mng rämibühl

Mathematisch-Naturwissenschaftliches Gymnasium

Microcontroller-Driven Robotic Arm with Software Integration

Author: Alejo Restrepo

Supervisor: Christoph Vogel

Matura paper in computer science

4c MNG Rämibühl

January 8th 2024

Abstract

This paper exhibits the design and construction of a six-degrees-of-freedom robot arm using consumer electronics and affordable materials in order to emulate the positioning capabilities of an industrial robot arm. To achieve this, an optimal mechanical prototype is modeled and built, using servo motors as the main actuators. Furthermore, two mathematical control methods are proposed: a trigonometric exact inversion and an approximate local inversion via derivatives. These methods are implemented into an open source C++ library. The implementation successfully performed an inverse kinematic analysis, allowing for two-dimensional trajectory control and pivoting of the end-effector using input devices. A series of user interface and stability tests were conducted with positive results in precision and repeatability in most scenarios.

Contents

1 Introduction

1.1 Objectives

This project has two main objectives: 1. To design and build a robot arm prototype with six degrees of freedom (forward/back, up/down, left/right, yaw, pitch, roll) and 2. To program both approximate and exact control algorithms for maneuverability, as well as a solution for pivoting the end-effector. The end result is a proof of concept that the same movement and control capabilities of an industrial robot arm is possible using only consumer electronics and affordable materials.

1.2 Inspiration

I have had a great fascination with robots and robotic systems since I was very young. The idea of automating such complex tasks to a degree of perfection comparable to human dexterity has always intrigued me. Seeing how modern-day industrial robots perform has always led me to wonder if I would be able to design and build a system that could come close to the capabilities of one of these machines using cheap components I could get online in electronic kits.

1.3 Project Overview

The next chapters are structured the following way:

The second chapter provides a theoretical background on robot arms and robotics as a whole. Furthermore, the inner workings of servo motors as well as microcontrollers are explained. Subsequently, the theory of kinematics, which is the basis behind the two main methods used in the project to control the robot is presented.

The third chapter delves into the design and construction process of the robot arm, both from a hardware and an electronic point of view. First, the mechanical design of the prototype is explained, taking into account optimal weight distribution and physical limitations such as the torque of the motors. Next, the process of building and assembling each joint of the robot arm is detailed. At the end, a graphic design of the structure and a diagram of the wiring between the microcontroller and the motors are shown along with a joystick, the main input device.

The fourth chapter presents the software solutions for controlling the robot. Starting with a forward kinematic analysis, two methods for calculating the rotation of the joints having given the position of the end-effector are introduced. The first method is a trigonometric solution for the positioning of the robot's end-effector. The mathematical equations and geometric proof behind this method are explained. The second method is a derivative solution that calculates the needed change in joint angles when the position of the end-effector is displaced using vector and matrix representations. Additionally, a generalization to the algorithms is made to allow pivoting the end-effector. Finally, the solutions obtained through these methods are then implemented into C++ , creating necessary classes and structures for the motors and joints, making them functional for the Arduino microcontroller.

In the fifth chapter, a series of tests are conducted using participant "pilots" to compare and evaluate the performance of the robot arm when maneuvered using the input device. Test individuals were first instructed to pilot the robot arm along a horizontal, a vertical, and a diagonal path. Then, they were asked to assess the robot on criteria such as precision, fluidity of movement and input delay. A second test was conducted to see if the individuals were capable of piloting the arm towards a target object and tipping it over. Finally, the accuracy of the pivoting solution is measured for both methods via a stability test which does not need a participant pilot .

The sixth and last chapter interprets the results described in the fifth chapter and underlines the possible sources of the problems encountered. Next, steps in the hardware and software development process where an improvement could have been made, are explained. Finally, a final reflection on the paper is given, analyzing the overall cost of the project as well as the learning experience that was gained throughout the process.

2 Theoretical Basis

2.1 A brief introduction to robotics

Robotics as a field of science has been mainly driven with the goal of automating manual tasks that may be repetitive or dangerous for humans. Early industrial robots, dating back to the 1960s, were big stationary machines that aided factories in tasks that required a significant amount of physical effort or precision. As an example, the first industrial robot arm was introduced at the General Motors plant in order to automate the diecasting of automobile parts (Moran, M.E., 2007).

The development of robots has since spread past industrial applications, including regions like healthcare, with the daVinci Surgical System (DiMaio et al., 2011), space exploration robots such as the mars rovers (Maurette, M., 2003), and many other domains. The evolution of intelligent systems, machine learning and advancements in sensors have expanded the impact of robotics on humanity, as robots have become more capable of making complex decisions, learning and interacting with the environment.

2.2 Robot arms

The most common type of robot is the robotic arm. It is typically programmed to perform tasks that mimic the dexterity and range of motion that a human arm possesses. The robot arm's links are connected together by joints, providing linear motion such as in a cartesian robot (*Fig. 1*) , or rotational motion, as may be in an articulated robot (*Fig. 2*) (Bejczy, A. K., 1974).

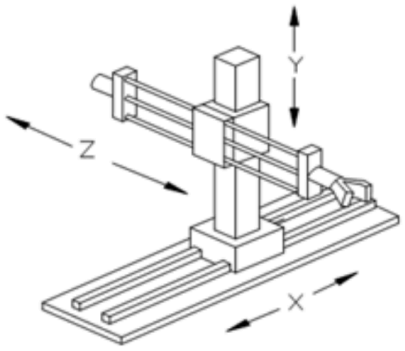

Figure 1.

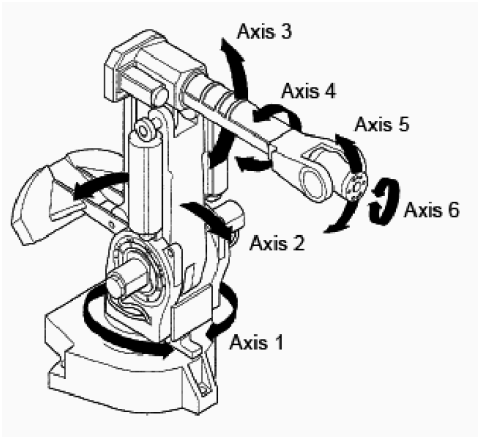

Figure 2.

A regular industrial robot arm has six degrees of freedom, enabling movement in six different ways: forward/back, up/down, left/right, yaw, pitch and roll. For comparison, a human arm has seven degrees of freedom (Kim et al., 2011). Just as a human arm has the function of moving the hand from one place to another, the purpose of a robotic arm is to move its end-effector to different locations.

Tesla's factories use robot arms to perform difficult and repetitive tasks such as welding the chassis of their cars along an assembly line (Casgains Academy, 2021). In order to give the robot instructions, a handheld controller is often used. It stores a sequence of events, then executes them every time it's needed. In this process the main actuator of the arm has a very important role, as they are equipped with advanced sensors that communicate with the robot and tell it where it is.

For more intricate tasks such as tightening bolts, the motors need to receive very accurate feedback from its sensors in order to adjust the robot's actions and ensure consistent movements. This precision is especially important in the computing industry, where the production of microchips and circuit boards relies on the accurate nature of servo motors (Tanaka, T., & Honda, Y., 2016).

2.3 Servo motors

Servo motors are a type of electrical motor which use continuous feedback from sensors to operate (Firoozian, R., 2014). A user can instruct the motor to rotate to a certain position and the controller inside the motor will send power to the motor until it reaches that angle. Servos have a constant feedback loop with their sensor that makes adjustments to the motor while it's rotating.

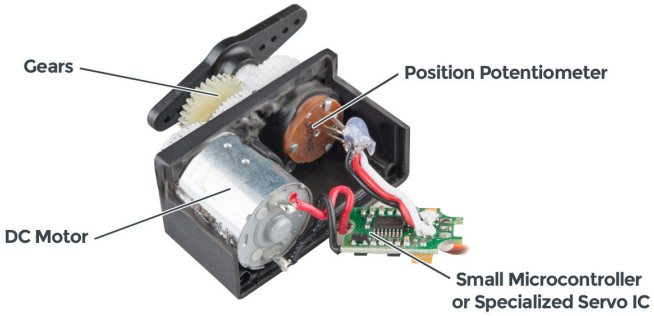

Figure 3.

As illustrated in Figure *3*, a servo motor is formed by a DC motor, gears, a feedback device and a control circuit. To activate the motor, an input signal must be received by the control circuit. The motor then propels the gear train, which is connected to the output shaft and to an electrical positioning sensor, usually a potentiometer. Positional sensors not only allow tracking the rotation of the servos output shaft, but also transmit feedback error signals to the control circuit. This process facilitates adjusting the power to minimize error in the desired rotation angle.

Figure *4* depicts this feedback loop. The motor responds to an input signal and adjusts its output position based on the feedback of the actuator's sensor.

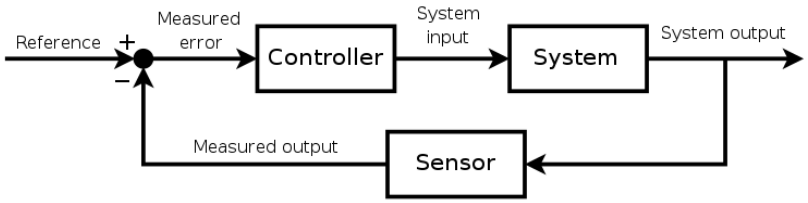

Figure 4.

Given that servo motors allow the precise control and movement needed, they are the perfect choice for the robot arm in this project. Furthermore, they provide a very high torque-to-cost ratio, with motors in the price range of 10-30 CHF having 20 kg/cm torque or more.

2.3.1 Pulse width modulation (PWM)

The following explains how servos are controlled. Most servos do not have the ability to turn continuously, but are limited to 180°. This is due to the use of simple potentiometers as positional feedback devices. Often, these potentiometers are not able to rotate continuously. Continuous rotational servo motors are available, but use a different positional feedback device. In professional servo motors, an encoder is almost always used instead of a potentiometer, even if continuous rotation is not required.

Servo motors require the voltage to be regulated through a method called “Pulse width modulation” or PWM in short. PWM is a method which switches the voltage from the power source from an “ON” state to an “OFF” state repetitively. The period of time which the power is switched on is called a pulse.(Sun, J., 2012).

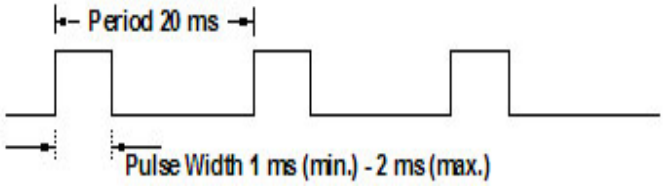

Figure 5.

Normally, the pulse rate of most servos is between 1 and 2 milliseconds, this means that the power is switched on, then after 20 milliseconds it is switched back off again. The servo motor constantly calculates if it needs a longer or a shorter pulse of voltage in order to move the output shaft to the desired position (*Fig. 5*).

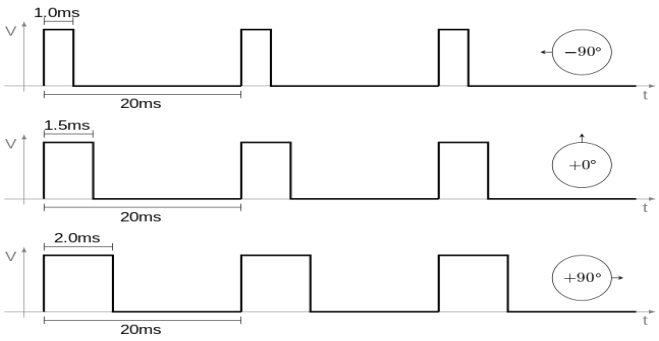

Figure 6.

The position of the motor is dependent on the width of the pulse. As most servos are restricted to a maximum +90° of rotation, the neutral position of the servo would be in the middle, so 0° and the minimum position -90°. A neutral pulse is 1.5ms long and results in the servo rotating to the neutral position. The minimal and maximum pulse lead the servo to rotate to the minimum and maximum positions respectively (*Fig. 6*).

2.4 Microcontrollers

Microcontrollers are an integrated circuit that contain a processor core, memory core and IN/OUTPUT pins (Barret, S., & Pack, D., 2006). Microcontrollers are designed for embedded systems and can execute user programmed instructions in order to perform various tasks and functions. They are commonly used in electrical appliances, automotive systems and open-source electronic prototypes platforms like Arduino boards.

2.4.1 Arduino

According to Louis (2016, p. 21), the Arduino platform was introduced in 2005 to provide affordable and simple electronic materials for constructing and programming electronic devices. This open source computing platform is supported by simple microcontroller boards that can be programmed and reprogrammed easily at any time using programming languages such as C, C++ or Python (Arduino, S. A., 2015). The platform can also act as a mini computer, taking inputs and controlling the outputs for the creation of a variety of electronics devices such as the robot arm prototype in this project.

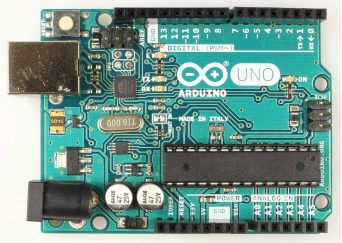

Figure 7.

2.5 Kinematics

Kinematics deals with the study of the movement of an object without considering the forces that cause it. Here, the position, velocity and acceleration of the object relative to time, are the main variables considered.

2.5.1 Forward Kinematics

In forward kinematics the end-effector position is determined using the lengths and angles of each individual joint. In this project, forward kinematics was used in order to understand the relationship between the end-effector's location and the joint angles (Kucuk, S. & Bingul, Z. 2006).

2.5.2 Inverse Kinematics

Inverse kinematics is the complementary process of forward kinematics. It determines the needed joint angles and movements in order to allow the end-effector to reach a specific location or orientation in space, later this process will be used to derive the trigonometric solution for the joint angles.

3 Hardware

3.1 Synopsis

The goal of this section is to describe the mechanical construction process of our robotic arm prototype. The end product has six degrees of freedom and working dimensions large enough to allow it to reach and manipulate inanimate objects within a given workspace. The robot arm will have three vertical joints allowing for movement along the vertical axis, as well as three rotational joints, which pivot along the horizontal axis.

3.2 Mechanical design

The initial implementation of the robotic arm prototype uses six servo motors. Each motor rotates one the joints along its respective axis. The focus of this section is detailing the thought behind the mechanical design of the 3 main vertical joints, the base joint, and the rotational joints.

3.2.1 Conventional 6DOF robotic arm joint distribution

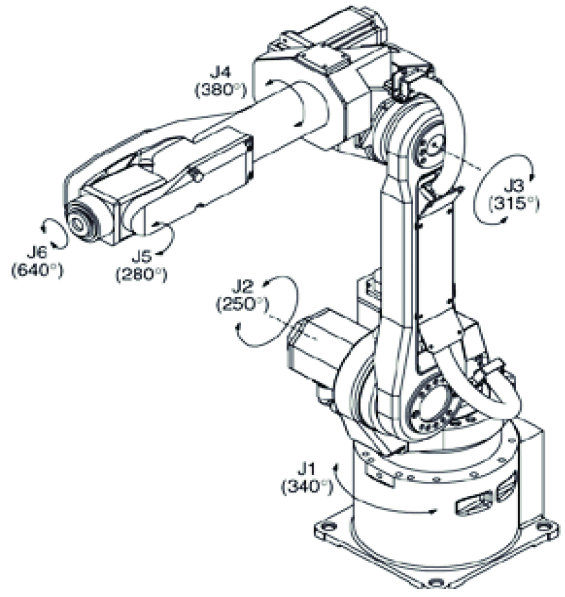

Figure 8. (Fanuc Robotics, 2008)

Base Joint: the base joint (*J1 from Figure 8)* provides a stable foundation for the entire robotic arm. It is designed to rotate around a horizontal axis, it allows the arm to pivot from side to side. The base joint is equipped with the strongest servo motor, having 40 kg per cm of torque, since this joint supports the weight of all other joints. Vertical joints: the three main vertical joints (*J2, J3, and J5 from Figure 8)* of the robotic arm are crucial for achieving a wide range of movements. They are commonly named after human anatomy parts, J2 being the shoulder joint, J3 the elbow joint and J5 the hand joint. These joints are designed to rotate across the vertical axis, enabling the arm to move up, down, forward and back. Each vertical joint consists of a servo motor connected to a rigid linkage mechanism.

Rotational Joints: The further two axes (*J4 and J6 from Figure 8)* revolve and swivel the forearm and hand joints allowing it to move freely. Rather than the vertical and horizontal joints, which move the robot to a specific position, these further two joints allow the end-effector, the last point

of the arm, to rotate just like a human wrist can rotate a hand and allow its fingertips to reach any point in space with any orientation.

3.2.2 Design prototype

Ensuring the stability of the base is vital in the design of the robotic arm prototype. The base balances the whole structure and prevents it from tipping over. To achieve this, the base should be flat and cover a large area, while being firmly connected with the servo of the first joint. A wide base plate made of durable materials will enhance the stability of the robot. For materials, 2 mm thick plywood sheets are used. Several types of servo motors are used, with different model names and torque for each joint (*Table 1*).

Table 1. Servo motors specifications

	Joint 1 (J1)	**Joint 2 (J2)**	**Joint 3 (J3)**	**Joint 4 (J4)**	**Joint 5 (J5)**	**Joint 6 (J6)**
Servo Model	WOAEIUOS	GXServo	MG996R	MG996R	MG90S	MG90S
Torque (kg/cm)	40	40	10	10	2.2	2.2

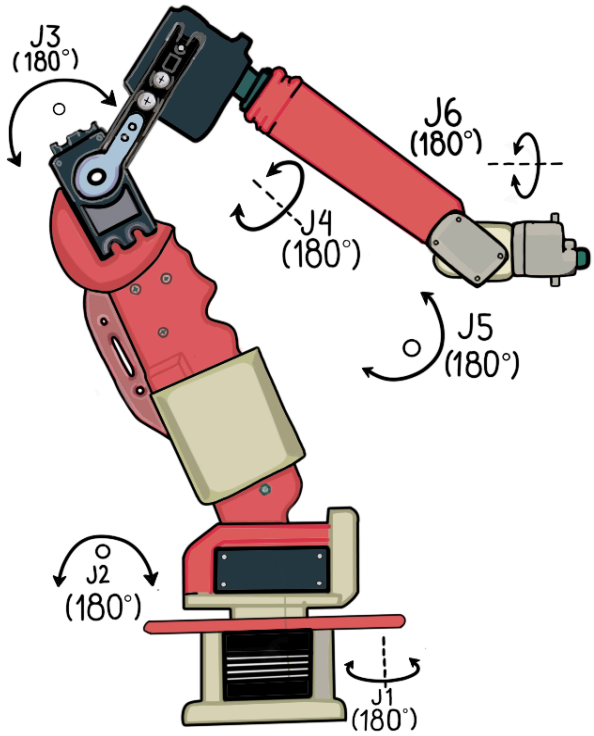

Figure 9. Sketch of prototype design. (own work)

In order to make the plywood sheets harden, the sheets are cut to the desired size and glued one on top of each other with a cyanoacrylate-based adhesive solution (instant glue). This process binds the sheets together and solidifies the wood completely, as the glue solves itself through the fibers of the wood sheet.

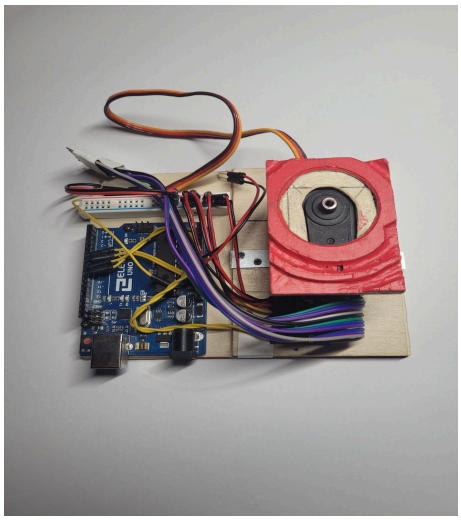

Figure 10. Base Plate, upper view with Arduino

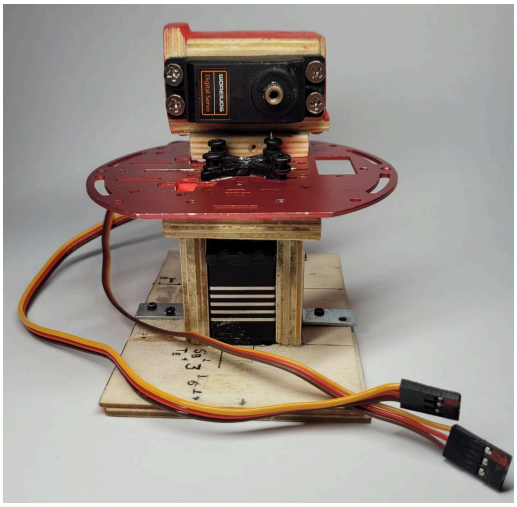

Figure 11. Base Plate front view with shoulder servo

The base plate consists of three solid parts, a flat slab that is attached with the base servo (*Fig 10 and Fig 11*) and a housing structure that surrounds the servo. The later structure has a flat upper section that allows the metallic servo output shaft to rotate along it minimizing vertical oscillation.

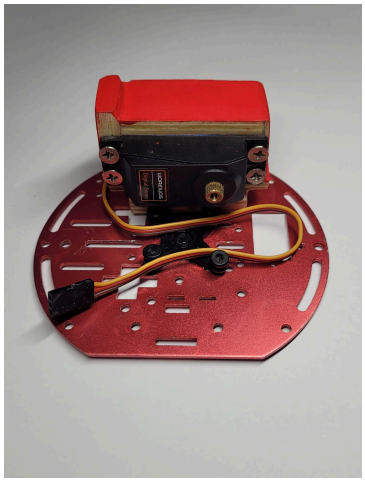

Figure 12. Shoulder servo front view

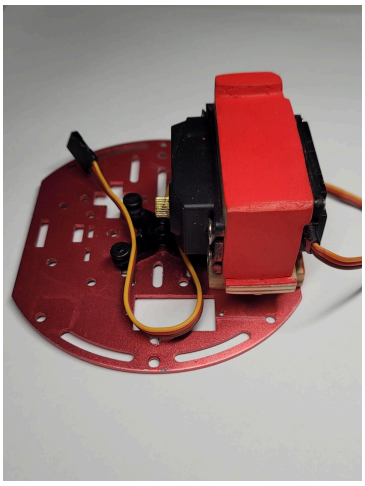

Figure 13. Shoulder servo side view

On top of the flat upper section is the shoulder servo, secured by several layers of plywood housing to prevent oscillation (*Fig 12, 13*).

The output shaft of the servo is made out of aluminum and is attached to the metal body (*Fig 14*), having a length of 18.7 cm from the shoulder to the elbow servo joint. This metal body structure is reinforced by several layers of cut plywood, having been molded to the exact shape of the aluminum skeleton to reduce weight. These layers are secured by M3 (3 mm diameter) bolts and hexagonal nuts.

Figure 14. Metal body front view

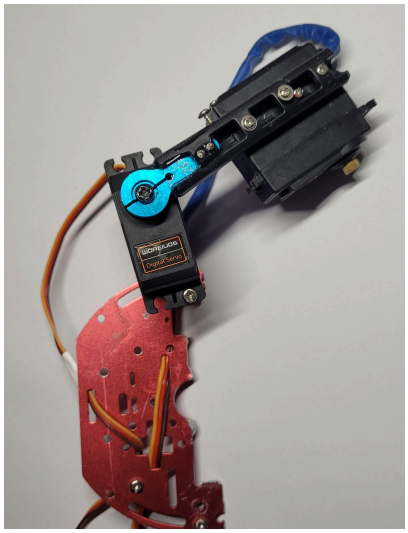

Figure 15. Metal body connected with elbow servo

Attached to the elbow servo (*Fig. 9, J3*) there is a plastic extension structure (*pictured here in Fig. 15*) that allows another servo to be the second rotational joint (*Fig. 9, J4*). This servo will rotate the rest of the forearm and the last two joints (*J5, J6, from Fig 9, pictured in Fig. 16, Fig. 17*), being powered by micro MG90D Servos (*see Table 1*).

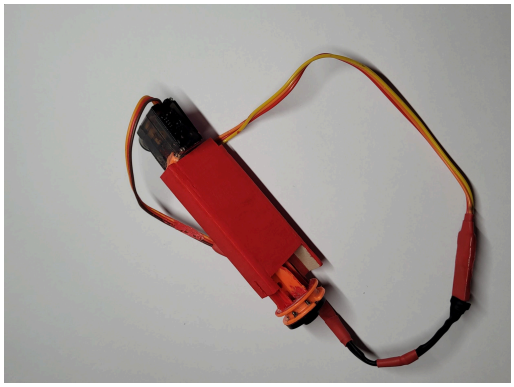

Figure 16. Forearm joint

Figure 17. Hand joint

While assembling the final parts, every joint needs to be secured in with bolts, taking into account that every servo motor is correctly aligned to the joint. Below is a true to scale graphic with the nomenclature for the parts and joints (*Fig. 18*).

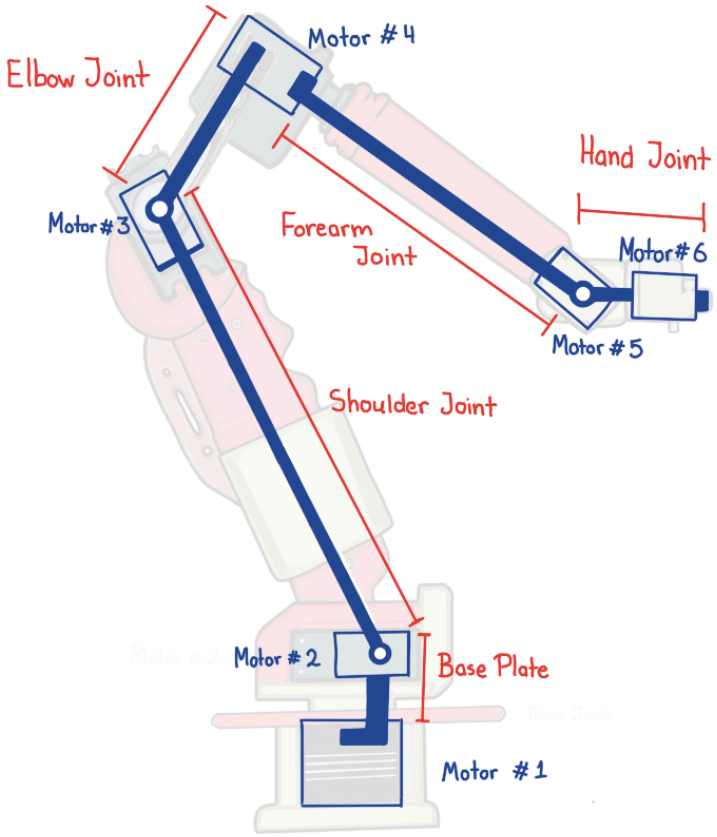

Figure 18. Design of full robot arm structure with nomenclature (own work)

Table 2. Measurements and weight of the vertical joints and the base plate

	Weight	**Length**
Hand joint	16.403g	6cm
Forearm joint	31.707g	15cm
Elbow joint	65.393g	6.7cm
Shoulder joint	70.255g	18.7cm
Base plate	112.573g	3cm

3.2.3 Vertical displacement at the second joint

A very important modification to the mechanical design that needs to be taken into account in order to develop the software for movement and trajectory calculation is the noticeable displacement at the second vertical joint (*J3, Fig. 9*), that elevates the fourth servo (*Motor #4, Fig. 18*) 6.7 cm (*Table 2, elbow joint length*) above of the third servo axis (*Motor #3, Fig. 18*) and rotates it by 90°. This results in the relative range of the elbow joint, which we can call θ_2, going from 0°-180° to 90°-270°.

Regardless of the absolute range staying the same (180°), because of the displacement, the elbow joint does not go into the shoulder joint, so it allows the robot to completely fold into itself and avoid colliding with its own structure. This is vital not to constrict the possible range of motion on the 2-dimensional plane. The maximum extended range is also made longer, as at its maximum stretch the robot has 6.7 cm more forward reach.

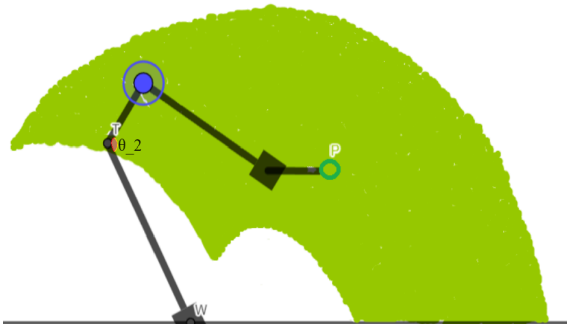

Figure 19. work range when $\theta_2 < 180°$

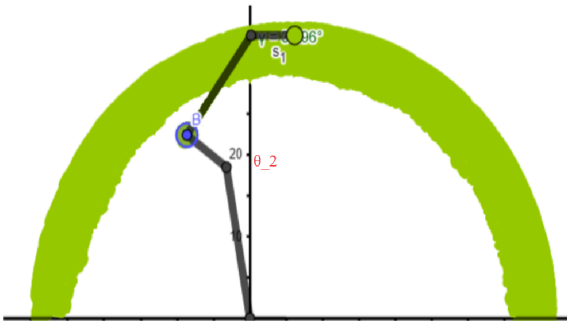

Figure 20. work range when $\theta_2 > 180°$

3.2.4 Assembly and cable routing

Each of the servos has three cables, the first one (often yellow in color), is for the signal input of the servo. The other two cables are for the power supply of the motor. One is for the 5 volt (5V) input, the other to ground the voltage (GND). They are routed into either the Arduinos default 5V and GND pins, or an external power supply, for example four AA batteries connected in series. This arrangement creates a voltage of 6 volts, well inside the servo's operating range. They are colored red and black for 5V and GND respectively. The cables need to be routed efficiently through the body of the robot arm, as to not impede the range of motion. This is an important step and thus requires the cables to be let with a certain slack between the joints to accomplish this. After all cables are efficiently routed into the base using jumper cables, all the signal wires from the servos need to be connected to the output (OUT) signal pins of the Arduino. In order to get the most working torque out of the motors, an external power supply is needed to provide each one of them with sufficient voltage and to not overload the Arduino.

3.2.5 External power routing

The Servo motors for the base, first and second joint have much greater load on them, so their power supply comes from an external source, and not the Arduino itself, to allow them to exert maximal torque and also to not overload the Arduino. Four AA batteries are connected in seires, to reach a voltage of 6 volts. Another four AA batteries are used and a parallel circuit is formed between the two sets, having the same voltage of 6 volts, but double the amps.

$$Watts = Amps \times Volts$$

Amps multiplied by volts equals watts, which is the measurement used to determine the amount of energy. The higher the wattage is, the more power and output from the motor (Cummings, P. G. et al, 1981). The last three servo motors are supplied by the Arduino with power, all three connecting their 5 V and GND pins to the respective slots of the Arduino.

3.2.6 Final circuit board diagram

The circuitry on the robot looks like this, viewing it practically, with all the cables routed into their correct power sources and analog pins. Compared with the theoretical circuit board diagram, made using an open source software called TinkerCad that shows the routing of each cable in a very compact form (*Fig. 21*), as it allows for an easier overview of the whole circuit.

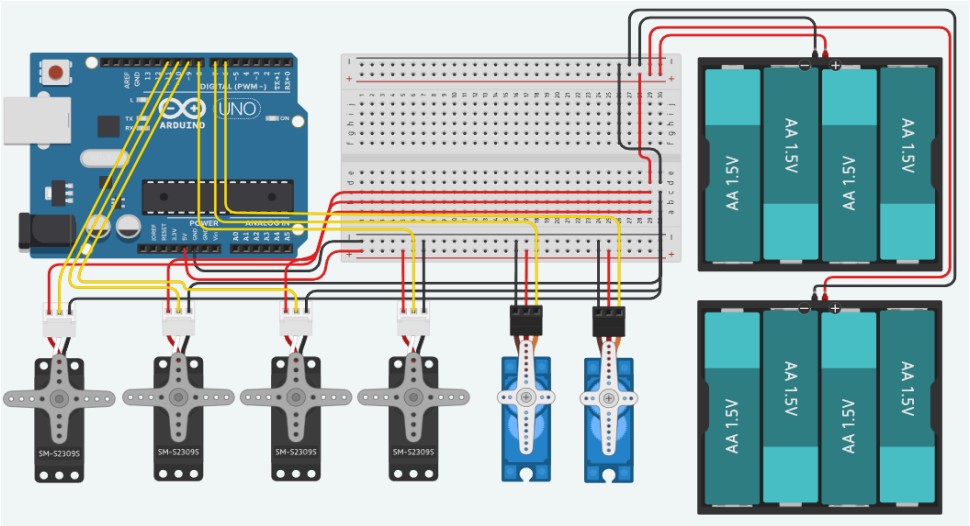

Figure 21. Circuit designed with TinkerCad (own work)

3.2.7 Input devices

The main input devices to allow the user to control the robot in an easy and intuitive way are 2-axis joystick modules. Joystick is composed of two potentiometers positioned square with each other, and one push button. It provides two values, an x-coordinate and a y-coordinate, running from 0 to 1023, which represents where the joystick is currently placed. The neutral position of the joystick would then be at the middle of this x-y grid, roughly at the (x, y) coordinate (512 , 512). The joystick module has five pins, a 5V and a GND pin to provide it with current from the Arduino and two x and y value pins, which connect to the analog pins A0 and A1 of the Arduino respectively. Finally the push button pin of the joystick, marked with “SLN” is connected to a digital Arduino pin, allowing it to read the state of the push button, HIGH or LOW (on or off).

4 Software

4.1 Synopsis

In this section, the main focus shifts towards the intelligent control and coordination of the robot arm. This involves developing inverse kinematic algorithms that allow precise and efficient manipulation of the end-effector position and orientation. We will propose and implement two methods, as well as a solution for end-effector pivoting, and evaluate each one in chapter *5* on parameters such as accuracy, smoothness, repeatability and user interface usability.

4.2 Forward kinematic analysis

Forward Kinematics, as explained in section *2.5*, refers to determining the position of the end-effector with the given values for the joint angles.

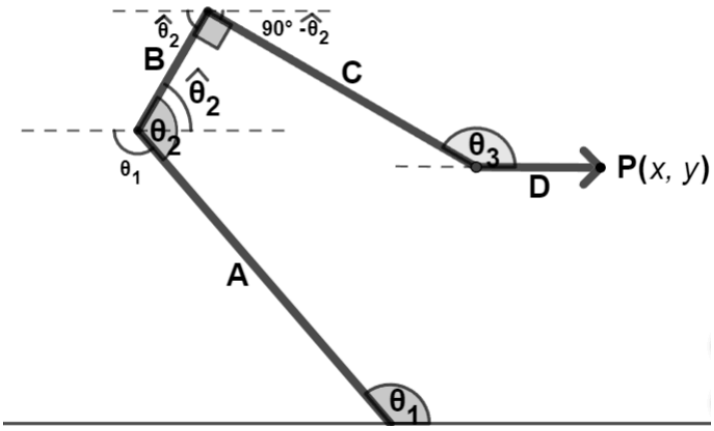

Figure 22. Vector graphic of the arm with joint angles

We can derive the two equations for the position of the end-effector by using geometric analysis. In particular, the position of the effector is determined by the angles θ_1 and θ_2.

We can trace a line parallel to the base which goes through θ_2. This allows θ_2 to be defined in terms of θ_1, as they are alternate angles. and a more useful angle $\hat{\theta}_2$. We express θ_2 as:

$$\theta_2 = \hat{\theta}_2 + 180° - \theta_1$$

Using this new angle $\hat{\theta}_2$along with the existing θ_1, we can now add up the horizontal and vertical components of the joint vectors *A*, *B*, *C*, *D* using cosine and sine functions in order to write equations for the endpoints *x* and *y*. In the following analysis, for simplicity, we will assume that the joint is in a horizontal position. In section *4.5*, however, we will extend the solution to the case in which the arm joint is rotated, by means of a simple transformation.

$$x = A \cdot cos\,\theta_1 + B \cdot cos\,\hat{\theta}_2 + C \cdot cos\,(90°\text{-}\,\hat{\theta}_2) + D$$

$$y = A \cdot sin\,\theta_1 + B \cdot sin\,\hat{\theta}_2 - C \cdot sin\,(90°\text{-}\hat{\theta}_2)$$

In angles (90° - θ) there is a relation between all trigonometric ratios. It is by definition, that cos (90° - θ) = sin θ and sin (90° - θ) = cos θ. We can rewrite our equations as:

$$x = A \cdot cos\theta_1 + B \cdot cos\hat{\theta}_2 + C \cdot sin\hat{\theta}_2 + D$$

$$y = A \cdot sin\theta_1 + B \cdot sin\hat{\theta}_2 - C \cdot cos\hat{\theta}_2$$

Furthermore, our definition of $\theta_2 + \hat{\theta}_2 + 180° - \theta_1$ from above, can be transformed into $\hat{\theta}_2 = \theta_1 + \theta_2 - 180°$ and inserted. We now have our final equations:

$$x = A \cdot cos(\theta_1) - B \cdot cos(\theta_1 + \theta_2) - C \cdot sin(\theta_1 + \theta_2) + D$$

$$y = A \cdot sin(\theta_1) - B \cdot sin(\theta_1 + \theta_2) + C \cdot cos(\theta_1 + \theta_2)$$

Eq. 4.2.1 Forward kinematic equations for the position of the end-effector

Now that we have planted our forward kinematic equations, we use them in our derivative approach in the coming section *4.4*, however it is essential to also put forward the inverse kinematic equations, such as to make a comparison between the two approaches possible. In the next section, a derivative solution in order to calculate our inverse kinematic equations is explained.

4.3 Trigonometrical approach

In the trigonometrical solution, the goal is to calculate the angles of the three vertical joints necessary to reach a given point *P(x, y)* on the vertical plane. This method relies on fundamental trigonometric principles, allowing for an visualization of the arm's possible configurations intuitively.

To better visualize, consider a scenario where a robotic arm is tasked with reaching a point *P(x, y)* in the vertical plane (*Figure 23*).

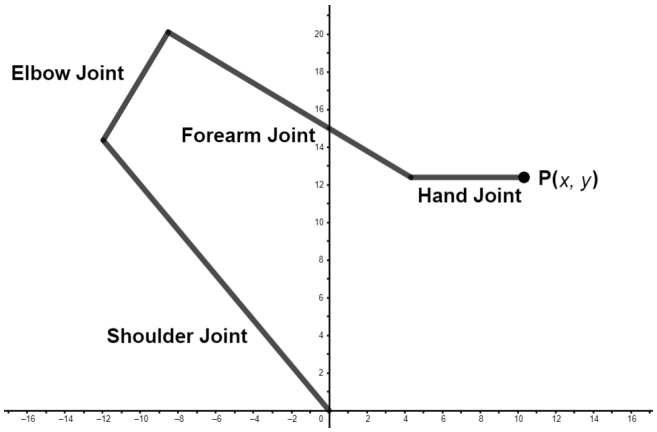

Figure 23. Diagram of joints

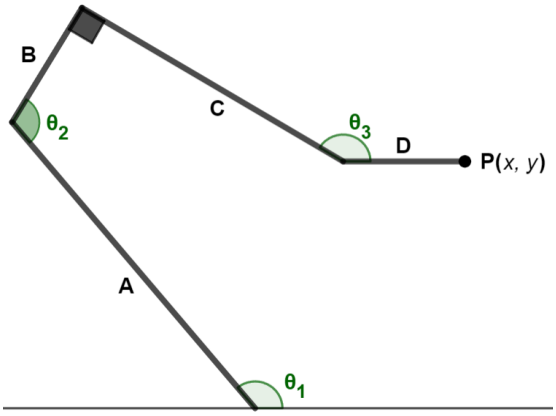

Figure 24. Diagram of joint angles

We can then define angles θ_1, θ_2 and θ_3 as the desired angles for the vertical joints (*Fig. 24*).

The joints will also be renamed as *A*, *B*, *C* and *D* to keep the equations simple, as we already know their exact lengths (*Table 2*).

There are, however, infinite solutions for 3-jointed systems, so we have to set the orientation of the end-effector to a fixed angle. Let's take the case, where the end-effector is always oriented parallel to the ground, such as is in the diagram above. This means that the hand joint has an orientation of 0° relative to the horizontal axis.

We next draw a vertical line across point *P* and the x-axis, in order to form a polygon and derive a geometrical definition for θ_3 (*Fig. 25*).

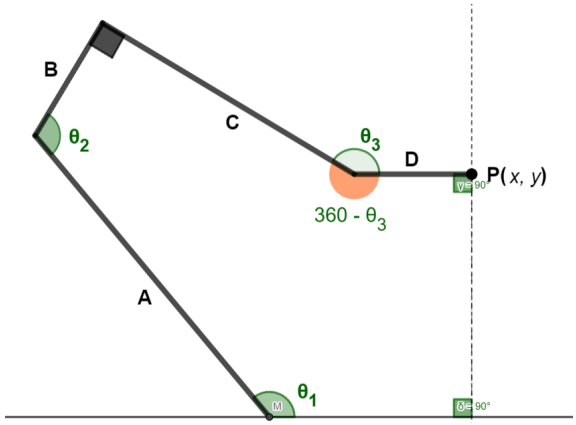

Figure 25.

We can see that this polygon can be divided into four triangles *(Fig. 26*), all of which must fulfill the basic property that their internal angles add up to 180°.

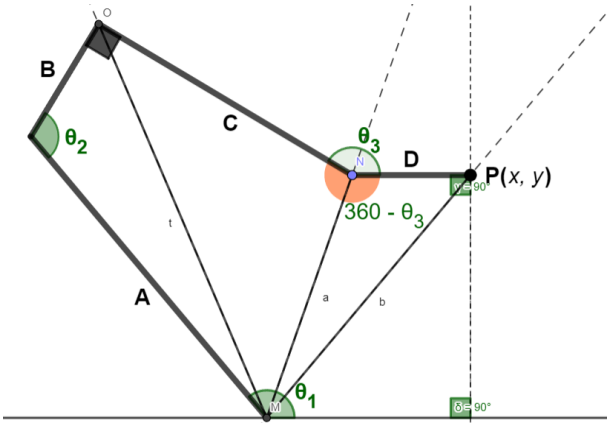

Figure 26.

We can define the sum of all internal angles of the polygon, then solve for θ_3:

$$3\cdot(180°) = 3\cdot(90°) + \theta_1 + \theta_2 + (360° - \theta_3)$$

$$\theta_3 = \theta_1 + \theta_2 - 90^o$$

This relation leaves θ_3 in terms of θ_1 and θ_2, leaving us with only two unknown angles and thus reduces the method to a two-joint kinematic problem. Solving for these last two angles is possible using further definitions of trigonometry. Let's revisit our graph from before.

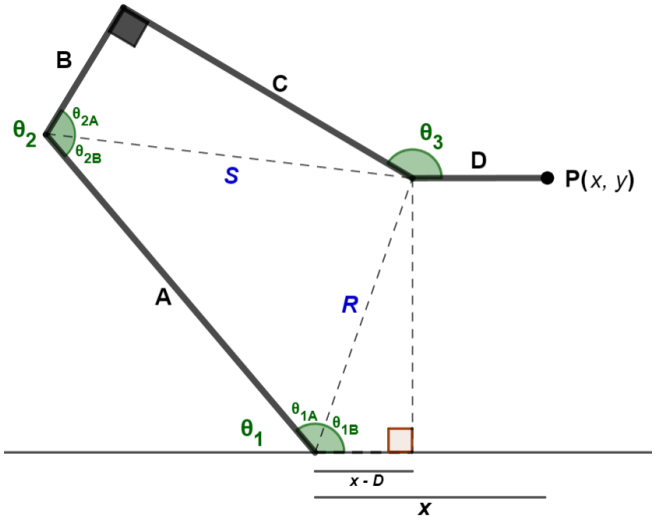

Figure 27.

Tracing two lines, *S* and *R (Fig. 27)*, we bisect both θ_1 and θ_2 into θ_{1A}, θ_{1B} and θ_{2A}, θ_{2B} respectively. We start by defining the segment *S*. Using the pythagorean theorem, we can say:

$$S = \sqrt{B^2 + C^2}$$

Now we can use the law of cosines, and define $S^2 = A^2 + R^2 - 2AR \cdot cos(\theta_{1A})$, and solve for:

$$\theta_{1A} = arccos(\frac{A^2+R^2-S^2}{2AR})$$

θ_{1B} can be expressed as $\theta_{1B} = arccos(\frac{x-D}{R})$, as the adjacent cathetus to this angle has a length of $x - D$, given that the vertex of θ_3 is positioned at the point (*x-D* / *y*).

It is important to note, this value is only true for the case where the end-effector is parallel to the ground such as was our forward kinematic example in section *4.2*. In section *4.5* an adjustment to the equation for all orientations of the end-effector is discussed.

R can be thus be written as $R = \sqrt{(x - D)^2 + y^2}$.

θ_{2B} is solved using the law of cosines, $\theta_{2B} = arccos(\frac{S^2+A^2-R^2}{2AS})$, and the last expression would be $\theta_{2A} = arccos(\frac{B}{C})$. As defined, $\theta_1 = \theta_{1A} + \theta_{1B}$ and $\theta_2 = \theta_{2A} + \theta_{2B}$, so we can derive equations for each angle θ_1, θ_2 and θ_3 using constants we already know the value of.

$$\theta_1 = arccos(\frac{B}{C}) + arccos(\frac{S^2+A^2-R^2}{2AS})$$

$$\theta_2 = arccos(\frac{x-D}{R}) + arccos(\frac{A^2+R^2-S^2}{2AR})$$

$$\theta_3 = \theta_1 + \theta_2 - 90°$$

Eq. 4.3.1 Inverse kinematic equations for the rotation of angle joints

These three equations are our inverse kinematic equations, as already stated, they take the value of the desired coordinate, and return the orientation of the vertical joints in degrees which is then transmitted to the servos to move the robot's end-effector to the point *P(x, y)*.

4.4 Derivative Approach

The derivative approach is different from the trigonometrical method, as it does not calculate the angles θ_1, θ_2, θ_3 given an absolute coordinate *P(x,y)* on the vertical plane, but rather given a "small" change in position $(\Delta x, \Delta y)$, it determines the approximates values $(\Delta\theta_1, \Delta\theta_2, \Delta\theta_3)$ for the change in arm rotation that would attain such a change in position.

The forward kinematic functions *f* for *x* and *g* for *y* are dependant on the values θ_1, θ_2 and are expressed as:

$$f(\theta_1, \theta_2) = x$$

$$g(\theta_1, \theta_2) = y$$

The functions for a change in position by the factor Δx and Δy would be dependant on θ_1, θ_2 and their corresponding changes in rotation $\Delta\theta_1$ and $\Delta\theta_2$:

$$f(\theta_1 + \Delta\theta_1, \theta_2 + \Delta\theta_2) = x + \Delta x$$

$$g(\theta_1 + \Delta\theta_1, \theta_2 + \Delta\theta_2) = y + \Delta y$$

Having planted these equations, as f and g are differentiable in every point of the domain of θ_1, θ_2, as defined by derivative identity (see Stewart, J. (2001)), the following identity is true:

$$\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} = \begin{bmatrix}\frac{\partial f}{\partial \theta_1}(\theta_1,\theta_2) & \frac{\partial f}{\partial \theta_2}(\theta_1,\theta_2)\\ \frac{\partial g}{\partial \theta_1}(\theta_1,\theta_2) & \frac{\partial g}{\partial \theta_2}(\theta_1,\theta_2)\end{bmatrix}\begin{bmatrix}\Delta\theta_1\\ \Delta\theta_2\end{bmatrix} + \begin{bmatrix}o(\Delta\theta_1)\\ o(\Delta\theta_2)\end{bmatrix}$$

Moreover, since f and g are continuously differentiable in every point of the domain of θ_1, θ_2, it is the case that so long as the matrix above (the "Jacobian" $J(\theta_1, \theta_2)$) is not singular in $\theta_1,\ \theta_2$, the following holds due to the inverse function theorem (see Stewart, J. (2001)): .

$$\begin{bmatrix}\Delta\theta_1\\ \Delta\theta_2\end{bmatrix} = \left(\begin{bmatrix}\frac{\partial f}{\partial \theta_1}(\theta_1,\theta_2) & \frac{\partial f}{\partial \theta_2}(\theta_1,\theta_2)\\ \frac{\partial g}{\partial \theta_1}(\theta_1,\theta_2) & \frac{\partial g}{\partial \theta_2}(\theta_1,\theta_2)\end{bmatrix}\right)^{-1}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} + \begin{bmatrix}o(\Delta x)\\ o(\Delta y)\end{bmatrix}$$

The terms of the Jacobian can be easily calculated, because we already have the forward kinematic functions f and g:

$$\frac{\partial f}{\partial \theta_1}(\theta_1,\,\theta_2) = -A\cdot\sin\theta_1 + B\cdot\sin(\theta_1+\theta_2) - C\cdot\cos(\theta_1+\theta_2)$$

$$\frac{\partial f}{\partial \theta_2}(\theta_1,\,\theta_2) = B\cdot\sin(\theta_1+\theta_2) - C\cdot\cos(\theta_1+\theta_2)$$

$$\frac{\partial g}{\partial \theta_1}(\theta_1,\,\theta_2) = A\cdot\cos\theta_1 - B\cdot\cos(\theta_1+\theta_2) - C\cdot\sin(\theta_1+\theta_2)$$

$$\frac{\partial g}{\partial \theta_2}(\theta_1,\,\theta_2) = -B\cdot\cos(\theta_1+\theta_2) - C\cdot\sin(\theta_1+\theta_2)$$

In order to determine the singularity of the Jacobian, the easiest computational method (for dimension 2) is via the determinant. In particular, we have that:

$$\begin{aligned} det(J(\theta_1,\theta_2)) &= \frac{\partial f}{\partial \theta_1}(\theta_1,\theta_2)\frac{\partial g}{\partial \theta_2}(\theta_1,\theta_2) - \frac{\partial f}{\partial \theta_2}(\theta_1,\theta_2)\frac{\partial g}{\partial \theta_1}(\theta_1,\theta_2)\\ &= A\sin(\theta_1)\left[B\cos(\theta_1+\theta_2) + C\sin(\theta_1+\theta_2)\right]\end{aligned}$$

Finally, we define the approximate angle movements as:

$$\begin{bmatrix}\Delta\widetilde{\theta}_1\\ \Delta\widetilde{\theta}_2\end{bmatrix} = \left(\begin{bmatrix}\frac{\partial f}{\partial \theta_1}(\theta_1,\theta_2) & \frac{\partial f}{\partial \theta_2}(\theta_1,\theta_2)\\ \frac{\partial g}{\partial \theta_1}(\theta_1,\theta_2) & \frac{\partial g}{\partial \theta_2}(\theta_1,\theta_2)\end{bmatrix}\right)^{-1}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}$$

From which, using Cramer's rule (Cramer, G., 1750), we have that:

$$\Delta\widetilde{\theta_1} = \frac{\Delta x \frac{\partial g}{\partial \theta_2}(\theta_1, \theta_2) - \Delta y \frac{\partial f}{\partial \theta_2}(\theta_1, \theta_2)}{det(J(\theta_1, \theta_2))}$$

$$\Delta\widetilde{\theta_2} = \frac{\Delta y \frac{\partial f}{\partial \theta_1}(\theta_1, \theta_2) - \Delta x \frac{\partial g}{\partial \theta_1}(\theta_1, \theta_2)}{det(J(\theta_1, \theta_2))}$$

Eq. 4.4.1 Derivative equations for the rotation of angle joints

4.5 Pivoting problem

With the methods described previously, it is possible to move the robot accurately to whatever position is needed by using the input from the joystick. The orientation of the end effector however, stays parallel to the ground, and limits the robot's ability of orientating the end-effector vertically. Put in a different way, it does not allow the robot to manipulate objects from the top or from the sides.

The solution is planted the following way: Once the end-effector has reached its final position, we must rotate the hand joint without moving the position of the end-effector.

The input device will consist of a one dimensional joystick indicating if the end effector must be rotated clockwise or counter-clockwise.

If we call the angle of the end effector respect to the x-axis, $\hat{\theta}_3$ then we can say that:

$$\theta_3 = \theta_1 + \theta_2 - 90^o - \hat{\theta}_3$$

While the new coordinates of the end of the hand joint that we must move towards in order not to change the end-effector position, will be given by:

$$x' = x_0 - D \cdot cos(\hat{\theta}_3)$$

$$y' = y_0 - D \cdot sin(\hat{\theta}_3)$$

Therefore, the required angles will be given by:

$$\theta_1 = arccos(\frac{B}{C}) + arccos(\frac{S^2+A^2-R'^2}{2AS})$$

$$\theta_2 = arccos(\frac{x'}{R'}) + arccos(\frac{A^2+R'^2-S^2}{2AR'})$$

Eq. 4.5.1 Trigonometric pivot solution

where $R' = \sqrt{x'^2 + y'^2}$, while S is already defined as $S = \sqrt{B^2 + C^2}$ in chapter *4.3*.

The derivative solution can also be applied to this pivot problem:

Under a change in $\hat{\theta}_3$ given by $\Delta\hat{\theta}_3$, the respective approximate change in the angles θ_1, θ_2 will be given by equation *4.4.1*, where:

$$\Delta x = - D(\Delta\hat{\theta}_3) \sin(\hat{\theta}_3)$$

$$\text{and } \Delta y = D(\Delta\hat{\theta}_3) \cos(\hat{\theta}_3).$$

Eq. 4.5.2 Derivative pivot solution

4.6 Software implementation

Having presented both the inverse kinematic and derivative solutions for the angles θ_1, θ_2 and θ_3, we can code these same mathematical equations into the C++ Arduino interface. The implementation created an interactive robotic system that can respond to user input for movement control using an input device. In order to be as efficient as possible, we have to consider optimization techniques and code structuring. This allows for faster computation and also ensures smooth execution of the control algorithm, given the limited resources of the Arduino microcontroller.

4.6.1 Software Architecture

The software architecture is to be modular, making it more organized and readable. First, essential libraries for servo motor control are included (`#include <Servo.h>`) and utility functions for logging and enumerations for logging levels are defined. Function pointers for logging callbacks allows for monitoring the system's behavior in real-time, and is useful in the debugging process.

```
enum class LoggingEnum {FATAL, ERROR, WARN, INFO, DEBUG};

typedef void (*LoggingCallback)(LoggingEnum, String);

String LoggingEnumToString(LoggingEnum level) {
   switch (level) {
       case LoggingEnum::DEBUG: return "DEBUG";
       case LoggingEnum::INFO:  return "INFO";
       case LoggingEnum::WARN:  return "WARN";
       case LoggingEnum::ERROR: return "ERROR";
       case LoggingEnum::FATAL: return "FATAL";
       default:    return "UNKNOWN";
   }
}
```

The logging system is designed to categorize log messages into different levels of severity. This is done with an enumeration called `LoggingEnum`. It enumerates various log levels, `Fatal`, `Error`, `Warn` *(warning)*, `Info` *(informational),* and `Debug`.

4.6.2 ServoArm Class

A class structure `ServoArm` is defined, with the purpose of encompassing all of the logic of controlling an "Arm" via a servo motor. Particularly `ServoArm` abstracts the behavior of an arm being rotated against a reference halfline (*Fig. 28*).

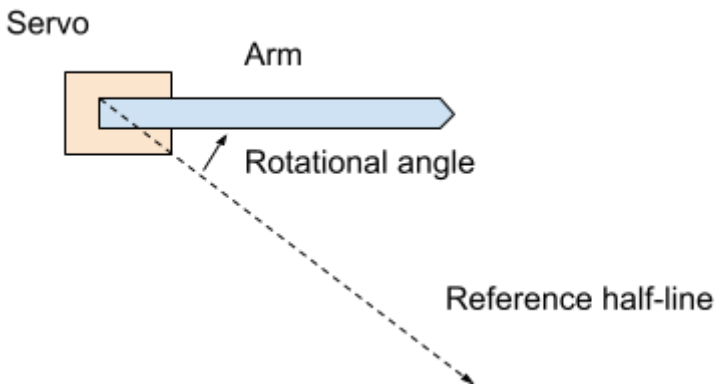

Figure 28. `ServoArm` abstracts the behavior of an arm being rotated against a reference halfline.

Key functionalities of the class include methods for moving the arm to a specified angle (`moveTo`), moving the arm by a delta angle (`moveBy`), and retrieving the current angle (`currentAngle`). The following is the class declaration, its implementation can be found at https://github.com/alerest285/robotic_arm/blob/main/src/servo_arm.cpp.

```
class ServoArm {

  const String _name;
  const double _length;
  const int _pin;
  const LoggingCallback _logging;
  const Servo _servo;

  struct MapRange {
    double lower_bound;
    double upper_bound;
    double servo_to_lower_bound;
    double servo_to_upper_bound;
  };
  const MapRange _map_range;

  double _current_angle;
  bool _is_current_angle_set;

  double _transformArmAngleToServoAngle(double angle);

  int _nearestIntegerAngle(double angle);

  String _rangeToString();

  public:

    ServoArm(String name, double length, int pin, MapRange map_range,
      LoggingCallback logging_callback);

    void moveTo(double angle);

    void moveBy(double delta);

    double currentAngle();
};
```

In particular, the class incorporates mapping functions to ensure that the desired arm angles are translated into corresponding servo angles within defined lower and upper bounds for the arm movement. This is important due to (i) the controlling servo being rotated or reflected respect to the arm rotation origin; (ii) the controlling servo being imprecise in its range and therefore a software calibration being required. Such a mapping is specified via the struct `MapRange`, that maps the actual lower and upper bounds of the arm angles into servo instructions.Finally, error handling was also implemented to check if the specified angle is within the allowed range, and warning messages are logged in case of out-of-range movements.

4.6.3 Robot Class

The `Robot` class is designed to organize the movement of the entire robotic system. It provides a comprehensive set of methods that facilitate precise control and monitoring of the system's position and orientation. This class defines key functionalities, such as moving to specific cartesian coordinates, moving by incremental coordinates, and selecting between exact and derivative-based movement methods. The following is the declaration of the class. Its implementation can be seen at https://github.com/alerest285/robotic_arm/blob/main/src/robot.cpp, although below we will explain the most important bits of its implementation.

```
class Robot {

  LoggingCallback _logging;

  ServoArm *_shoulder;
  ServoArm *_elbow;
  ServoArm *_hand;

  const double _forearm_length;

  const double _differential_stability_threshold = 1e-3;

  struct AngularCoordinates {
    double shoulder_angle;
    double elbow_angle;
    double hand_reference_angle;
    String toString();
  };

  struct AngularDerivatives {
    double x_by_shoulder_angle;
    double x_by_elbow_angle;
    double y_by_shoulder_angle;
    double y_by_elbow_angle;
  };

  enum class MethodEnum {
    EXACT,
    DERIVATIVE
  };

  MethodEnum _method;

  PlaneCartesianCoordinates _calculateCartesianCoordinates(AngularCoordinates
```

```
angular_coordinates);

  AngularDerivatives _calculateAngularDerivatives(AngularCoordinates
angular_coordinates);

  AngularCoordinates _calculateAngularCoordinates(PlaneCartesianCoordinates
cartesian_coordinates, double hand_reference_angle);

  double _calculateDeterminant(AngularDerivatives angular_derivatives);

  void _moveByWithExactMethod(
    PlaneCartesianCoordinates delta_cartesian_coordinates,
    double delta_hand_reference_angle);

  void _moveByWithDerivativeMethod(
    PlaneCartesianCoordinates delta_cartesian_coordinates,
    double delta_hand_reference_angle);

  double _calculateHandAngle(AngularCoordinates angular_coordinates);

  double _getCurrentHandReferenceAngle();

  public:

    Robot(LoggingCallback logging_callback);

    PlaneCartesianCoordinates currentCartesianCoordinates();

    AngularCoordinates currentAngularCoordinates();

    void moveArmsTo(AngularCoordinates angular_coordinates);

    void moveBy(PlaneCartesianCoordinates delta_cartesian_coordinates);

    void rotateHandBy(double delta_hand_reference_angle);

    void setMethodToExact();

    void setMethodToDerivative();

};
```

To ensure the accuracy of the robotic system's movements, we first implement various mathematical utilities. We started by defining trigonometric identities needed for the calculations in our kinematic algorithms, as these by default are given by the built in functions `sin()` and `cos()` in radians. Thus a transformation into degrees is needed. The constant Pi is also defined. (*see* https://github.com/alerest285/robotic_arm/blob/main/src/math.h).

```
constexpr double pi = 3.14159265358979323846264433;

double cosDegrees(double x){
  return cos(x * pi / 180);
}

double cosDegreesDerivative(double x){
  return - (pi / 180) * sin(x * pi / 180);
}

double sinDegrees(double x){
 return sin(x * pi / 180);
}

double sinDegreesDerivative(double x){
 return (pi / 180) * cos(x * pi / 180);
}

double acosDegrees(double x){
 return acos(x) * 180 / pi;
}
```

Inside the class lie the two main methods for controlling the robot arm, the exact trigonometric method, and the approximate derivative method. The implementations for the exact method takes the cartesian coordinates as an input value and runs them through a function `_calculateAngularCoordinates` a function based entirely on the trigonomic equations presented in section *4.3* which returns the angles θ_1 and θ_2 (where θ_3 is determined due to the condition of horizontality of the hand effector).

```
Robot::AngularCoordinates Robot::_calculateAngularCoordinates(
  PlaneCartesianCoordinates cartesian_coordinates, double
hand_reference_angle) {
  double A = _shoulder->length();
  double B = _elbow->length();
  double C = _forearm_length;
  double D = _hand->length();

  // Coordinates of the end point of the forearm.
  double x_prime = cartesian_coordinates.x - D *
```

```
cosDegrees(hand_reference_angle);
  double y_prime = cartesian_coordinates.y - D *
sinDegrees(hand_reference_angle);

  double R = sqrt(pow(x_prime, 2) + pow(y_prime, 2));
  double S = sqrt(pow(B, 2) + pow(C, 2));

  double shoulder_angle = acosDegrees(x_prime / R) + acosDegrees((pow(A, 2)
+ pow(R, 2) - pow(S, 2)) / (2 * A * R));
  double elbow_angle = acosDegrees(B / S) + acosDegrees((pow(S, 2)+ pow(A,
2)- pow(R, 2)) / (2 * A * S));
  return {shoulder_angle: shoulder_angle, elbow_angle: elbow_angle,
hand_reference_angle: hand_reference_angle};
}
```

If the projected angular coordinates the robot would have in the next step do **not** exceed the mechanical limitations of the servo motors or the working range of the robot itself, no logging error message is printed and the servo `moveTo` method in the `ServoArm` class is executed with the new angular coordinates.

```
void Robot::_moveByWithExactMethod(
  PlaneCartesianCoordinates delta_cartesian_coordinates,
  double delta_hand_reference_angle) {
  // Current state.
  PlaneCartesianCoordinates current_cartesian_coordinates =
currentCartesianCoordinates();
  double hand_reference_angle = _getCurrentHandReferenceAngle();

  // Projected state.
  PlaneCartesianCoordinates projected_cartesian_coordinates = {
    x: current_cartesian_coordinates.x + delta_cartesian_coordinates.x,
    y: current_cartesian_coordinates.y + delta_cartesian_coordinates.y};
  double projected_hand_reference_angle = hand_reference_angle +
delta_hand_reference_angle;
  AngularCoordinates projected_angular_coordinates =
    _calculateAngularCoordinates(projected_cartesian_coordinates,
projected_hand_reference_angle);
  if (isnan(projected_angular_coordinates.shoulder_angle)
    || isnan(projected_angular_coordinates.elbow_angle)) {
    return;
  }
```

```
  _shoulder->moveTo(projected_angular_coordinates.shoulder_angle);
  _elbow->moveTo(projected_angular_coordinates.elbow_angle);
  _hand->moveTo(_calculateHandAngle(projected_angular_coordinates));
}
```

The derivative method has a more extensive implementation, but it is similar to the exact method. First we define the cartesian coordinates where the robot is currently at, and the ones the robot will be after a movement by the increment $(\Delta x, \Delta y)$. The derivative matrix is then calculated, checking also if the current point is unstable or not. The criteria employed for stability is the derivative to be greater than the threshold value 0.001. However, stability is ensured in the algorithm itself as we will see in the next paragraph and such a check is purely for running time safety.

```
void Robot::_moveByWithDerivativeMethod(
  PlaneCartesianCoordinates delta_cartesian_coordinates,
  double delta_hand_reference_angle){
  double D = _hand->length();

  // Current state.
  PlaneCartesianCoordinates current_cartesian_coordinates =
currentCartesianCoordinates();
  double hand_reference_angle = _getCurrentHandReferenceAngle();

  // Expected state.
  PlaneCartesianCoordinates expected_cartesian_coordinates = {
    x: current_cartesian_coordinates.x + delta_cartesian_coordinates.x,
    y: current_cartesian_coordinates.y + delta_cartesian_coordinates.y};

  // Jacobian calculation.
  AngularDerivatives angular_derivatives = _calculateAngularDerivatives(
    {shoulder_angle: _shoulder->currentAngle(), elbow_angle:
_elbow->currentAngle()});
  double determinant = _calculateDeterminant(angular_derivatives);
  if (abs(determinant) < _differential_stability_threshold) {
      _logging(
        LoggingEnum::FATAL,
        "Currently we are in an unstable position " +
current_cartesian_coordinates.toString());
    return;
  }
  // ...
```

The delta angles are calculated and a comparison is made between the point where the robot was expected to go and the real point reached with the derivative method. A further check is made if the point the robot is moving to is unstable, finally the `moveBy` method is called to perform the rotation of the servos.

```
        "Trying to move to an unstable position " +
projected_cartesian_coordinates.toString());
    return;
  }
  _shoulder->moveBy(delta_shoulder_angle);
  _elbow->moveBy(delta_elbow_angle);
  _hand->moveBy(delta_shoulder_angle + delta_elbow_angle -
delta_hand_reference_angle);
  return;
}
```

4.6.4 Input method

The input method, a 2 axis joystick is used to provide an interface between the user and the robot. The mechanical movement of the joystick is converted into electrical signals to the Arduino, to provide a numerical value of a cartesian coordinate as explained in section *3.2.7 (input device)* The values of the joystick, which range from 0 to 1023, are mapped onto a variable *delta_x* and *delta_y*. The velocity of the robot's movement depends on the parameter `max_displacement_per_loop` and the loop frequency. The following is the declaration of the class. Its implementation can be seen at:

https://github.com/alerest285/robotic_arm/blob/main/src/joystick.cpp.

```
class CartesianJoystick {

  int _horizontal_input_pin;
  int _vertical_input_pin;
  double _max_displacement_per_loop;

  const int _min_milis = 100;

  public:
```

```
    CartesianJoystick(int horizontal_input_pin, int vertical_input_pin,
double max_displacement_per_loop);

    PlaneCartesianCoordinates getDeltaCartesianCoordinates();
};
```

In a similar way, an interface for a single axis joystick is implemented, in order to control the hand inclination when pivoting the end effector's location.

```
class AngularJoystick {

  int _input_pin;
  double _max_displacement_per_loop;

  const int _min_milis = 100;

  public:

    AngularJoystick(int input_pin, double max_displacement_per_loop);

    double getDeltaAngle();
};
```

4.6.5 Control loop

The final part of the software implementation is the control loop, which initializes the interface with the input and outputs and directs the instructions to be executed while the system is active.

```
#define HORZ_PIN A0
#define VERT_PIN A1
#define ANGLE_PIN A2

void logging(robotic_arm::LoggingEnum level, String message) {
  Serial.println(robotic_arm::LoggingEnumToString(level) + ": " + message);
}

robotic_arm::Robot robot(logging);
robotic_arm::CartesianJoystick cartesian_joystick(HORZ_PIN, VERT_PIN,
/*max_displacement_per_loop=*/0.1);
```

```
robotic_arm::AngularJoystick angular_joystick(ANGLE_PIN,
/*max_displacement_per_loop=*/0.1);

int loop_counter = 0;
const int SECONDS_PER_LOOP = 5000;

void setup() {
  Serial.begin(9600);
}

void loop() {

  logging(
    robotic_arm::LoggingEnum::INFO,
    "\nLoop No " + String(loop_counter) + "."
  );
  switch (loop_counter) {
    case 0:
      // INITIALIZATION
      robot.setMethodToDerivative();
      robot.moveArmsTo({shoulder_angle: 90, elbow_angle: 180,
hand_reference_angle: 0});
      break;
    default:
      // DEFAULT EXECUTION LOOP
      robotic_arm::PlaneCartesianCoordinates delta_coordinates =
cartesian_joystick.getDeltaCartesianCoordinates();
      double delta_hand_reference_angle = angular_joystick.getDeltaAngle();
      robot.moveBy(delta_coordinates);
      robot.rotateHandBy(delta_hand_reference_angle);
      break;
  }

  //
  delay(SECONDS_PER_LOOP);
  loop_counter++;
}
```

We can manually change the positioning method used for control by either calling the function `robot.setMethodToDerivative();` or `robot.setMethodToExact();`. The modular software architecture makes maintenance and future expansion easier. The `ServoArm` and `Robot` classes encapsulate necessary functionalities, such as servo motor control and precise system movement and orientation. The software implementation detailed in this section developed an accessible and interactive robotic system which is available on github.com for free public use:

https://github.com/alerest285/robotic_arm

Scan this QR Code to go to the github link:

5 Results

5.1 Synopsis

To interpret the robot arm's movement capabilities with concrete criteria, a series of experiments were conducted using test persons. A group of 5 individuals was asked to pilot the robot arm to follow a predetermined path, simulating a task requiring precise movement. In the second experiment they were asked to go to a certain object on the table and tip it over with the end-effector. They were then asked several binary questions relating to the maneuverability and accuracy of the robot while performing both tests. Such tasks were performed twice: once with the trigonometrical approach and once more with the derivative approach. Why were these tests done? For a consumer product like a robot arm, quality assurance tests should be done in order to see what the weak points and drawbacks for each method are, necessary corrections can then be implemented from a user point of view. Finally, the pivoting solutions of the trigonometric and derivative methods were tested for their accuracy and repeatability.

5.2 Protocol one: Following predetermined path

The test individuals are instructed to pilot the robot using the joystick in three different ways: Horizontal movement, vertical movement and diagonal movement using both methods, exact and derivative. They had to answer the following questions with a **yes** or **no** answer:

- Question #1.1: *Did the robot follow a straight path in the commanded direction ?* - (testing correctness)
- Question #1.2: *Did the robot follow a straight path without oscillations ?* - (to test for smoothness)
- Question #1.3: *Does the robot react immediately to the joystick command ?* - (this shows if there is a perceivable delay in the input)

5.2.1 Horizontal Movement (Left ⇒ Right / Right ⇒ Left)

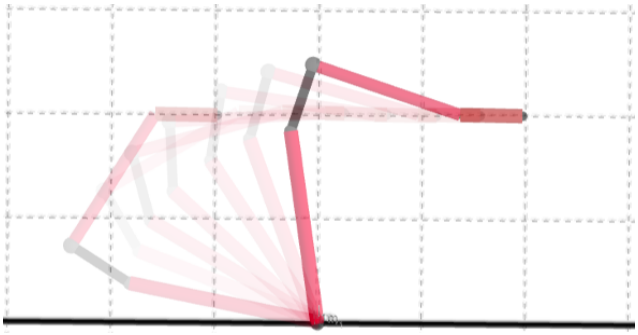

Figure 29. Horizontal movement

The test individuals were instructed to pilot the robot using the joystick following a straight horizontal line (*Fig. 29*), both from left to right and right to left, each with both control methods. They were next asked the questions from above, below is a table with their answers.

Nomenclature:

P1,P2..P5 refers to test person 1-5. *M1/M2* refers to the control methods tested upon, *M1* is the exact control method with trigonometric solution while *M2* is the approximate method with derivative solution.

Table 3. Vertical path results (both sides are the same table)

	P1/M1	P2/M1	P3/M1	P4/M1	P5/M1
Q1.1	Yes	Yes	Yes	Yes	Yes
Q1.2	Yes	Yes	Yes	Yes	Yes
Q1.3	Yes	Yes	Yes	Yes	Yes

P1/M2	P2/M2	P3/M2	P4/M2	P5/M2
Yes	Yes	Yes	No	No
Yes	No	Yes	No	Yes
Yes	Yes	Yes	Yes	Yes

5.2.2 Vertical Movement (Up ⇒ Down / Down ⇒ Up)

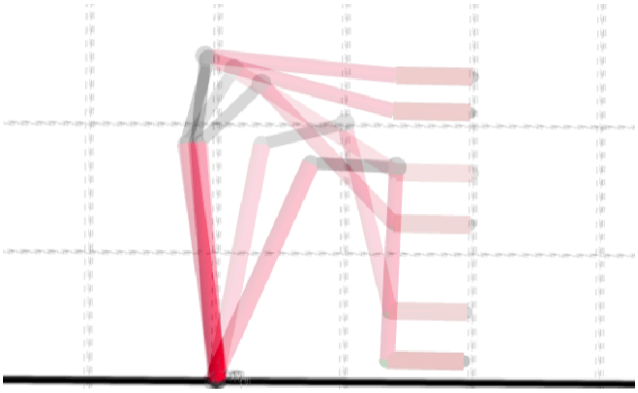

Figure 30. Vertical movement

The individuals piloted the robot using the joystick in a straight vertical line (*Fig. 30*), first starting from above going down, then from below going up, each only one attempt. They were then asked the same questions.

Table 4. Horizontal path results

	P1/M1	P2/M1	P3/M1	P4/M1	P5/M1
Q1.1	Yes	Yes	Yes	No	Yes
Q1.2	Yes	Yes	Yes	Yes	No
Q1.3	Yes	Yes	Yes	Yes	Yes

P1/M2	P2/M2	P3/M2	P4/M2	P5/M2
Yes	Yes	Yes	Yes	No
No	Yes	No	No	Yes
Yes	Yes	Yes	Yes	Yes

5.2.3 Diagonal Movement (45° ⇳, 135° ⇳)

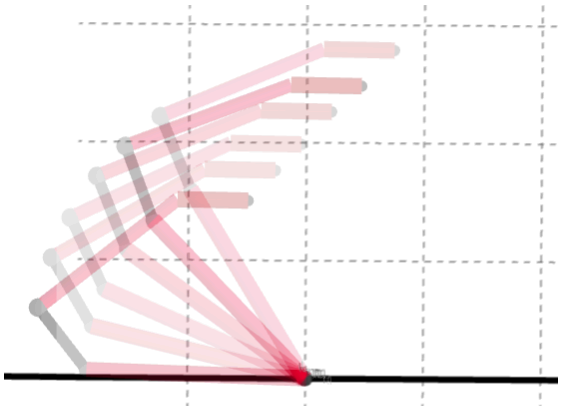

Figure 31, Diagonal Movement 45°

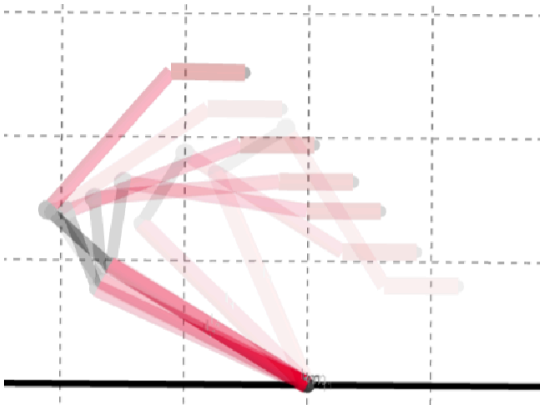

Figure 32, Diagonal Movement 135°

The test individuals piloted the robot using the joystick in order to follow a diagonal line as closely as possible. The subjects would follow the two possible diagonal trajectories; Bottom left to top right (*Fig. 31*) or top left to bottom right (*Fig. 32*), in both directions.

Table 5. Diagonal Movement 45° results

	P1/M1	P2/M1	P3/M1	P4/M1	P5/M1
Q1.1	Yes	Yes	Yes	No	Yes
Q1.2	Yes	Yes	Yes	Yes	No
Q1.3	Yes	Yes	Yes	Yes	Yes

P1/M2	P2/M2	P3/M2	P4/M2	P5/M2
Yes	Yes	No	Yes	No
No	Yes	No	No	No
Yes	Yes	Yes	Yes	Yes

Table 6. Diagonal Movement 135° results

	P1/M1	P2/M1	P3/M1	P4/M1	P5/M1	P1/M2	P2/M2	P3/M2	P4/M2	P5/M2
Q1.1	No	Yes	Yes	No	Yes	No	Yes	No	No	No
Q1.2	No	Yes	Yes	Yes	No	No	No	No	No	No
Q1.3	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

5.3 Protocol two: Trajectory towards object

The robot is to start at a canonical, the individuals are then instructed to pilot the robot with the joystick towards a fixed object on the table, simulating a common pick and place scenario, and touch a marker, causing it to fall over. Each person is free to guide the end-effector using any path they wish, but has four attempts to do so within 10 seconds, two with the exact method and two with the approximate method. They are then asked the following questions:

- Question #2.1: *Could you command the robot to the target object and tip it over ?* - (testing correctness)
- Question #2.2: *Did you have to do readjustments to reach the target object?* - (to test for smoothness)
- Question #2.3: *Could the robot achieve the target object in a repeatable way ?* - (repeatability)

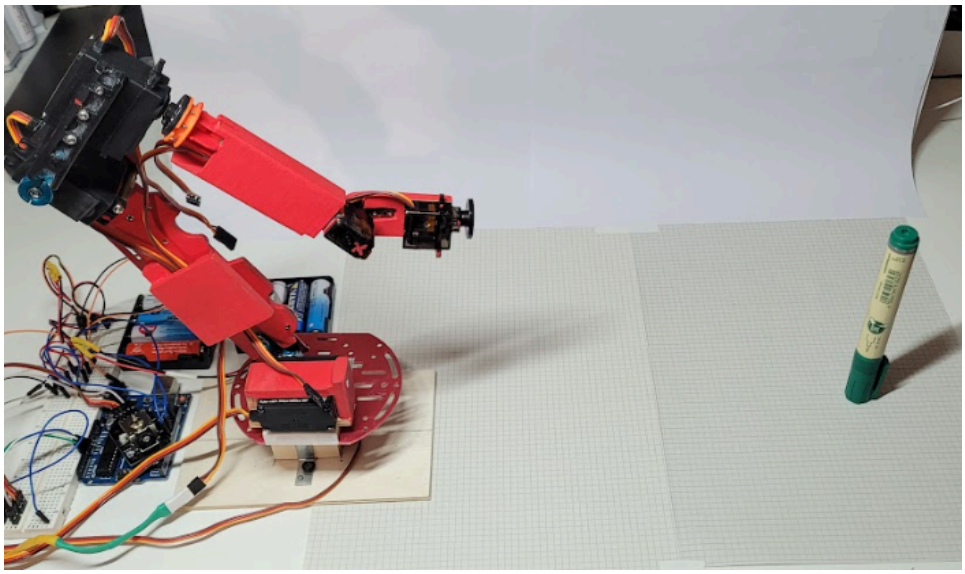

Figure 33. Experiment environment side view (own work)

Table 7. Object trajectory results (right side exact method, left side derivative method).

	P1/M1	P2/M1	P3/M1	P4/M1	P5/M1	P1/M2	P2/M2	P3/M2	P4/M2	P5/M2
Q2.1	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Q2.2	Yes	No	No	Yes	No	Yes	No	Yes	Yes	Yes
Q2.3	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	No

For Question #2.2: *Did you have to do readjustments to reach the target object?* an answer with "No" is **preferred,** and a "Yes" is **not**. Hence the inverted coloring scheme.

Each person conducted the experiment 2 times with the exact method and 2 with the derivative method, resulting in a total of 20 runs. The real time coordinates of the robot were printed out to the serial monitor of the Arduino every 100ms, so it was possible to plot every run using GeoGebra by saving the output coordinates in a text file and inputting them on the graph (*Fig. 34*). Of the total 20 runs, 18 were successful in tipping over the object. The two unsuccessful runs were both from test person 4, taking 13.4 and 11.9 seconds.

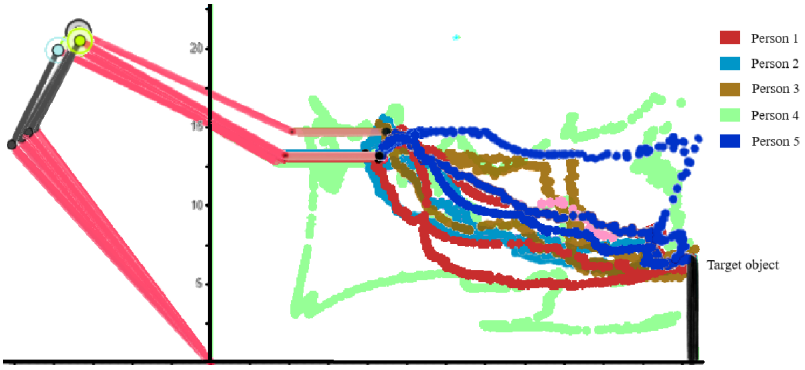

Figure 34. Test trajectories plotted on a graph (own work)

The time it took the test participant from the first input of the joystick to the successful displacement of the target object was measured in seconds for each of the four attempts in a chart (*Fig. 35*).

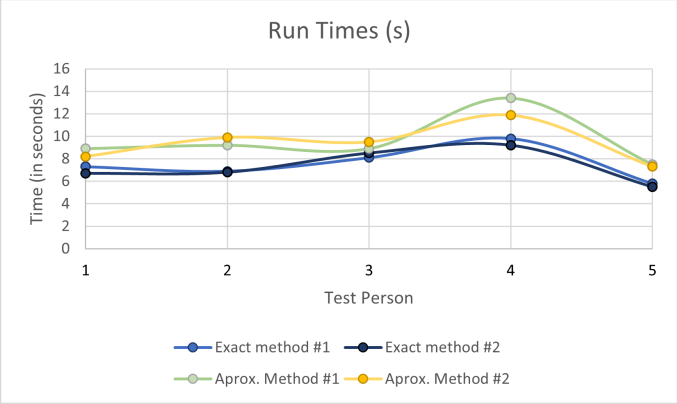

Figure 35. Test times of 20 runs (own work)

5.4 Pivoting tests

As explained in section *4.5*, after modifying the equations for both methods, it is possible to pivot the end-effector joint on a single point while changing its orientation. This is an essential ability that robot arms need to have in any setting so they can then manipulate the desired target object from any orientation.

In order to test the pivoting algorithms, we proposed the following setup: The robot is to be positioned at a stationary position, and the horizontal value of the joystick's potentiometer will indicate if the end effector must be rotated clockwise or counter-clockwise. It will then be measured if the end effector was able to maintain a stationary position while pivoting using both the exact method and the approximate method's implementation of pivoting. Deviations from the starting point will be observed and counted as errors. A full rotation is to be conducted, taking the end-effector from an orientation of 0° to 180°, then back down to 0°.

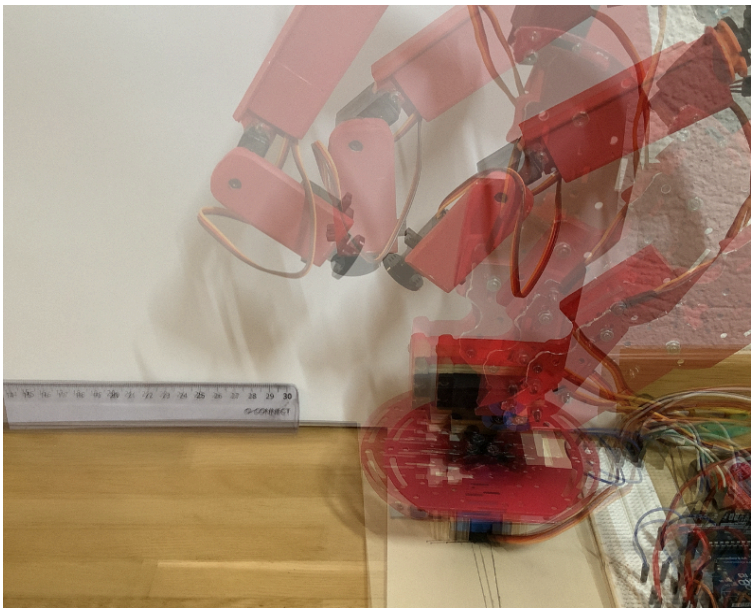

Figure 35. Multiple pivoting positions

After conducting the rotation of the hand joint for each pivoting method, the following observations were made:

1. The exact method accurately pivots around the same position in space, not deviating from the starting location more than **1.5 cm** (a ruler was used as scale, *Fig. 35*). The rotation is smooth, however there is slight oscillation when the end effector reaches 180°. This is due to the hand servo reaching its mechanical limit before it reaches an actual rotation of 180°, something that doesn't occur in the other servos, causing them to continue rotating for a fraction of a second.

2. The derivative method is inaccurate when pivoting, as the approximate coordinates always differ by a slight margin from the actual desired position. This however is amplified as the end-effector is continuously pivoted, the margins of error accumulate along the rotation and after a full of back and forward rotation, the end effector has deviated from its original position by **4.1 cm**, making the derivative-based solution for the pivoting problem not practically viable.

6 Discussion

6.1 Interpretation of results

6.1.1 User protocol tests

The results showed mostly positive results from the participants, especially in the robot's ability to follow horizontal and vertical paths accurately. While there may have been oscillations, I suspect these were due to mechanical limitations of the motors, not the software. Movement along a diagonal path was not terribly unstable, but it did have more oscillation than the past two directions. My main theory is that the measurement of each joint length is not perfect, as it is quite difficult to measure exactly where the exact center of the axis for each joint is located. Accurate measurements for the lengths of the links are tricky, and the values given are just a very good estimation, I will discuss more of this in the next section, *6.2.4*. The results of the second protocol hint at the main difference between both control methods. Using the derivative method, test individuals took on average more time to reach the target object (*Fig. 35*) and had to do more readjustments, while the exact method led them on a straighter path towards the object. The only two runs that were unsuccessful, and above the 10 second mark were using the derivative method. Another remark to be made is that the test subject almost always had a faster second run with both methods, due most likely to the intuitive nature of the control interface.

6.1.2 Method comparison

Comparing the two methods, we can get an insight into the advantages and drawbacks for each solution. The exact method with trigonometric solution is very accurate and provides smooth movements along all paths. Possible oscillations are only due to the imperfections of the servo motors and joint length inaccuracies, as the method solved for the exact joint angles in a perfect environment. The drawback of this method is that it is not easily scalable. It works wonders for 2-jointed systems, such as plotting robots, or the legs of a quadruped robot, even performing well for 3-jointed systems like mine, but solving for three dimensional coordinates becomes extremely complicated, so tasks like drawing on a flat piece of paper require the basis of the method to be reevaluated.

In contrast to this, the approximate method with derivative solution does not calculate an exact coordinate, but a positional displacement, and solves for the angle adjustments. It has some inaccuracies between the desired position the robot is asked to move to, and the actual position the robot currently is in, but it constantly adjusts these errors and moves in directional vectors to correct for this, so an oscillation is slightly more noticeable when controlling the robot with this method. The main advantage however is the scalability and expandability of this method. Because

it doesn't solve for an exact coordinate, but rather moves using directional vectors, one can move the robot in the three dimensional space, by adapting the equation in *4.4.1* for 3 dimensional vectors.

6.1.3 Pivoting control

Both implementations of the pivoting solution were tested, and it was unanimously concluded that the exact pivoting method is far more accurate than the derivative method. This is due to the derivative method being an approximation, and after a certain rotation of the pivot, the errors in approximation add up, until the deviation is too much to neglect. The approximate solution was not without faults however, as slight oscillations were observed when rotating the pivot joint completely to 180°. An explanation for this is that the servo motor for the hand joint is limited to a rotation of <175° due to cheap manufacturing, as it stops rotating slightly before the other two vertical joint motors do, causing the end effector to deviate slightly.

6.2 Sources of error

6.2.1 Servo Restrictions

The main suspect for sources for errors in precision is primarily the hardware. Mechanical limitations of the cheap servo motors restrict the movement of the robot because the quality of rotation is only as good as the quality of the potentiometers. The quality of rotation depends on how many steps a full rotation is divided into. For example 360 steps would result in a quality of 1° per step, larger numbers of steps would result in fractions of a degree, thus increasing the quality of the motor's rotation. This is usually dependent on the price. So, servos in the 10-20 CHF price range generally only reach an accuracy of 1°.

6.2.2 Power supply

Each servo has an operating range of around 5 volts. Powering two servos with the same 5V source like the Arduino cuts the amps each servo can access in half (assuming equal power requirement for each servo), thus reducing the wattage and maximum torque (see *3.2.5*). With this in mind, I used an external power supply of two sets of four AA batteries connected in parallel for all three vertical joint motors (also increasing the maximum consumed amperage), while the rest were powered with the internal Arduino power source, as they comparatively didn't need as much torque as the others. The question arises, if theoretically any oscillation would be dampened when having an external power supply for every individual motor, thus providing it with the maximum possible amperes. This is only theoretical, as it is not economically viable. For each motor there would have to be two sets of four AA batteries connected in parallel, resulting in 48 AA batteries.

6.2.3 Mechanical imperfections

Imperfections in the construction, such as non-flush surfaces are also a possible cause for oscillation. In a perfect environment, there are no gaps between joints, allowing for perfect rotation, this however, is not realistic. Each joint is attached on the output shaft of its servo motor directly using M3 (3mm diameter) screws. Securing it too tight results in little to no oscillation, but hampers the rotation as there is too much friction. Thus, a sweet spot between not too tight and not too loose must be found. Secondly, the spacing between the base plate and the shoulder servo housing could also be a cause of oscillation, as there is a < 1 mm wide gap between the two plates. This difference doesn't seem much, but it may have minimal effect while controlling the end-effector.

6.2.4 Incorrect joint measurements

The trigonometric solution and the derivative solution both assume that the robot is a flat, perfect, 2-dimensional linkage system, which it's obviously not. Measuring the robot's links extremely accurately becomes quite difficult, as there is often no straight line to be measured along, and multiple measurements of the same joint length have given different results. This causes the real values to differ slightly from the values imputed into the control algorithm, which is one of the main causes for inaccuracies.

6.3 Limitations

Each method of robot control comes with its own advantages and drawbacks. Both methods provide a practical solution to 2D coordinate control for a 3-jointed system. The trigonometric method is very exact, but is not extensible since there is no exact solution for 3D coordinate control and iterative algebraic methods must be used (see Chrysanthou, Y et al, 2017). The derivative method is an alternative that provides decent accuracy, and easily allows for 3D expansion. So not only is it possible to find a solution for 3-jointed systems, but also for machines with 4 or more degrees of freedom. This was an original goal for me, to be able to implement the rotational joints into the derivative solution itself, but was ultimately not feasible due to time constraints. The main drawback of the derivative method is that there are no solutions for points where the determinant of the matrix A is 0 as explained in section *4.4*, so there are regions where the code has to prevent the robot from going to, something that is not a problem with the exact solution.

6.4 Improvements and Additions

An improvement for the mechanical design would be the use of a 3D printer, and a professional CAD (computer assisted design) software. This would reduce the mechanical faults, as every intricate part would be compatible with each other and not made by hand, eliminating human error.

Two methods for two-dimensional coordinate control were achieved, the next step would intuitively be to program a solution that works with three-dimensional coordinates, as these have more real world applications. As the project stands, I was able to coordinate the three vertical joints with the joystick control, but not the rotational joints. These have to be controlled individually with another input device.

As it was mentioned in section *6.1.1,* the quality of the servos is a main source of oscillation and inaccuracy, as they have cheap gearboxes that provide decent torque for the price, but in doing so sacrificing the accuracy of rotation. A possibility could be using more expensive servos, that use an optical sensor instead of a potentiometer to track the position of the rotor, allowing (i) precise positioning without post-assembly adjustments; and (ii) absolute measurement, deeming unnecessary the initialization of the servos to a canonical position each time the robot is reset.

Using other actuators, such as professional robot actuators, since these use stepper motors and cycloidal reducers, mechanical gearboxes that use a cycloidal mechanism to reduce the speed. They are compactly designed, have high torque density, and a reduced backlash that make it particularly well-suited for robotic applications. They are however not affordable or practical for a hobby project like the one in this paper.

6.5 Conclusion

After assessing the robots performance throughout various tests has given me an insight into its qualities and imperfections, making it possible to perform improvements where they are needed. Tests using human participants were a fun experiment allowing a typical user to pilot a robotic system and see if the control system i designed was intuitive enough for the individual to succeed in a set of challenging tasks having never been familiar with a machine like a robot arm. I was especially satisfied with the input device having no delay between moving the joystick and the robot adjusting its positioning. This means the electronics and software execute efficiently enough with no perceivable delay to be involved.

In the future, improvements, tweaks and additions are essential for advancing the capabilities of the robot. While it was not possible to test using three-dimensional coordinates, this is only the next step in evolving the control capabilities of the prototype. In summary, areas for improvement were noted, the main sources of error were identified and solutions to these areas were considered. Refining the control methods and incorporating a couple of design modifications would create a more robust and versatile robotic system in the future, with many potential real-world applications.

6.6 Reflection

Throughout the project, I was faced with a lot of challenges and setbacks. In the construction phase, I often found myself redoing several parts and investing entire days in cutting and filing new components. The absence of a 3D printer proved to be a limiting factor, as its use could have significantly expedited the process and resulted in a more robust mechanical design.

The electronic components presented their own set of difficulties, with motors occasionally ceasing to function. Troubleshooting these issues became a recurring task, demanding time and attention to ensure the consistent operation of the robotic system. Despite the setbacks, these challenges provided invaluable insights and hands-on experience in addressing real-world problems in robotics.

Working with both trigonometric identities and derivative definitions during the project helped with my understanding of these mathematical concepts. The practical application of these theoretical principles in the implementation of control algorithms gave me knowledge and proficiency in programming, specifically in the C++ language. The utilization of object-oriented programming to code robust and well defined interfaces not only enhanced the control algorithms' effectiveness but also contributed to my overall skill set in software development.

The working product of the project allowed me to test both control algorithms on a machine constructed from consumer electronics and affordable materials. This practical approach was required due to the extremely high cost of industrial robots, even those smaller in scale as the one I built. In reflection, while the project presented its share of challenges, the iterative nature of problem-solving that engineering requires and the hands-on experience garnered have been crucial in my learning journey. The amalgamation of theoretical knowledge, practical skills in construction and electronics, as well as programming proficiency has provided a holistic learning experience, equipping me with a versatile skill set applicable in the dynamic field of robotics.

7 Bibliography

Arduino Software Documentation. https://www.arduino.cc/en/software [accessed on January 6, 2024]

Barrett, S. F., Pack, D. J. (2006). Microcontrollers Fundamentals for Engineers and Scientists. Switzerland: Morgan & Claypool Publishers.

Bejczy, A. K. (1974). *Robot arm dynamics and control*. (No. NASA-CR-136935).

Casgains Academy, (2021, March). *Inside Tesla's Crazy AI Manufacturing Revolution*. https://streetfins.com/inside-teslas-crazy-ai-manufacturing-revolution/. [accessed on December 28, 2023]

Cramer, G. (1750). *Introduction à l'analyse des lignes courbes algébriques*. chez les frères Cramer et C. Philibert.

Chrysanthou, Y., Aristidou, A., Shamir, A., & Lasenby, J. (2018, September). Inverse Kinematics Techniques in Computer Graphics: A Survey. In *Computer Graphics Forum* (Vol. 37, No. 6). https://doi.org/10.1111/cgf.13310

Cummings, P. G., Bowers, W. D., & Martiny, W. J. (1981). Induction motor efficiency test methods. *IEEE Transactions on industry Applications*, (3), 253-272.

DiMaio, S., Hanuschik, M., Kreaden, U. (2011). *The da Vinci Surgical System*. In: Rosen, J., Hannaford, B., Satava, R. (eds) Surgical Robotics. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-1126-1_9

Firoozian, R. (2014). *Servo motors and industrial control theory*. Springer.

Kim, H., Miller, L. M., Al-Refai, A., Brand, M., & Rosen, J. (2011, August). Redundancy resolution of a human arm for controlling a seven dof wearable robotic system. In *2011 Annual international conference of the IEEE engineering in medicine and biology society* (pp. 3471-3474). IEEE.

Kucuk, S., & Bingul, Z. (2006). *Robot kinematics: Forward and inverse kinematics* (pp. 117-148). London, UK: INTECH Open Access Publisher.

Louis, L. (2016). Working Principle of Arduino and Using It As a Tool for Study and Working Principle of arduino and using IoT. *International Journal of Control, Automation, Communication and Systems (IJCACS)*, *1*(2).

Mahadevkar, S. V., Khemani, B., Patil, S., Kotecha, K., Vora, D. R., Abraham, A., & Gabralla, L. A. (2022). A review on machine learning styles in computer vision—Techniques and future directions. *Ieee Access*, *10*, 107293-107329. doi: 10.1109/ACCESS.2022.3209825

Maurette, M. (2003). *Mars Rover Autonomous Navigation*. Autonomous Robots 14, 199–208. https://doi.org/10.1023/A:1022283719900

Moran, M.E., 2007. *Evolution of robotic arms*. J Robotic Surg 1, 103–111. https://doi.org/10.1007/s11701-006-0002-x or https://rdcu.be/du9ne

Stewart, J. (2001). Multivariable calculus: concepts and contexts. Brooks/Cole.

Sun, J. (2012). Pulse-width modulation. In *Dynamics and Control of Switched Electronic Systems: Advanced perspectives for modeling, simulation and control of power converters* (pp. 25-61). London: Springer London. https://doi.org/10.1007/978-1-4471-2885-4_2

Tanaka, T., & Honda, Y. (2016). *Improving accuracy with servo motors and high speed network*. White paper in Yaskawa Electric Corporation [retrieved in Jan 5, 2023] https://www.yaskawa.com.sg/assets/upload/news_file/1453368223Yaskawa%20article%20in%20IAA.pdf

8 Figures

Cover figure.

MNG Rämibühl. Mathematisch-Naturwissenschaftliches Gymnasium. url: https://www.mng.ch. [accessed 7 Jan. 2024]

Figure 1.
robohub. (2008). [Cartesian robot]. Wikimedia Commons. [accessed 21 Nov. 2023]
https://robohub.org/wp-content/uploads/2015/12/2391_Cartesian_Coordinates.png

Figure 2.
Pan , Z., & Zhang, H. (2008). [6-DOF ABB IRB 6400 robot]. [accessed 21 Nov. 2023]
https://www.researchgate.net/figure/Structure-of-6-DOF-ABB-IRB-6400-manipulator_fig1_4170565

Figure 3.
Sparkfun.com. (2023). [Servo motor parts]. [accessed 28 Nov. 2023]
https://cdn.sparkfun.com/assets/custom_pages/5/6/0/servo-parts.jpg

Figure 4.
Orzetto. (2008). [A typical, single-input, single-output feedback loop with descriptions for its various parts.]. Wikimedia Commons. [accessed 3 Dec.. 2023]
https://commons.wikimedia.org/w/index.php?curid=5000019

Figure 5.
Khadav. (2008). [Servo pwm]. Wikimedia Commons. [accessed 3 Dec.. 2023]
https://commons.wikimedia.org/wiki/File:ServoPwm.png

Figure 6.
Tauner, S. (2013). [Servo diagram]. Wikimedia Commons. [accessed 3 Dec.. 2023]
https://commons.wikimedia.org/wiki/File:Servo_Diagram.svg

Figure 7.
Make MagazinE (2018). [Arduino Uno R3]. Wikimedia Commons. [accessed 5 Dec.. 2023]
https://commons.wikimedia.org/wiki/File:Arduino_uno_r3.jpg

Figure 8.
Fanuc Robotics (2008), cited by Tamin, M. N. & Rahman, F. (2010) [The axes of the reference robot manipulators]. [accessed 17 Nov. 2023]
https://www.researchgate.net/figure/The-axes-of-the-reference-robot-manipulators-Fanuc-Robotics-2008_fig4_221908770

Every other Figure used in this paper was created by the author using authorized software.

Eigenständigkeitserklärung

Der Unterzeichnete bestätigt mit Unterschrift, dass die Arbeit selbständig verfasst und in schriftliche Form gebracht worden ist, dass sich die Mitwirkung anderer Personen auf Beratung und Korrekturlesen beschränkt hat und dass alle verwendeten Unterlagen und Gewährspersonen aufgeführt sind.

Unterschrift: _____ Datum: _____