Design and implementation of a purely rotary tobotic manipulator with 4 DOF to pick up and place objects from one place to another through smartphone control

Diseño e implementación de un manipulador tobótico puramente rotativo con 4 GDL para recoger y colocar objetos de un lugar a otro a través del control de un teléfono inteligente

Juan Esteban Chiriboga[1] Melissa Arias[2] Ricardo Yépez[3] Jean Carlo Collaguazo[4]
[1]Escuela de Ingeniería Mecatrónica, Universidad Internacional del Ecuador
• *e-mail: juan.esteban.ch3110@gmail.com* • *e-mail: melyariash@hotmail.com* • *e-mail: ricardoy97@hotmail.com*
• *e-mail: jecollaguazohi@uide.edu.ec*

***Abstract*— The following study describes the design and implementation of a robotic arm that has 4 degrees of freedom, used to lift objects from a certain specific point and leave them in another place. In the document, the kinematic study of the robot can be evidenced, both the mechanical and electronic design, in addition, an application for cell phones was designed and programmed, through which the robot and all its movements will be controlled, in the application it can be controlled in a manual controls by sliding or automatically by direct and inverse kinematics and you can also see the respective tests carried out on the prototype, to obtain smooth and easily controllable movements.**

***Keywords*—robotic arm, direct kinematics, inverse kinematics, degrees of freedom.**

***Resumen*— En el estudio realizado a continuación se describe el diseño e implementación de un brazo robótico que cuanta con 4 grados de libertad, utilizado para levantar objetos desde cierto punto específico y dejarlos en otro lugar. En el documento se puede evidenciar el estudio cinemático del robot, el diseño tanto mecánico como electrónico, en adición se diseño y programó una aplicación para celular, mediante la cual se controlará al robot y todos sus movimientos, en la aplicación se puede controlar de manera manual mediante controles deslizantes o de manera automática mediante cinemática directa e inversa y además se puede apreciar las respectivas pruebas realizadas en el prototipo, para obtener movimientos suaves y fácilmente controlables.**

***Palabras Clave*—brazo robótico, cinemática directa, cinemática inversa, grados de libertad.**

I. Introduction

The term 'manipulator' in industrial robotics refers to mechanisms specifically created to perform tasks such as moving and holding objects. What defines manipulative robots as multifunctional. Furthermore, these devices are basically and easily handled by humans on an external device.

The manipulative robots are made of rigid bars. United through different joints that allow the movement and firmness of the well-known robotic arm. It is even totally affordable with a human arm. That is why the human figure is replaced for different tasks with this useful tool.

Manipulative robots are a tool capable of carrying out the activities indicated by the human. This tool only brings benefits to the company. Well, it prevents your workers from getting hurt by something or simply allows something to be held with precision and balance. In addition to using common tools to do the job easily and quickly.

II. MECHANICAL DESIGN

For the design of the robot, some design ideas were sketched, one of them is presented in Figure [1], whose inspiration design is also visible there.

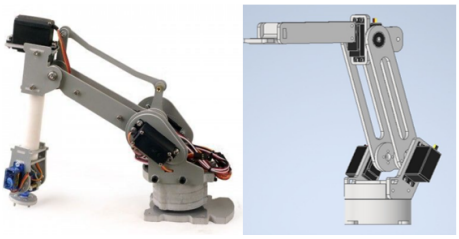

Fig. 1. Inspiration design / Initial prototype idea.[1]

Starting with that basic design, several changes were made, such as:

- Redesigning certain parts according to the dimensions of the elements available.
- The dimensions of the holes were changed and adapted to bults and nuts that exist, in order to assemble it without problems.
- A gripper was added as the end effector, including its mechanism to open and close it, visible in Figure 2. It was chosen to be an end effector that makes a parallel grip instead of an encompassing grip, which means, it makes a movement that catches the elements as shown in Figure [2].

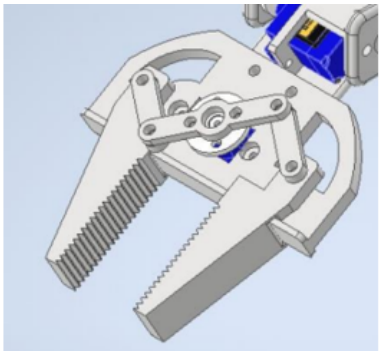

Fig. 2. Gripper.

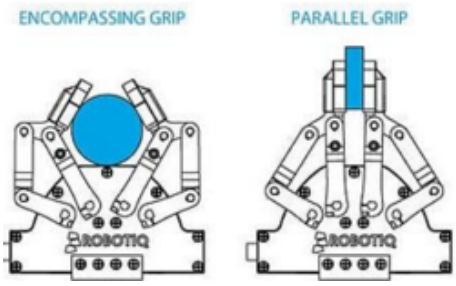

Fig. 3. Types of Grippers.[2]

- • A new degree of freedom was added, considering that this design completely enables 6 movements that are going to be accomplished by 6 servomotors of 2 different types, from these movements, 5 enables the degrees of freedom and 1 enables the gripper mechanism, they are going to be done according to Figure [4], which are:

1. Base
2. Arm
3. Elbow
4. Forearm
5. Wrist
6. Gripper

Fig. 4. Movement's representation .[3]

With all the applied changes in the design, using the software Autodesk Inventor, the final prototype, ready for printing is exposed in the Figure [5]:

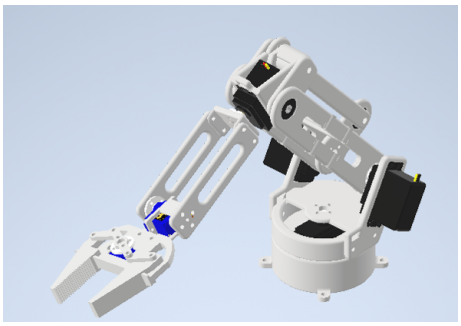

Fig. 5. Isometric view of the final prototype design.

The design is made up of 3 fundamental sections: the base, the principal arm, and the forearm with the gripper.

A. *Base:*

It is the section that contains most of the heavy elements (MG996R servomotors), this was made with the purpose of taking off the weight at higher parts of the robot and prevent the servomotors from making an extra effort that may damage them or even, stop them from working correctly. This section contains the servomotors that act like joints and activate the movement of the base, the shoulder, and the elbow, As is shown in Figure [6].

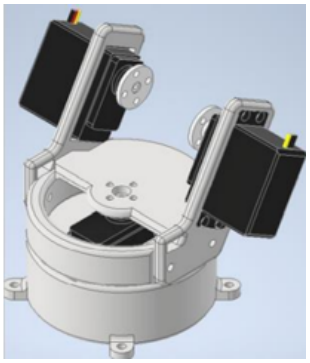

Fig. 6. Subsection 1: Base.

B. *Arm:*

Is made up of two parallel links that act as one as they are joined together by a bridge, this geometry was made to avoid including a solid part that might cause the arm to become extremely heavy; apart from that, there's another joint that is activated by a mechanism that transmits the rotation of the servomotor at the base through the large element at the top that pushes what will be the elbow, as is shown in [7].

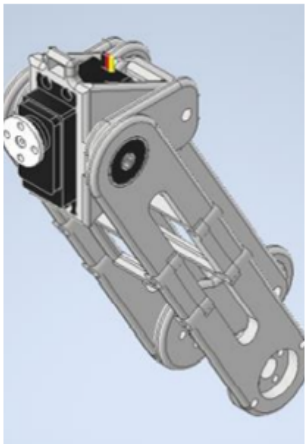

Fig. 7. Subsection 2: Arm.

C. *Gripper:*

It includes the forearm that is also made up of two parallel elements connected by a bridge to keep them in the correct position and using the minimal material as possible; they are joined by a small servomotor that act as the wrist, this electrical element is joined to the end effector whose mechanism is also enabled by a small servomotor. As can be seen, the using small servomotors will help in the objective of making the top of the robot, the less heavy as it can be, as is shown in Figure [8].

This division of the design is also going to be considered when printing, to quick up the manufacturing process of the parts.

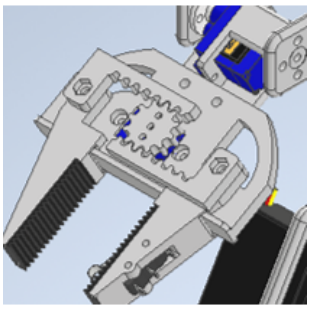

Fig. 8. Subsection 3: Gripper.

III. Prototype simulation

Once the final design was obtained the simulation of the mechanism's movement tests were carried out in the CAD software, by testing different poses of the robot as Figure [9] shows and checking that the mechanism won't crash any part with any other part.

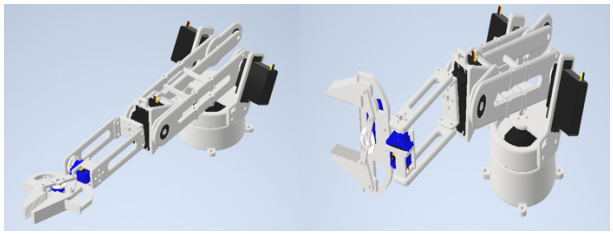

Fig. 9. Mechanism test simulation.

IV. Applying the Denavit-Hartenber Convention

Due to the geometry of the complete assembly (Figure [5]), applying the convention used in class, the scheme of the design will be the one in the Figure [10].

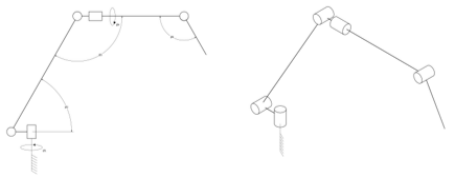

Fig. 10. Robotic manipulator scheme.

Therefore, adding the frames into the diagram using MATLAB toolkit used in class, will look as the on in the Figure [**??**].

Once the frames have been declared, the table corresponding to the Denavit-Hartenberg parameters is made, as can be seen in Table [I].

Then, to get the forward kinematics of the robot MATLAB was used in order to calculate each transformation matrix using algorithm shown in Figure [12].

To final find the homogeneous transformation matrix of the robot by multiplying this transformations matrix previously obtained, also considering that the values of L0-L5 are 8, 3,

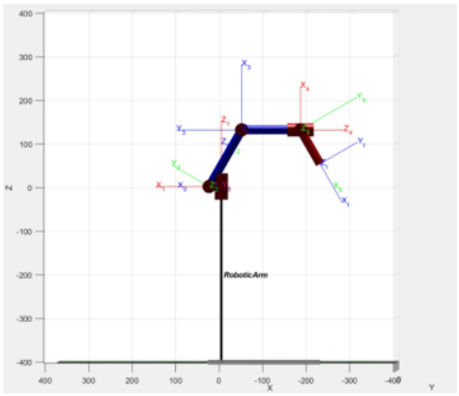

Fig. 11. Frames according to the Denavit Hartenberg convention.

TABLA I
DENAVIT HARTENBERG PARAMETERS

i	$\alpha_i - 1$	$a_i - 1$	d_i	θ_i
1	0	0	L0	q1
2	$[\pi/2]$	-L1	L0	q2
3	0	L2	L0	$[q3 - \pi/2]$
4	$[\pi/2]$	L3	L4	$[q4 - \pi/2]$
5	$[\pi/2]$	0	0	q5

```
for i=1:5
   frame{i}=[cos(dh(i,4)) -sin(dh(i,4)) 0 dh(i,2)
    sin(dh(i,4))*cos(dh(i,1)) cos(dh(i,4))*cos(dh(i,1)) -sin(dh(i,1)) -dh(i,3)*sin(dh(i,1))
    sin(dh(i,4))*sin(dh(i,1)) cos(dh(i,4))*sin(dh(i,1)) cos(dh(i,1)) dh(i,3)*cos(dh(i,1))
    0 0 0 1];
end
for i=1:5
    fprintf('frame %d with respect to %d ',i,i-1)
    disp(frame{i})
end
```

frame 1 with respect to 0

$$\begin{pmatrix} \cos(q_1) & -\sin(q_1) & 0 & 0 \\ \sin(q_1) & \cos(q_1) & 0 & 0 \\ 0 & 0 & 1 & L_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

frame 2 with respect to 1

$$\begin{pmatrix} \cos(q_2) & -\sin(q_2) & 0 & -L_1 \\ 0 & 0 & -1 & 0 \\ \sin(q_2) & \cos(q_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

frame 3 with respect to 2

$$\begin{pmatrix} \cos\left(q_3 - \frac{\pi}{2}\right) & -\sin\left(q_3 - \frac{\pi}{2}\right) & 0 & L_2 \\ \sin\left(q_3 - \frac{\pi}{2}\right) & \cos\left(q_3 - \frac{\pi}{2}\right) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

frame 4 with respect to 3

$$\begin{pmatrix} \cos\left(q_4 - \frac{\pi}{2}\right) & -\sin\left(q_4 - \frac{\pi}{2}\right) & 0 & L_3 \\ 0 & 0 & -1 & -L_4 \\ \sin\left(q_4 - \frac{\pi}{2}\right) & \cos\left(q_4 - \frac{\pi}{2}\right) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

frame 5 with respect to 4

$$\begin{pmatrix} \cos(q_5) & -\sin(q_5) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin(q_5) & \cos(q_5) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

frame 6 with respect to 5

$$\begin{pmatrix} 1 & 0 & 0 & L_5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Fig. 12. Algorithm used to get the transformation matrix.

15, 1, 12 and 9 respectively, additionally using the command "simplify" on MATLAB to get the matrix in the most simplified way.

$$Homogtransmatrix = \begin{pmatrix} t11 & t12 & t13 & t14 \\ t21 & t22 & t23 & t24 \\ t31 & t32 & t33 & t34 \\ t41 & t42 & t43 & t44 \end{pmatrix} \quad (1)$$

where:

- $t11 = cos(q1) * sin(q2) * sin(q3) * sin(q5) - cos(q1) * cos(q2) * cos(q3) * sin(q5) - cos(q4) * cos(q5) * sin(q1) + cos(q1) * cos(q2) * cos(q5) * sin(q3) * sin(q4) + cos(q1) * cos(q3) * cos(q5) * sin(q2) * sin(q4)$
- $t12 = cos(q4) * sin(q1) * sin(q5) - cos(q1) * cos(q2) * cos(q3) * cos(q5) + cos(q1) * cos(q5) * sin(q2) * sin(q3) - cos(q1) * cos(q2) * sin(q3) * sin(q4) * sin(q5) - cos(q1) * cos(q3) * sin(q2) * sin(q4) * sin(q5)$
- $t13 = -sin(q1) * sin(q4) - cos(q1) * cos(q2) * cos(q4) * sin(q3) - cos(q1) * cos(q3) * cos(q4) * sin(q2)$
- $t14 = 15 * cos(q1) * cos(q2) - 3 * cos(q1) - 12 * cos(q1) * cos(q2) * cos(q3) + cos(q1) * cos(q2) * sin(q3) + cos(q1) * cos(q3) * sin(q2) - 9 * cos(q4) * cos(q5) * sin(q1) + 12 * cos(q1) * sin(q2) * sin(q3) - 9 * cos(q1) * cos(q2) * cos(q3) * sin(q5) + 9 * cos(q1) * sin(q2) * sin(q3) * sin(q5) + 9 * cos(q1) * cos(q2) * cos(q5) * sin(q3) * sin(q4) + 9 * cos(q1) * cos(q3) * cos(q5) * sin(q2) * sin(q4)$
- $t21 = cos(q1) * cos(q4) * cos(q5) - cos(q2) * cos(q3) * sin(q1) * sin(q5) + sin(q1) * sin(q2) * sin(q3) * sin(q5) + cos(q2) * cos(q5) * sin(q1) * sin(q3) * sin(q4) + cos(q3) * cos(q5) * sin(q1) * sin(q2) * sin(q4)$
- $t22 = cos(q5) * sin(q1) * sin(q2) * sin(q3) - cos(q2) * cos(q3) * cos(q5) * sin(q1) - cos(q1) * cos(q4) * sin(q5) - cos(q2) * sin(q1) * sin(q3) * sin(q4) * sin(q5) - cos(q3) * sin(q1) * sin(q2) * sin(q4) * sin(q5)$
- $t23 = cos(q1) * sin(q4) - cos(q2) * cos(q4) * sin(q1) * sin(q3) - cos(q3) * cos(q4) * sin(q1) * sin(q2)$
- $t24 = 15 * cos(q2) * sin(q1) - 3 * sin(q1) + 12 * sin(q1) * sin(q2) * sin(q3) + 9 * cos(q1) * cos(q4) * cos(q5) - 12 * cos(q2) * cos(q3) * sin(q1) + cos(q2) * sin(q1) * sin(q3) + cos(q3) * sin(q1) * sin(q2) - 9 * cos(q2) * cos(q3) * sin(q1) * sin(q5) + 9 * sin(q1) * sin(q2) * sin(q3) * sin(q5) + 9 * cos(q2) * cos(q5) * sin(q1) * sin(q3) * sin(q4) + 9 * cos(q3) * cos(q5) * sin(q1) * sin(q2) * sin(q4)$
- $t31 = cos(q5) * sin(q2) * sin(q3) * sin(q4) - cos(q3) * sin(q2) * sin(q5) - cos(q2) * cos(q3) * cos(q5) * sin(q4) - cos(q2) * sin(q3) * sin(q5)$
- $t32 = cos(q2) * cos(q3) * sin(q4) * sin(q5) - cos(q3) * cos(q5) * sin(q2) - cos(q2) * cos(q5) * sin(q3) - sin(q2) * sin(q3) * sin(q4) * sin(q5)$
- $t33 = cos(q2 + q3) * cos(q4)$
- $t34 = 15 * sin(q2) - cos(q2) * cos(q3) - 12 * cos(q2) * sin(q3) - 12 * cos(q3) * sin(q2) + sin(q2) * sin(q3) - 9 * cos(q2) * sin(q3) * sin(q5) - 9 * cos(q3) * sin(q2) *$

$sin(q5) - 9 * cos(q2) * cos(q3) * cos(q5) * sin(q4) + 9 * cos(q5) * sin(q2) * sin(q3) * sin(q4) + 8$

- $t41 = 0$
- $t42 = 0$
- $t43 = 0$
- $t44 = 0$

V. Jacobian calculation

For the calculation of the Jacobians, the position previously obtained with the forward kinematics is used, to after by using the Jacobians definition (partial derivatives) method get the values of the Jacobians:

$$P = \begin{pmatrix} t14 \\ t24 \\ t34 \end{pmatrix} \tag{2}$$

MATLAB was used to calculate the partial derivatives, using the algorithm of the Figure [13].

```
q={q1 q2 q3 q4 q5};
p=fkinematics(1:3,end);
for i=1:3
    for j=1:5
        Jtool(i,j)=simplify(diff(p(i),q{j}));
    end
end
disp(Jtool)
```

Fig. 13. Algorithm used to calculate the partial derivatives (Jacobian)

To finally have the Jacobians, also using command "simplify" to get the values in the most simplified way.

$$J^0 = \begin{pmatrix} J11 & J12 & J13 & J14 & J15 \\ J21 & J22 & J23 & J24 & J25 \\ J31 & J32 & J33 & J34 & J35 \end{pmatrix} \tag{3}$$

where:

- $J11 = 3 * sin(q1) - 15 * cos(q2) * sin(q1) - 12 * sin(q1) * sin(q2) * sin(q3) - 9 * cos(q1) * cos(q4) * cos(q5) + 12 * cos(q2) * cos(q3) * sin(q1) - cos(q2) * sin(q1) * sin(q3) - cos(q3) * sin(q1) * sin(q2) + 9 * cos(q2) * cos(q3) * sin(q1) * sin(q5) - 9 * sin(q1) * sin(q2) * sin(q3) * sin(q5) - 9 * cos(q2) * cos(q5) * sin(q1) * sin(q3) * sin(q4) - 9 * cos(q3) * cos(q5) * sin(q1) * sin(q2) * sin(q4)$
- $J12 = cos(q1) * (cos(q2) * cos(q3) - 15 * sin(q2) + 12 * cos(q2) * sin(q3) + 12 * cos(q3) * sin(q2) - sin(q2) * sin(q3) + 9 * cos(q2) * sin(q3) * sin(q5) + 9 * cos(q3) * sin(q2) * sin(q5) + 9 * cos(q2) * cos(q3) * cos(q5) * sin(q4) - 9 * cos(q5) * sin(q2) * sin(q3) * sin(q4))$
- $J13 = cos(q1) * (cos(q2) * cos(q3) + 12 * cos(q2) * sin(q3) + 12 * cos(q3) * sin(q2) - sin(q2) * sin(q3) + 9 * cos(q2) * sin(q3) * sin(q5) + 9 * cos(q3) * sin(q2) * sin(q5) + 9 * cos(q2) * cos(q3) * cos(q5) * sin(q4) - 9 * cos(q5) * sin(q2) * sin(q3) * sin(q4))$
- $J14 = 9 * cos(q5) * (sin(q1) * sin(q4) + cos(q1) * cos(q2) * cos(q4) * sin(q3) + cos(q1) * cos(q3) * cos(q4) * sin(q2))$
- $J15 = 9 * cos(q4) * sin(q1) * sin(q5) - 9 * cos(q1) * cos(q2) * cos(q3) * cos(q5) + 9 * cos(q1) * cos(q5) * sin(q2) * sin(q3) - 9 * cos(q1) * cos(q2) * sin(q3) * sin(q4) * sin(q5) - 9 * cos(q1) * cos(q3) * sin(q2) * sin(q4) * sin(q5)$
- $J21 = 15 * cos(q1) * cos(q2) - 3 * cos(q1) - 12 * cos(q1) * cos(q2) * cos(q3) + cos(q1) * cos(q2) * sin(q3) + cos(q1) * cos(q3) * sin(q2) - 9 * cos(q4) * cos(q5) * sin(q1) + 12 * cos(q1) * sin(q2) * sin(q3) - 9 * cos(q1) * cos(q2) * cos(q3) * sin(q5) + 9 * cos(q1) * sin(q2) * sin(q3) * sin(q5) + 9 * cos(q1) * cos(q2) * cos(q5) * sin(q3) * sin(q4) + 9 * cos(q1) * cos(q3) * cos(q5) * sin(q2) * sin(q4)$
- $J22 = sin(q1) * (cos(q2) * cos(q3) - 15 * sin(q2) + 12 * cos(q2) * sin(q3) + 12 * cos(q3) * sin(q2) - sin(q2) * sin(q3) + 9 * cos(q2) * sin(q3) * sin(q5) + 9 * cos(q3) * sin(q2) * sin(q5) + 9 * cos(q2) * cos(q3) * cos(q5) * sin(q4) - 9 * cos(q5) * sin(q2) * sin(q3) * sin(q4))$
- $J23 = sin(q1) * (cos(q2) * cos(q3) + 12 * cos(q2) * sin(q3) + 12 * cos(q3) * sin(q2) - sin(q2) * sin(q3) + 9 * cos(q2) * sin(q3) * sin(q5) + 9 * cos(q3) * sin(q2) * sin(q5) + 9 * cos(q2) * cos(q3) * cos(q5) * sin(q4) - 9 * cos(q5) * sin(q2) * sin(q3) * sin(q4))$
- $J24 = 9 * cos(q5) * (cos(q2) * cos(q4) * sin(q1) * sin(q3) - cos(q1) * sin(q4) + cos(q3) * cos(q4) * sin(q1) * sin(q2))$
- $J25 = 9 * cos(q5) * sin(q1) * sin(q2) * sin(q3) - 9 * cos(q2) * cos(q3) * cos(q5) * sin(q1) - 9 * cos(q1) * cos(q4) * sin(q5) - 9 * cos(q2) * sin(q1) * sin(q3) * sin(q4) * sin(q5) - 9 * cos(q3) * sin(q1) * sin(q2) * sin(q4) * sin(q5)$
- $J31 = 0$
- $J32 = 15 * cos(q2) - 12 * cos(q2) * cos(q3) + cos(q2) * sin(q3) + cos(q3) * sin(q2) + 12 * sin(q2) * sin(q3) + 9 * sin(q2) * sin(q3) * sin(q5) - 9 * cos(q2) * cos(q3) * sin(q5) + 9 * cos(q2) * cos(q5) * sin(q3) * sin(q4) + 9 * cos(q3) * cos(q5) * sin(q2) * sin(q4)$
- $J32 = 15 * cos(q2) - 12 * cos(q2) * cos(q3) + cos(q2) * sin(q3) + cos(q3) * sin(q2) + 12 * sin(q2) * sin(q3) + 9 * sin(q2) * sin(q3) * sin(q5) - 9 * cos(q2) * cos(q3) * sin(q5) + 9 * cos(q2) * cos(q5) * sin(q3) * sin(q4) + 9 * cos(q3) * cos(q5) * sin(q2) * sin(q4)$
- $J33 = cos(q2) * sin(q3) - 12 * cos(q2) * cos(q3) + cos(q3) * sin(q2) + 12 * sin(q2) * sin(q3) + 9 * sin(q2) * sin(q3) * sin(q5) - 9 * cos(q2) * cos(q3) * sin(q5) + 9 * cos(q2) * cos(q5) * sin(q3) * sin(q4) + 9 * cos(q3) * cos(q5) * sin(q2) * sin(q4)$
- $J34 = -9 * cos(q2 + q3) * cos(q4) * cos(q5)$
- $J35 = 9 * cos(q2) * cos(q3) * sin(q4) * sin(q5) - 9 * cos(q3) * cos(q5) * sin(q2) - 9 * cos(q2) * cos(q5) * sin(q3) - 9 * sin(q2) * sin(q3) * sin(q4) * sin(q5)$

VI. Electronics design

Once the mechanical design and the servomotors necessary for the correct operation of the robot had been defined, the necessary components for the wireless of the robot were dimensioned. It is necessary at this point to define the main functionalities of the robot to establish which electronic

components, modules and sensors are going to be necessary for the construction of the robot.

1. The robotic manipulator has 5 degrees of freedom and a grip movement, so in this case as we've defined in the mechanical design, we are going to need 4 high torque servomotors and 2 mini servomotors.
2. It is necessary to use protections for the robot control circuit.
3. The control of the robot will be through a cell phone application, so in this case the Bluetooth connection is the best option for wireless control, considering that the HC-05 will connect the microcontroller with the cell phone application.

After the requirements of the robot were defined, an electrical diagram was made to better visualize how the operation will be in the electrical part, connections and communication between the different components as can be seen in Figure [14].

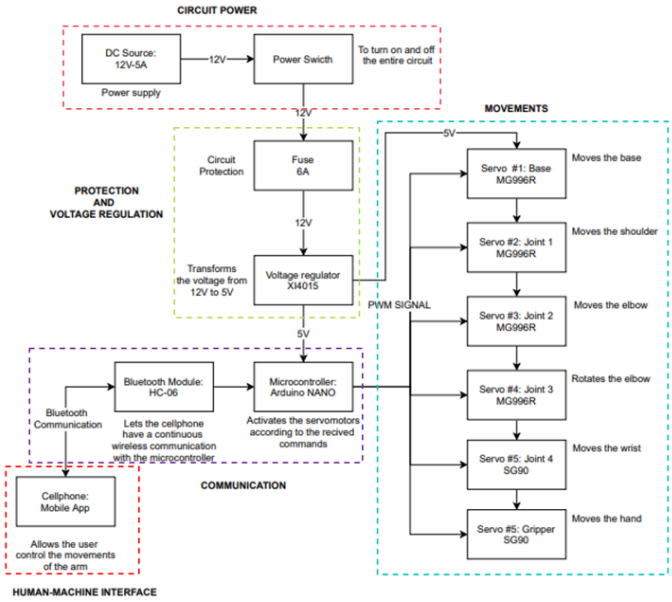

Fig. 14. Electronic scheme.

Each of the modules and electronic components need to transmit the information to the microcontroller, therefore they need communication pins with it, a connection analysis is performed for the microcontroller in Table [II].

Considering the required digital pins, the Arduino nano satisfies the requirements of the robot connections. The total current consumption of the robot was calculated, taking into account the individual consumptions obtained from the data sheets of each component as shown in Table [III].

So, considering these current consumptions of the electronics components the robot will need a 12v DC - 5A source to power the whole circuit, also a 6A protection fuse is added considering any unexpected power rise.

TABLA II
CONNECTION ANALYSIS FOR MICROCONTROLLER.

Connection	Pins	Consideration
High torque servomotors	[4]	The servomotors will need a pwm pins to be controlled.
Mini servomotors	[2]	The servomotors will need a pwm pins to be controlled.
Bluetooth HC-05	[2]	This sensor needs the transmit and receive pins of the microcontroller.
Total	$[Digitals : 8]$	-

TABLA III
POWER SUPPLY SIZING.

Component	Quantity	Consumption (mA)
Servomotor MG996R	[3]	1400 mA.
Servomotor SG90	[2]	250mA.
Arduino Nano	[1]	19mA.
Bluetooth module	[1]	8.5mA
Voltage Regulator XI4015	[1]	50mA
Total	[−]	4.77 A

VII. ELEMENT'S ACQUISITION

The main electronics elements for the construction of the designed prototype were acquired, as the Figure [15] shows.

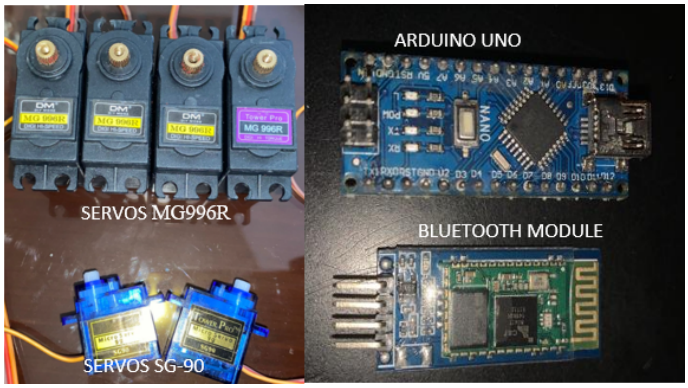

Fig. 15. Electronics elements acquired for the robotic manipulator.

One of the most important components that we are going to use are the bluetooth module, because this will be the one that allows the robot to work remotely, together with a cell phone application.

VIII. STRUCTURE IMPLEMENTATION:

With the model already designed and simulated and the electronics components acquired, the prototype can be assembled, for which the following elements must be acquire:

- PLA (for 3D printing).
- Multimeter (to check electronics connections).

- Bults and nuts (for construction).

The main advance in these 2 weeks of the robotic arm was the design of the model, the correction of tolerances errors of the prototype and the 3D printing of all the previously designed and improved parts.
The pieces were printed with 4 different printers that were available in the group which are:

- Ender v3
- Ender v3 pro
- Prusa MK2
- Tronxy XY2

To obtain a similar result between the pieces, taking into account that they were not all going to be printed on the same printer, the same software for slicing the .STL of the design was used, exactly as the main printing parameters were configured as Figure 16 shows, without considering the parameters that are unique in each printer.

Printing parameters used to print the robotic manipulator:

- Layer height: 0.2
- Infill: 20 %
- Printing speed: 60m/s
- Wall line count: 4

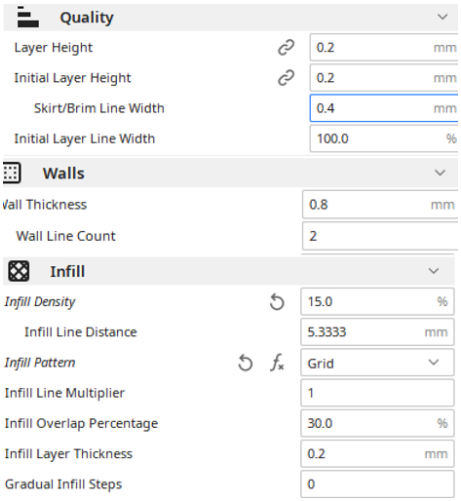

Fig. 16. Printing parameters.

The printing parameters used were selected based on previous experiences to optimize material consumption and in the same way that the pieces have good resistance.
The parts necessary for the assembly of the robot are:

- Base
- Base rotational
- 2 Arm lateral support
- 2 Arm
- 2 arm supports
- 2 arm joints
- 4 bearings
- 1 lever
- 1 forearm motor support
- 2 forearm joints
- 1 forearm motor joint
- 2 arm wrists
- 1 wrist servo joint
- 1 base gripper
- 2 grippers
- 1 large gripper link
- 2 small gripper links
- 4 servomotors MG996R
- 4 servos joints for MG996R
- 2 servomotors SG90
- 2 servos joints for SG90

In order to save printing time, the aforementioned available 3D printers were used, dividing the printing of the parts into the gripper as shown in Figure 19, the arm as shown in Figure [17], and finally the base.

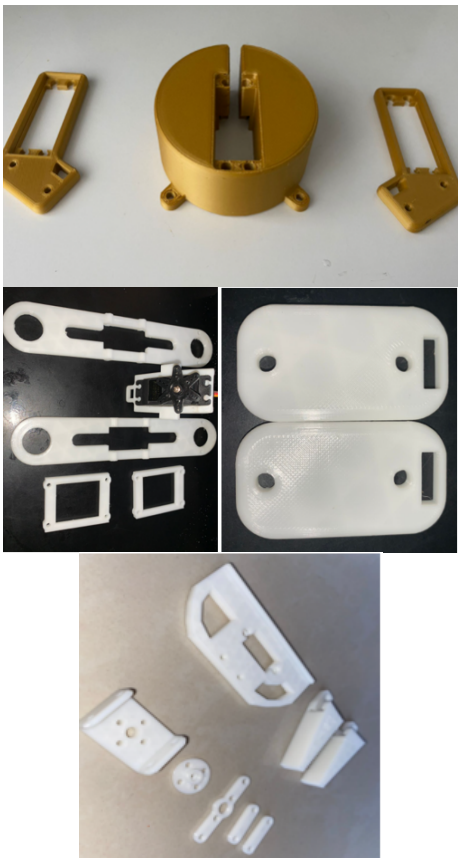

Fig. 17. 3D printed base pieces, 3D printed arm pieces, 3D printed gripper pieces

The total printing time of all the pieces took around 5 days, considering that the printers were not kept printing 24 hours, in total printing time there was around 50h.

IX. Assembly

Once having the parts of the structure and the electronic components available, we proceeded to identify all the parts considering that there are parts that only work on the left or right side of the prototype and are similar, this so as not to

have confusion in the assembly and that the robot works in the most optimal way.
As the mechanical design explanation, it is important to know that the assembly of the prototype is divided into 3 parts:

- Base
- Arm
- Gripper

The assembly of the robot was divided in this way to save us time individually and in the end to join the 3 main assembled parts.

X. Gripper phase

For the first phase of the assembling a problem was found with respect to the bolts and nuts in the functionality of the gripper mechanism, since the first design of the gripper as shown in Figure 20 collided and did not allow the gripper to move freely, so a new design of the mechanism was carried out as shown in Figure [18].

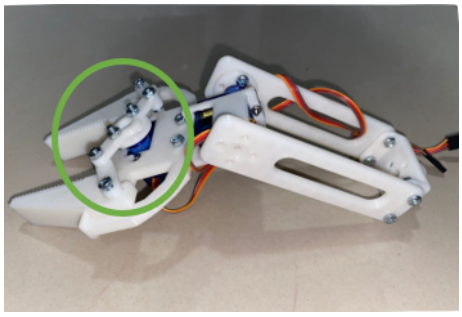

Fig. 18. First gripper mechanism model.

Fig. 19. Final gripper mechanism model.

XI. Arm phase

This phase of the robot was carried out without problems, once tolerances errors of the prototype had been solved, it was easier and faster to assemble the robot, also with the servomotors acquired, the final prototype was assembled with the bearings and motors as Figure [20] shows, for later proceeding with functional tests, explained in the test section.

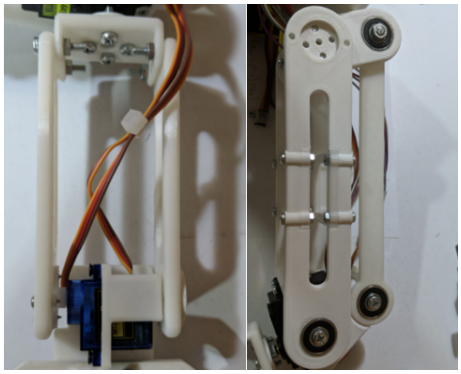

Fig. 20. Assembled Arm.

XII. Base phase

For this last phase of the assembly of the robot, the height of the base had to be enlarged by 5cm since the first design didn't have enough space for the electronic components, so we proceeded to expand it and avoid problems in the future with the electrical connections, in this way, the final model of the base was obtained, as can be seen in Figure [21].

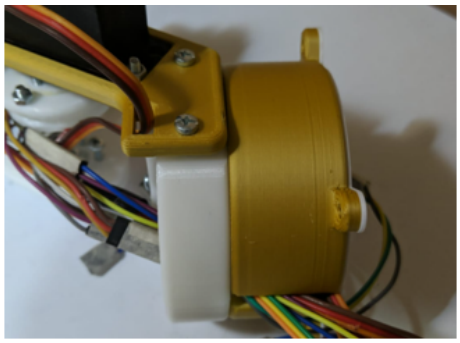

Fig. 21. Assembled Base.

Once the 3 phases of the assembly of the prototype were finished, we proceeded to join them to have the final prototype assembled as Figure [22] shows and able to start with the functional tests of the different movement mechanisms that the robot has.

XIII. Problems and solutions

- Unconsidered tolerances: One of the main errors that occurred when assembling the robot was not considering the tolerances at the time of design, since as soon as the first parts of the robot were printed, the parts needed an additional finish, to link with other parts of the prototype. Once the error was recognized, slight changes were made to increase the tolerance of the pieces so that they can be linked to each other without any inconvenience and in this way this error could be solved.

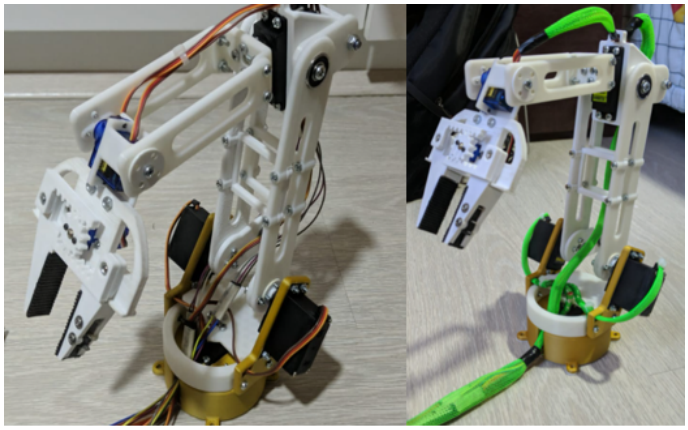

Fig. 22. Final prototype assembled.

- Bults and nuts unconsidered: Since the gripper was the first to be built, it was possible to detect the main errors in the design, the error already mentioned in the tolerances and in the first design of the gripper[18] the bolts and nuts were not taken into account, so they collided with the mechanism, in order to solve this problem it was designed and printer the gripper parts again as Figure [19] shows.
- Bearing's size: Another aspect without considering were the commercial sizes of the bearings, although this was a small problem, this type of considerations must be considered when designing the prototype, since to fix this problem, a small 3D printed coupling as shown in Figure 25 of 0.5mm thickness.

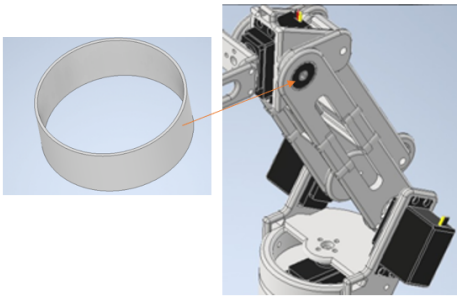

Fig. 23. Coupling for bearing

XIV. Tests of the operation of the mechanism

For the tests of the assembled prototype, Arduino was used to make a basic code and, in this way, test 2 different poses of the robot to make sure that the movements are carried out correctly and whole prototype mechanism works in a functional way and doesn't have any crashes in any movement.

A. *1st Pose tested*

For the first pose tested in the prototype, the displacement of the forearm, the gripper and the wrist was tested as can be seen in Figure [24].

Fig. 24. First pose tested on the final prototype.

B. *2nd pose tested*

For the second tested pose, we test the elbow rotation, the base and the shoulder movements as Figure [25] shows.

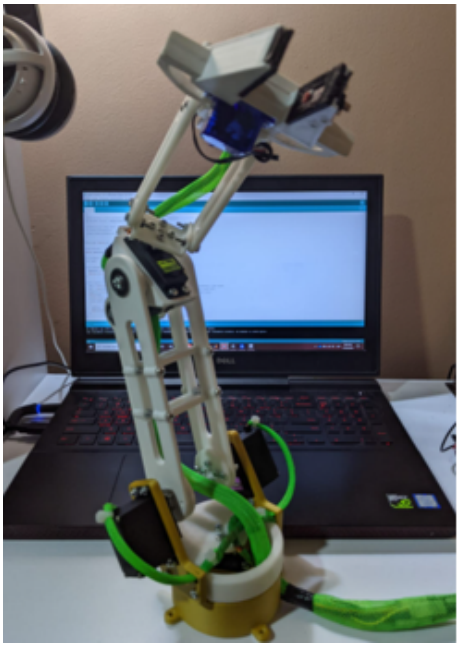

Fig. 25. Second pose tested on the final prototype.

These basic and specific poses were performed in order to test each movement of the robot and see if it works correctly. The previously mentioned tests can be better viewed in the videos attached to the repository.

XV. Results of tests of the operation of the mechanism

Once the mechanisms of the final prototype built had been tested as we could see in the last section and having taken different poses to detect any crashes at any movement of the robot, there were no problems or crashes with the final prototype mechanisms, which is why it is determined that

every movement is functional to proceed to the final part of the project that is the programming and communication with the mobile application.

XVI. Control circuit design

As the last stage of the construction but not less important for the operation of the robot, it is the control circuit design to communicate and supplies energy to each component necessary for the operation of the robot, for which, considering the elements of Table 3, all the connections was made first as Figure [26] shows, then the circuit paths were drawn for the PCB [27], subsequently the PBC of the control circuit was simulated as Figure [28] shows to check the dimensions and if the space that it will occupy is enough.

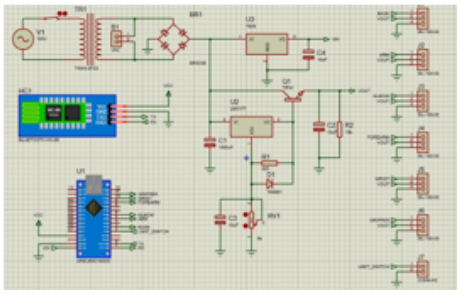

Fig. 26. Connections of control circuit.

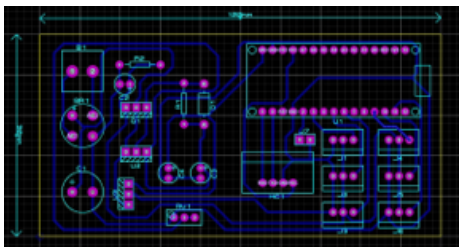

Fig. 27. Designed PCB control circuit.

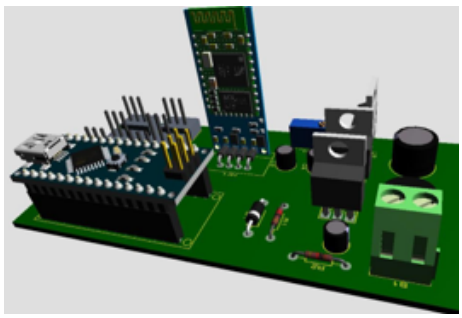

Fig. 28. Control circuit design simulation.

Once the circuit board that will control the robot has been designed, the final dimensions are: 100x50x50 [mm]
So, the size is ideal for the space allocated in the base for the control circuit.

XVII. Development of the phone app

The mobile application will be used to control the robot, as mentioned above, for the development of the application, the MIT App inventor application was used to create the interface and send and receive data with the robot.
The control through the app, works in 3 main parts as shows Figure [29]:

- Gripper control, speed configuration and mode select.
- Inverse kinematics
- Forward kinematics

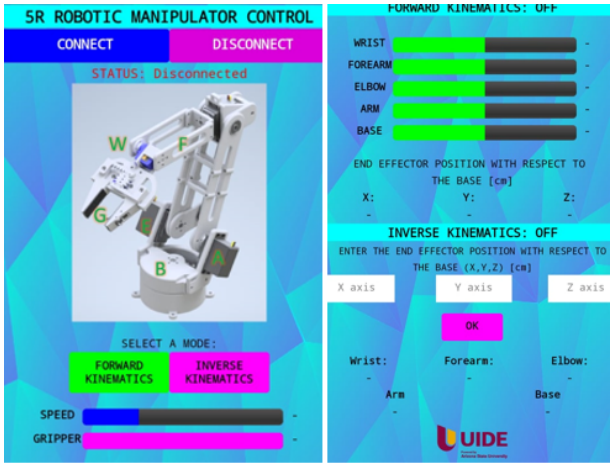

Fig. 29. Phone app to interface.

In the first part, buttons connect and disconnect are showed, with this you are able to see the available devices that are nearby the robot, and displays a message on the screen which indicates the status of the connection, this connections as before mentioned is made through the bluetooth module H0-06, in addition you can view a picture of the robot to distinguish the different movements that the robot can perform.
In the second part you can see the interface of forward kinematics, by means of which each movement of the robot is controlled, by means of sliders, and as these change the angle of the robot changes at the same time, also the position of the end effector with respect to the base in x, y and z coordinates.
Last but not least, the inverse kinematics mode of operation works by entering the coordinates of the end effector, so that the robot calculates the different possible solutions and reaches the requested position.

XVIII. Programming the whole code

For the next project, in order to distinguish and reduce the processing time of each command, it will be divided into the following parts:

A. *Main control program: Robot Arm*

This program captures and processes the application control strings, its main variables contain the positions of the grip point in x, y, z and the angles of each degree of freedom; q0, q1, q2, q3 and q4, as can be seen on the figure [30].

```
void setup() {
  Serial.begin(9600);
  setup_servos(q(0),q(1),q(2),q(3),q(4),180);
  T=fkine(q);
  x=T(0,3);
  y=T(1,3);
  z=T(2,3);
  delay(1000);
}

void loop() {
  t1=millis();
  if(t1-t2>=1000)
  {
    q=get_angles();
    T=fkine(q);
    x=T(0,3);
    y=T(1,3);
    z=T(2,3);
    Serial<<x<<"I"<<y<<"I"<<z<<"I"<<q(4)<<"I"<<q(3)<<"I"<<q(2)<<"I"<<q(1)<<"I"<<q(0)<<'\n';

    t2=t1;
  }
}
```

Fig. 30. Setup and loop.

Its first function "fkine mode", named like this because it is the function of the kinematic movement that extracts the value of the string in the reading vector to send them to the position related subprograms such as "motors servos.cpp.ªnd that of "fkine.cpp.ºn figure [31], which will be explained later.

```
void fkine_mode()
{
  String pos;
  for(int i=s.length();i>=0;i--)
  {
    if(s[i]=='B')
    {
      pos=s.substring(i+1,s.length());
      base(pos.toInt());
      move_servos(vel);
      break;
    }
    else if(s[i]=='A')
    {
      pos=s.substring(i+1,s.length());
      arm(pos.toInt());
      move_servos(vel);
      break;
    }
    else if(s[i]=='E')
    {
      pos=s.substring(i+1,s.lengt
      elbow(pos.toInt());
      move_servos(vel);
      break;
    }
    else if(s[i]=='F')
    {
      pos=s.substring(i+1,s.lengt
      forearm(pos.toInt());
      move_servos(vel);
      break;
    }
    else if(s[i]=='W')
    {
      pos=s.substring(i+1,s.lengt
      wrist(pos.toInt());
      move_servos(vel);
      break;
    }
```

Fig. 31. fkine mode.

In this way, the position to which the entire arm will move will be determined.
The second ïkine mode"function, named after the inverse kinematic motion function, also extracts the string values and sends them to the servo and ïkine.cppçodes to determine their motion components.

The main difference of this program is that it extracts the entered coordinate values to later process them with a reverse movement and devour the angles that it has to perform to move the arm to the desired position, seen on figure [32].

The last main function "SerialEventïs in charge of recognizing the interruptions of the Arduino (figure [33]) for

```
void ikine_mode()
{
  String pos;
  float x,y,z;
  bool enable=false;
  for(int i=0;i<s.length();i++)
  {
    if(s[i]=='X')
    {
      enable=true;
      pos=s.substring(i+1,s.length());
      x=pos.toFloat();
    }
    if(s[i]=='Y')
    {
      enable=true;
      pos=s.substring(i+1,s.length());
      y=pos.toFloat();
    }
    if(s[i]=='Z')
    {
      enable=true;
      pos=s.substring(i+1,s.length());
      z=pos.toFloat();
    }
  }
  if(enable)
  {
    //Serial<<x<<" "<<y<<" "<<z<<" "<<'\n';
    set_angles(ikine(x,y,z));
    //Serial<<get_angles()<<'\n';
    move_servos(vel);
    enable=false;
  }
```

Fig. 32. “ikine mode”

when it detects a signal of sending or receiving data with the application connected by bluetooth to the arduino. And in this way choose the mode to be used according to the previously described functions.

```
void serialEvent()
{
  String sp,pos;
  if(Serial.available() > 0)
  {
    s=Serial.readString();
    for(int i=0;i<s.length()-1;i++)
    {
      c[i+2]=' ';
      c[i+1]=' ';
      c[i]=s[i];
    }
    if(charIsEqual(c,"fkine"))
    {
      //Serial.println("Fkine mode");
      mode=1;
    }
    else if(charIsEqual(c,"ikine"))
    {
      //Serial.println("Ikine mode");
      mode=2;
    }
    for(int i=s.length();i>=0;i--)
    {
      if(s[i]=='S')
      {
        sp=s.substring(i+1,s.length());
        vel=sp.toInt();
        break;
      }
      if(s[i]=='G')
      {
        pos=s.substring(i+1,s.length());
        gripper(pos.toInt());
        move_servos(vel);
        break;
      }
    }

    switch(mode)
    {
      case 1:
      fkine_mode();
      break;
      case 2:
      ikine_mode();
      break;
      default:
      break;
    }
```

Fig. 33. “serialEvent”

B. *Subprogram: fkine*

This applet emphasizes the use of libraries such as "BasicLinearAlgebra.h.ªnd .ElementStorage.h", which help complex array operations.

With this defined, you can create matrices such as 4x4 for each sector, of which the main one is the last "GrelB"viewed on figure [34], which is the matrix of the gripper’s position with respect to the base. These will contain the complete cinematics.

In this section the four kinematics with respect to the previous one are contemplated: B, A, E, F, W, G, and the

```
#include <BasicLinearAlgebra.h>
#include <ElementStorage.h>
#include <Arduino.h>4


using namespace BLA;

float L0=9.5;
float L1=3;
float L2=15;
float L3=1;
float L4=12;
float L5=9;


BLA::Matrix<4,4> B;
BLA::Matrix<4,4> A;
BLA::Matrix<4,4> E;
BLA::Matrix<4,4> F;
BLA::Matrix<4,4> W;
BLA::Matrix<4,4> G;
BLA::Matrix<4,4> GrelB;
```

Fig. 34. “fkine”

```
BLA::Matrix<6,1> get_Links()
{
  BLA::Matrix<6,1> Links={L0,L1,L2,L3,L4,L5};
  return Links;
}
float deg2rad(int deg)
{
  return deg*PI/180.0;
}
float rad2deg(int rad)
{
  return rad*180.0/PI;
}
BLA::Matrix<4,4> fkine(BLA::Matrix<5,1>Q)//(int Q1,int Q2, int Q3, int Q4, int Q5)
{

  float q1=deg2rad(Q(0));
  float q2=deg2rad(Q(1));
  float q3=deg2rad(Q(2));
  float q4=deg2rad(Q(3));
  float q5=deg2rad(Q(4));


  B={cos(q1),-sin(q1),0,0,sin(q1),cos(q1),0,0,0,0,1,L0,0,0,0,1};
  A={cos(q2),-sin(q2),0,-L1,0,0,-1,0,sin(q2),cos(q2),0,0,0,0,0,1};
  E={cos(q3-PI/2),-sin(q3-PI/2),0,L2,sin(q3-PI/2),cos(q3-PI/2),0,0,0,0,1,0,0,0,0,1};
  F={cos(q4-PI/2),-sin(q4-PI/2),0,L3,0,0,-1,-L4,sin(q4-PI/2),cos(q4-PI/2),0,0,0,0,0,1};
  W={cos(q5),-sin(q5),0,0,0,0,-1,0,sin(q5),cos(q5),0,0,0,0,0,1};
  G={1,0,0,L5,0,1,0,0,0,0,1,0,0,0,0,1};
  GrelB=B*A*E*F*W*G;

  return GrelB;
}
```

Fig. 35. “Matrix calculation”

last one "GrelB"which will be the multiplication of all the previous ones, whose value will be returned to the main program Robot Arm ïn function"fkine mode "([35]).

C. *Subprogram: ikine*

This subprogram can be said to be the most complicated, because it has five degrees of freedom, and its processing is longer so it is more difficult to obtain the result. Which is followed by non-linear equations that can hardly be solved.

In order to solve this obstacle, a simple code of genetic algorithms was used that determines the 5 angles that each joint must move ([36]).
As a first step, the first generation of the algorithm starts with values of 0 in each possible matrix.
Subsequently, starting from a "while", random variables are created within range of motion / distance restrictions, to

```
#include <Arduino.h>
#include <BasicLinearAlgebra.h>
#include <ElementStorage.h>
#include "fkine.h"

using namespace BLA;

BLA::Matrix<5,1> ikine(float x, float y, float z)
{

  int d_forearm=90;
  int j;
  int cont=0;

  float er=1;
  float Min[4];
  float spx[4];
  float spy[4];
  float spz[4];

  BLA::Matrix<4,4> T;
  BLA::Matrix<5,1> q{0,0,0,d_forearm,0};
  BLA::Matrix<4,1> sQ1={0,0,0,0};
  BLA::Matrix<4,1> sQ2={0,0,0,0};
  BLA::Matrix<4,1> sQ3={0,0,0,0};
  BLA::Matrix<4,1> sQ5={0,0,0,0};
  BLA::Matrix<6,1> Links=get_Links();
```

Fig. 36. “ikine”

prevent the motor from hitting its physical limits. An example of this is in the upper body where the sum of the elbow and the arm cannot be more than 230 degrees, since the motor would break the part or the elbow or the arm.
In this way, we have an initial population with which we can work with the "fkine"subprogram and establish the corresponding angle movements.

With the returned values, the minimum error is calculated, which is stored the smaller the error, in this way iterations are created until the error is as small as possible or reaches 1000 iterations ([36]).
Consequently, it is verified if the value of the return angle is less than the accepted error, which must be less than 1 cm.

D. *Subprogram: Servo$_{m}otores$*

This is a basic and generic open software program that can be found online, which determines the signals that will be sent to each motor to determine a particular movement in degrees of the axis.

XIX. Functionality test and results

For the tests and results obtained in table [I] and [II], for the case of the forward kinematics mode, parameters were placed in the movements of the robot as shown in figure [38] and for the case of the inverse kinematics mode, a final position was placed at the which end effector has to arrive as shown in figure [39].

```
while(1)
{
  for(int i=1;i<4;i++)
  {
    sQ1(i)=random(0,181);
    sQ5(i)=random(0,181);
    while(1)
    {
      sQ2(i)=random(30,161);
      sQ3(i)=random(30,141);
      if(sQ2(i)+sQ3(i)<=230)
      {
        break;
      }
    }
  }
  for(int i=0;i<4;i++)
  {
    q(0)=sQ1(i);
    q(1)=sQ2(i);
    q(2)=sQ3(i);
    q(4)=sQ5(i);
    T=fkine(q);
    spx[i]=T(0,3);
    spy[i]=T(1,3);
    spz[i]=T(2,3);
  }
```

```
for(int i=0;i<4;i++)
{
  Min[i]=abs(x-spx[i])+abs(y-spy[i])+abs(z-spz[i]);
}
j=0;
for(int i=1;i<4;i++)
{
  if(Min[i]<Min[j])
  {
    j=i;
  }
}
sQ1(0)=sQ1(j);
sQ2(0)=sQ2(j);
sQ3(0)=sQ3(j);
sQ5(0)=sQ5(j);

if((abs(x-spx[0])<=er) && (abs(y-spy[0])<=er) && (abs(z-spz[0])<=er))
{
  break;
}
else if(cont>1000)
{
  break;
}
cont++;
}
q(0)=sQ1(0);
q(1)=sQ2(0);
q(2)=sQ3(0);
q(4)=sQ5(0);
return q;
}
```

Fig. 37. “Genetic algoritm”

For the real measurement of the position of the end effector with respect to the base, a ruler was used, and it was proceeded to measure manually as Figure [**??**] shows, to later compare the calculated values with those measured.

Fig. 38. “Parameters set for forward kinematics.”

TABLA IV
TEST 1 OF FORWARD KINEMATICS

Test 1	Calculated	Real	% Error
x	0	0	L0
y	18	18.3	1.63
z	25.5	25.2	1.19

TABLA V
TEST 2 OF FORWARD KINEMATICS

Test 2	Calculated	Real	% Error
x	11.84	13.7	6.27
y	11.16	12.2	8.52
z	6.24	6.5	4

TABLA VI
TEST 3 OF FORWARD KINEMATICS

Test 3	Calculated	Real	% Error
x	11.83	11.4	3.7
y	9.92	10.8	8.1
z	36.26	36.7	1.3

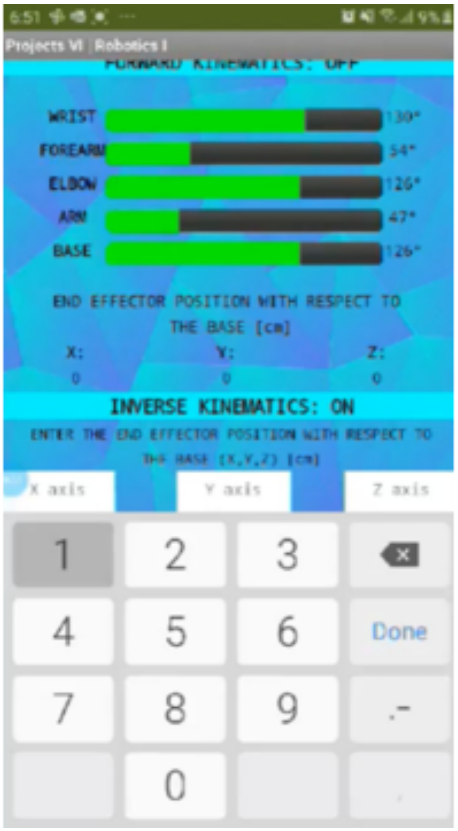

Fig. 39. “Input parameters for inverse kinematics.”

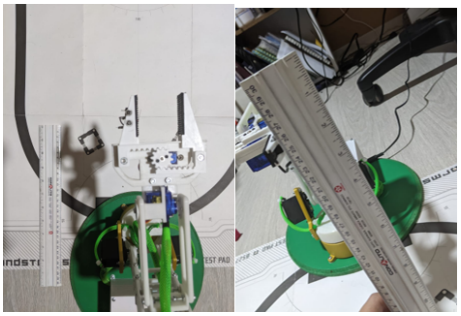

Fig. 40. “Position of the end effector with respect to the base measured.”

TABLA VII
TEST 1 OF INVERSE KINEMATICS

Test 1	Wanted	Calculated	Real	% Error
x	0	0.2	0.3	33.3
y	10	11.31	11.8	4.1
z	30	29.51	28.9	2.1

A. *Results*

As you can see on Table (IV-IX) the percentage of error between the real thing and the calculated ones, it does not exceed 8 % except for 1 test, so in this way it can be verified that the calculations are quite close to reality, and it can be concluded that the programming and the robot work satisfactorily.

TABLA VIII
TEST 2 OF INVERSE KINEMATICS

Test 2	Wanted	Calculated	Real	% Error
x	10	9.15	8.8	3.9
y	15	15.85	16	0.93
z	15	14.16	14.8	4.3

TABLA IX
TEST 3 OF INVERSE KINEMATICS

Test 3	Wanted	Calculated	Real	% Error
x	-20	-19.2	-19.5	1.5
y	20	20.39	19.8	2.9
z	5	7.58	7.2	5.2

XX. CONCLUSIÓN

The manipulative robots have been designed in order to eliminate the efforts caused by the lifting of the merchandise and are intended for various and diversified fields. They are distinguished by their simplicity of use and by their resistant and solid structure, characteristics that give these products a high level of reliability.

And it can also be concluded that the set of links and joints is called a kinematic chain. It is said that a kinematic chain is open if each link is connected by means of joints exclusively to the previous one and the next, except for the first, which is usually fixed to a support, and the last, whose end is free. A terminal element or final actuator can be connected to this: a special tool that allows the general-purpose robot to carry out a particular application, which must be specifically designed for said application, that is, a holding tool.

Their movement is called by degrees of freedom (g.d.l.) to each of the independent coordinates that are necessary to describe the state of the robot's mechanical system (position and orientation in space of its elements).

This is how after finishing the design and especially the manufacturing of our manipulator arm, and taking into account the entire process that was carried out, we can mention the following; With our design, you can carry out simple and concise explanations about servo motors, DC motor control, parts of an arm, as well as the definition of link, degrees of freedom. In addition to giving an idea of how to program a robot. On the other hand, due to its characteristics, its implementation is not only didactic, it can also be used in other areas, either production or monitoring. And finally, the cost of making our arm is much lower than many other commercials. In addition to having the advantage that parts that break or fail can be changed, since in the case of commercial robots, spare parts and repairs are relatively expensive.

REFERENCIAS

[1] "Robotic arm control traines." *For Industrial*, 2021.
[2] "More than just a parallel gripper: How does it works?" *Blog.Robotic*, 2014.
[3] "Parts of an arm," *Pinterest*, 2019.