to it being difficult to find any solution for my case, the only solution was to make another PCB that will switch devices on I2C manually. This board contains two multiplexors that connect both SCL and SDA pins. The final I2C switching board can be found in the appendixes.

The next goal is to measure the distance to the object. For that purpose, the GY-VL53L0X was chosen as a distance-measuring sensor. It has a different address so it can be connected to the I2C directly.

When the robotic arm is powered on, it does not know the actual position of the stepper motors. For that purpose the end switches were used, so the robot can be automatically set to a home position.

The last important sensor is the current sensor for current flows through the servo motor to determine if the servo is holding any object or not. The ACS712 sensor was the right solution in my case. Values can be read via an AD convertor, so no I2C communication was not needed.

Sensor type	Sensor name
Accelerometer and gyroscope	MPU-6500
Distance measuring	VL53L0X
Encoder	AS5600
Current sensor	ACS712

Table 5.1: Sensors used in this thesis

Table 5.1 above shows all sensors used for this robotic arm solution to make it transparent.

Actuators and drivers for robotic arm

All actuators used in my thesis were basically the actuators, that were used in the original BCN3D robotic arm. The only problem was finding exactly the same stepper motors. All I found was the same type and torque. Types such as NEMA14, NEMA17 and NEMA23, gave me the distances for mounting holes. So if the same stepper motor was found, it was automatically used, otherwise, the stepper motor with the same type but higher torque was used instead.

Another actuator used in this project is the servo motor. It is used for opening and closing gripper. It was not a problem to find the same servo motor, but due to my Bachelor thesis, where the servo was too weak, I decided to design it for DS3235SG digital servo. Which offers high torque of 35Kg.cm, which is strong enough. Table 5.2 below lists all actuators used in this project.

Part	Motor	τ [kg.cm]	I_{max} [A]	Motor ID
Joint 1	NEMA17 stepper motor	5	1.7	17HS8401
Joint 2	NEMA23 stepper motor	30.6	4.2	57HS11242A4D8
Joint 3	NEMA17 5:1 stepper motor	17.1	1.68	17HS19-1684S-PG5
Joint 4	NEMA17 stepper motor	2.85	1.3	17HS3401
Joint 5	NEMA14 stepper motor	1.84	0.42	PHB35Y34-401
Gripper	Servo motor	35	2.3	RDS3235

Table 5.2: Stepper motor used in this thesis

All mentioned stepper motors listed above need a stepper motor driver to be controlled. Stepper motors are described in a more detailed way in section 3.1. For all NEMA14 and NEMA17 stepper motors the Toshiba TB67S109 stepper motor drivers were chosen.

This driver was suitable for my use case because it is a „dump“ driver that has no stall detection implemented, so it can be programmed by me. Another reason is my personal bad experience with the cheapest A4988 driver, that was noisy, mostly failed and it had to be changed often. The last reason for choosing this stepper driver is its size, which is suitable because steppers that use this driver do not consume much current. This driver is not mentioned in section 3.1, because it is not used very often, but I wanted to try it.

The NEMA23 stepper motors needed stronger stepper motor drivers, such as HY-DIV268N-5A. This driver allows maximal current up to 5A, which is suitable for a big NEMA23 stepper motor, but in the end, it turned out to be a bad solution. It was difficult to find different drivers that would fulfil my goals but at last, I found the DM860H driver,

which is not used in any mentioned solution in 2.3, but it offers up to 7.2A, which is good enough.

To prevent stepper motor failure, it was necessary to add two fans on the top of the main control cover box. When the fans are not used, the drivers can overheat and can be destroyed. Two fans were used and were directly connected to the 12V branch via free pins next to the powering part. The fans can be seen in appendix C.

Powering for robotic arm

From all chapters above, three types of voltage levels are needed. The 12-volt branch is for the stepper motors powering, the 5-volt branch is for stepper motor logic values, ESP32 itself, Raspberry Pi and servo motor and the last 3.3V branch is for ESP32 as well and all sensors. The scheme for powering can be found in appendix B. The total minimal current for my application is approximately 12A for actuators, and up to 3A for Raspberry Pi. It is 15A for the most power-consuming components. The chosen power supply is **S-400-12V** which provides 12V/33A, where half of it is only the reserve.

Because in the end, my previous drivers were changed to different types as for what the main control board was designed, the two voltage regulators were added. The reason is that the previous drivers needed at least 8V for powering (my solution offered 12V), but the new type uses at least 24V. These regulators moved the voltage from 12V to 35V, which is good enough for new drivers.

5.3 Software for robotic arm

This section focuses on the software part of this thesis such as firmware and graphical user interface. The user can find there the design of the Graphical User Interface of my robotic arm, which describes how the GUI is designed and describes every single tab that the user can use. Another part is the firmware for the robotic arm, which explains what parts were necessary to implement, what features were implemented etc.

Graphical User Interface of robotic arm

The goals of my graphical user interface (GUI), specified in section 5.2 are intuitive appearance, ease to use and different ways of controlling. Another goal was to make GUI without a webserver to eliminate problems with multiple connections, bad connections, finding IP addresses etc. I decided to program it in Python programming language, because the Raspberry Pi, where the GUI is placed, can be controlled via Python, so it is a suitable option.

I made the GUI in wxPython, described in section 3.4, using the wxFormBuilder editor, which allows me to make it in an easy way by clicking on widgets without coding. The final GUI is described below.

Figure 5.10 shows the first GUI tab, that offers individual stepper motor controlling (joint controlling mode). For each stepper motor, the angular position can be set by one of five slider bars, and the gripper can be controlled via the button at the bottom as well. The image on the right is just an illustration for a fast understanding of which slider bar controls desired joint.

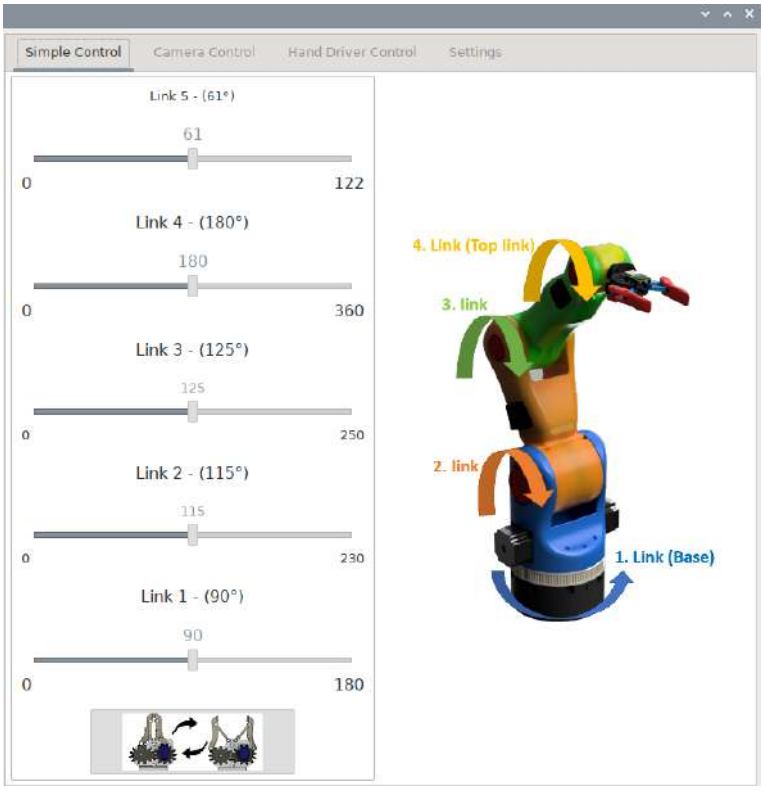

Figure 5.10: The first GUI tab used for manual controlling mode

Figure 5.11 below shows another GUI tab, which is the camera controlling tab. This tab is responsible for the visualization of the camera view and for the item recognition feature. In this tab, the user cannot control the robotic arm by itself. All that the user can do is scan the surroundings to detect all known objects around. Another useful feature is the move option at the bottom of the tab. This allows the user to specify coordinates in the space and the robotic arm tries to reach it. Every time the user sets at least one coordinate the 3D visualization is updated. The grab object and find object features do not work, due to time limitations.

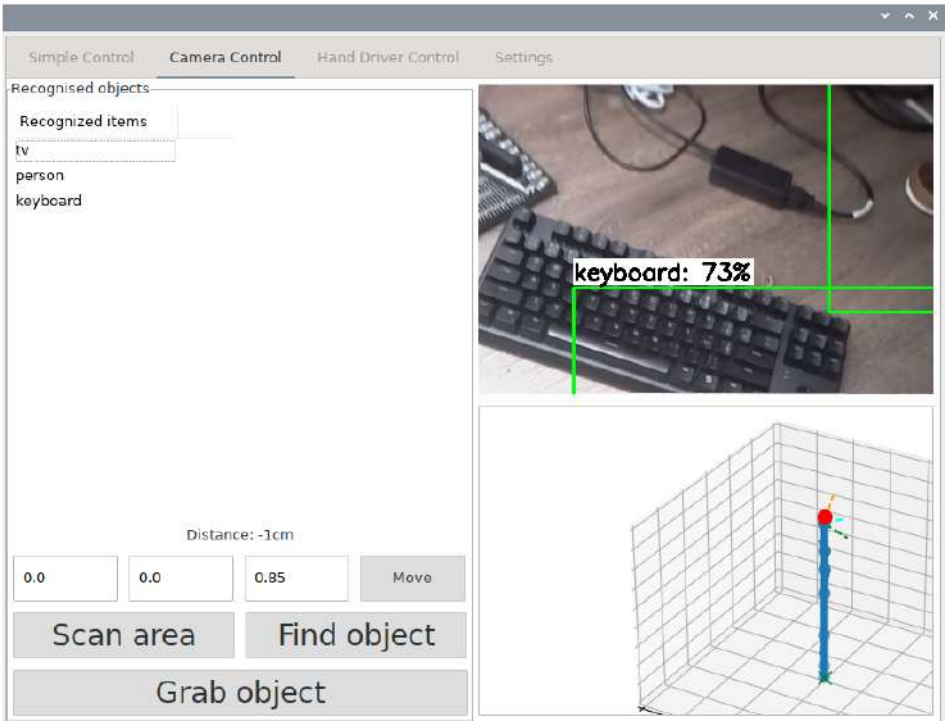

Figure 5.11: The second GUI tab used for camera controlling mode

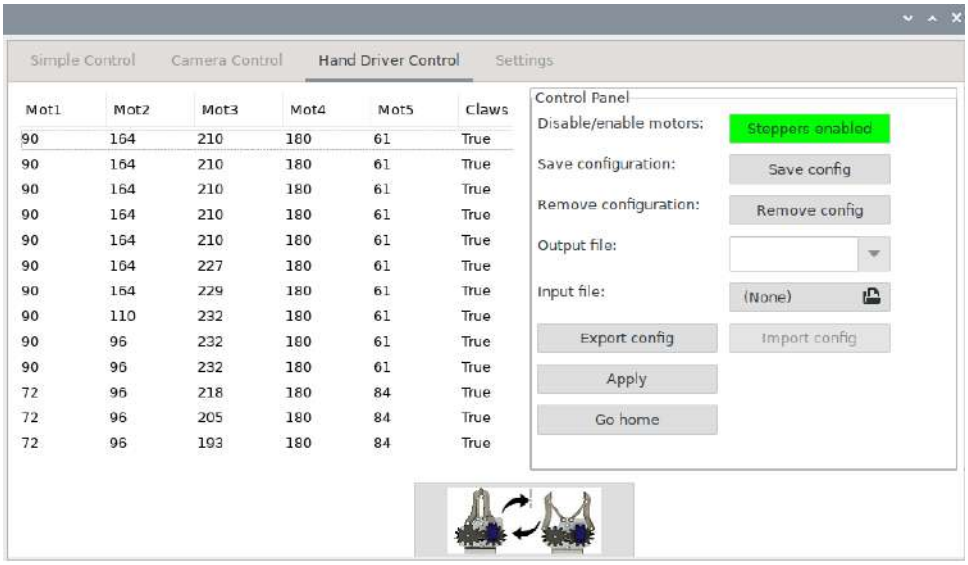

Figure 5.12: The third GUI tab used inverse kinematics

Figure 5.12 shows the last controlling tab. It can be used for position capturing. The tab is divided into two parts. The first contains a list, where all stepper motor positions and the state of the gripper (closed or opened) are stored. The second is the controlling part where the user can disable or enable stepper motors (disable for hand tracking), save and remove the configuration, import configuration from file or export configuration to file

and apply the full configuration set. The configuration is the row with 5 stepper motor angles and gripper status (true for opened grippers).

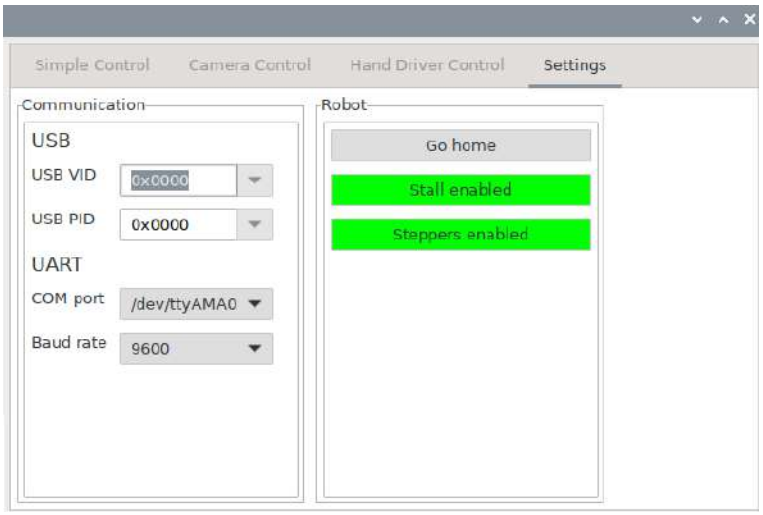

Figure 5.13: The settings tab

The last tab is used for settings. It allows the user to specify the COM port that is used for ESP32 to Raspberry Pi communication. The user can also set the robotic arm to a home position, enable or disable the stall detection feature and enable or disable stepper motors as well as in the previous tab.

Firmware for compute processor

The firmware for compute processor is located in the Raspberry Pi4 computer. The responsibility of this computer is to visualize GUI, communicate with the control processor, compute inverse kinematics and work with a camera. The whole firmware including GUI is stored in the following directory structure of GUI/src:

- **main_gui.py** - Frontend of the graphical user interface, generated from the wxFormBuilder tool (for Python)- it contains only the design with no functionality.
- **main_gui_ctrl.py** - Backend of the graphical user interface defined in main_gui.py. This file is not generated as the main_gui.py, but it is fully handwritten.
- **main.py** - The main part that allows running complete the GUI, by creating an instance of the GUI defined in main_gui_ctrl.py
- **communication.py** - A script for communication between Raspberry Pi and the ESP32. It is a wrapper for the PySerial Python module.
- **camera_control** - A script for basic camera item recognition used in main_gui_ctrl.py module.
- **ik_python.py** - A module for computation of the inverse and forward kinematics and visualization of the resolution. To compute the kinematics the IKPy Python module is used. All that it needs is just urdf file that contains a description of the robotic arm.

- **URDF** - Folder with defined urdf file for ik_python.py module.
- **Sample_TF-Lite_model** - Folder with important files for item recognition used in the second tab for camera control.

The Raspberry Pi should take care of updating the camera view and it should recognize if the control board detected stall. Both these features are handled by two timers, where the first timer is responsible for the camera, which updates the camera view only when the camera tab is selected to decrease the load. The second timer is responsible for stall detection and repeatedly checks the serial port for that purpose. When the control board sends information about the detected stall, the GUI pops up the message dialogue that provides the user with two possibilities to solve it, such as continuing or returning home.

Firmware for control processor

My firmware for control is located in ESP32, which is the main control board. The goal of this control board is to read all sensors, communicate with the main compute processor and control the entire robotic arm.

The most important part is the parallel **controlling of stepper motors**. To do so my solution uses the internal timers that trigger the timer interrupts. Every time the interruption occurs, the appropriate stepper motor makes a step. The advantage of this method is that before the interrupt occurs the rest of the code is not blocked. The problem was that the ESP32 has only 4 timers, so only 4 motors can be controlled via this method. This limitation led me to a solution, where one motor (the base motor) is controlled normally without interrupts and the rest with interrupts.

The interrupts must be precise, so there is no big time space for stall detection, which was one of my desired goals. Luckily the ESP32 has **two cores**, so the problem was solved by moving the whole firmware into two cores.

- **The first core** - used for the main loop that reads and parse data coming from UART from Raspberry Pi and controls the entire robotic arm.
- **The second core** - used for stall detection if the feature is enabled by the user.

Thanks to this method, the robotic arm can be controlled and the stall detection can be detected without mutual interference. The code snippet below shows what the interrupt method looks like for a given stepper motor.

```
void IRAM_ATTR on_motor2_timer(){
    //If all steps were done, stop the timer
    if(motor2_step_cntr <= 0){
        motor2_step_cntr = 0;
        stepper_go_home = false;
        timerAlarmDisable(motor2_timer);
        timerStop(motor2_timer);
    }
    //Change the polarity of the step pin
    if(!motor2_state){
        digitalWrite(MOT2_STEP_PIN, HIGH);
    }else{
        digitalWrite(MOT2_STEP_PIN, LOW);
        motor2_step_cntr--;
    }
    motor2_state = !motor2_state;
}
```

The stall detection is my other goal to design. It was previously implemented in two different ways. The first one was with encoders and the second was implemented with an accelerometer.

- **The encoder version** - Quite time-consuming for computing (for each joint the difference between the previous and actual position should be computed) and it was not reliable. The problem was that the range of the encoder was only 0 to 359 degrees. When the degrees overflowed (e.g. to 361) the encoder represented it as 1 degree, but due to almost all joints being geared, the original range is insufficient, e.g. when the joint has a gear ratio of 5:1, the stepper motor must make 1800 degrees for a joint full rotation.

  This problem should be fixed by the cumulative method, so when overflow occurs the difference will be computed and added to the cumulative angle. To give an example, the angle of 400 degrees, which would be represented as 40 degrees will now detect overflow and adds 40 (diff between 400 and 360) to 360. The only problem is that this value must be captured very often or the overflow can be missed and the stall detection would be useless.

- **The accelerometer version** - This version of stall detection is much easier to implement and it is quite reliable. All that is needed is to capture the accelerometer value and check if it exceeds the threshold value. When the threshold value is exceeded, the stall is detected. The only limitation of this version is that this detection cannot be applied for a full range of motion, because when the robotic arm starts moving or stops moving, the acceleration value is high and it will be indicated as a stall.

The arguments above led me to the decision to implement stall detection via an accelerometer. The following code snippet shows how the stall detection is captured for the base link.

```
//Check stall detection only if the feature is enabled from GUI (disabled by default)
if(stall_enabled){
  //Check it only when it is time to detect stall (the joint is in the range where the
      detection is allowed)
  if(arm.actuators.check_stall(1)){
    //Check if the accelerometer value is greater than the threshold
    if(abs(arm.sensors.get_accel_value(0).x) > 7500){
      stall_detected = true;
    }
  }
}else{
  stall_detected = false;
}
```

Another goal to implement was the detection if the gripper grabbed any object. For that purpose, the ACS712 current sensor, mentioned in section **Sensors for my robotic arm solution** is a good choice. While the gripper is closing, the value from ADC that converts the value from the current sensor is compared against the threshold value. When the value is exceeded, the gripper is holding an object.

Figure 5.14: Visualization of gripper state (fully opened, closed with an object, fully closed)

Figure 5.14 Shows three states of the gripper. The first state is open (the most left figure), closed gripper holding object (middle figure) and fully closed gripper (the most right figure).

```
void BobArmActuators::claw_close( void ){
    //Change gripper claws position tick by tick (from fully opened to fully closed)
    for(int i = 0; i < 60; i++){
        myservo.write(i);
        //Wait 5ms to slow the motion
        delay(5);
        //If the current exceeds the threshold value, open the claws by 3 ticks to
        //Reduce tension and stop the claws
        if(sensors.get_servo_current() > 1580){
            myservo.write(i-3);
            break;
        }
    }
}
```

The code snippet for object detection is mentioned above. The servo starts closing and when the current value exceeded the threshold value, the servo is slightly opened to reduce tension and the servo stops closing.

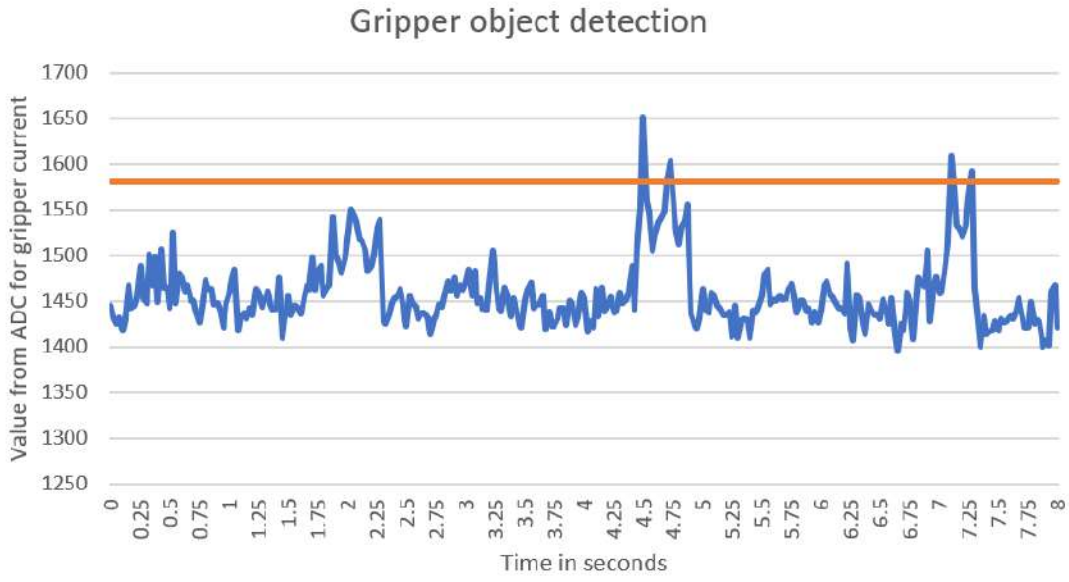

Figure 5.15: Result for my gripper object detection

Figure 5.15 above shows the captured values for the gripper. The test used for this graph was toggling the gripper (repeat open and close) every 500 ms, so the object was put twice into the claws space and it successfully detected the object. The values are quite noisy, which can be caused, in my opinion, by surrounding noise, or changing the direction of the stepper motor. The threshold value is represented by the orange line. As was mentioned the object was put two times into the claws, but in the figure, there are four spikes. The second lower spike can be caused, in my opinion, by changing direction after it grabbed the object. So the current rises and the gripper is slightly opened to reduce tension when the object is grabbed. This causes a current decrease, but due to the gripper starting to open, the current raises again and caused a second spike.

5.4 Testing of the robotic arm

The chapter 4, describes all the goals that my robotic arm solution should have and the chapter 5, shows the implementation of that goal. These goals are what define my own solution.

This section focuses on testing all mentioned goals to determine which of them were successfully implemented and which were not. All the tests mentioned in this section will follow the testing procedure, specified in section 4.4. The following list shows all tests, tested in this thesis:

- Total weight of the robotic arm
- Payload test
- Accuracy test of the robotic arm
- Repeatability test of the robotic arm
- Stall detection test of the robotic arm
- Gripper object detection
- Test of the GUI

Due to time limitations on this thesis, not all suitable tests were realized. For example, a long-duration test if the robotic arm is capable to run a few hours, days, etc. Another possible test can be automatic GUI testing instead of hand testing.

Total weight of robotic arm

The total weight of my entire robotic arm solution is 13,6kg and as was mentioned in section 4.4 this value is not bad. This is just good to know value. My solution does not contain any heavier parts, than what the original BCN3D MOVEO has, so it can be assumed that the weights of both solutions are quite similar.

Payload test

This test was made in iterations as mentioned in section 4.4. The test started with an object of weight 100g and in each iteration, the payload was increased by 100g. The following table shows the tested weight and its results. The criterion for this test was very simple. When half of the robotic arm stepper motors can handle a given load, the test will be considered successful.

Object weight [g]	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5
100	Passed	Passed	Passed	Passed	Passed
200	Passed	Passed	Passed	Failed	Failed
300	Passed	Passed	Passed	Failed	Failed
400	Passed	Passed	Passed	Failed	Failed
500	Passed	Passed	Passed	Failed	Failed
600	Passed	Passed	Passed	Failed	Failed

Table 5.3: Payload test of my robotic arm solution

Table 5.3 shows that the robotic arm lifts up to 600g. The only problem was that around 200g, the last joint (the joint that holds the claws) and the fourth joint had a problem getting through its middle position, due to the load. The rest of the robotic arm joints had no problem. My personal goal was 300g, which the last two joints did not fulfil, but the rest of the joints did. So this test can be considered successful.

Accuracy test of the robotic arm

In this test, the robotic arm was set into a specific position and the real coordinates of the end effector was measured. The result is the difference between the measured coordinates and the target coordinates. Table 5.4 shows the distances the differences between the goal and real position. Coordinates are given in meters.

Target position [m]	**Reached position [m]**	**X diff [m]**	**Y diff [m]**	**Z diff [m]**
0.0; 0.0; 0.8	0.0; 0.0; 0.81	0.0	0.0	+0.1
0.03; -0.5; 0.1	0.029; -0.50; 0.08	-0.01	0.0	-0.02
0.13; -0.3; 0.2	0.125; -0.31; 0.185	-0.005	-0.01	-0.015

Table 5.4: Accuracy test for translation motion

Table 5.4 shows that the maximal error between the target position and the computed position was 20mm on the Z axis and 10mm on the other axis. The precision of measured value could be affected by the human factor because all values were measured by meter and ruler and weak link 5. This test was only an informative test, so it can not be considered a successful or failed test.

Repeatability test of the robotic arm

This test did the same as what the previous Accuracy test of the robotic arm test did, but it was repeated 10 times, 15 times and 20 times. All of that was made for translation and rotation motion as well as accuracy tests. The goal of this test was to find accuracy between each repetition. In all cases, the worst accuracy was +-1mm. This test was only an informative test, so it can not be considered a successful or failed test.

Stall detection test of the robotic arm

I put the robotic arm in a default straight position and then moved to a different position. The goal was to stop the robotic arm when the obstacle was put into the trajectory.

This part was complicated and the stall detection did not work properly with the encoders. Sometimes it detected a stall even when the stall did not happen and sometimes it detect nothing when the stall should occur. The solution with an accelerometer worked properly when the robotic arm was standing still and hit by an object. When the robotic arm was moving, it sometimes detected stall even when stall did not happen. After com

The stall detection for the servo (gripper), which indicates the gripper is holding some object, worked. When the gripper was closing and no object was between the claws, the claws completely closed themselves. When the object was put in the middle of the claws, the servo stops closing to prevent increasing the current through the servo and protect it from damage. This test can be considered as successful.

Gripper object detection

I run the basic program, where the gripper was toggled, so when the claws were opened, it closed it etc. When the object was put in the claws' space, the gripper stopped. Figure 5.15 shows values captured during the testing phase. Due to that, the test can be considered as successful.

Test of the GUI

This test was only visual. The Graphical User Interface was run many times and every time I checked if all widgets, such as buttons, radio buttons, edit bars, slider bars, etc are visible and they work as they should. This test was successful, because all widgets were visible, when the widget should be disabled or enabled, it worked. Every time the widget was used I checked if the behaviour is correct as well and it worked too. This test can be considered as successful.

Test conclusion

To make it clear, the following list shows the result of all tests made in this chapter. It contains the name of the test above and the final result of it.

- **Total weight of robotic arm**: 13.5kg
- **Payload test**: 100g for all stepper motors and more than 600g without two last links
- **Accuracy test**: +-20mm in Z axis and +-10mm for other axis
- **Repeatability test**: -+1mm
- **Stall detection test**: successful
- **Gripper object detection test**: successful
- **Test of the GUI**: successful

All tests were successfully made and almost all results are satisfactory, except for the accuracy test. The problem could be caused by a weak fourth joint, which is quite inaccurate and it could cause big differences.

Chapter 6

Conclusion

The goal of this thesis was to make my own robotic arm solution, by using RC components and servos. This goal was successfully fulfilled.

I made my own robotic arm solution that reused the existing BCN3D MOVEO robotic arm design, which was modified according to my needs. The robotic arm uses an accelerometer for stall detection and encoders for detecting actual stepper motors positions that can be stored for further repetition. The gripper uses a current sensor to detect if the robotic arm is holding any object. The robotic arm has a camera and distance-measuring sensor attached to the gripper mechanism. The camera is used for object detection and recognition via the OpenCV Python library. The distance meter is used for measuring the distance to the desired object to get its position in space. Despite these features the automatic grabbing object, detected by the camera, is not implemented due to time limitations.

The whole solution is divided into two processors, such as ESP32 for controlling the whole robotic arm and the Raspberry Pi4 computer for computations and for GUI, which was designed as well. The Graphical User Interface allows the user to work with a robotic arm in three different ways. The first is controlling each joint separately via slider bars. The second option is by using a camera view and object recognition on it. With this option, the user can detect objects around the robotic arm. This option offers the user to specify a point in 3D and the robotic arm will automatically reach this position. The last possibility how to control the robotic arm is the capturing mode, where each configuration(forward kinematics) can be captured and stored for further autonomous repetition.

Future work includes the motion of the robotic arm should be smoother, automatical object grabbing can be finished, tests can be improved, by adding long-term tests and cable management could be improved. The problem is that each sensor or stepper motor requires wires and the robotic arm design is quite messy. The reason why these tests were not made is the time limitation of this thesis.

Bibliography

[1] *Encoder Communications Handbook*. Virginia, USA: Dynapar, 2007 [cit. 28. December 2022]. Available at: https://www.dynapar.com/hubfs/uploadedFiles/Downloads/Encoder%20Communications%20Hanbook.pdf.

[2] *PWM DAC Using MSP430 High-Resolution Timer*. Texas, USA: Texas Instruments Incorporated, 2011 [cit. 18. December 2022]. Available at: https://www.ti.com/lit/an/slaa497/slaa497.pdf.

[3] *PyQt tutorial*. India: Tutorials Point, 2015 [cit. 19. January 2023]. Available at: https://www.tutorialspoint.com/pyqt/pyqt_tutorial.pdf.

[4] *WxPython GUI TOOLKIT*. India: tutorialspoint, 2015 [cit. 19. January 2023]. Available at: https://www.tutorialspoint.com/wxpython/wxpython_tutorial.pdf.

[5] *Cost-effective Open Source fully 3D printed Robot Arm*. Spain: BCN3D, 2016 [cit. 17. January 2022]. Available at: https://www.3dprinter.ee/media/3dprinter/Juhendid/usecases/BCN3D%20moveo%20-%20Cost-effective%203D%20printed%20Robot%20Arm%20-%20BCN3D%20Technologies.pdf.

[6] *Forward Kinematics*. Maryland, USA: United States Naval Academy, 2016 [cit. 2022-12-23]. Available at: https://www.usna.edu/Users/cs/crabbe/SI475/current/arm-kin/kinematics.pdf.

[7] *Technical Explanation for Servomotors and Servo Drives*. Japan: Omron Corporation, 2016 [cit. 13. December 2022]. Available at: https://www.ia.omron.com/data_pdf/guide/14/servo_tg_e_1_1.pdf.

[8] *TOSHIBA BiCD Integrated Circuit Silicon Monolithic TB6600HG*. Japan: Toshiba, 2016 [cit. 1. May 2023]. Available at: https://toshiba.semicon-storage.com/info/TB6600HG_datasheet_en_20160610.pdf?did=14683&prodName=TB6600HG.

[9] *Closed Loop Stepper Motor Design With Encoder for StallDetection Reference Design*. Texas, USA: Texas Instruments Incorporated, 2017 [cit. 28. December 2022]. Available at: https://www.ti.com/lit/ug/tiducy6/tiducy6.pdf?ts=1672239654234.

[10] *ESP32-WROOM-32 Datasheet*. China: Espressif Systems, 2019 [cit. 12. January 2023]. Available at: https://www.laskakit.cz/user/related_files/esp32-wroom-32_datasheet_en.pdf.

[11] *Niryo One User Manual*. France: Niryo, 2019 [cit. 15. January 2022]. Available at: https://www.generationrobots.com/media/niryo-one-user-manual-03-09-2019.pdf.

[12] *Raspberry Pi Pico and Pico W*. England, UK: Raspberry Pi Foundation, 2021 [cit. 18. December 2022]. Available at: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf.

[13] *A4988*. New Hampshire, New Mexico: Allegro MicroSystems, 2022 [cit. 1. May 2023]. Available at: https://www.allegromicro.com/-/media/files/datasheets/a4988-datasheet.pdf.

[14] *AR2 3D Printed Robotic Arm* [online]. Netherlands: Wevolver, 2022 [cit. 15. January 2022]. Available at: https://www.wevolver.com/specs/ar2.3d.printed.robotic.arm.

[15] *Introduction to Stepper Motors* [online]. California, USA: Omega Engineering Inc., 2022 [cit. 10. December 2022]. Available at: https://www.omega.co.uk/prodinfo/stepper_motors.html.

[16] *Ned2 User Manual*. France: Niryo, 2022 [cit. 15. January 2022]. Available at: https://docs.niryo.com/product/ned2/v1.0.0/generated_pdfs/pdf_en.pdf.

[17] *Stepper motor basics* [online]. Germany: Faulhaber, 2022 [cit. 10. December 2022]. Available at: https://www.faulhaber.com/fileadmin/Import/Media/AN001_EN.pdf.

[18] *Arduino® UNO R3*. Italy: Arduino®, 2023 [cit. 20. April 2023]. Available at: https://docs.arduino.cc/static/a597adc8f334878e612824e2bc210e6e/A000066-datasheet.pdf.

[19] *TMC2130 DATASHEET*. Germany: TRINAMIC Motion Control, 2023 [cit. 1. May 2023]. Available at: https://www.trinamic.com/fileadmin/assets/Products/ICs_Documents/TMC2130_datasheet_rev1.16.pdf.

[20] *TMC2208/2 & TMC2224 family Datasheet*. Germany: TRINAMIC Motion Control, 2023 [cit. 1. May 2023]. Available at: https://www.trinamic.com/fileadmin/assets/Products/ICs_Documents/TMC2202_TMC2208_TMC2224_datasheet_rev1.14.pdf.

[21] Andreas, A. and Joan, L. *Inverse Kinematics: a review of existing techniques and introduction of a new fast iterative solver*. England, UK: University of Cambridge, 2009.

[22] Annin, C. *AR4 Robot Manual*. Idaho, USA: Annin Robotics, 2022 [cit. 17. January 2022]. Available at: https://drive.google.com/file/d/1T7tWd5-ZMWTbNEdSN-Tvi7uAK0rFrwcS/view.

[23] Apoorve. *What is a Servo Motor? - Understanding the basics of Servo Motor Working* [online]. India: Circuit Digest, 2015 [cit. 13. December 2022]. Available at: https://circuitdigest.com/article/servo-motor-working-and-basics.

[24] Bobčík, P. *Robotické rameno s modelářskými servy*. Brno, CZ, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/22369/.

[25] Cubero, S. *Industrial Robotics*. Rijeka: IntechOpen, 2006. ISBN 3-86611-285-8. Available at: https://doi.org/10.5772/44.

[26] Earl, B. *All About Stepper Motors*. New York, USA: Adafruit Industries, 2022 [cit. 10. December 2022]. Available at: https://cdn-learn.adafruit.com/downloads/pdf/all-about-stepper-motors.pdf.

[27] Fiore, C. *Stepper Motors Basics: Types, Uses, and Working Principles.* Washington, USA: MPS, 2022 [cit. 2022-09-12]. Available at: https://media.monolithicpower.com/mps_cms_document/2/0/2020-stepper-motors-basics-types-uses-and-working-principles_r1.0.pdf.

[28] Halfacree, G. *THE OFFICIAL Raspberry Pi Beginner's Guide How to use your new computer.* England, UK: Raspberry Pi Trading Ltd, 2020. ISBN 978-1-912047-73-4. Available at: https://magazines-attachments.raspberrypi.org/books/full_pdfs/000/000/038/original/BeginnersGuide-4thEd-Eng_v2.pdf.

[29] Hamza, S. A. E. *The Common Use of Pulse Width Modulation "PWM" Technique in Power Electronics.* Sudan: Al Neelain University, 2020 [cit. 16. December 2022]. Available at: https://www.ijsr.net/archive/v3i11/T0NUMTQxMTYw.pdf.

[30] jha, P. *Inverse Kinematic Analysis of Robot Manipulators.* India: National Institute of Technology, 2015 [cit. 18. December 2022]. Available at: https://core.ac.uk/download/pdf/80147797.pdf.

[31] Kunal, S. *WORKING OF ROBOTIC ARM IN INDUSTRIES.* India: Vel Tech - Technical University, 2021 [cit. 19. January 2023]. Available at: https://www.researchgate.net/profile/Kunal_Singh12/publication/353478116_WORKING_OF_ROBOTIC_ARM_IN_INDUSTRIES/links/60ff66b20c2bfa282a02f109/WORKING-OF-ROBOTIC-ARM-IN-INDUSTRIES.pdf.

[32] Lara, M. M. . I. *Differences Between Optical and Magnetic Incremental Encoders.* Texas, USA: Texas Instruments Incorporated, 2022 [cit. 28. December 2022]. Available at: https://www.ti.com/lit/an/slya061/slya061.pdf.

[33] Mitra, D. *Sensorless Stall Detection With the DRV8889-Q1.* Texas, USA: Texas Instruments Incorporated, 2020 [cit. 27. December 2022]. Available at: https://www.ti.com/lit/an/slvaei3/slvaei3.pdf.

[34] Nilsson, R. *Inverse kinematics.* Luleå, EE, 2009. Master's thesis. Luleå University of Technology. Available at: http://www.diva-portal.org/smash/get/diva2:1018821/FULLTEXT01.pdf.

[35] Pinewski, P., Reiter, J. and Borland, G. *S12HY and S12XHY Stepper Stall Detect.* Netherlands: Freescale Semiconductor, 2010 [cit. 28. December 2022]. Available at: https://www.nxp.com/docs/en/application-note/AN4024.pdf.

[36] Rouse, M. *Pulse Width Modulation* [online]. GEngland, UK: Techopedia, 2022 [cit. 18. December 2022]. Available at: https://www.techopedia.com/definition/9034/pulse-width-modulation-pwm.

[37] Sabhadiya, J. *What Is Servo Motor?- Definition, Working And Types* [online]. Engineering Choice, 2022 [cit. 13. December 2022]. Available at: https://www.engineeringchoice.com/what-is-servo-motor/.

[38] Shipman, J. W. *Tkinter 8.5 reference: aGUI for Python.* New Mexico, USA: New Mexico Tech, 2013 [cit. 19. April 2023]. Available at: https://tkdocs.com/shipman/tkinter.pdf.

[39] Syed, N. R. *Inverse kinematics using the Jacobian inverse, part 2.* San Francisco, USA: nrsyed, 2017 [cit. 23. December 2022]. Available at: https://nrsyed.com/2017/12/10/inverse-kinematics-using-the-jacobian-inverse-part-2/.

[40] Söderby, K. and Hylén, J. *Getting Started with Arduino IDE 2.0.* Italy: Arduino, 2022 [cit. 20. December 2022]. Available at: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2.

[41] Yimeng Shi, H. L. *Beginner's Guide for Raspberry Pi Pico.* Singapoure: SeeedStudio, 2021 [cit. 18. December 2022]. Available at: https://files.seeedstudio.com/wiki/Grove_Shield_for_Pi_Pico_V1.0/Begiinner%27s-Guide-for-Raspberry-Pi-Pico.pdf.

[42] Zach, P. *Uživatelské rozhraní pro polohování robotické ruky BCN3D MOVEO.* Czech Republic: Czech Technical University, 2020 [cit. 17. January 2023]. Available at: https://dspace.cvut.cz/handle/10467/90355.

Appendix A

Printed circuit boards

Design of the sensor board for accelerometer and encoder module. First figure is the final design with all treces and other two figures are the final product.

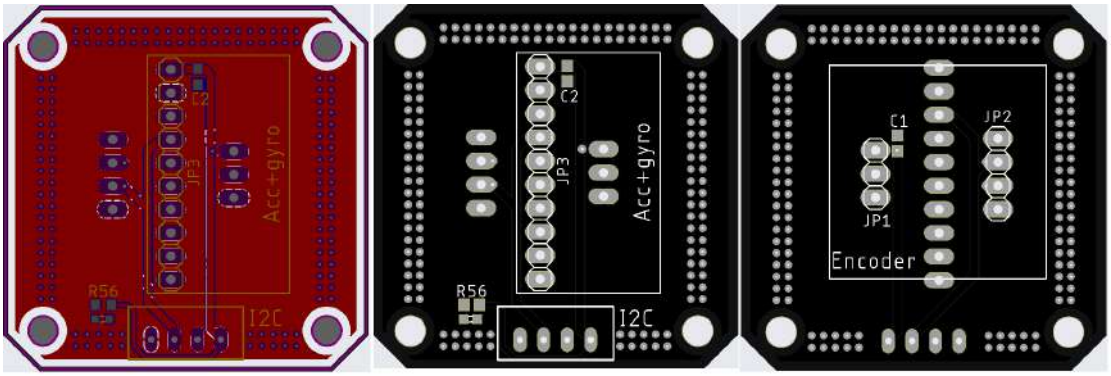

Figure A.1: The sensor board with accelerometer, gyroscope and encoder

Design of the switching board used for switching I2C between sensor boards. The left figure is the final design with all traces before realization and figure on the right is the final product.

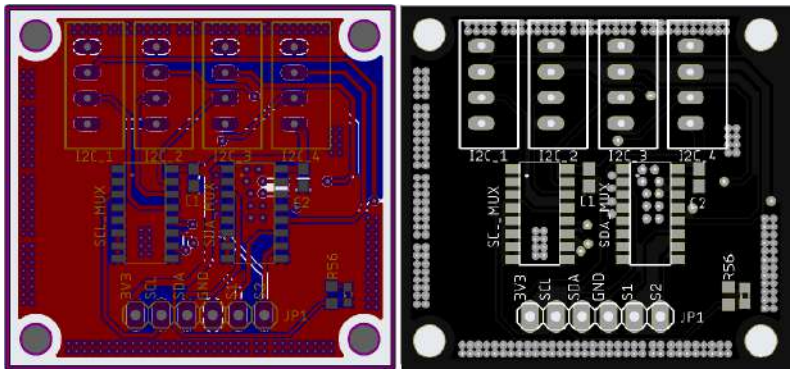

Figure A.2: The switching board for switching sensors in I2C

The following figure shows the design of the main control board with all traces. It also indicates four main parts of this board with color rectangle. The blue for powering

part, the green is for the stepper motors (stepper motor drivers), servo and shift registers for resolution and direction. The yellow rectangle is a slot for the main control processor, which is ESP-32. The last black rectangle indicates the input sensor part with end sensors and the UART communication.

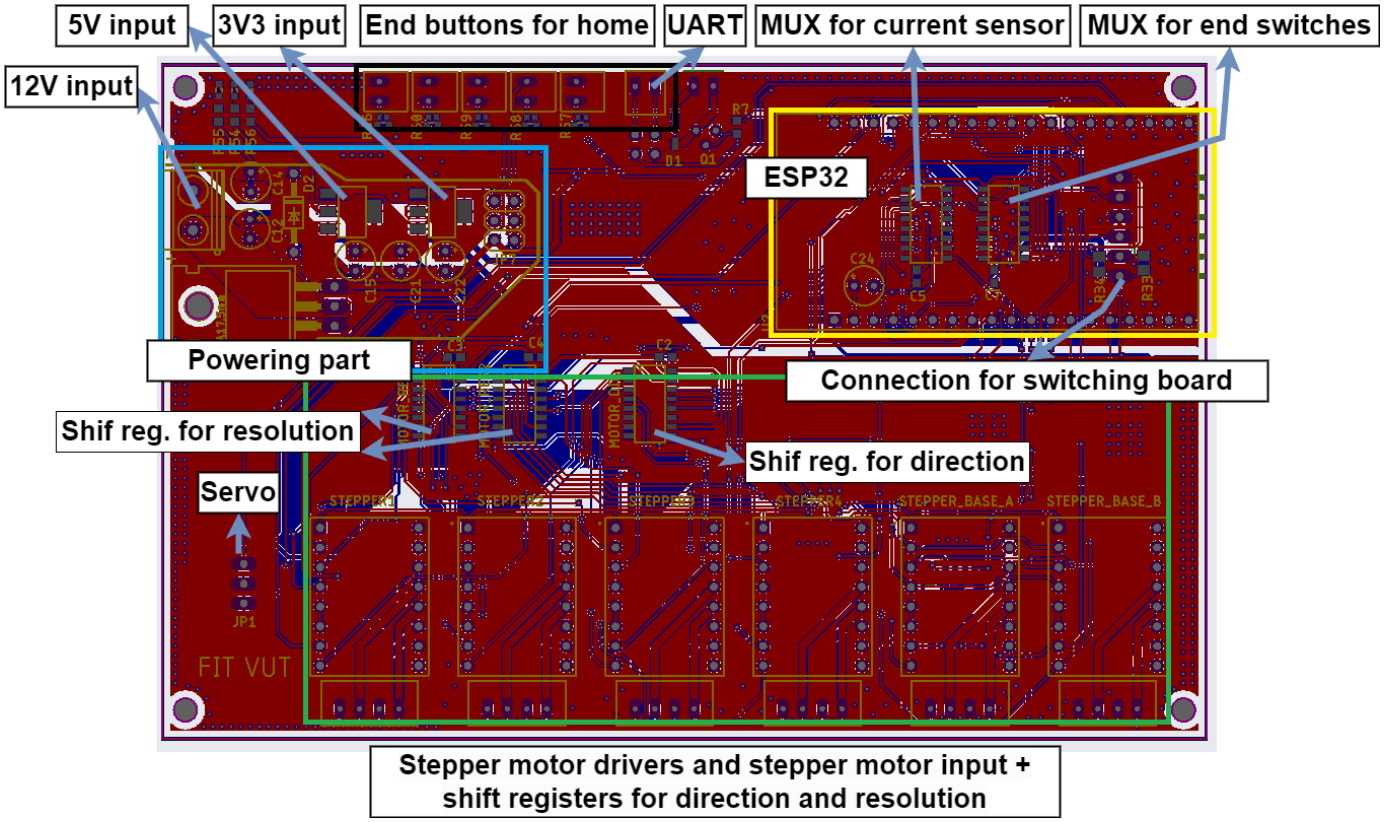

Figure A.3: The main control board

Appendix B

Printed circuit boards

Schema for main control board powering. There are three branches such as 12V as input and then 5V and 3V3.

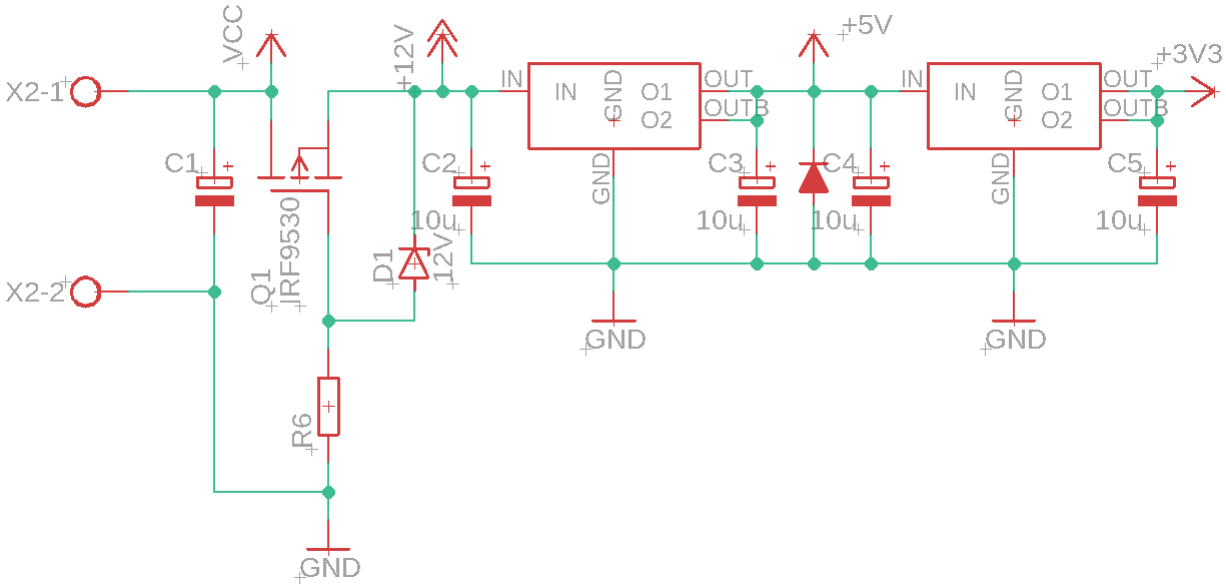

Figure B.1: The schema of the main controlling board powering

Pinout for the ESP32 located in main control board.

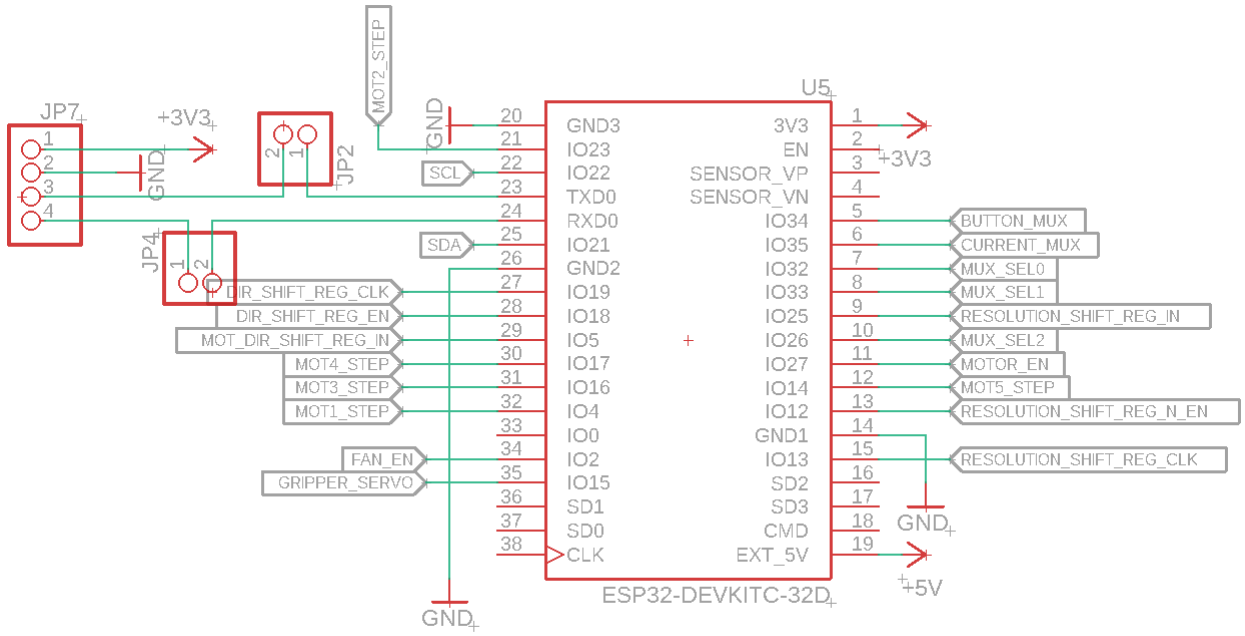

Figure B.2: The ESP32 connection

The shift registers fixed problem with lack of pins, because resolution and direction for all stepper motors are controlled by three shift registers.

Appendix C

Main board cover

The main control board was placed in the cover box with fans.

Figure C.1: The covering box for main control board

Appendix D

Final realization

The final realization of the robotic arm described in this thesis.

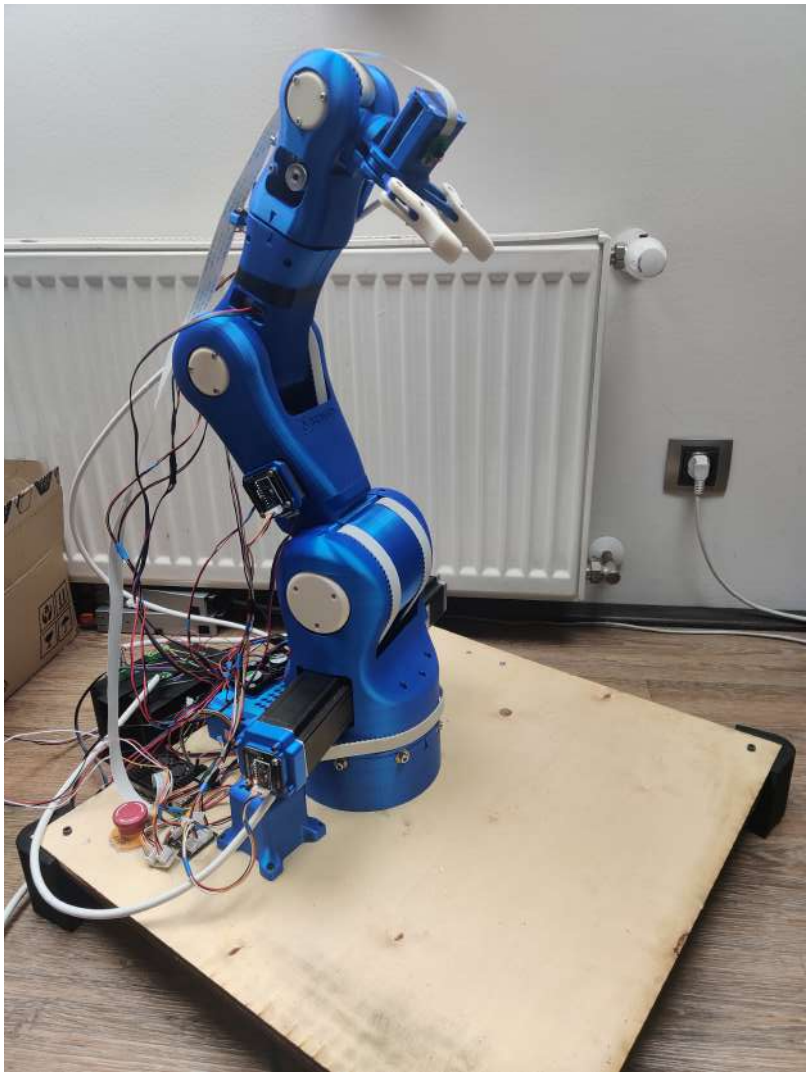

Figure D.1: The final result of the thesis