mng rämibühl
Mathematisch-Naturwissenschaftliches Gymnasium

Designing a Microcontroller-Driven Robotic Arm with Software Integration

Alejo Restrepo

4c, MNG Rämibühl

Maturaarbeit

Submitted on Januar 8th 2024

Abstract

Contents

1. Introduction

1.1. Objective

The goal of this project is to design and build a robot arm with six degrees of freedom (forward/back, up/down, left/right, yaw, pitch, roll) and program both approximate and exact control algorithms for maneuverability. The end result is a proof of concept that the same movement and control capabilities of an industrial robot arm, such as coordinate positioning, path following and axis pivoting is possible with 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 capability 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 even come close to the capabilities of one of these machines using electronic components I could get in hobby kits or online.

1.3. Project Overview

The next section provides a theoretical background on the selection of motors and microcontrollers. Subsequently, the main methods for controlling the robot are discussed using both a geometric approach and a matrix solution with inverse kinematics[1].

The following two sections delve into the design and construction processes of the robot arm, both from the point of view of the hardware and software used. In the software section, (missing). In the hardware section, first the mechanical design of the prototype is explained, taking into account weight distribution and torque limitations. Next, the process of building and assembling each of the joints of the robot arm is detailed. At the end, a graphical design of the structure and a wiring diagram between the microcontroller and the motors are shown.

The third chapter presents the software solutions for controlling the robot, using both inverse and forward kinematic solutions in C++. The first method is a trigonometrical solution for solving positioning of the robot end effector using three-degree of freedom movement. The mathematical equations and geometry behind this solution are then explained. The reliability of the kinematic solutions is observed, letting the

[1] https://www.mathworks.com/discovery/inverse-kinematics.html

robot follow a predetermined path. A last algorithm for pivoting on a given axis is proposed and a solution using derivatives is presented.

In the fourth chapter, various tests are performed to compare and evaluate the expected performance of the robot arm with the actual behavior of the finished product. For example, it is verified that the robot arm can perform a movement following a predetermined path. The robot´s precision while positioning and orienting the end-effector, is also measured taking into consideration different operating speeds as well as endurance tests. Finally, the maximum weight the robot arm can functionally handle without overloading the motors is measured by pushing the robot arm to its mechanical limit.

The fifth and last chapter reflects on the project, the overall cost, compares the results with both industrial and hobbyist robotic systems and details steps of the hardware and software development process where an improvement could have been made, recounts the learning experience and final thoughts of the paper.

2. Theoretical Basis

2.1. A brief introduction to robotics

The evolution of robotics has been driven by the pursuit of enhancing the efficiency, precision, and versatility of various industries. The first industrial robots were introduced in the 1960s as the need for automation of manpower-intensive tasks in manufacturing increased. These early robots were often large, stationary machines, limited in their capabilities and so were primarily designed for heavy lifting tasks (Moran, M.E., 2007).

Robotics as a field has since expanded beyond industrial applications, encompassing a diverse range of domains, including healthcare, such as the daVinci Surgical System (DiMaio et al., 2011), space exploration robots like the mars rovers (Maurette, M., 2003), agriculture, household assistance and many others. The development of intelligent systems, machine learning and advanced sensors have expanded the impact of robotics on humanity as they 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, which functions as a mechanical model arm. It is typically programmed to perform tasks and mimics the dexterity and range of motion of a human arm. The links of the robot are connected by joints, facilitating either rotational motion, as observed in an articulated robot (*Fig. 1*), or linear displacement, such as a cartesian robot (*Fig. 2*) [1].

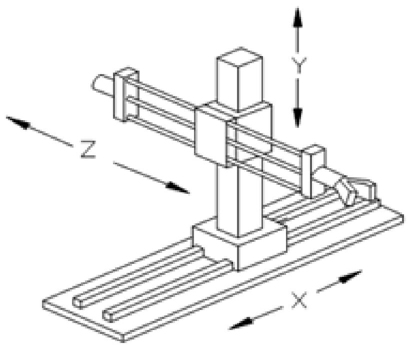


Figure 1.

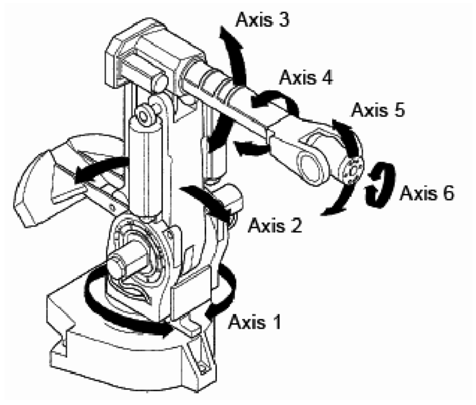


Figure 2.

An industrial arm, resembling a human arm with six joints equivalent to a shoulder, an elbow, and a wrist, usually has a stationary shoulder mounted on a base structure. This type of robot has six degrees of freedom, enabling movement in six different ways. For comparison, a human arm has seven degrees of freedom [2]. Just as a human arm's purpose is to move the hand from one place to another, a robotic arm's function is to move its end-effector to different locations.

Industrial robots are specifically engineered for repetitive tasks within controlled environments, such as the automation of soldering the chassis of a car along an assembly line like in Tesla factories. To instruct a robot in its designated task, the user guides the robotic arm through precise motions using a handheld controller. The robot then stores the exact sequence of movements in its memory, executing it each time a new unit arrives at the assembly line (Casgains Academy, 2021). In this process, the actuators of the arm, servo motors, play a **pivotal** role. These motors, equipped with sensors, enable the robot to achieve precision and accuracy in its movements. For instance, when tightening bolts or performing intricate tasks, the servo motors receive feedback from sensors, adjusting the robot's actions to ensure consistency. These motors enhance the robot's efficiency by maintaining consistent drilling locations and bolt-tightening forces, regardless of the duration of operation. This precision is especially critical in the computer industry, where the assembly of tiny microstrips relies on the accurate functioning of servo motors [4].

2.3. Servo motors

A servo motor is by definition a motor that uses feedback to operate. You can program a servo motor to go to a certain position and it will keep sending power to the motor until it reaches that position. A true servo motor has a constant feedback loop and makes adjustments to the motor as it is in the process of rotating, instead of checking after the rotation has ended and adjusting from there.

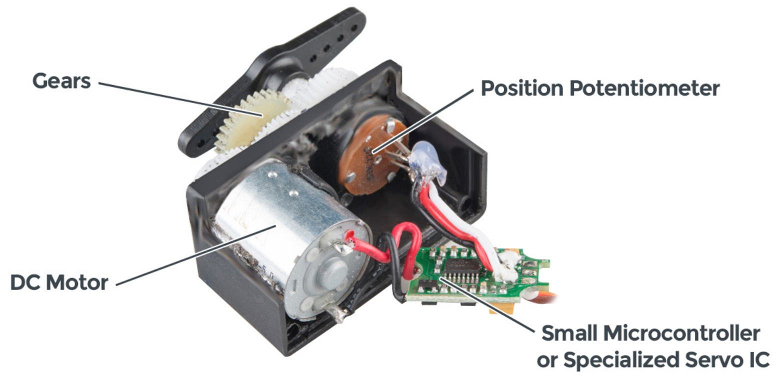


Figure 3.

A servo motor comprises a DC motor, gears, a feedback device, and a control circuit, as illustrated in *Figure 3*. In its simplest form, the control circuit receives an input signal to activate the motor. The motor propels the gear train, connected to the output shaft and a positioning sensor, typically a potentiometer, which is an electrical sensor that tracks rotation in cheaper servos or an encoder in more advanced models.

The potentiometer transmits feedback error signals to the control circuit, instructing it to adjust power to minimize the error and achieve the desired output angle of the motor shaft. This feedback loop is depicted in *Figure 4*.

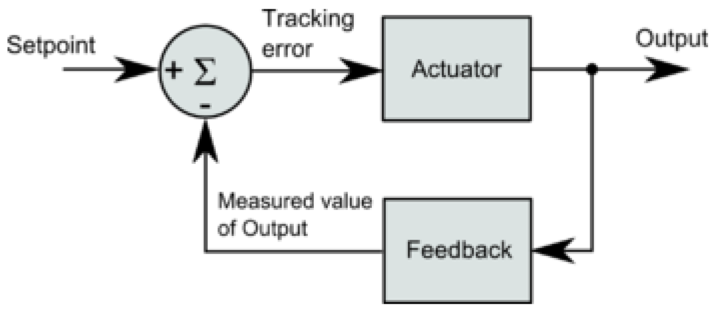


Figure 4.

The overall design involves the motor responding to input signals, adjusting its position based on feedback, and continually refining until the target angle is reached.

Servo motors are the ideal solution with a dynamic payload that needs to start and stop really fast. This occurs because their advanced control algorithms can measure the amount of force being delivered and can check that to the acceleration of the load. This also helps reduce noise. The reserve power they can deliver for a short burst of time makes them good at accelerating loads. When handling fragile loads, servos are the best choice, since they can give feedback on the amount of pressure they exert. The downsides to servos are that they cost more, and that their more advanced operating system makes the setup difficult.

The choice of servo motors for the robotic arm is crucial to achieve precise control and movement. Factors such as torque, speed, and accuracy are considered when selecting appropriate servo motors for each joint. With servo motors possessing reliable precision, a low cost and high torque compared to their weight they are a perfect choice for this project.

2.3.1. Pulse width modulation (PWM)

The difference in internal design consequently results in a different way of controlling the motor both, in hardware and software. Following is an explanation of 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. These potentiometers are often not able to rotate continuously. Continuous rotational servo motors are available and use a different positional feedback device. Professional servos almost always use an encoder **[]** instead of a potentiometer even if continuous rotation is not required.

Servos are not controlled by simply increasing the voltage of the power source which is the case in regular DC motors. In the basic operation of a servo motor, this voltage is regulated through a process known as pulse width modulation (PWM). In PWM, the voltage is rapidly switched between on and off states, controlling the effective voltage delivered to the motor []. The standard repetition rate of most servos is 20 milliseconds, meaning that every 20ms the control circuit calculates if the motor needs more, less or equal power to keep or move to the desired position (*Fig. 5*).

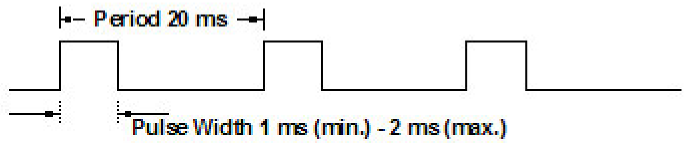
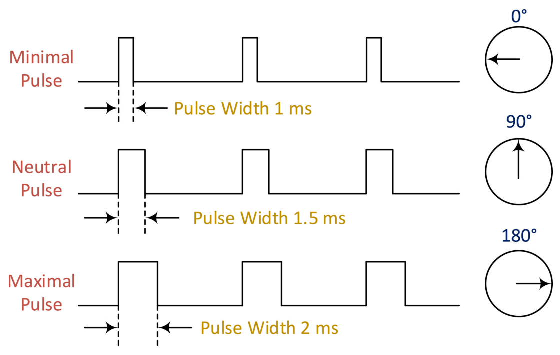


Figure 5.

The width of the pulse determines the desired motor position. Given the rotational constraints of typically 180° maximum, the neutral position of the servo arm is at 90° and the minimal position is 0° . In typical servos, the neutral position requires a pulse with signal of around 1.5ms, the minimal position around 1ms and the maximum position a pulse of around 2ms, seen in *Fig. 6* below.



These exact values can be different for different servo manufacturers and servo models of the same manufacturer, but the general control idea stays the same.

Figure 6: Different control signals The accuracy of the standard potentiometer servos is dependent on the accuracy of the sensor's feedback. When accuracy is an important aspect in design a servo with a encoder as a feedback system should be used.

2.4. Microcontrollers

A microcontroller is a compact integrated circuit that contains a processor core, memory core and IN/OUTPUT pins (Barret, S., & Pack, D., 2022; Keim, R, 2019). 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 prototyping platforms like Arduino boards (Arduino, S. A., 2015).

2.4.1. Arduino

Arduino is an open source microcontroller which can be easily programmed, erased and reprogrammed at any time. Introduced in 2005, the Arduino platform was designed to provide an inexpensive and easy way for hobbyists, students and professionals to create devices that interact with their environment using sensors and actuators. Based on simple microcontroller boards, it is an open source computing platform that is used for constructing and programming electronic devices. It is also capable of acting as a mini computer just like other microcontrollers by taking inputs and controlling the outputs for a variety of electronics devices. Arduino uses hardware known as the Arduino development board and software for developing the code known as the Arduino IDE (Integrated Development Environment). Built up with the 8-bit Atmel AVR microcontroller that are manufactured by Atmel or a 32-bit Atmel ARM, these microcontrollers can be programmed easily using the C or C++ language in the Arduino IDE.

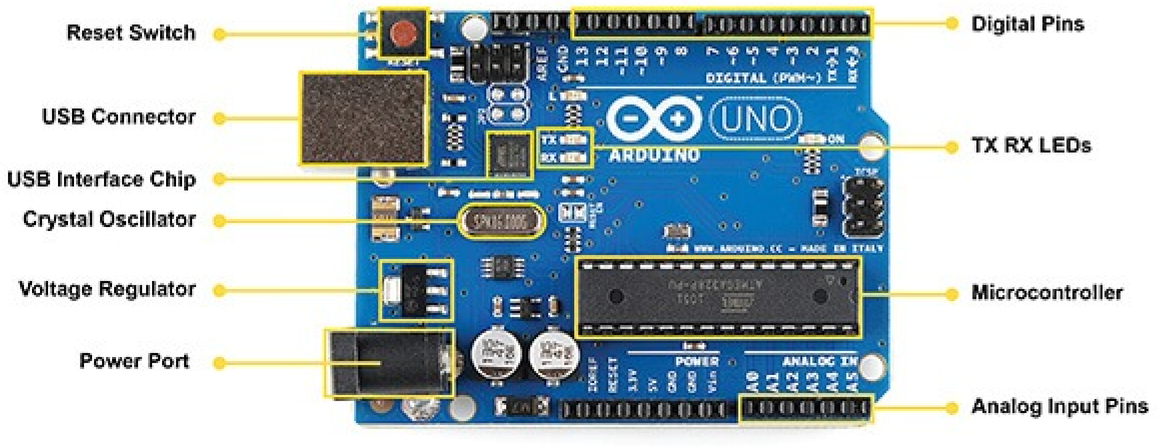


Figure 7.

2.5. Kinematics

Kinematics describes the rotational and translational motion of points, bodies (objects) and systems of bodies (groups of objects) without consideration of what causes the motion or any reference to mass, force or torque. Inverse kinematics (IK) was initiated in robotics as the problem of moving a redundant kinematic arm with specific degrees of freedom (DoFs) to a predefined target. Beyond its use in robotics, IK has found applications in computer graphics, generating particular interest in the field of animating articulated subjects. This survey focuses on IK applications in computer graphics, aiming to provide insights about IK to young researchers by introducing the mathematical problem, and surveying the most popular techniques that tackle the problem (Aristidou et al., 2017).

Positive kinematic analysis of the robotic arm refers to obtaining the position of the end of the robotic arm relative to the reference coordinate system based on these angles and information about the connecting rod, given that the rotation angles of the motor at each joint between the robotic arms are known. The structure of the four-axis robotic arm in this paper is shown in **Figure 5** where Joint0 is the base coordinate system of the robotic arm, Joint1, Joint2, and Joint3 are all rotational joints, and Joint4 is the wrist joint.[2]

[2] Mahadevkar, S.V.; Khemani, B.; Patil, S.; Kotecha, K.; Vora, D.R.; Abraham, A.; Gabralla, L.A. A Review on Machine Learning Styles in Computer Vision—Techniques and Future Directions. *IEEE Access* **2022**, *10*, 107293–107329. [**Google Scholar**] [**CrossRef**]

In contrast to the positive kinematic analysis of the arm, the inverse kinematic analysis of the arm uses calculations to solve for the angle of rotation at each joint when the relevant parameters of the connecting rod and the position of the end of the arm concerning the reference coordinate system have been obtained..[3]

Inverse Kinematics (*IK*) is a critical aspect of robotic arm control, involving the determination of joint angles that result in a desired end-effector position and orientation.[] Among the methods employed for solving inverse kinematics problems, the trigonometrical solution stands out as a straightforward and accessible approach. This method is particularly advantageous when dealing with robotic arms operating in a two-dimensional vertical plane.

3. Hardware

3.1. Synopsis

The objective is to build a robot arm with six degrees of freedom with working dimensions large enough to allow for grasping and manipulating a range of inanimate objects within a defined workspace. The robot arm will have six rotational joints, enabling it to have fluid movement along a three-dimensional space, having an end-effector at the end for manipulation tasks.

3.2. Mechanical design

This chapter presents the design and initial implementation of the robotic arm prototype that utilizes six servo motors to rotate all of its joints. The focus of this section is detailing the mechanical design of the 4 main vertical joints, the base horizontal joint and the rotational joint. The end effector design will be discussed in subsequent sections. The

[3] Meribout, M.; Baobaid, A.; Khaoua, M.O.; Tiwari, V.K.; Pena, J.P. State of Art IoT and Edge Embedded Systems for Real-Time Machine Vision Applications. *IEEE Access* **2022**, *10*, 58287–58301. [**Google Scholar**] [**CrossRef**]

importance of a stable base to support the weight of the entire structure is emphasized in this initial prototype.

Conventional 6DOF robotic arm joint distribution

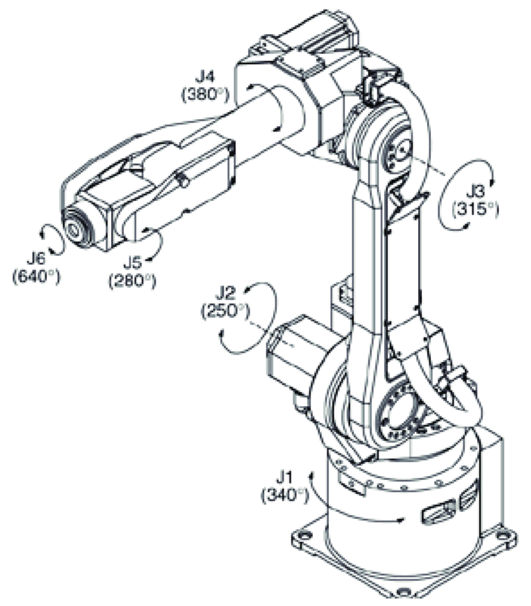


Figure 8.

Base Joint; the base horizontal joint (*Joint 1 from Fig.1*) is responsible for providing a stable foundation for the entire robotic arm. It is designed to rotate around a horizontal axis, allowing 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 *(Joint 2, 3, 5 from Fig. 1)* of the robotic arm are crucial for achieving a wide range of motion. These joints are designed to rotate across the vertical axis, enabling the arm to move up and down. Each vertical joint consists of a servo motor connected to a rigid linkage mechanism, allowing precise control of angular displacement.

Rotational Joints; The further two axes, (*J4 and J6 From Fig.1*) , revolve and swivel the forearm and wrist respectively, allowing it to move freely. Rather than the vertical and horizontal joints, which move the robot to a specific position, the 6-axis arrangement allows the end effector, the last point of the arm, to rotate to every angle just like a human wrist can rotate the hand and allow your fingertips to reach any point in space with any orientation.

3.2.1. Design prototype

Ensuring the stability of the base is vital in the design of the robotic arm prototype. The base must support the weight of the entire structure, maintaining balance and preventing tip-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 for complex and intricate parts that need to be molded by hand whilst having durability. The process for achieving hardness is by cutting the sheets to the desired size and gluing them on top of eachother with a cyanoacrylate-based adhesive solution, this attaches the sheets together and solidifies the wood altogether, as it solves itself inside through the fibers of the wood sheet.

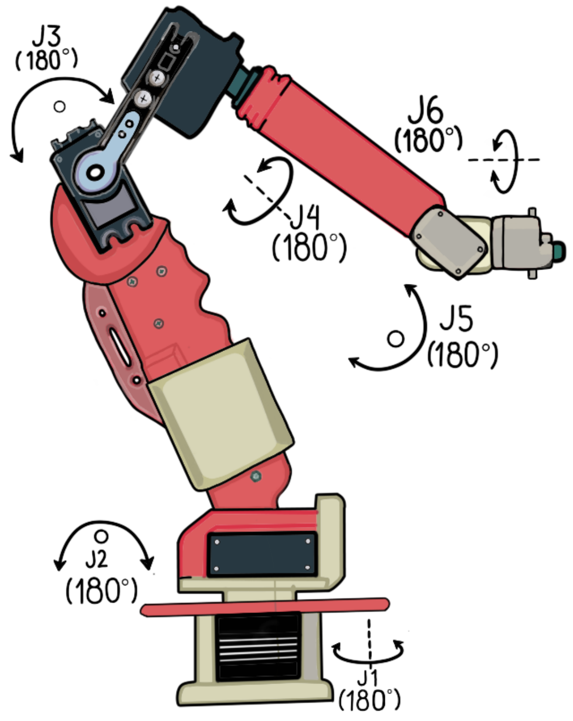


Figure 9. (own work)

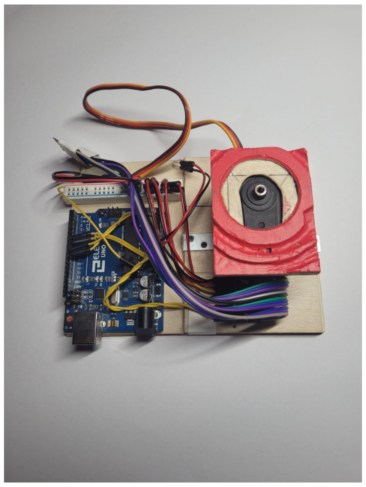


Figure 10. Base Plate upper view (own work)

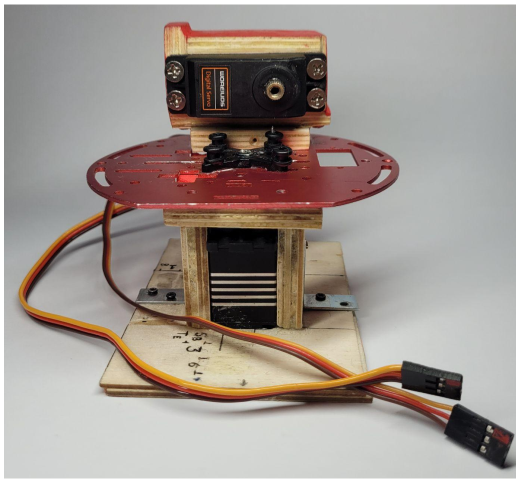


Figure 11. Base Plate side view with shoulder servo (own work)

The base plate consists of three solid parts; a flat slab that is attached with the base servo[4] (fig. 1) and a housing structure that surrounds the servo and has a flat upper section allowing the metallic servo output shaft to rotate along it minimizing vertical oscillation.

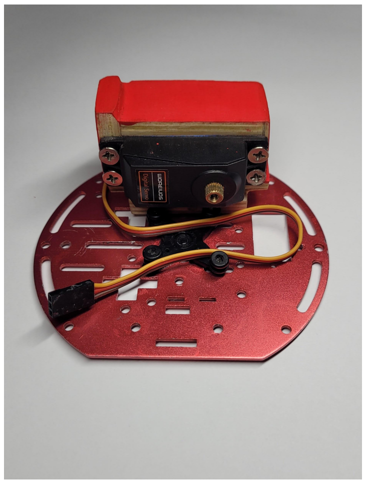


Figure 12. Shoulder servo front view (own work)

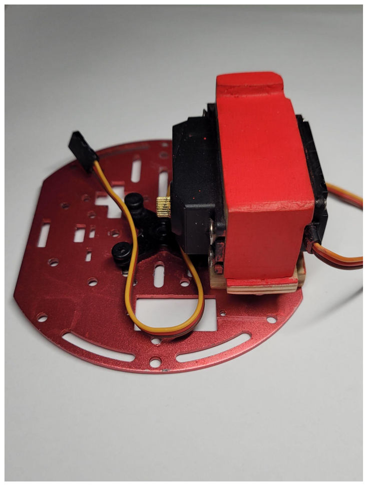


Figure 13. Shoulder servo side view (own work)

On top of the flat upper section is the shoulder servo[5], secured several layers of plywood housing to prevent oscillation. The output shaft of the servo is made out of aluminum and is attached to the metal shoulder (Fig.1, L_1), having a length of 18.7cm from the shoulder to elbow servo joint.

[4] Qy3240mg High Speed Metal Gear 40kg
[5] WOAEIUOS 25KG RC Digital Servo

This metal body is reinforced by several layers of cut plywood, having been molded to the exact shape of the metallic skeleton to reduce weight. These layers are secured by bolts and hexagonal nuts.

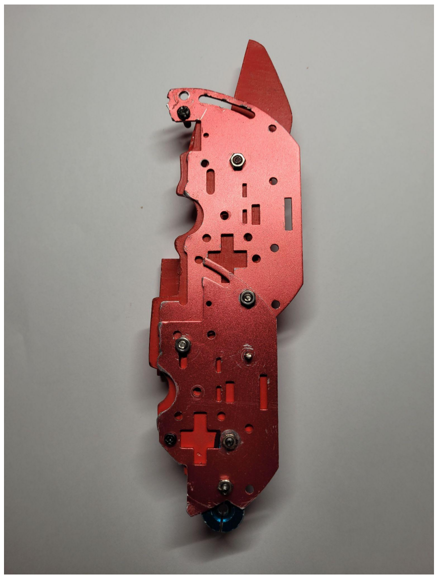


Figure 14. Metal body front view (own work)

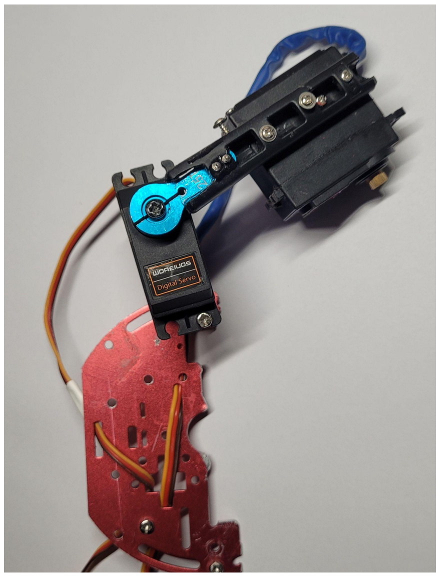


Figure 15. Metal body connected with elbow servo (own work)

Attached to the elbow servo (J3, Fig.1) there is a plastic extension structure that allows another servo to be the second rotational joint (J4, Fig.1).

This servo will rotate the rest of the forearm and the last two joints, being powered by Micro MG90D Servos (J5, J6, Fig.1). The end effector will be able to be attached and removed at the end of the sixth joint according to the needed task.

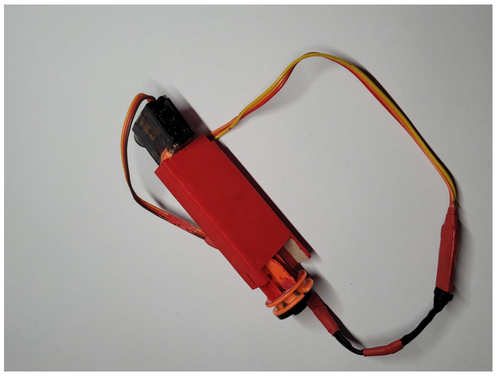


Figure 16. Forearm structure (own work)

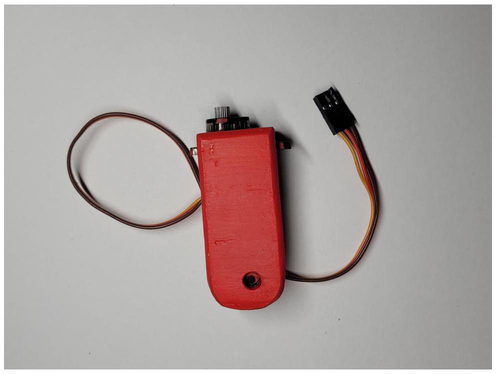


Figure 17. Hand joint (own work)

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.

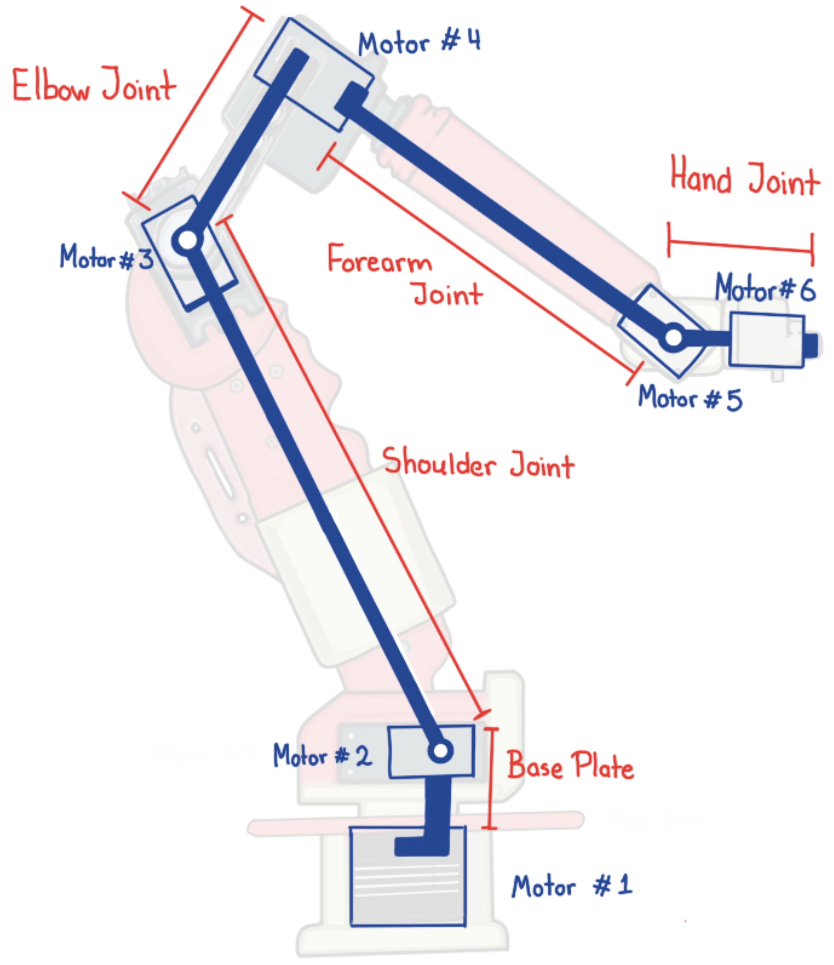


Figure 18. Design of full robot arm structure with nomenclature (own work)

Table 1. with measurements and weight of the vertical joints and the base plates needed for torque calculation

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

3.2.2. 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, that elevates the third servo 6.7 cm above of the second servo axis and rotates it by 90 degrees. This results in the relative range of the shoulder joint going from 0-180 to 90-270. Regardless of the range staying the same, because of the displacement, the elbow joint does not go into the shoulder joints workspace, so it allows the robot to completely fold into itself and avoid colliding with its own joints.

This is vital not to constrict the possible range of motion on the 2d planar space. The maximum extended range is also made greater, as at its maximum stretch the robot has around 6 cm more range.

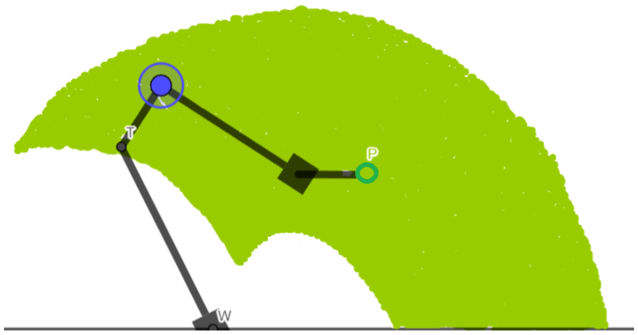


Figure 19. work range when Theta2 < 180°

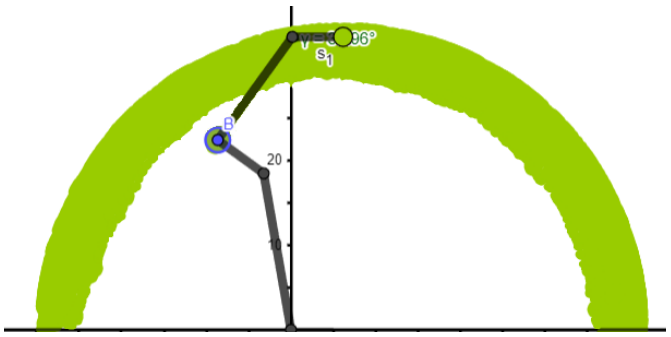


Figure 20. ork range when Theta2 > 180°

3.2.3. Calculation of theoretical motor torque limitations

Using the information provided by the servo manufacturers about the torque capabilities of the servo motors using ideal voltage, alongside measurements of length and weight of the motors, joints and cables, we can use physical formulas in order to derivate how much torque at the end effector is maximal possible, as well as how much torque each of the servo motors are subjected to.

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 (5 V) input, the other to ground the voltage (GND). They are routed into either the Arduinos default 5 V power supply and ground, or an external power supply, for example four AA batteries connected in series. They are colored red and black for 5 V and GND respectively. The cables are to be rooted efficiently through the body of the robot arm, as to not impede the range of motion. This is important to take into account 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 male/female jumper cables, individual male/male jumper cables need to be connected to the "OUT" signal pins of the arduino. According to the power requirements, and torque calculations of each servo motor, an efficient way of connecting all electrical circuits 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 along a serial circuit, to reach a voltage of 6 Volts. Another four AA batteries are used and a parallel circuit is formed, 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 appliance []. 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.

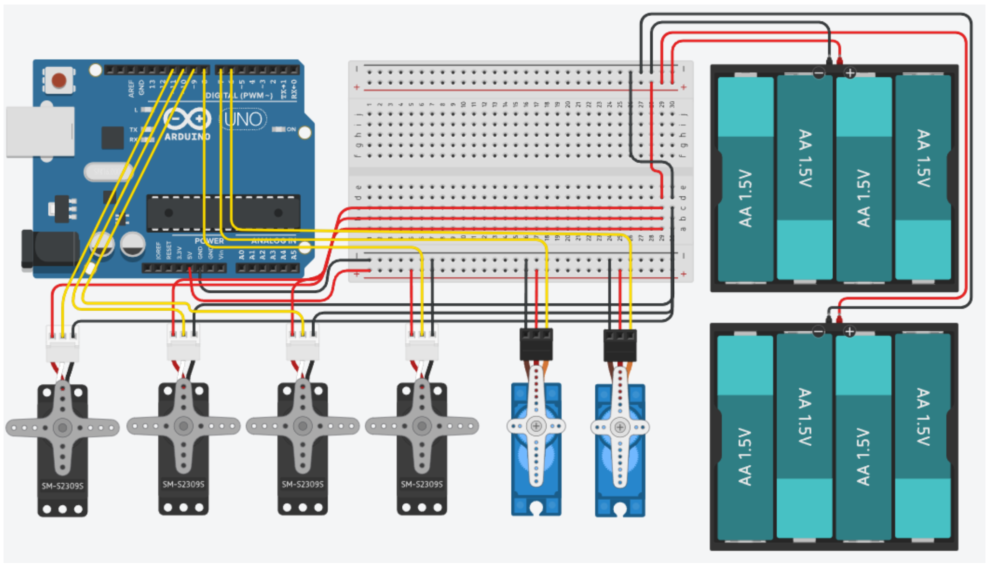


Figure 21. Circuit designed with TinkerCad (own work)

Compared with the theoretical circuit board diagram, made using an open source software called TinkerCad[6] that shows the routing of each cable in a very compact form, allowing for an easier overview of the whole circuit.

Individual motor range errors

3.2.7. Input device

[6]https://www.tinkercad.com/things/klqVVg9XwPh-shiny-gaaris/editel?returnTo=%2Fdashboard%3Ftype%3Dcircuits%26collection%3Ddesigns

4. Software

4.1. Synopsis

In this section, the main focus shifts towards the intelligent control and coordination of the six degrees of freedom of the servo robot. This entails the development of inverse kinematic algorithms that allow precise and efficient manipulation of the end effector position and orientation. We will propose and implement two methods for such a control and evaluate each of them in the results chapter on the following parameters: precision, smoothness, flow of movement, repeatability and user interface usability.

4.2. Forward kinematic analysis

Forward Kinematics, as explained in chapter *2.5* refers to determining the positioning and orientation of the end effector with the given values for the joint angles [7] We can derive

the two equations for the position of the end effector by using geometric definitions.

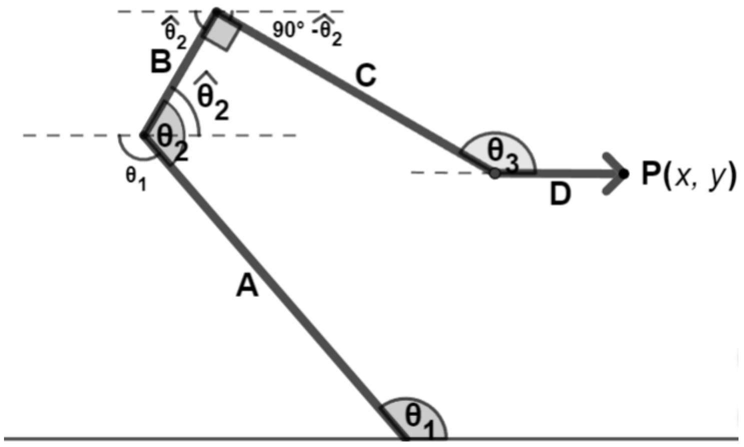


Figure 22. Vector graphic of the arm with joint angles (own work)

We can derive the two equations for the position of the end effector by using geometric analysis. In particular, the position of all the effectors is determined by the angles θ_1 and $\theta_2$we reach the coordinates of the end point *P (x,y).*

[7] https://ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/multiscaleroboticlab-dam/documents/trm/2017/04_2017-10-23_ForwardKinematics.pdf

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 [8]. and a more useful angle θ_{2v}. We express θ_2 as

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

Using this new angle $\widehat{\theta}_2$along with the existing θ_1, we can now add up the horizontal components using cosine, and the vertical components of the joint vectors *A*, *B*, *C*, *D* using sine functions in order to write equations for the endpoints *x* and *y.* The vector for the hand joint D $\begin{pmatrix}4\\0\end{pmatrix}$ is assumed to be.

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

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

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. [9] We can rewrite our equations as:

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

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

Furthermore, our definition of $\theta_2 + \widehat{\theta}_2 + 180° - \theta_1$ from above, can be transformed into $\widehat{\theta}_2 = \theta_1 + \theta_2 - 180°$ 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

8 https://thirdspacelearning.com/gcse-maths/geometry-and-measure/angles-in-parallel-lines/

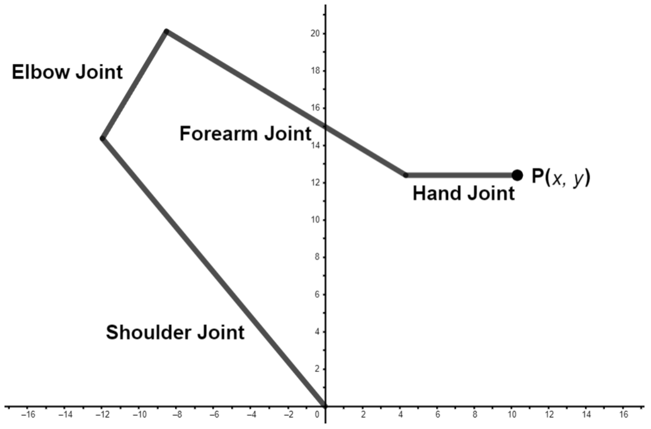
9 https://www.math-only-math.com/trigonometrical-ratios-of-90-degree-minus-theta.html

Now that we have planted our forward kinematic equations, we use them in our derivative approach in the coming chapter *3.3*, 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 subchapter, a trigonometric solution to derive our inverse kinematic equations is explained.

4.3. Trigonometrical approach

In the context of the trigonometrical solution, the primary objective 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 intuitive visualization of the arm's possible configurations.[]

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



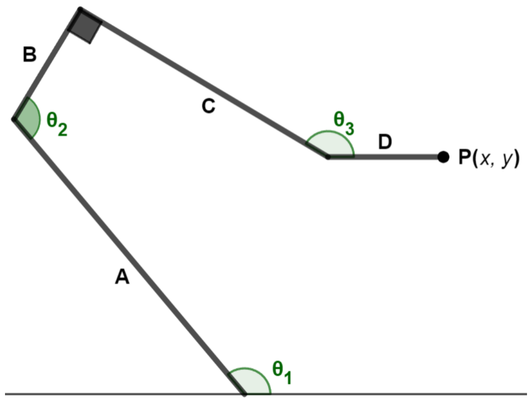


Figure 23. Figure 24.

We can then define angles θ_1, θ_2 and θ_3 as the desired angles for the vertical joints (Figure 24, right). These joints will also be renamed as *A*, *B*, *C* and *D* for the simplicity of the equations, we already know their exact lengths.

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.

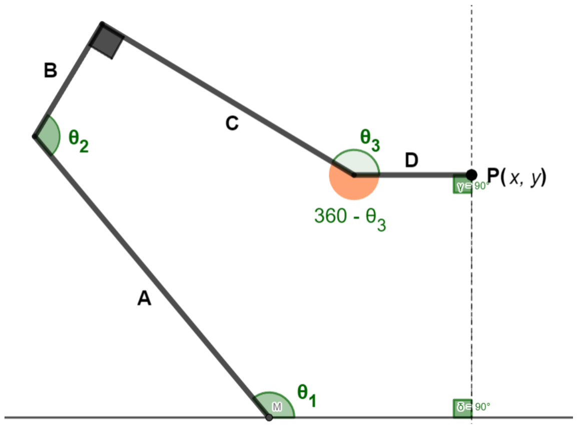


Figure 25.

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

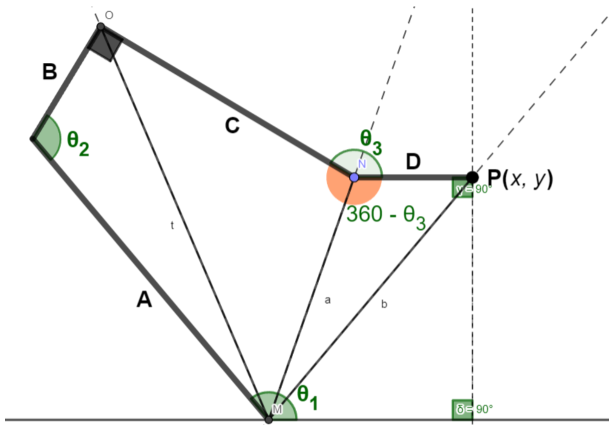


Figure 26.

We can define the sum of all internal angles of the polygon, then solve for theta3.

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

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

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.

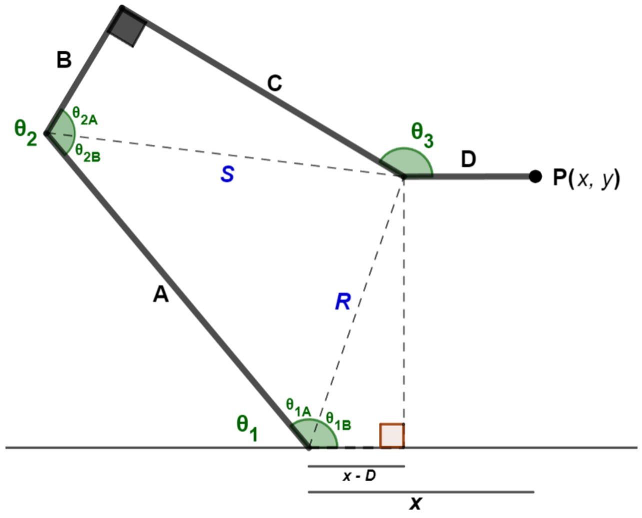


Figure 27.

Tracing two lines, *S* and *R*, 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 chapter *3.2*. In a further chapter a small 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 for θ_{2A} would be $\theta_{2A} = arccos(\frac{B}{C})$.

As defined, $\theta_1 = \theta_{1A} + \theta_{1B}$ and $\theta_2 = \theta_{2A} + \theta_{2B}$, and we can derive equations for each angle θ_1, θ_2 and θ_3.

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

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

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

These three equations are our inverse kinematic equations, as already mentioned, 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 to the point *P(x, y)*.

4.4. Derivative Approach

The derivative approach differentiates itself 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 (Δx, Δy), 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 θ_1, θ_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, we have that

... [complete]

4.5. 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 aims to create an accessible and interactive robotic system capable of responding to user input for movement control. To enhance the efficiency and guarantee the correctness and extensibility of the implementation, it is essential to consider optimization techniques and code structuring. This not only facilitates faster computation but also ensures the smooth execution of the control algorithm within the limited resources of the Arduino microcontroller.

4.5.1. Software Architecture

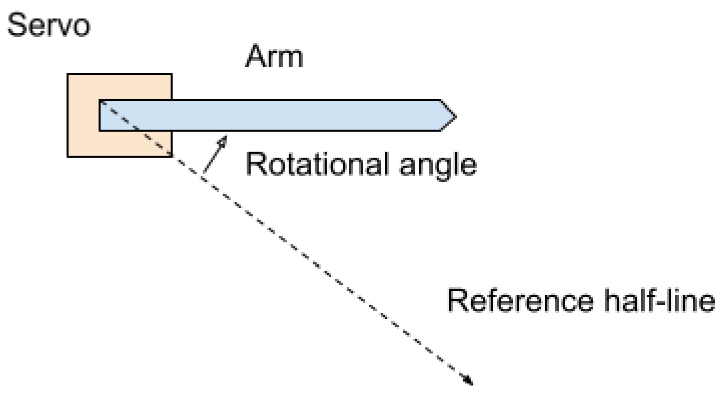
The software architecture is designed with modularity in mind, promoting code organization and readability. Essential libraries for servo motor control are included (`#include <Servo.h>`) and utility functions for logging and enumerations for logging levels are defined. The use of function pointers for logging callbacks allows monitoring the system's behavior in real-time, and is crucial 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 (Fig.#). This is achieved with an enumeration called `LoggingEnum`. It enumerates various log levels, namely `Fatal`, `Error`, `Warn` *(warning)*, `Info` *(informational),* and `Debug`.

4.5.2. 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.



(Figure 28.) Figure 4.5.2. `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`).

```
class ServoArm {

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

   void moveTo(double angle);

   void moveBy(double delta);

   double currentAngle();

   double length();
};
```

Additionally, 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. Such a mapping is specified via the struct `MapRange`.

```
   struct MapRange {
    double lower_bound;
    double upper_bound;
    double servo_to_lower_bound;
    double servo_to_upper_bound;
  };
```

Finally, error handling is implemented to check if the specified angle is within the allowed range, and warning messages are logged in case of out-of-range movements.

4.5.3. 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.

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.

```
constexpr double pi = 3.14159265358979323846264 33;

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 chapter 3.3 which returns the angles θ_1 and θ_2 (where θ_3 is determined due to the condition of horizontality of the hand effector).

```
AngularCoordinates _calculateAngularCoordinates(
   CartesianCoordinates cartesian_coordinates) {
   double x = cartesian_coordinates.x;
   double y = cartesian_coordinates.y;
   double A = _shoulder->length();
   double B = _elbow->length();
   double C = _forearm_length;
   double D = _hand->length();
   double R = sqrt(pow((x - D), 2) + pow(y, 2));
   double S = sqrt(pow(B, 2) + pow(C, 2));
   double shoulder_angle = acosDegrees((x - D)/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};
 }


double _calculateHandAngle(AngularCoordinates angular_coordinates) {
   return angular_coordinates.shoulder_angle +
      angular_coordinates.elbow_angle - 90;
}
```

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 _moveByWithExactMethod(double delta_x, double delta_y) {
   CartesianCoordinates current_cartesian_coordinates =
currentCartesianCoordinates();
   CartesianCoordinates projected_cartesian_coordinates = {
     x: current_cartesian_coordinates.x + delta_x,
     y: current_cartesian_coordinates.y + delta_y};
   AngularCoordinates projected_angular_coordinates =
     _calculateAngularCoordinates(projected_cartesian_coordinates);
   if (isnan(projected_angular_coordinates.shoulder_angle)
     || isnan(projected_angular_coordinates.elbow_angle)) {
     _logging(
     LoggingEnum::WARN,
       "Moving the robot to the impossible position "
         + projected_cartesian_coordinates.toString());
     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.

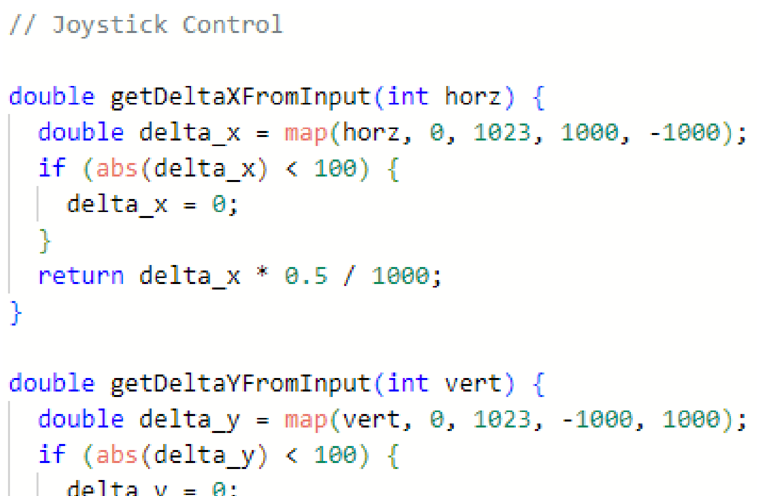
```
void _moveByWithDerivativeMethod(double delta_x, double delta_y) {
  CartesianCoordinates current_cartesian_coordinates =
    currentCartesianCoordinates();
  CartesianCoordinates expected_cartesian_coordinates = {
    x: current_cartesian_coordinates.x + delta_x,
    y: current_cartesian_coordinates.y + delta_y};
  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.

```
  // ...
  double delta_shoulder_angle = (
    delta_x * angular_derivatives.y_by_elbow_angle
    - delta_y * angular_derivatives.x_by_elbow_angle) / determinant;
  double delta_elbow_angle = (
    delta_y * angular_derivatives.x_by_shoulder_angle
    - delta_x * angular_derivatives.y_by_shoulder_angle) / determinant;
  AngularCoordinates projected_angular_coordinates =  {
    shoulder_angle: _shoulder->currentAngle() + delta_shoulder_angle,
    elbow_angle: _elbow->currentAngle() + delta_elbow_angle};
  CartesianCoordinates projected_cartesian_coordinates =
    _calculateCartesianCoordinates(projected_angular_coordinates);
  _logging(
    LoggingEnum::INFO,
    "Target: " + expected_cartesian_coordinates.toString()
      + ". Actual: " +  projected_cartesian_coordinates.toString());
  AngularDerivatives projected_angular_derivatives =
    _calculateAngularDerivatives(projected_angular_coordinates);
  double projected_determinant =
    _calculateDeterminant(projected_angular_derivatives);
  if (abs(projected_determinant) < _differential_stability_threshold) {
    _logging(
      LoggingEnum::WARN,
      "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);
  return;
}
```

4.5.4. 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 chapter 2.*. 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 scale of these variables.

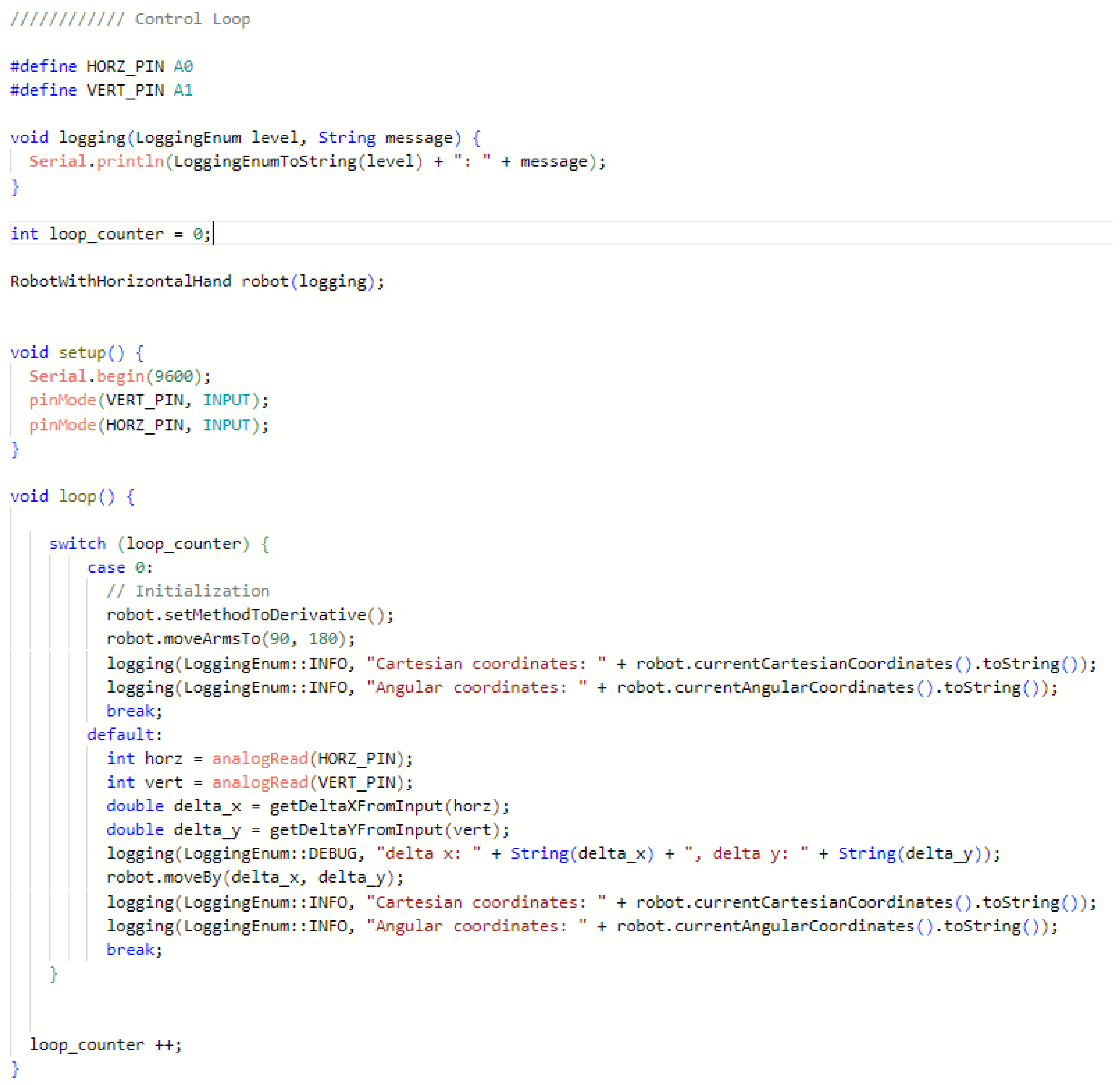


```
// Joystick Control

double getDeltaXFromInput(int horz) {
  double delta_x = map(horz, 0, 1023, 1000, -1000);
  if (abs(delta_x) < 100) {
    delta_x = 0;
  }
  return delta_x * 0.5 / 1000;
}

double getDeltaYFromInput(int vert) {
  double delta_y = map(vert, 0, 1023, -1000, 1000);
  if (abs(delta_y) < 100) {
    delta_y = 0;
```

4.5.5. Control Loop

The final part of the software implementation is the control loop, which defines the analog pins that the joystick is connected to and prints the logging messages through the serial monitor of the Arduino.

We can manually change the method used for trajectory by either calling `robot.setMethodtoDerivative();` or `robot.setMethodtoExact();`



```
//////////// Control Loop

#define HORZ_PIN A0
#define VERT_PIN A1

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

int loop_counter = 0;

RobotWithHorizontalHand robot(logging);


void setup() {
  Serial.begin(9600);
  pinMode(VERT_PIN, INPUT);
  pinMode(HORZ_PIN, INPUT);
}

void loop() {

  switch (loop_counter) {
    case 0:
      // Initialization
      robot.setMethodToDerivative();
      robot.moveArmsTo(90, 180);
      logging(LoggingEnum::INFO, "Cartesian coordinates: " + robot.currentCartesianCoordinates().toString());
      logging(LoggingEnum::INFO, "Angular coordinates: " + robot.currentAngularCoordinates().toString());
      break;
    default:
      int horz = analogRead(HORZ_PIN);
      int vert = analogRead(VERT_PIN);
      double delta_x = getDeltaXFromInput(horz);
      double delta_y = getDeltaYFromInput(vert);
      logging(LoggingEnum::DEBUG, "delta x: " + String(delta_x) + ", delta y: " + String(delta_y));
      robot.moveBy(delta_x, delta_y);
      logging(LoggingEnum::INFO, "Cartesian coordinates: " + robot.currentCartesianCoordinates().toString());
      logging(LoggingEnum::INFO, "Angular coordinates: " + robot.currentAngularCoordinates().toString());
      break;
  }


  loop_counter ++;
}
```

The software implementation detailed in this section establishes a robust foundation for the development of an accessible and interactive robotic system. The modular software architecture promotes ease of maintenance and future expansion. The `ServoArm` and `RobotWithHorizontalHand` classes encapsulate necessary functionalities, such as servo motor control and precise system movement and orientation.

5. Results

5.1. Synopsis

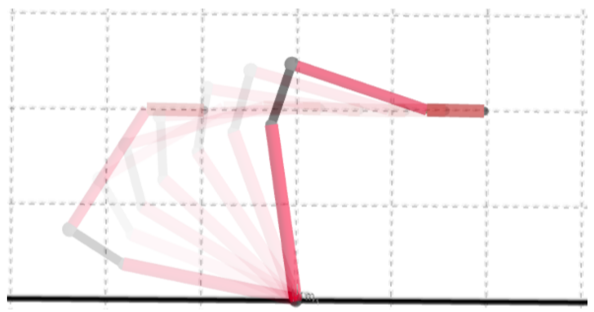
To interpret the robot arm's movement capabilities with concrete criteria, a series of experiments were conducted using test persons. A group of 5 individuals was asked to pilot the robot arm to follow a predetermined path, simulating a task requiring precise movement. In the second experiment they were 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.

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. 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)

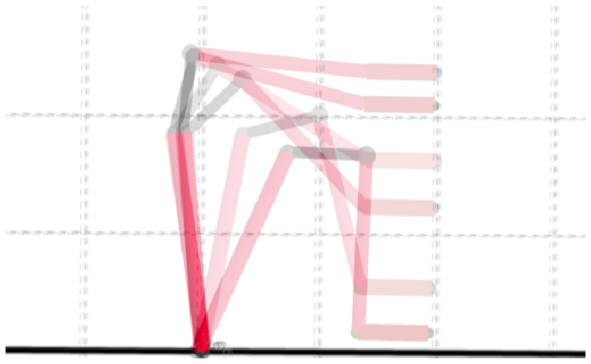
5.2.1. Horizontal Movement (Left ⇒ Right / Right ⇒ Left)



The test individuals were instructed to pilot the robot using the joystick following a straight horizontal line, both from left to right and right to left each only one attempt. They were next asked the questions from above, below is a table with their answers.

	Person 1	Person 2	Person 3	Person 4	Person 5
Q1.1	Yes	Yes	Yes	Yes	Yes
Q1.2	Yes	Yes	Yes	Yes	Yes
Q1.3	Yes	Yes	Yes	Yes	Yes

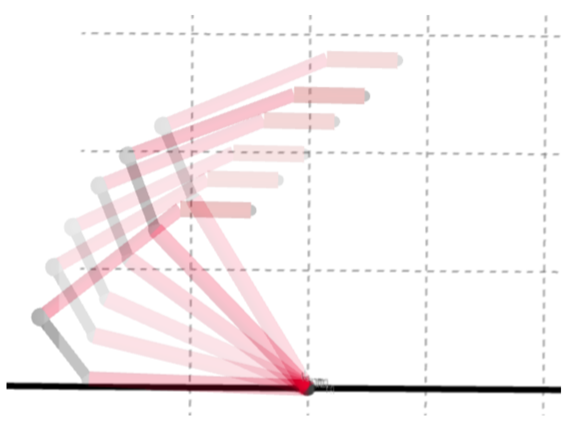
5.2.2. Vertical Movement (Up ⇒ Down / Down ⇒ Up)

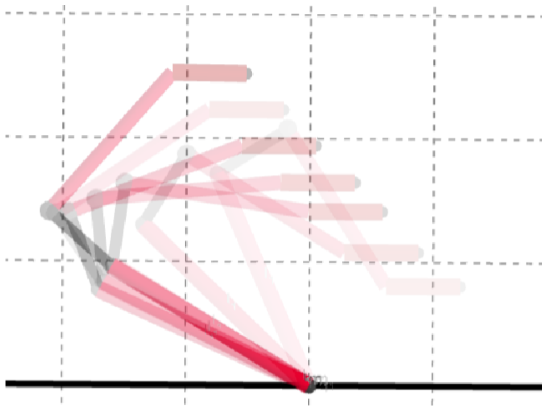


The individuals were told to pilot the robot using the joystick in a straight vertical line, first starting from above going down, then from below going up, each only one attempt. They were then asked the same questions.

	Person 1	Person 2	Person 3	Person 4	Person 5
Q1.1	Yes	Yes	Yes	Yes	Yes
Q1.2	No	Yes	Yes	No	Yes
Q1.3	Yes	Yes	Yes	Yes	Yes

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


Diagonal Movement 45°


Diagonal Movement 135°

The test individuals were instructed to pilot 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 or top left to bottom right, in both directions.

Diagonal Movement 45°

	Person 1	Person 2	Person 3	Person 4	Person 5
Q1.1	Yes	No	No	Yes	Yes
Q2	No	Yes	Yes	Yes	Yes
Q3	Yes	Yes	Yes	Yes	Yes

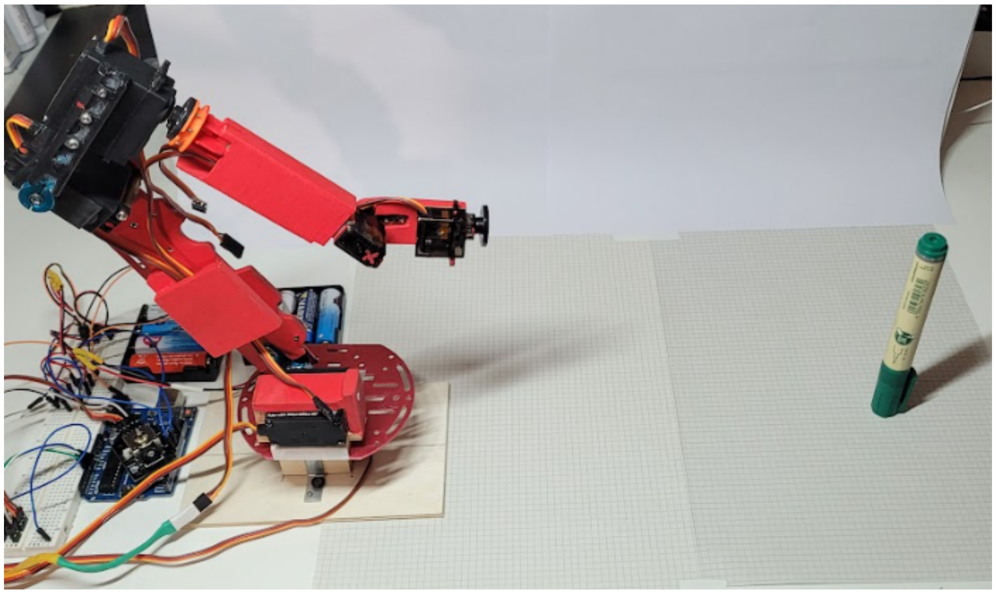
Diagonal Movement 135°

	Person 1	Person 2	Person 3	Person 4	Person 5
Q1.1	No	Yes	Yes	Yes	No
Q1.2	No	No	Yes	No	Yes
Q1.3	Yes	Yes	Yes	Yes	Yes

5.3. Protocol two: Trajectory towards given point

The robot is to start at a canonical predetermined position, the test individuals are then instructed to pilot the robot with the joystick towards a fixed point 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 three attempts to do so within 10 seconds. They are then asked the following questions:

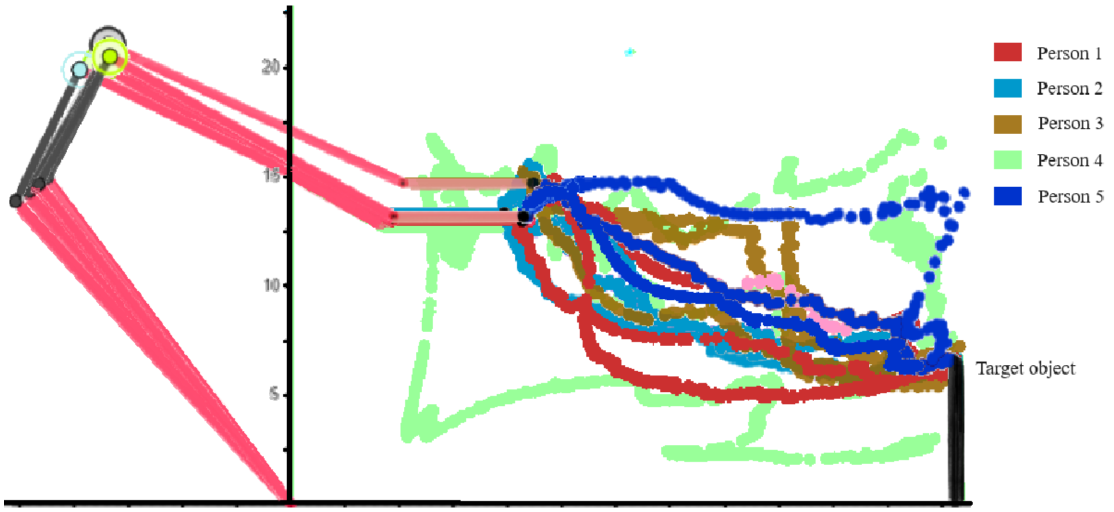
- Question #2.1: *Could you command the robot to the target object and tip it over ?* - (testing correctness)
- Question #2.2: *Did you have to do readjustments to reach the target object?* - (to test for smoothness)
- Question #2.3: *Could the robot achieve the target object in a repeatable way ?* -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


Experiment environment side view

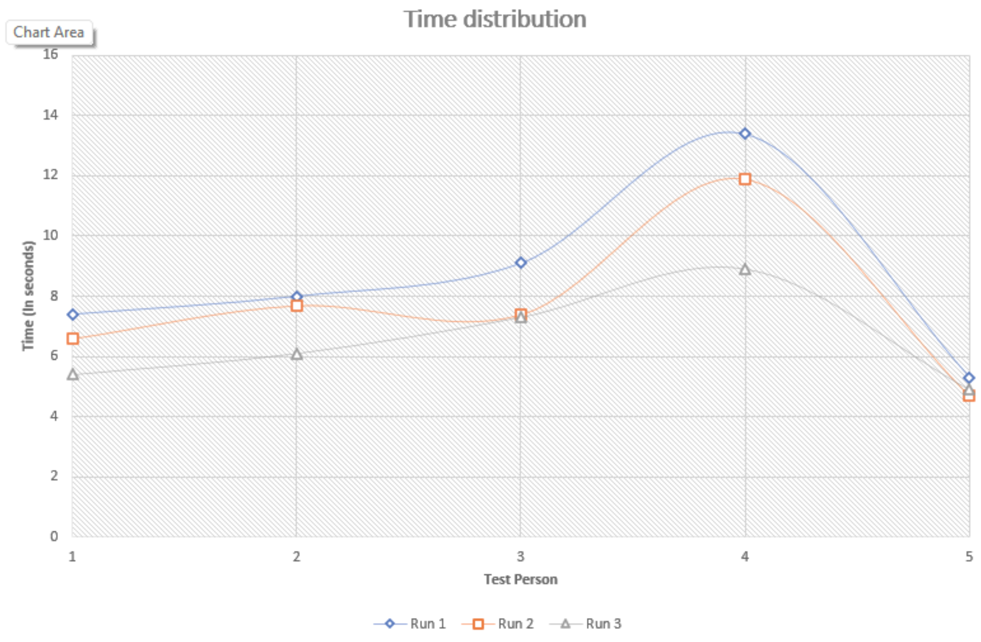
	Person 1	Person 2	Person 3	Person 4	Person 5
Q2.1	Yes	Yes	Yes	Yes	Yes
Q2.2	Yes	No	No	Yes	No
Q2.3	Yes	Yes	Yes	Yes	Yes

For Question #2: *Did you have to do readjustments to reach the target object?* an answer with “No” is **preferred,** and a “Yes” is **not**. Thus the different coloring scheme.

Each person conducted the experiment 3 times, resulting in a total of 15 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. Of the total 15 runs, 13 were successful in tipping over the object. The two unsuccessful runs were both from test person 4, taking 13.4 and 11.9 seconds.



The time it took the test person from the first input of the joystick to the successful displacement of the target object was measured in seconds for each of the three attempts and plotted in a graph.



Quality assurance in order to see what are the weak points (drawbacks) for every method.

Corrections should be done from a user point of view

6. Discussion

6.1 Results

4.1.1 Interpretation

The findings of this study entail a comprehensive and objective evaluation, undertaking a comparative analysis of various factors and performance metrics through multiple test iterations. We specifically compared two main ways of controlling the robot, as explained in the software chapter. One method relies on exact calculations using trigonometry, while the other uses a solution based on derivatives, looking at relative changes in values. The results from lots of tests point out the subtle differences in how well these two control methods work. Subjective evaluations were also conducted using test persons controlling the arm with an input device mentioned in chapter 2.3.2, asking the test persons to rate the maneuverability and comprehension of the controls.

This analysis helps us better grasp the pros and cons of using precise trigonometry-based control versus the relative control based on derivatives in the software we've presented.

Repeatability - is how well the robot will return to a programmed position. This is not the same as accuracy. It may be that when told to go to a certain X-Y-Z position that it gets only to within 1 mm of that position. This would be its accuracy which may be improved by calibration. But if that position is taught into controller memory and each time it is sent there it returns to within 0.1mm of the taught position then the repeatability will be within 0.1mm.
Accuracy and repeatability are different measures. Repeatability is usually the most important criterion for a robot. ISO 9283 sets out a method whereby both accuracy and repeatability can be measured. Typically a robot is sent to a taught position a number of times and the error is measured at each return to the position after visiting 4 other

positions. Repeatability is then quantified using the standard deviation of those samples in all three dimensions.

6.1. Cost Effectiveness

6.2. Improvements

6.3. Learning Bits

7. Bibliography

Arduino, S. A. (2015). *Arduino*. Arduino LLC, 372. https://search.iczhiku.com/paper/TFzDJhGhd6VMaDsI.pdf

Aristidou, A.. Lasenby, J., Chrysanthou, Y., &, Shamir, A. (2017). Inverse Kinematics Techniques in Computer Graphics: A Survey https://doi.org/10.1111/cgf.13310

Barret, S., & Pack, D., 2022. Microcontrollers fundamentals for engineers and scientists. Springer Nature Switzerland AG 2022

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

Keim, R. (March, 2019). *Defining Characteristics and Architecture of a Common Component*. https://www.allaboutcircuits.com/technical-articles/what-is-a-microcontroller-introduction-component-characteristics-component/

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

Mahadevkar, S. V., Khemani, B., Patil, S., Kotecha, K., Vora, D.R., Abraham, A., Abdelkareim, L.G. (2022). *A Review on Machine Learning Styles in Computer Vision—Techniques and Future Directions*. In IEEE Access, vol. 10, pp. 107293-107329, 2022, doi: https://ieeexplore.ieee.org/document/9903420

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
https://rdcu.be/du9ne

Figures

Figure 1.
https://robohub.org/wp-content/uploads/2015/12/2391_Cartesian_Coordinates.png
Figure 2.
https://www.researchgate.net/profile/Mario-Saenz-Espinoza/publication/259009627/figure/fig9/AS:668221367660551@1536327797352/Typical-6DOF-robot-manipulator.ppm

Figure 3. https://cdn.sparkfun.com/assets/custom_pages/5/6/0/servo-parts.jpg
Figure 4. https://granitedevices.com/w/images/d/dd/Controller_general.png
Figure 5. https://upload.wikimedia.org/wikipedia/commons/7/7b/ServoPwm.png
Figure 6. https://ars.els-cdn.com/content/image/1-s2.0-S2405844022006880-gr6_lrg.jpg
Figure 7.
https://www.etechnophiles.com/wp-content/uploads/2023/07/Arduino-UNO-R3-Hardware-overview.jpg?ezimgfmt=ng:webp/ngcb41
https://blog-c7ff.kxcdn.com/blog/wp-content/uploads/2016/10/board-01.jpg

Figure 8.
https://www.researchgate.net/profile/Mohd-Tamin/publication/221908770/figure/fig4/AS:671528576376851@1537116297733/The-axes-of-the-reference-robot-manipulators-Fanuc-Robotics-2008.png