Control of a 3DOF anthropomorphic robot

Dr. Alejandro González de Alba

22/11/2022

Industrial Robotics

Cesar Eduardo Monterrubio Morales A01067371,

Luis Mario Flores Campos – A01065662,

Fernando Mejía Laguna – A01246227

I. Abstract

This paper describes the analysis, design of the elements involved to control an anthropomorphic robot with 3 degrees of freedom. The first part of the project was the design process, where the MATLAB HMI and CAD were designed. For the HMI, the variables which the user is working with were identified for a later development on the app designer. For the CAD, the models of the used components were looked for and used to assemble the arm digitally. Afterwards, the box was drawn with a working area of 10 cm, including the measurements for the electronic components for a proper visualization of the arrangement. Then, the code was developed to calculate the end-effector position with 3 angles as inputs, and the interaction of the user with the HMI is defined by a knob that selects the working mode of the robot. After making the human-machine interface functional, the system related to Arduino to be able to send the values to the robot that will properly move according to the joint values.

II. Objectives

The main objective of this project is to build an anthropomorphic robot arm, which is capable of being manipulated through a Human-Machine-Interface (HMI) implemented on MATLAB.

III. Introduction

The following document explains how an anthropomorphic robot arm was built. Giving a brief, but concise description of the current designs that are available in the research and in the industry as can be seen on the state of start. This depicts a general idea of what are the available tools that can be used to implement this robot. And it helps to obtain a set of parameters, thus we can compare what the differences are between the one that we implemented in the course and the others found in the literature. Additionally, the report is intended to demystify both the process of creating it, the methods applied, and the material used as seen on the materials and methods section. Moreover, through experiments and tests realized along its process, we collected a considerable amount of data, so we could present it in a more intuitive way on the results section, to analyze it and find new insights for future additions or projects the conclusions section should be checked.

IV. State of art

Before going deeper about our robot, we want to emphasize the current developments that have been done in the industry and in research. According to the researcher Lelia [1], one of the major obstacles that have been faced around the industry is a new approach to designing a lightweight anthropomorphic arm for service applications without creating a “softer” one. In other words, the main problem with optimizing the weight is that as you decrease it is likely to obtain a system with undesirable vibrations. So, in this paper, they considered adding the structural dimensions and the drivetrains of a robotic arm as additional parameters for the optimization. Another trend that is increasing exponentially is the implementation of a control system with remote gesture tracking to the development of a valuable tool to interact with situations where high precision is required. In the paper written by Trapeznikov [2], they proposed developing a sensor that can track a device for remote control of an anthropomorphic manipulator. To achieve this, they use a system made of a photo resistive bending sensor for transmitting the movements to the slave mechanism. Finally, another evolving implementation that has been worked on is the possibility to drive it by artificial muscles using a variable structure control with the aim of getting a lighter and more flexible robot. As reported by M. Chamberlain [8], it wasn’t only possible to create it, but also it could create outstanding performance by comparing it with a PID control.

V. MATERIALS AND METHODS

A. *Materials*

Currently, the robot is composed of the following materials:

Table 1: Materials

Name	Quantity
Arduino MEGA with cable	1
SN74LS241N	1
Servo Robotis Dynamixel AX-12A	3
Dynamixel Cable	3
FP04-F2	1
FP04-F3	2
FP04-F4	1
FP04-F9	1
FP04-F11	1

B. *Direct Geometric Model*

To obtain the direct geometric model of the robot, the DH parameters should be obtained. For this a kinematic diagram was drawn to define the frames of each joint and the direction of each axis from its respective frame.

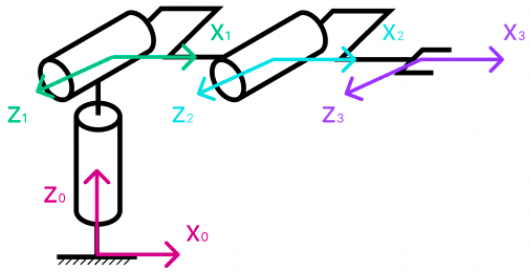


Figure 1: Kinematic diagram

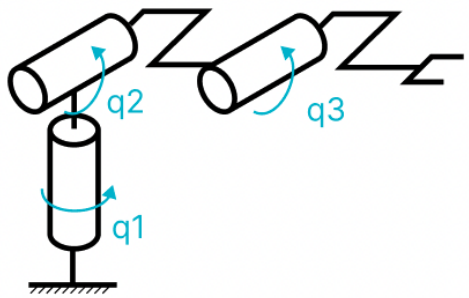


Figure 2: Identification of positive joint direction

With this information the DH parameters can be obtained, so that values are placed in a table for easy understanding.

Table 2: DH Parameters

i	a	α	d	θ
1	0	$\pi/2$	L_1	q_1
2	L_2	0	0	q_2
3	L_3	0	0	q_3

The DH parameters on this table are general for any 3R anthropomorphic robot with the initial configuration shown on the image. L1 is the distance between reference frame 0 and reference frame 1, L2 is the distance between reference frame 1 and reference frame 2 and L3 is the distance between reference frame 2 and reference frame 3.

For our specific robot the CAD models of the pieces were used to compute those distances for a higher precision, so the distances are L1= 89 mm, L2=67.5mm, L3=98.43 mm.

With all this values the homogeneous transformation matrix of all the frames can be computed, then should be multiplied in the correct order, giving as a result the direct geometric model of the robot.

For each homogeneous transformation matrix, the general formula is the next one:

$$^{i}T_{i+1} = rotZ(\theta_i)transZ(d_i)rotX(\alpha_i)transX(a_i)$$

For the direct geometric model all the transformation matrices need to be multiplied in the next order:

$$^{0}T_3 = {}^{0}T_1\ {}^{1}T_2\ {}^{2}T_3$$

As the objective is to control the position of the robot the important part of that matrix is the fourth column. From that column the position equations for X, Y and Z can be obtained, giving the next formulas:

$$X = \cos(q_1)\,(L_2\cos(q_2) + L_3\cos\,(q_2 + q_3))$$

$$Y = \sin(q_1)\,(L_2\cos(q_2) + L_3\cos\,(q_2 + q_3))$$

$$Z = L_1 + L_2\sin(q_2) + L_3\sin\,(q_2 + q_3)$$

For test purposes, and additional transformation matrix is used to change the world reference frame, this reference frame has the same orientation as the reference frame 0, so the only operations needed are some translations, thus the transformation matrix between this new reference frame and reference frame 0 is:

$$^{-1}T_0 = \begin{bmatrix} 1 & 0 & 0 & 43.68 \\ 0 & 1 & 0 & 97 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

By multiplying this matrix to the left of the complete transformation will give us the new position equations that just have a constant added to the original equations.

$$X = \cos(q_1)\,(L_2\cos(q_2) + L_3\cos\,(q_2 + q_3)) + 43.68$$

$$Y = \sin(q_1)\,(L_2\cos(q_2) + L_3\cos\,(q_2 + q_3)) + 97$$

$$Z = L_1 + L_2\sin(q_2) + L_3\sin\,(q_2 + q_3)$$

With this the whole implementation of the DGM is completed.

C. *Inverse Geometric Model*

To compute the inverse geometric model, two methods can be used, the numerical one or the analytical one. It does not matter which method will be used, both need the DGM to be computed, the numerical method needs the Jacobian that needs the transformation matrices to be computed, and the analytical one needs the position equations. Both methods will be used so the first thing to be computed will be the Jacobian.

The formula for the Jacobian of any robot which transformation matrices are known is the next one:

$$J_V = \sigma\, a_{i-1}^0 + not(\sigma)a_{i-1}^0 X\left(P_n - P_{i-1}^0\right)$$
$$J_\omega = not(\sigma)a_{i-1}^0$$

Both formulas are used to compute the complete Jacobian matrix and each one generates in this case a 3x3 matrix, we only care about position, so only the first one will be used:

$$J = \begin{bmatrix} -s1(L_3c23 + L_2c2) & -c1(L_3s23 + L_2s2) & -L_3s23c1 \\ c1(L_3c23 + L_2c2) & -s1(L_3s23 + L_2s2) & -L_3s23s1 \\ 0 & (L_3c23 + L_2c2) & L_3c23 \end{bmatrix}$$

Now the singular configurations are to be determined using the determinant of the Jacobian matrix:

$$|J| = \frac{L_3}{2}(-\sin(q2) + \sin(q_2 + 2q_3) + L_2\cos(q_2)\sin\,(q_3)$$

The determinant of the Jacobian only has q2 and q3 so only these 2 joint variables can reach singular configurations independently of what value q1 has. If L2 and L3 are substituted, and the determinant is equalized to cero we get the next graph of singular configurations on the joint plane formed by q2 and q3:

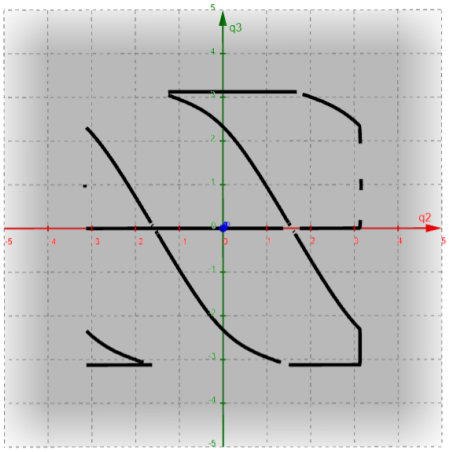


Figure 2: Singular configurations of the robot.

This is important because in all the positions of the task space corresponding to these values of q2 and q3, there is no solution for the IGM so the analytical method will fail, and the numerical will continue to do iterations until another condition stops the computation. With all this information both methods will be implemented.

The first method implemented was the **analytical method**, using Paul's method to obtain equations that compute the angles, so first the additional translation must be subtracted, then the desired position must be multiplied by the left with the inverse transformation matrix from 0 to 1:

$$^0T_1^{-1}U = {}^1T_3$$

Once this is done, we compare each row of the fourth column of both sides to solve for the joint variables. After many operations, the next equations were obtained:

$$q_1 = atan2(Y, X)$$

$$A = Xcos(q_1) + Ysin(q_1)$$

$$B = Z - L1$$

$$\cos\,(q_3) = \frac{A^2 + B^2 - L2^2 - L3^2}{2L2L3}$$

$$q_3 = atan2(\pm(1 - \cos^2(q_3))^{0.5}, cq3a)$$

$$C = L2 + L3cos(q_3)$$

$$D = L3sin(q_3)$$

$$\sin\,(q_2) = \frac{CB - DA}{C^2 + D^2}$$

$$\cos(q_2) = \frac{CA + DB}{C^2 + D^2}$$

$$q_2 = atan2(\sin(q_2), \cos(q_2))$$

It is important to note that there are four different solutions for this equations, two different q3 can be obtained changing the sign of the square root, thus also giving two solutions for q2. Also, two values for q1 can be obtained getting 2 different values for A, that's how four q3 are obtained.

As four solutions are computed, the difference between the previous angle vector and each solution is computed, the solution with the smaller differences is the one selected to be send to the Arduino.

For the **numeric solution** of the IGM the next formula is needed:

$$q_{k+1} = J^{-1}(q_k)(X_{k+1} - X_k) + q_k$$

On the implementation, a inicial joint vector and position vector is needed. For this, the initial joint vector is build by the angles already written on the HMI. To guarantee that the position vector is correct, the DGM is also applied on this method, not only to compute the Jacobian, but to compute the corresponding coordinates of the joint vector. Using the same initial joint vector the inverse Jacobian is computed to aproximate the solution of the desired position. Here is were the singular configurations shown on the joint plane have a lot of importance, this is because there are two problems that could appear because of this configurations.

The numerical method need the initial condition, so when the initial condition is already al singular configuration, the determinant of th Jacobian is equal to cero thus there is no inverse Jacobian matrix for this joint vector making imposible to compute the solution this way. The second problem is when the desired position is a singular condiguration, maybe there is no problem when the computation starts, but as the iterations aproach the solution two things could happen, the program stucks on a infinite loop because of the values of the determinant or you get an error. This last problem also appears when a point outside the workspace is selected.

To solve the first problem, the determinant is computed before the inverse Jacobian, if the result is near 0, the initial conditions are changed. This happens until a determinant different form cero is found.

For the second problem, some cases can be solved, when a desired position is near a singular configuration, the values calculated for the joint vector tend to be pretty big, and our motors work between 0° and 300°. In this case, the robot has only rotational joints, so it does not matter the size of the value, because it is a multiple of a value between the capabilities of our robot, so to solve this cases where the joint values are big, the trigonometric functions are applied and then the inverse trigonometric functions are aplied to obtain the corresponding angle in the interval form minus pi to pi. However not all the singular configurations can be solved this way and obviously positions outside the workspace cant be computed.

D. Trajectories

For the last phase of this project, the capacity to follow trajectories was asked for the robot. At least straight lines and circles were required to complete this phase. Nevertheless, this mode of the robot can be implemented to follow any trajectory that the user can imagine if it can be written in form of parametric equations.

The method used for the route tracing is a numerical approximation, so the Jacobian matrix is used again with the same formula that the numeric solution of the IGM, so the formula is the next one:

$$q_{k+1} = J^{-1}(q_k)(X_{k+1} - X_k) + q_k$$

To use this formula for route tracing, there are three mayor differences to the IGM approach, the first one would be that the goal position changes between each iteration. In the IGM lots of iterations would be done until the error between the desired position and the calculated one were less than 0.1 mm in theory, on the trajectories mode the desired position changes between each iteration and only one joint vector is computed for each desired position.

The second difference is that the error depends on the quantity of intermediate points you desire to divide your trajectory, because of the first difference, just one iteration is done per desired position, so if the only point computed is the final position of the route, lot of error will appear. This can be solved dividing the route on many points between the route, for straight lines this is easy but on more complex routes, you require the parametric equations to compute many points inside the route.

The last difference is that on the IGM computation, many joint vectors were calculated, but only the last one was saved and used, on trajectories mode each joint vector must be saved to send it to the robot so it can follow the trajectory, this is pretty easy to solve defining a matrix that on each iteration concatenates the new joint vector so when the trajectory computation is finished, the result is a matrix with 3 rows and as many columns as points you decided to compute, after this another loop sends each joint vector one per one until all the values have been sent.

To define the specific route, UI needs six inputs, the first 3 numerical inputs and the last 3 string inputs. The first 3 are the initial value of the parameter, the last value of the parameter and how many steps are needed. The next three are the parametric equation for X, Y and Z.

For example, if a straight line is needed, and the initial and last point are P1= (50,75,100) and P2= (80,150,120), the next operations are needed to determine each equation:

$$n_1 = 0\,, n_2 = 10$$
$$x_1 = 50, x_2 = 80$$
$$y_1 = 75, y_2 = 150$$
$$z_1 = 100, z_2 = 120$$

Three parametric lineal equations are needed so the formula of the straight line must be applied to each coordinate.

$$X = \frac{80-50}{10-0}(n) + 50 \rightarrow 3n + 50$$
$$Y = \frac{150-75}{10-0}(n) + 75 \rightarrow 7.5n + 75$$
$$Z = \frac{120-100}{10-0}(n) + 100 \rightarrow 2n + 100$$

With these three equations the line between those 2 points can be computed.

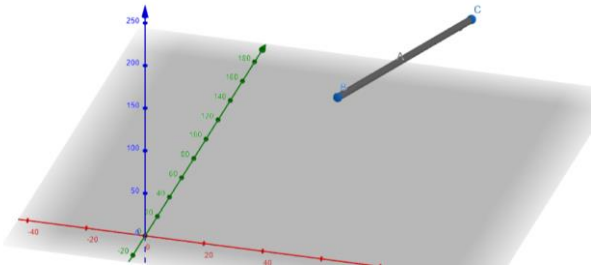


Figure 3: Linear parametric trajectory

Now if a circular trajectory is needed, it can be parametrized two ways, a 2D circumference combined with a line for the third dimension, or a 3D circumference.

For example, let's say that the same points on the previous example need to be reached with a circular route on X and Y and a linear movement on Z.

The Z parametric equation stays the same, so:

$$Z = \frac{120-100}{10-0}(n) + 100 \rightarrow 2n + 100$$

For X and Y, the center of the circle must be defined, with only 2 points you can determine a simple center just adding the coordinates and dividing them by 2:

$$C = \left(\frac{50+80}{2}, \frac{75+150}{2}\right) = (65\,, 112.5)$$

To compute each equation using this center you must use the distance between the center and one of the two points as the radii of the circle so:

$$R = \sqrt{(150-112.5)^2 + (80-65)^2} = 40.38$$

The last thing needed is to determine the angular frequency and the phase angle for each sine and cosine, so the circle matches the right points at the same parameter value:

Now that we know the radii, the phase angle must be determined, also the angular frequency depends on how many turns we want on the interval of parameters stablished, in this case only half of a circle will be implemented.

$$X(n) = 40.38\cos\left(\frac{\pi}{10}n + \phi\right) + 65$$
$$50 = 40.38\cos(\phi) + 65$$
$$\phi = \operatorname{acos}\left(\frac{50-65}{40.38}\right) = 1.9514$$
$$X(n) = 40.38\cos\left(\frac{\pi}{10}n + 1.9514\right) + 65$$
$$Y(n) = 40.38\sin\left(\frac{\pi}{10}n + \beta\right) + 112.5$$
$$75 = 40.38\sin(\beta) + 112.5$$
$$\beta = \operatorname{asin}\left(\frac{75-112.5}{40.38}\right) = -1.1908$$
$$Y(n) = 40.38\sin\left(\frac{\pi}{10}n - 1.1908\right) + 112.5$$

With these two examples the user can see that any route can be computed, obviously the process to obtain the equation is very different depending on how many points you know previously and the type of equation that represents the route.

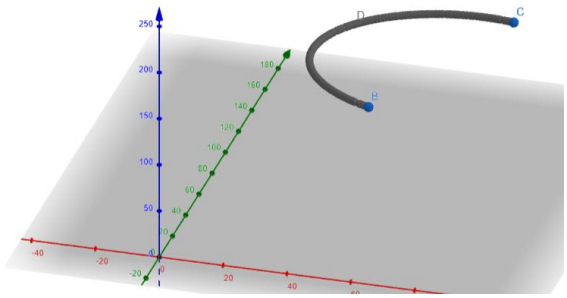


Figure 4: Circular parametric trajectory

E. Results

Results for DGM and IGM

Before computing the error of our robot, it is important to say that some offsets were required to align our motors 0 rad angles to the desired position. Those offsets were added on the arduino, for motor 1 and 3 the offset has a digital value of 512 that in degrees is equal to 150.14°, for motor 2 the offset was equal to 208 that is equivalent to 61°. With these offsets our desired position for the vector Q = [0,0,0] was achieved.

Ten set of Q vectors were used to compute the position with DGM and compare it with the measurements computed with the trilateration. The next table shows the results:

Q vector	X calculada(mm)	Y calculada(mm)	Z calculada(mm)	X medida(mm)	Y medida(mm)	Z medida(mm)	Error X(%)	Error Y(%)	Error Z(%)
[0,0,0]	205.2	97	89	196.9	99.95	110.6	4.04483	-3.04124	-24.2697
[0,0,1.57]	110.3	97	184	104.25	97.79	190.3	5.48504	-0.81443	-3.42391
[0.5,0,1.57]	102.1	128.9	184	102.4	137.7	195.3	-0.29383	-6.827	-6.1413
[1.3,-0.4,1.57]	69.98	191.7	150.6	72.73	201.72	153.6	-3.92969	-5.22692	-1.99203
[1.3,2,1.57]	13.16	-12.93	110	14.46	-5.63	110.91	-9.87842	56.4578	-0.82727
[2.3,0,0.8]	-44.73	195.9	157.1	-49.76	208.83	157.06	-11.2452	-6.60031	0.02546
[2.3,1.57,0.8]	89.02	46.26	221.7	102.14	45.36	221.38	-14.7383	1.94553	0.14434
[0,1.57,0]	43.81	97	250.5	41.56	95.77	253.33	5.13581	1.26804	-1.12974
[2,3,-0.7]	97.42	-20.42	169.2	113.91	-25.26	162.43	-16.9267	-23.7023	4.00118
[-2.4,-0.5,0.7]	-68.01	-5.309	75.99	-71.47	-6.34	72.49	-5.08749	-19.4199	4.60587

In some measurements there is a considerable amount of error, the problem here lies at the source of the error. There are at least 3 sources of error in the robot.

The first one is the resolution of the motor. It has 10 bits of resolution; this gives us steps of $\frac{300}{1024}$ degrees. It is a small resolution; the problem lies at the cosine and sine of 3 angles with this resolution. The input could be an angle that lies just between the calculated step, so the motor will choose the upper or lower value closest to the input value, if this happens with the 3 motors, mathematically you have the exact angle but physically you get 3 cosines and sines that all have a little bit of error because of the resolution. Then the error is amplified by the link size, this source of error is the bigger because the 3 position equations have cosines and sines:

$$X = L_2c1c2 + L3c1c23$$
$$Y = L2s1c2 + L3s1c23$$
$$Z = L1 + L2s2 + L3s23$$

As the equations show us, there are lot of cosine and sine products, so the resolution affects the 3 coordinates, also we can see that the Z coordinate is the least affected by this source of error based on the 2 sines that appear on its equation, this has sense with the results of Z error, in almost all measurements it has the smaller error. It should be mentioned that to achieve a desired position on q= [0,0,0], there is an offset angle, so this offset angle is the one that can fall between the steps. The last thing related to resolution is the rounding of results, as in the interface we input angles in radians, but the Arduino code use then in degrees, so the values are converted between these two units and some decimal values are lost.

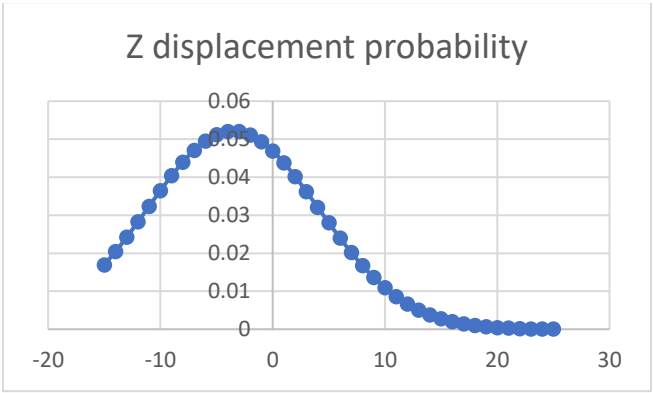
The second source of error could be the torque of the motors, in some positions the gravity could play an important role in the position of the motor 3 because of the smaller torque compared to motor 2, the motor 1 also has smaller torque but in its position the torque applied by gravity does not affect its rotation axis, the error here can be the speed of rotation, having the same specifications that the motor 2, just a smaller gearbox, the rotation speed of this motor is bigger, so it has a bigger inertia, this can leave the motor 1 a little bit out of the desired angle.

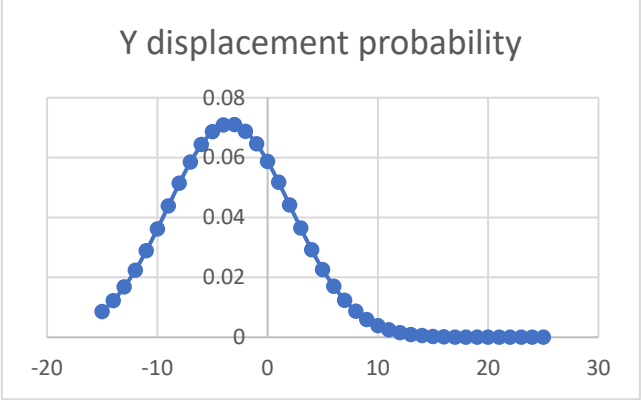
The third source of error could be the actual measurements of the sphere's radii, increasing the amount of error. This happens because we simple measured a wrong radius, while measuring we moved the robot, or both, increasing the error.

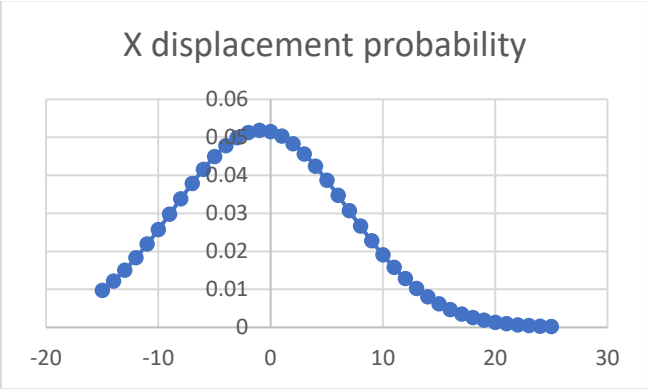
Thought the relative error appear to be big, the difference between the real and computed position is not that big, the next table shows the difference in millimeters between the real and computed positions.

	X diference	Y diferenence	Z diference
	8.3	-2.95	-21.6
	6.05	-0.79	-6.3
	-0.3	-8.8	-11.3
	-2.75	-10.02	-3
	-1.3	-7.3	-0.91
	5.03	-12.93	0.04
	-13.12	0.9	0.32
	2.25	1.23	-2.83
	-16.49	4.84	6.77
	3.46	1.031	3.5
Average	-0.887	-3.4789	-3.531
Variance	59.335681	31.34259089	58.528429
standard deviation	7.7029657	5.598445399	7.6503875

With this information we can see that the average difference is an average of five millimeters. With these data we can calculate the probability of different displacements, if these displacements have a normal distribution, our probability functions have the next graph.







With this probability distribution we can approximate the value of displacement that has a probability of 90% to occur, this is the limit of displacement between the area of the function probability has a value of 0.9.

Using a probability chart, we need to find the value that closes 0.95 of the area, this is because the graph is symmetrical so:

For any normal probability distribution, the value of Z, where Z is the amount of standard deviation between the desired value and the mean, that has 0.95 of the area is:

$$Z = \pm 1.645$$

The negative value is because the graph is symmetrical, and the negative value is having the rest of the value, so the more probable values of displacement are contained between this values.

For X displacement:

$$\Delta X_1 = 1.645\sigma + \mu = (1.645)(7.703) - 0.887 = 11.78\ mm$$

$$\Delta X_2 = -1.645\sigma + \mu = (-1.645)(7.703) - 0.887 = -13.55\ mm$$

This data says that with 90% of probability the real X coordinate of the end effector will have a difference between -2.36 mm and 14.17 mm with respect to the calculated position by the DGM. Knowing the meaning of this data, next we compute the limits of displacement for Y and Z

$$\Delta Y_1 = 1.645\sigma + \mu = (1.645)(5.6) - 3.47 = 5.742\ mm$$

$$\Delta Y_2 = -1.645\sigma + \mu = (-1.645)(5.6) - 3.47 = -12.682\ mm$$

$$\Delta Z_1 = 1.645\sigma + \mu = (1.645)(7.65) - 3.531 = 9.07\ mm$$

$$\Delta Z_2 = -1.645\sigma + \mu = (-1.645)(7.65) - 3.531 = -16.09\ mm$$

Results for trajectories mode

On this mode obviously the robot has the same issues with the mechanical error discussed previously, but on this mode numeric error also appears, so this section will only analyze the error related to the numeric approximation of the trajectories.

Probably the computation error will vary with the number of steps, in what iteration the code currently is and the complexity of the route. For this analysis, only the line example trajectory will be checked.

To visualize how the error increases, the number of steps will be changed. The first try will be with 20 steps between the initial point and the last point.

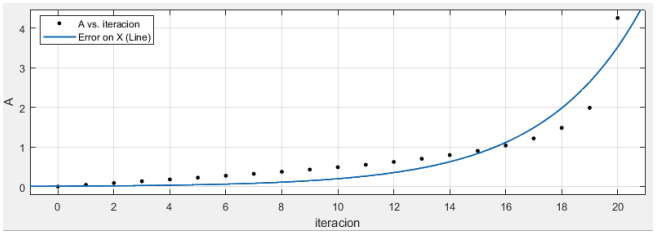


Figure 5: Error per iteration on X (Line, 20 iterations)

We can see that the error is like an exponential curve. Using curve fitting from MATLAB the R square value is of 0.892, this indicates that the fit is accurate. It has sense that the error is exponential because on each iteration the error adds up generating more and more error, also the rate of change of the error can be seen.

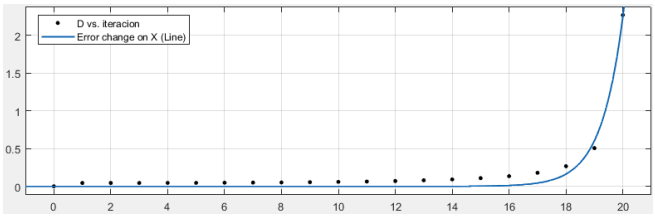


Figure 6: Error change per iteration on X (Line, 20 iterations)

The rate of change of the error is an approximation of the derivative, so the first curve being an exponential, it has sense the approximation of its derivative is also an exponential function. Now the same process will be done to Y and Z

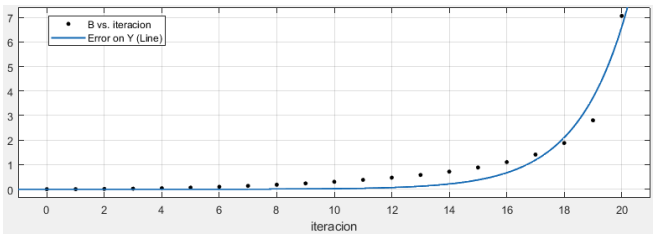


Figure 7: Error per iteration on Y (Line, 20 iterations)

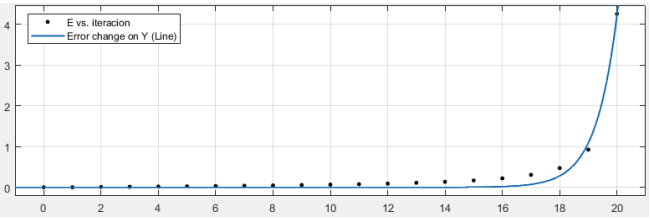


Figure 8: Error change per iteration on Y (Line, 20 iterations)

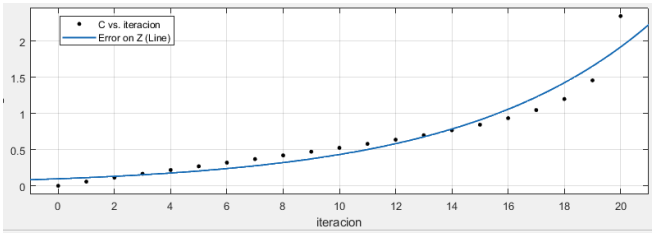


Figure 9: Error per iteration on Z (Line, 20 iterations)

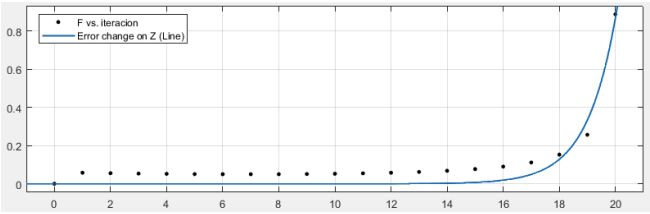


Figure 8: Error change per iteration on Z (Line, 20 iterations)

Now the number of steps will be increased to 40, the expected result on both graphs is that both will be flatter.

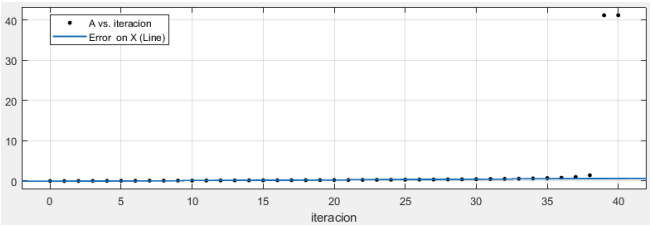


Figure 9: Error per iteration on X (Line, 40 iterations)

As it can be seen, the amount of error is almost cero, except in two points, this probably occurs because a singular configuration was reached, if the joint vector value is checked one of the angles will be near a singular configuration the value of the joint vector 39 is [1.0809,0.1904,-0.0239], the value of q3 is near 0, which is one of the singular configurations so it has sense that the error suddenly increased. Thus, the derivative has a spike on that value because of the sudden change as it can be seen on the next graph.

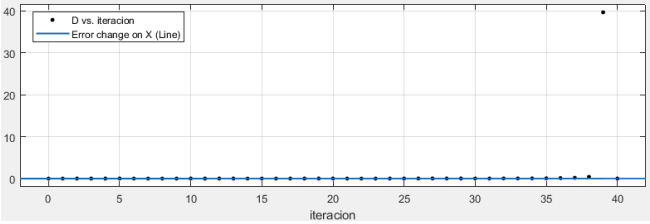


Figure 10: Error change per iteration on X (Line, 40 iterations)

If the same technique is applied to leave the angles between minus pi and pi some of the singular configurations could be dealt with, nevertheless the increase on the number of steps could be negative for the singular configurations, the more points you compute its more probable that one of those points it's a singular configuration, the less steps are used its less probable to compute a singular configuration but the error increases, so an equilibrium should be found to get the more advantages possible. The rest of the graphs look almost the same so, next what will be checked is the same trajectory but backwards. The equations for this are the following ones:

$$X = -3n + 80$$
$$Y = -7.5n + 150$$
$$Z = -2n + 120$$

If those equations are used as input the next error graph is obtained (using 20 and 40 iterations) :

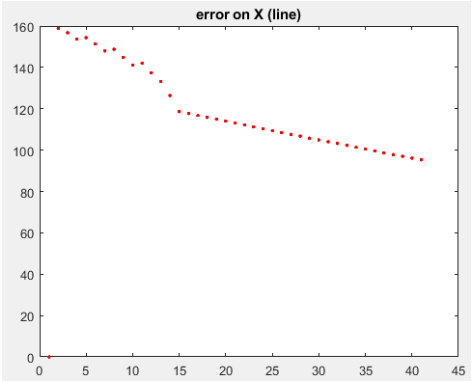


Figure 11: Error per iteration on backwards X (Line, 40 iterations)

The error is worse backwards because the singular configuration its at the start of the trajectory. Now if we check the same equations but with only 20 iterations:

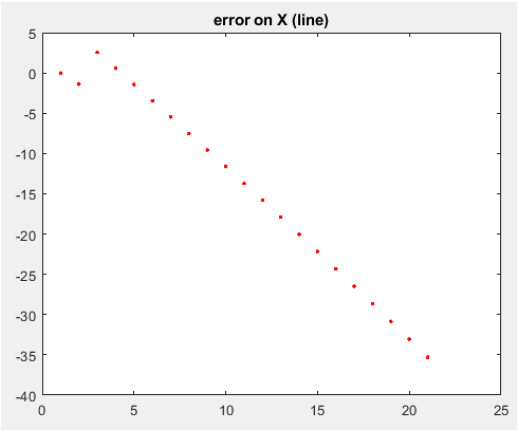


Figure 12: Error per iteration on backwards X (Line, 20 iterations)

Now the error is on the other direction but is smaller than with 40 iterations because the computed points were farther away from the singular configuration than with 40 iterations. Let's try a trajectory with no singular configuration to verify if the error is affected by the direction of the movement.

The next equations will be used:

$$X = 65$$

$$Y = 8n - 50$$
$$Z = 100$$

This corresponds to a straight line that only moves in Y.

The error in Y looks like this:

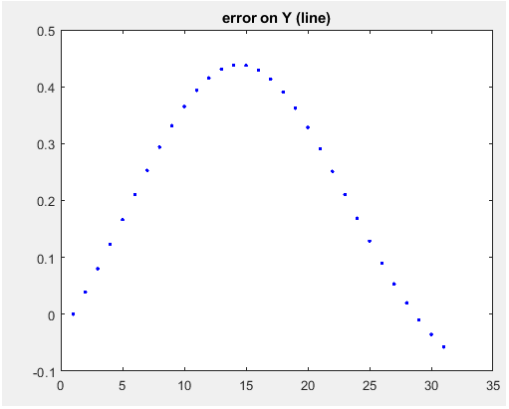


Figure 12: Error per iteration on Y (second example)

Visually the graph looks like it has lots of error but putting attention to the magnitude, the error is small, so probably the trajectory is far from any singular configuration. Now with the same number of iterations, the backwards trajectory will be computed:

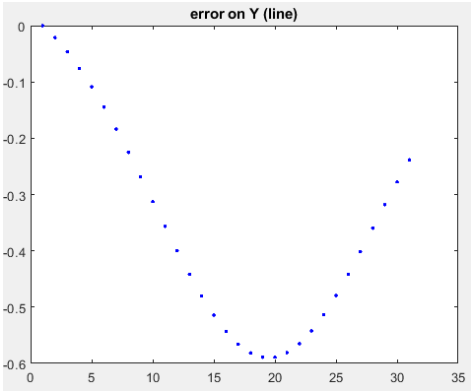


Figure 12: Error per iteration on backwards Y (second example)

There is an increase on the magnitude of the error and the direction of the error is also inverted, this means that the real position is greater than the ideal one. With this example it can be proved that the direction of the route changes the amount and direction of the error

With all this examples some variables that affect the error were seen, nevertheless there can be many more variables that have not been checked. So, to summarize, the amount of error can be affected by the direction of the movement on the trajectory, number of iterations and proximity to a singular configuration.

VI. CONCLUSION

DGM conclusions

César Eduardo Monterrubio Morales: Even though during the course we have calculated the direct geometric model of different kinds of robots, the results were always abstract because we couldn't see anything but some numbers. The first time we actually saw the real application of the forward kinematics was when we did the 6 DOF robot homework and used robot dk to simulate and check our results. Doing this part of the project gave an application to all the codes that have been done during the course in something physical, not just mathematical. Getting the expected results or some physical coordinates similar to the calculated ones was a great experience because first we couldn't get the correct coordinates until we finally added an offset to some initial angles.

Luis Mario Flores Campos: Developing a robot capable of being controlled through MATLAB and Arduino by using the methodologies seen in class was a success. Understanding how to merge an IDE (Arduino) with MATLAB was an extreme learning experience, not only because it pushed me to finish my ability to do research on my own, but also because I comprehend how to communicate with both tools. This project taught me that sometimes electronic devices will not have the requisites we need by default. To explain it better, we needed to add a constant of 2.66 if we wanted to align the motors with our y-axis and x-axis.

Fernando Mejia Laguna: The system was able to communicate successfully from matlab to Arduino and finally with the robot. I could understand first that the HMI was of extreme importance as the way the user interacts with the interface will determine how easy it is to manipulate the robot. Regarding the code, this was something that facilitated enormously our test sessions. The code, as we've seen during this course, worked as expected. However, the problems we encountered were mainly with the coding language of MATLAB app designer as it didn't allow us to call functions directly from MATLAB. Finally, we've encountered with problematics regarding the physical world, as the position 0 of the motors where different as the position 0 in the codes; To solve this, we added the differential of the angle in radians, a value of 2.66 in the code of Arduino, allowing the robot to move 150° to the left and right, with a total range that goes from 0 to 300°.

General conclusions:
During this phase of the project, we were able to develop a series of methods to control anthropomorphic robots arms the manual mode and DGM. However, we found a series of areas of opportunities; First, it was found that the measurements in the robot had an error of 1cm approximately, given that we are working with motors that do not have a proper torque and do not resist the inertia of the movement when it is too fast, which can be solved by either sending the values in smaller ranges to reduce the velocity. We also identified that the Manual and

DGM methods were working properly, allowing us to control the end effector by manipulating the joint variables either manually or sending the values in radians directly, but given that the Arduino has a delay in the code, this might represent a slow reaction time compared with the given input.

IGM conclusions

Cesar Eduardo Monterrubio Morales: This phase of the project was easier than the previous one because we already had a base. Solving all the problems of the DGM opens a lot the possibilities of the IGM because independently of the method you choose, numerical or analytical, both use information given by the DGM. The results obtained are pretty good thought some of the positions will always have error like the calculated on the results section, some additional error can be obtained from positions that were filtered to avoid damage to the robot. Other problem that sometimes happens is that the base of the robot blocks the movement, so the angles are correct, but this mechanical restriction makes impossible the movement. We already had a code for solving the IGM the analytical way but in my opinion the numerical method has the advantage of computing just one solution, so you don't need extra information or computations to decide which of the solutions to use. Also, an advantage of the numerical solution is that if suddenly we are asked to make a completely different robot, the same code of the numerical method can be used but the analytical would require lots of computations to get new equations for solving the IGM. Some could argue that for the numerical method you need the DGM because you need all the intermediate transformation matrices, while the analytical one needs only the complete transformation matrix, it doesn't matter because to get that final transformation matrix you need to compute the intermediate ones, so if you code is well done you can store all the transformation matrices and use the numerical method without any problems.

Luis Mario Flores Campos: Talking about the implementation of the IGM, I was able to see the strengths and drawbacks of both methods implemented: the analytical method and the numerical method. Speaking about the first method used, I was able to see that it requires more lines of coding to determine which result you must use, and the equations used are only usable for this robot chosen. In the other side, when we were implementing the numerical method. I was able to see that this one allows you to dispose of one of the results obtained automatically, which helps to simplify the process of selecting the right result.

Fernando Mejia Laguna: For this part of the project, we decided to calculate de inverse geometric model in two different manners, where the method of the Jacobian was, from our point of view, the best option to work with, given that it also allows to calculate any singularity with its determinant. We found a series of problems during this stage, such as having the software crashing after having a configuration with many possible solutions, sending values outside the workspace of the arm and a limitation on the robot displacement as the box interfered with the trajectories. However, we were able to solve most of these problems, for example, we add minimum and maximum values, so that the robot will be set to those values if the point is outside the workspace. Also, when computing a solution higher than the values in radians for the joints, we computed the cosine of it and then apply the inverse to reduce the range to a workable value. However, we still have difficulties when founding many solutions as the system, struggles to solve these singularities.

General conclusions:

On this phase of the project the only physical addition was the IGM mode. However, this is very important, the IGM was almost ready since the previous phase but was perfectioned for this phase, also the chance to implement the numerical method is pretty important because with it the trajectories mode will be implemented more easily.

Trajectories mode conclusion

Cesar Eduardo Monterrubio Morales: This was my favorite part of the project; it was exiting when the robot moves form one point to another but control the specific points the robot uses to reach that position was wonderful. Also, it is the first time that I put on practice the knowledge of parametric equations, for years I have been trying different equations to see what kind of routes those equations generate. Even thought I know many different trajectories, I do not know how to force a lot of those trajectories to pass through specific points, that's why the examples just have straight lines and a simple circle. Maybe it is for lack of information (additional points) o the process is to complex but at least the objective is completed with those two kinds of trajectories.

Fernando Mejia Laguna: For this final part of the project, we designed the trajectories part, which allows the user to draw a trajectory in three dimensions which the robot will follow.
For the system, we decided to separate x, y and z in order input manually the equation that describes the movement in each separate axis.

Respectively, each equation required a specific constant that will directly affect the behavior of the system.

As seen in class, we decided to use the method of the Jacobian to obtain the most efficient solution for the robot to follow the trajectory. However, this presented problems sometimes, as the most efficient trajectory was sometimes out of the work area of the robot.
The whole trajectory that the robot follows was divided in steps and, depending on that number, the difference between the past set of coordinates and the new set is significantly small, which can be solved with Matlab without any problem, on the other hand, the robot was physically unable to receive the command, so it would only move when the accumulation of this

commands where big enough that reached the minimum available movement of the servo.

Other problems we found were related to singular configurations, the robot tended to do random movements in every direction, after that specific point passed, it would return to the trajectory it was following.

To sum up, most of the inaccuracies that presented in front of us were solved and we were able to identify the problems that we couldn't solve due physical limitations.

Luis Mario: It was exciting to watch the robot go from one point to another, and the ability to control the precise spots that the robot used to get there was one of the project's highlights. One of the issues we discovered were associated with singular configurations; the robot exhibited a propensity for erratic movements in all directions, after which it would. We discovered that if connected to singular configurations, the robot tended to make haphazard movements in all directions, after which it would resume its original track. In order to find the most effective way for the robot to follow the trajectory, as was demonstrated in class, we choose to apply the Jacobian technique. Due to the most effective trajectory occasionally being outside of the robot's work area, this did occasionally provide issues.

VII. YOUTUBE VIDEO

https://youtu.be/8o4RfsvxuKg

https://youtu.be/DBADfP6y6vM

VIII. REFERENCES

[1] Wagieh, A., & Instructables. (2019, June 14). *Using Matlab app designer with Arduino*. Instructables. Retrieved November 1, 2022, from https://www.instructables.com/Using-MATLAB-App-Designer-With-Arduino/

[2] *Arduino support from Matlab*. Hardware Support - MATLAB & Simulink. (n.d.). Retrieved November 1, 2022, from https://www.mathworks.com/hardware-support/arduinomatlab.html

[3] Tuijthof, G. J. M., & Herder, J. L. (2000). *Design, actuation and control of an anthropomorphic robot arm. Mechanism and Machine Theory, 35(7), 945–962*.doi:10.1016/s0094-114x(99)00051-8

[4] Duffy, B. R. (2003). Anthropomorphism and the social robot. Robotics and Autonomous Systems, 42(3-4), 177–190. doi:10.1016/s0921-8890(02)00374-3

[5] Unanyan, N. N., & Belov, A. A. (2021). Anthropomorphic arm control system with remote gesture tracking. *IFAC-PapersOnLine*, *54*(13), 443–448. https://doi.org/10.1016/j.ifacol.2021.10.488

[6] Unanyan, N. N., & Belov, A. A. (2021, November 12). *Anthropomorphic arm control system with remote gesture tracking*. IFAC-PapersOnLine. Retrieved November 1, 2022, from https://www.sciencedirect.com/science/article/pii/S240589632101925X

[7] Robotis. (n.d.). Retrieved November 1, 2022, from http://en.robotis.com/service/downloadpage.php?ca_id=7040#

[8] Hamerlain, M. (n.d.). An anthropomorphic robot arm driven by artificial muscles using a variable structure control. Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots. doi:10.1109/iros.1995.525851