ETE 4751 Robotics

David Sumadsad

10/14/2020

Introduction

There are two ways to classify robots, mobile robots and fixed base robots. Mobile robots have a base that can move either on land, air, or water. Fixed base robots have one of their parts fixed to the ground or some solid structure. For this project we will be working on a fixed base robot like the one in this paper [3]. Many aspects of a robot are researched and studied on, for example this robot uses wires to control the direction and orientation of the arm and grasping hand instead of motors for each joint [4]. Another aspect of robots to research is the controls aspect of robots, where many different types systems are implemented to reduce the steady-state error. Some of these systems include the use of Proportion Integral Differentiation (PID) [8] controllers where the error of the robot is subject to each mathematical operation to determine the best course of action, another system is Fuzzy logic control, where Fuzzy logic is used to approximate human reasoning in other words making decisions on imprecise or non-numerical information [5].

There are many applications for robots especially in industry, these tasks may include assembly, grinding, painting, welding, and pick and place. An example of a robot in industry is this mobile robot with an arm, designed as a pick and place robot that can traverse within a facility [7]. Another robot for industrial use is this welding robot that uses dual-beam laser welding technology that is multi-arm [6]. However, besides industry there are other uses such as this robot which can be used to help people with disabilities for their activities of daily living (ADL) such as toileting, dressing, and eating [2].

Theoretical Background

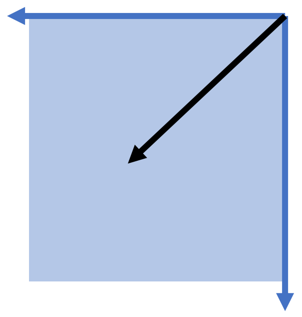
One of the most important aspects when it comes to simulating or modeling robotic arms are the Kinematics of robots, analyzing how each angle of the joint corresponds to the end effector's position. To model the robot, we first need to find the forward kinematic equations of the robot, from there we can find the inverse kinematic equations where the angle of each joint can be found by inputting the position and orientation of the robot. One of the most used methods is the Denavit-Hartenberg Representation of Forward Kinematic Equations, also known as D-H model, for robots which is used extensively for its simplicity.

The procedure for modeling with D-H goes as follow.

- All joints are represented by a z-axis, if the joint is revolute the z-axis points in the direction of motion by right-hand rule, if prismatic then the z-axis points along the direction of linear movement.
- The index number for a z-axis of joint *n*, is *n*-1, for example a robot with two joints, joint one will have a z_0 axis and joint two will have a z_1 axis.
- If the z-axes are not parallel or intersecting there exists a line between both that run perpendicular through both axes, this is called a common normal, and the direction of the common will be the direction of the x-axis. For example, if we have two z-axes, z_n and z_{n+1}, there is a common normal between the two with a distance of a_{n+1}, the direction of

the common will be our x_{n+1}, so when modeling our robot, it's important to keep in mind that the direction of the x-axis applies to the following z-axis.

- If two z-axes are parallel, an infinite number of common normal exists, we will choose the common normal that points in the same direction of the previous common normal
- If two z-axes intersect, the direction of the x-axis is perpendicular to plane formed by the z-axes, one way to visualize this is the above image, if the blue lines are intersecting z-



axes and the shaded area the plane formed by the two axes, the black line represents the x-axis that is perpendicular the plane, essentially the x-axis is *coming out of the plane*, of course the direction could be in the opposite direction where the x-axis is *going into the plane*, whichever direction simplifies the model is best.

After assigning the frames we input each variable into the following D-H parameters table.

#	ϴ	d	a	α
0-1	Θ_1	d_1	a_1	α_1
1-2	Θ_2	d_2	a_2	α_2
2-H	Θ_{n+1}	d_{n+1}	a_{n+1}	α_{n+1}

Where the following variables are described,

- ϴ, we rotate the x_n axis to match the orientation of the x_{n+1} axis, about the z_n axis by an amount Θ_{n+1}
- d, we translate the x_n axis to match the position of the x_{n+1} axis along the z_n axis by an amount d_{n+1}
- a, we translate the z_n axis to match the position of the z_{n+1} axis along the x_{n+1} axis by an amount a_{n+1}
- α, we rotate the z_n axis to match the orientation of the z_{n+1} axis, about the x_n axis by an amount α_{n+1}

We can see now that why D-H parameters are so successful and simple, it essentially describes the transformations between each following joint all the way towards the hand or end effector. Each transformation for each joint can be described by the following matrix equation.

$$A_{n+1} = \begin{bmatrix} C\theta_{n+1} & -S\theta_{n+1}C\alpha_{n+1} & S\theta_{n+1}S\alpha_{n+1} & a_{n+1}C\theta_{n+1} \\ S\theta_{n+1} & C\theta_{n+1}C\alpha_{n+1} & -C\theta_{n+1}S\alpha_{n+1} & a_{n+1}S\theta_{n+1} \\ 0 & S\alpha_{n+1} & C\alpha_{n+1} & d_{n+1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (Eq\ 1)$$

Once we have the matrixes for all joints, we can simply multiply all of them to derive the full model forward kinematic equation.

Modeling The Robot Using Matlab

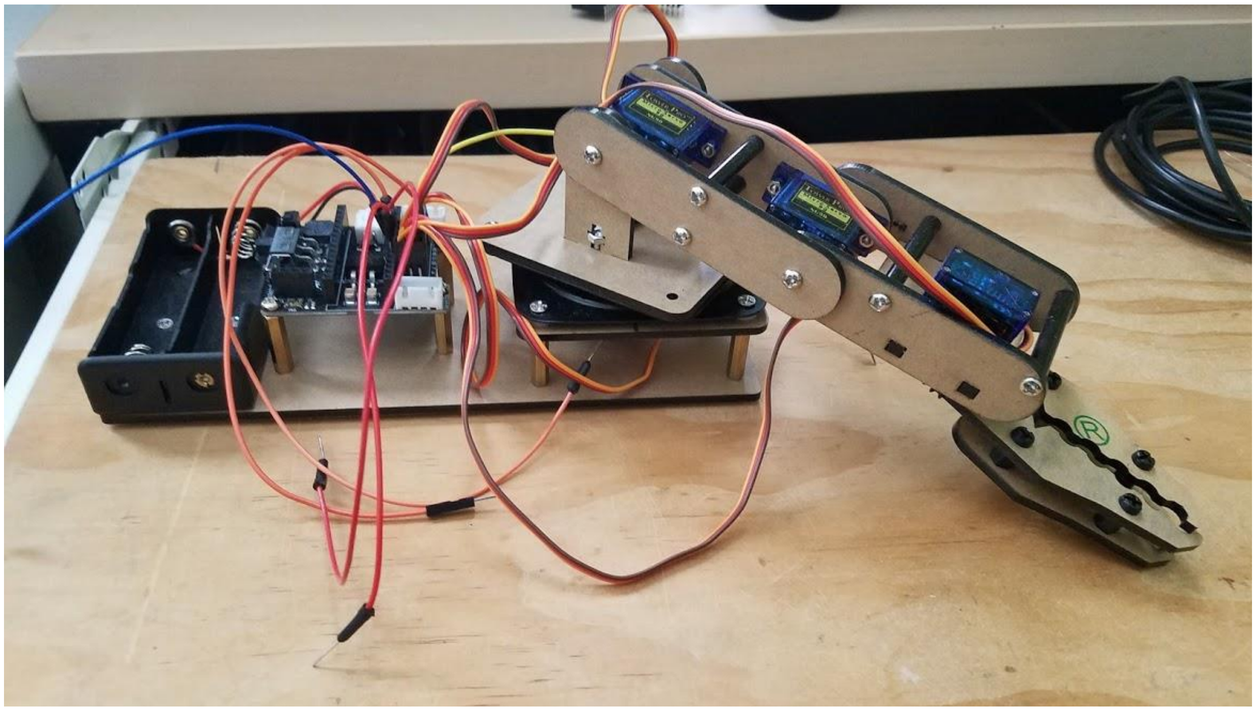


Figure 1. Picture of built robot arm

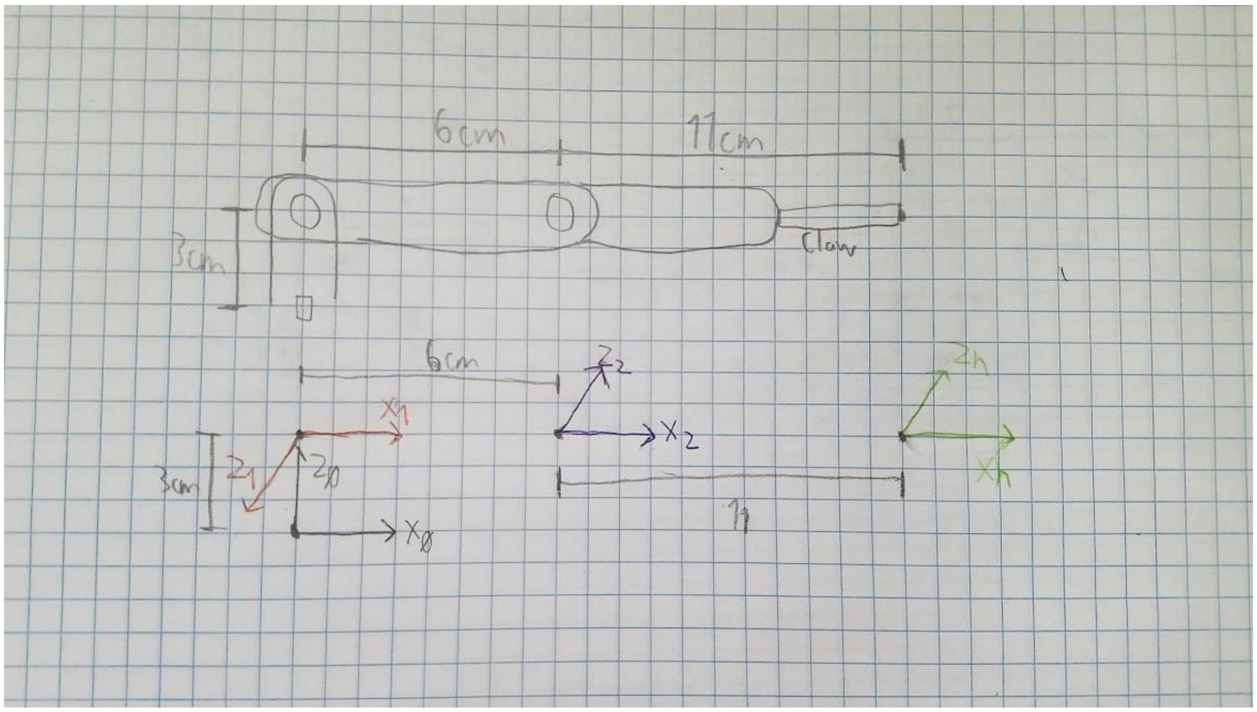


Figure 2. Assigned frames for each joint

With our robot built we can assign our frames and input our D-H table, one important aspect to note is that our z_2 axis points in the opposite direction from our z_1 axis, this is because of how our robot arm is built and is important to keep track of. This is also shown in our D-H table where the α value of 1-2 is 180°.

#	ϴ	d	a	α
0-1	Θ_1	3	0	90°
1-2	Θ_2	0	6	180°
2-H	Θ_3	0	11	0°

Using MatLab we first create table of our D-H parameters which is a simple three by four array. Then we create a general symbolic matrix equation that describes Eq. 1 from the previous section and using a for loop input our values from the D-H table to the equation. It’s important to note the use of curly brackets instead of parentheses to denote a cell array, from my research using a cell array seems to be the only way to create an array of matrices. Now that matrices for each joint is found we can multiply each matrix to get the full model.

```
>> D_H_Table = [...
Theta_1 3 0 pi/2; ...
Theta_2 0 6 pi; ...
Theta_3 0 11 0]

D_H_Table =
[ Theta_1, 3,  0, pi/2]
[ Theta_2, 0,  6,   pi]
[ Theta_3, 0, 11,    0]

>> Gen_A
Gen_A =
[ cos(Theta), -cos(Alpha)*sin(Theta),  sin(Alpha)*sin(Theta), a*cos(Theta)]
[ sin(Theta),  cos(Alpha)*cos(Theta), -sin(Alpha)*cos(Theta), a*sin(Theta)]
[          0,             sin(Alpha),             cos(Alpha),            d]
[          0,                      0,                      0,            1]
>>

>> for k = 1:3
A{k} = subs(Gen_A,[Theta, d, a, Alpha],[D_H_Table(k,1) ,D_H_Table(k,2) ,D_H_Table(k,3) ,D_H_Table(k,4)])
end

A =
  1×3 cell array
    {4×4 sym}    {4×4 sym}    {4×4 sym}
A =
  1×3 cell array
    {4×4 sym}    {4×4 sym}    {4×4 sym}
A =
  1×3 cell array
    {4×4 sym}    {4×4 sym}    {4×4 sym}

>> Full_Model = simplify(A{1}*A{2}*A{3})

Full_Model =
[ cos(Theta_1)*cos(Theta_2 - Theta_3), cos(Theta_1)*sin(Theta_2 - Theta_3), -sin(Theta_1), cos(Theta_1)*(6*cos(Theta_2) + 11*cos(Theta_2 - Theta_3))]
[ sin(Theta_1)*cos(Theta_2 - Theta_3), sin(Theta_1)*sin(Theta_2 - Theta_3),  cos(Theta_1), sin(Theta_1)*(6*cos(Theta_2) + 11*cos(Theta_2 - Theta_3))]
[              sin(Theta_2 - Theta_3),             -cos(Theta_2 - Theta_3),             0,             6*sin(Theta_2) + 11*sin(Theta_2 - Theta_3) + 3]
[                                   0,                                   0,             0,                                                          1]
```

The full model is simplified below.

$$Full\ Model = \begin{bmatrix} C_1C_{23} & C_1S_{23} & -S_1 & C_1(6C_2+11C_{23}) \\ S_1C_{23} & S_1S_{23} & C_1 & S_1(6C_2+11C_{23}) \\ S_{23} & -C_{23} & 0 & 6S_2+11S_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Where \begin{array}{l} S_{23} = Sin(\Theta_2 - \Theta_3) \\ C_{23} = Cos(\Theta_2 - \Theta_3) \end{array}$$

Finding the Inverse Kinematic Equation

Now that we have the forward kinematic equation, we must perform the inverse kinematic equation, we do this by multiplying the inverse of one the joint frames to the desired position and orientation frame. The process for doing so is taken from the Saeed B. Niku textbook [1].

$$A_1^{-1} * \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_1^{-1} * \begin{bmatrix} C_1C_{23} & C_1S_{23} & -S_1 & C_1(6C_2+11C_{23}) \\ S_1C_{23} & S_1S_{23} & C_1 & S_1(6C_2+11C_{23}) \\ S_{23} & -C_{23} & 0 & 6S_2+11S_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} n_xC_1+n_yS_1 & o_xC_1+o_yS_1 & a_xC_1+a_yS_1 & p_xC_1+p_yS_1 \\ n_z & o_z & a_z & p_z-3 \\ n_xS_1-n_yC_1 & o_xS_1-o_yC_1 & a_xS_1-a_yC_1 & p_xS_1-p_yC_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C_{23} & S_{23} & 0 & 6C_2+11C_{23} \\ S_{23} & -C_{23} & 0 & 6S_2+11S_{23} \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equating elements (3,4) for both sides we get

$$p_xS_1 - p_yC_1 = 0 \longrightarrow \Theta_1 = \tan^{-1}(\frac{p_y}{p_x})$$

Next, we equate elements (1,4) and (2,4) to find Θ_3.

$$6C_2 + 11C_{23} = p_xC_1 + p_yS_1$$

$$6S_2 + 11S_{23} = p_z - 3$$

$$(6C_2 + 11C_{23})^2 = \left(p_xC_1 + p_yS_1\right)^2$$

$$(6S_2 + 11S_{23})^2 = (p_z - 3)^2$$

$$36(S_2^2 + C_2^2) + 121(S_{23}^2 + C_{23}^2) + 132(S_2S_{23} + C_2C_{23}) = \left(p_xC_1 + p_yS_1\right)^2 + (p_z - 3)^2$$

$$(S_2S_{23} + C_2C_{23}) = \frac{\left(p_xC_1 + p_yS_1\right)^2 + (p_z - 3)^2 - 36 - 121}{132}$$

Using the trigonometric function

$$(S_2S_{23} + C_2C_{23}) = Cos[(\theta_2 - \theta_3) - \theta_2] = Cos(-\theta_3) = Cos(\theta_3)$$

So,

$$C_3 = \frac{\left(p_xC_1 + p_yS_1\right)^2 + (p_z - 3)^2 - 36 - 121}{132}$$

Since we know C_3 we can also find S_3 then using by inverse tangent Θ_3.

$$S_3 = \sqrt{1 - C_3^2}$$

$$\theta_3 = \tan^{-1}\frac{S_3}{C_3}$$

Finally, we need Θ_{23} to find Θ_2, which we find by using elements (1,1) and (2,1)

$$C_{23} = n_x C_1 + n_y S_1$$

$$S_{23} = n_z$$

$$\theta_{23} = \tan^{-1}\frac{S_{23}}{C_{23}}$$

$$\theta_{23} = \theta_2 - \theta_3 \longrightarrow \theta_2 = \theta_{23} + \theta_3$$

Now we can find the angles, we only need to input the position and the orientation of the x-axis of the hand.

Using Simulink and STM32 Nucleo board

For controller and method for programming, I decided to use the STM32 Nucleo Board and program it by using MatLab Simulink, we can simply use block diagrams to create the program, this is much different than programming with Arduino which requires someone to have a basic knowledge of the C programming language. When both methods are compared, Simulink creates a visual representation of the program, which makes readability of the program much easier than lines of C coding.

The first process we did was to create the calculations from the inverse kinematic equations. The calculations were then condensed into subsystems as to not clutter up the program.

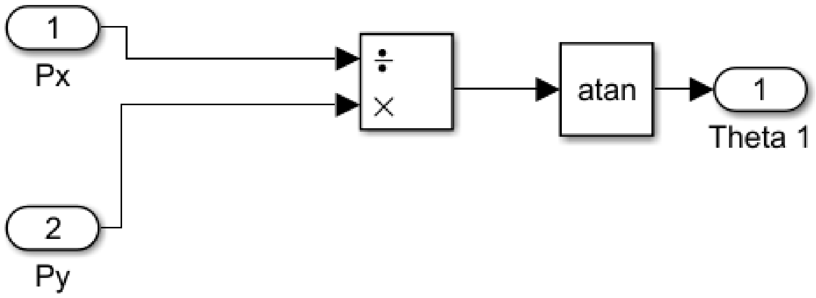


Figure 3. Simulink Block Diagram for Θ₁

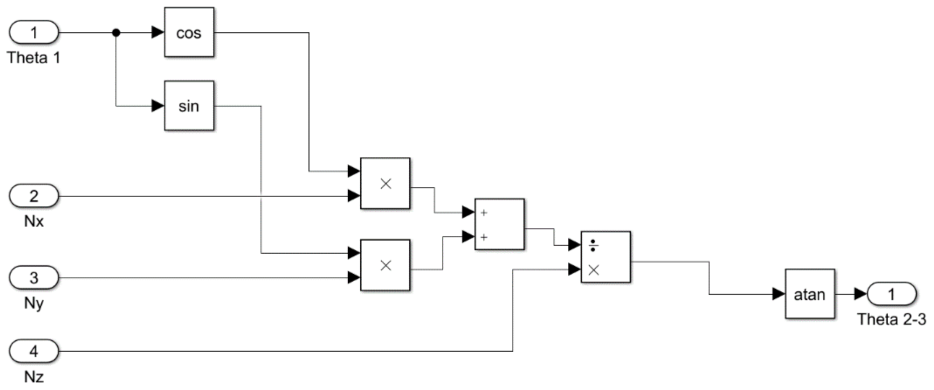


Figure 4. Simulink Block Diagram for Θ₂₃

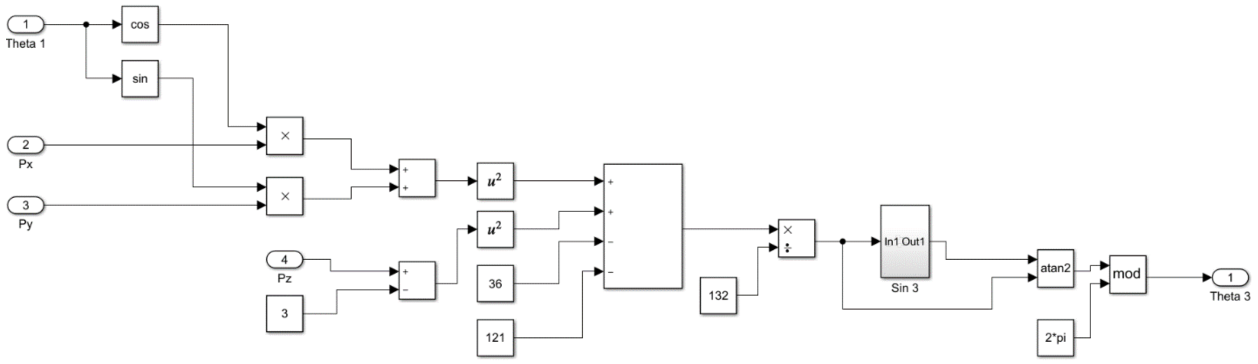


Figure 5. Simulink Block Diagram for Θ₃

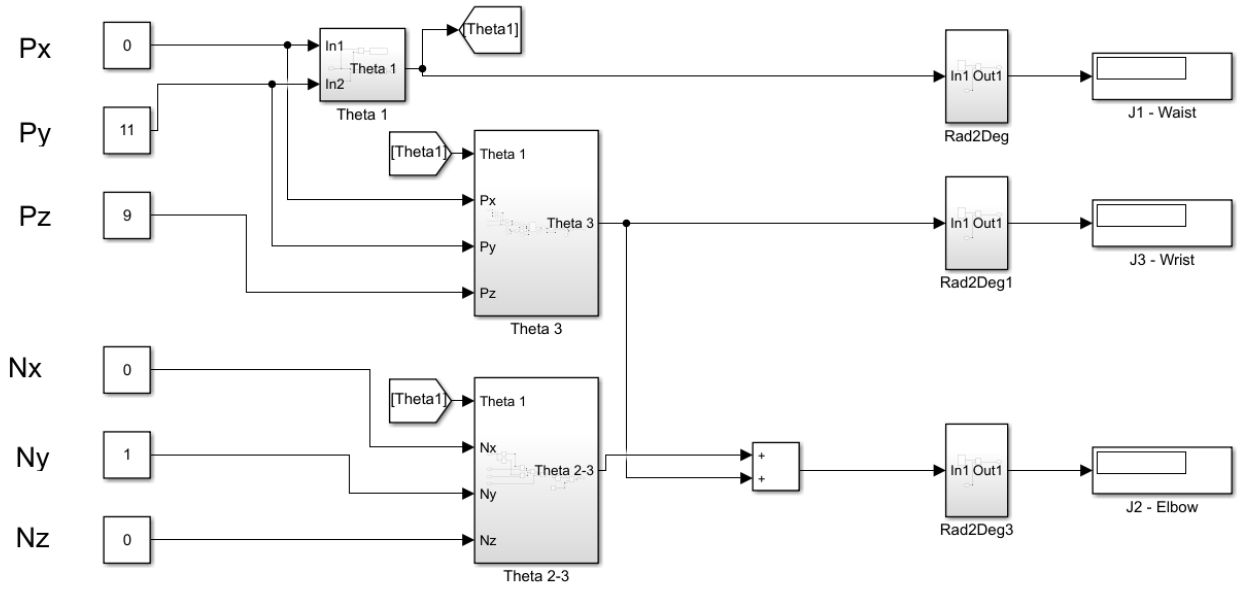


Figure 6. Simulink Block Diagram to find Θ compressed into subsystems

To validate our model and check if the equation is right, we input angles in our forward kinematic equation and input the position and orientation results into the model. If our Θ_1, Θ_2, Θ_3 are 45° ,90°, and 90° respectively. The result is

$$\begin{bmatrix} 0.707 & 0 & -0.707 & 7.778 \\ 0.707 & 0 & 0.707 & 7.778 \\ 0 & -1 & 0 & 9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

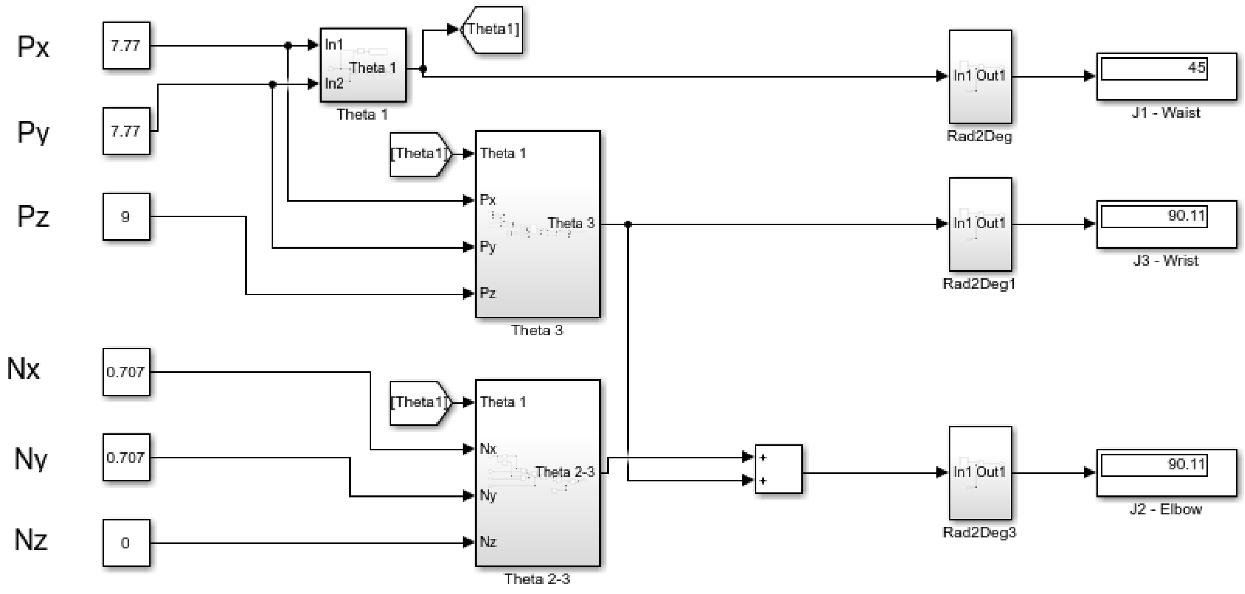


Figure 7. Simulink Block Diagram from before with inputted values

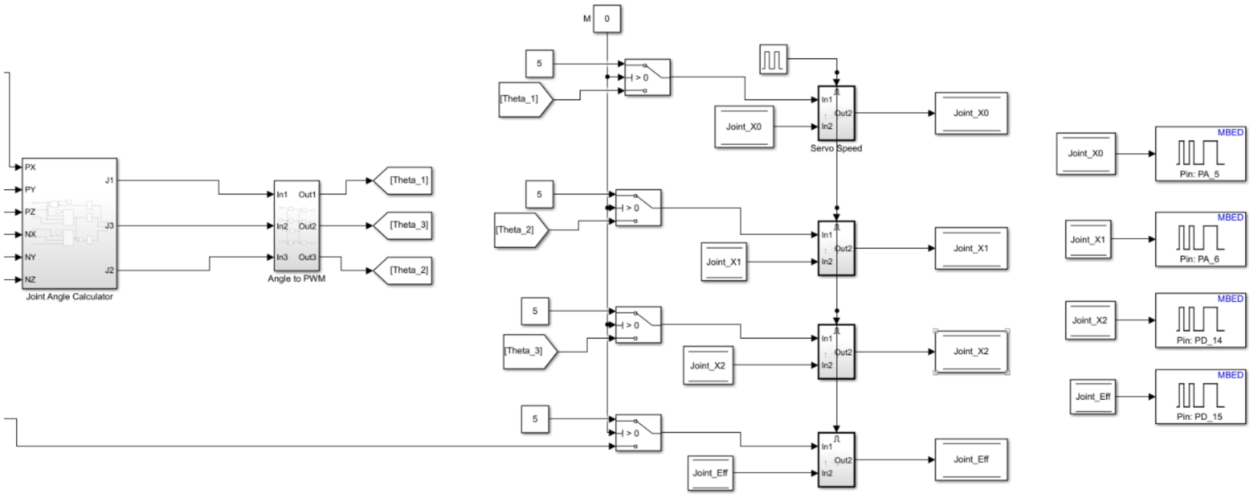


Figure 8. Θ subsystem with PWM blocks for Nucleo Board

If we input the p_x, p_y, p_z, n_x, n_y, and n_z values to our Simulink model we get the original angles. Which validates our equations.

Now that our equation is validated, we can convert the angle to a usable PWM signal that will help control the servo. One of the subsystems used is called "Servo Speed" which is used to change the speed of the servo by adding or subtracting the servo's present position by some user-defined increment. For example, if the servo's present angle is 0°, and the next angle is 90°, we add the increment value to the present value until we reach the next angle. We also use data store memory blocks, as well as subsequent Read, From, and Write To memory blocks, these are seen as Joint_X0, Joint_X1, Joint_X2, and Joint_Eff which controls the claw.

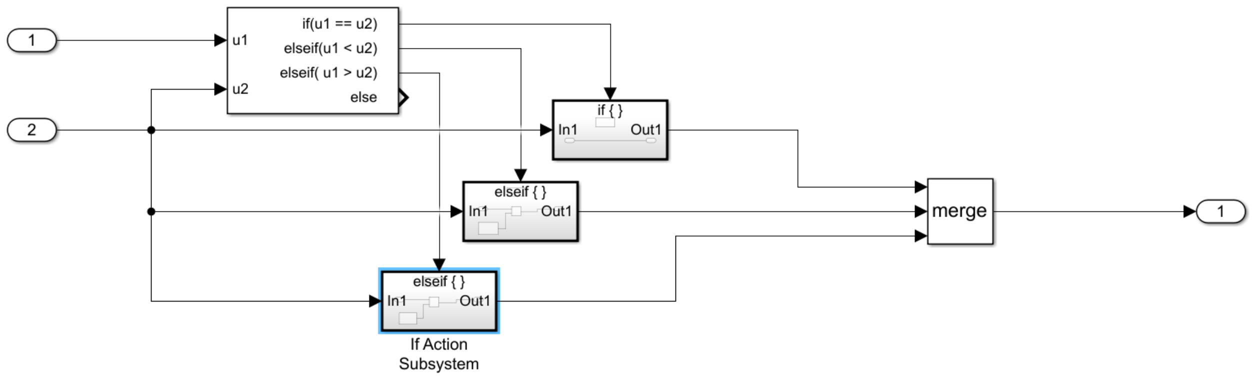


Figure 9. Internal Block Diagram for "Servo Speed" subsystem

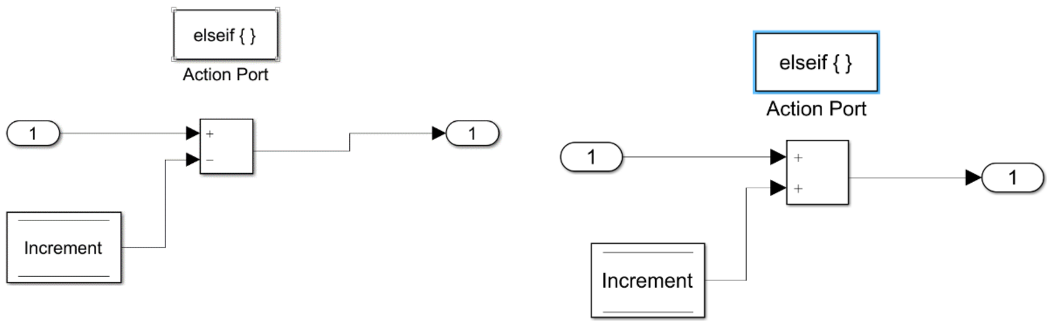


Figure 10. Internal Block Diagrams for "elseif" subsystems

One last function we can add is to create an array that stores all the data for multiple position, orientation, and claw states. Then by pressing a button we go down the array row by row and the robot arm proceeds to the next state. This is done by using a function block that outputs an array and using a mux selector block that extracts the relevant data from the right column for calculations. For the claw servo a value of 10 means the claw is open and a value of 5 means it closes.

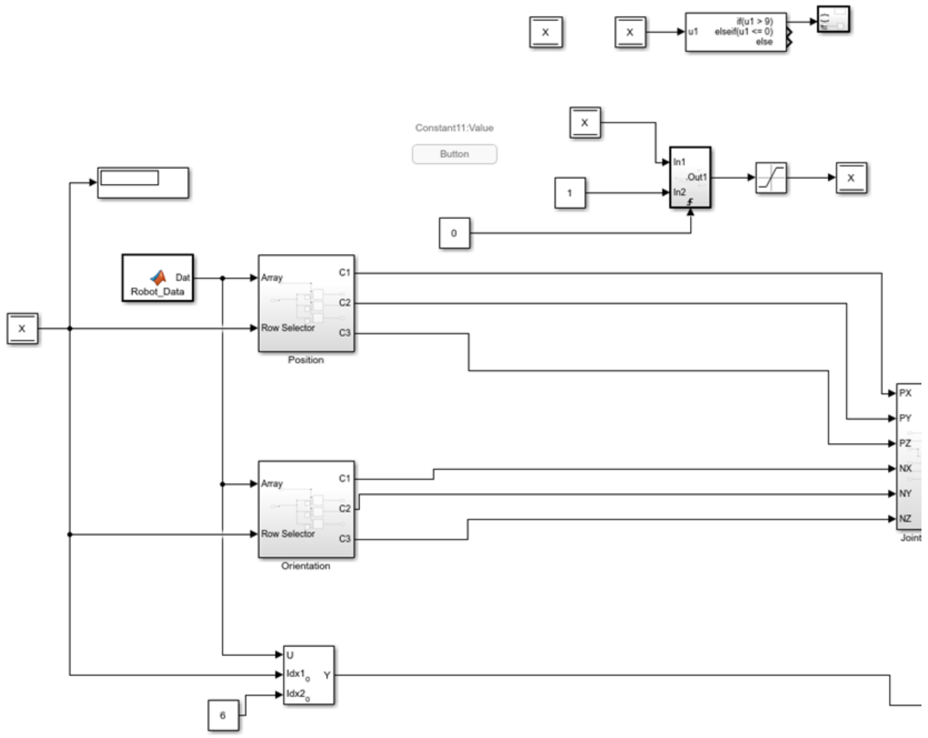
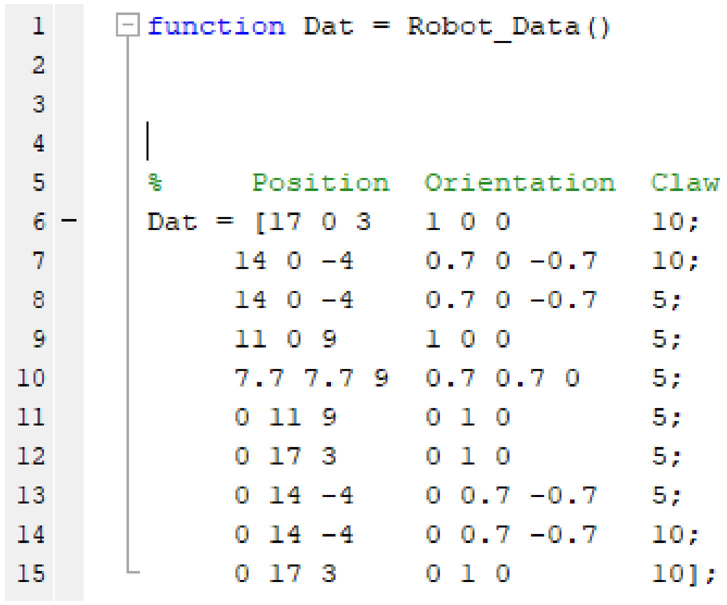


Figure 11. Block Diagram to move to the next state, subsystems contain multiple selector blocks

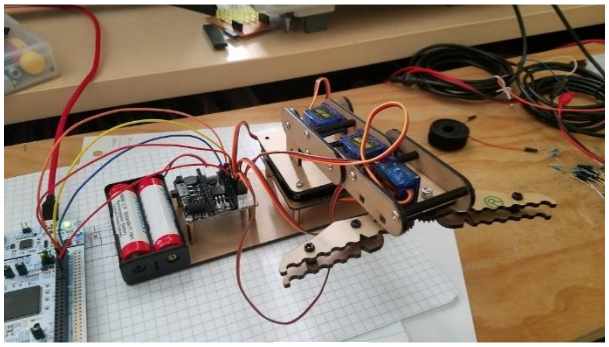


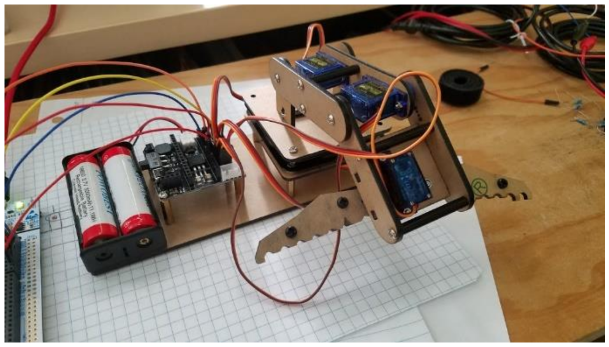
```
function Dat = Robot_Data()



%      Position   Orientation   Claw
Dat = [17 0 3   1 0 0          10;
       14 0 -4    0.7 0 -0.7    10;
       14 0 -4    0.7 0 -0.7    5;
       11 0 9     1 0 0         5;
       7.7 7.7 9  0.7 0.7 0     5;
       0 11 9     0 1 0         5;
       0 17 3     0 1 0         5;
       0 14 -4    0 0.7 -0.7    5;
       0 14 -4    0 0.7 -0.7    10;
       0 17 3     0 1 0         10];
```

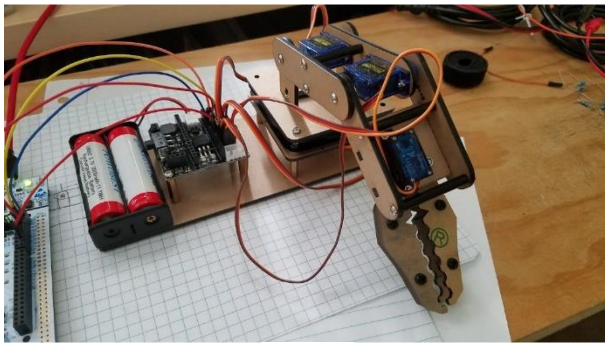
Figure 12. Function that stores values for position, orientation, and claw state as an array

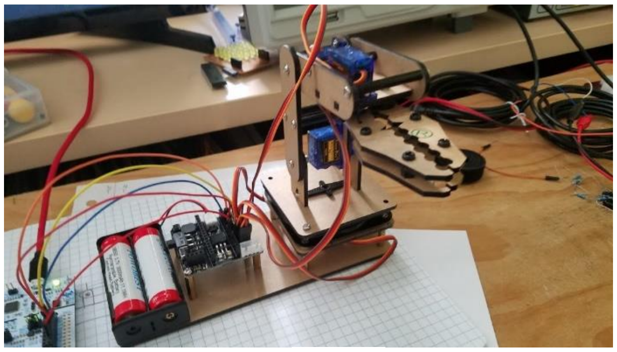
So, every time we press the button the x value increments from 0-9 then back to 0 again. When x is 0 data from the first row is extracted, when the button is pressed the x value increments to 1 and data from the second row is extracted and the process continues until the x value is reset, one thing to mention is that the selector is has designed to have an index mode of ‘Zero-Based’ instead of ‘One-Based’

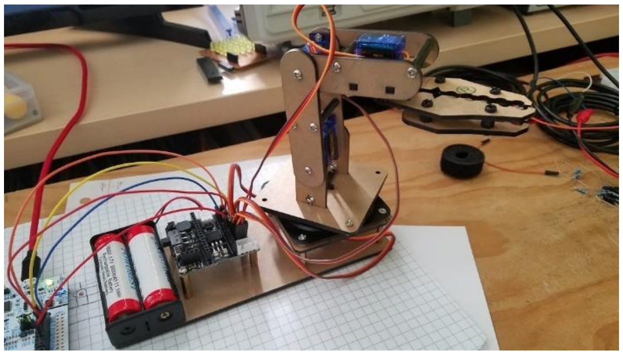
Experimentation and Results

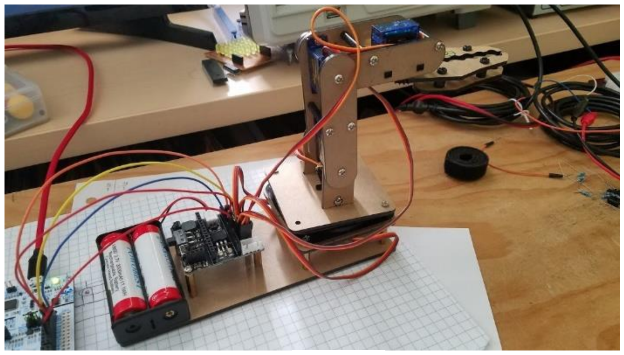

State 1

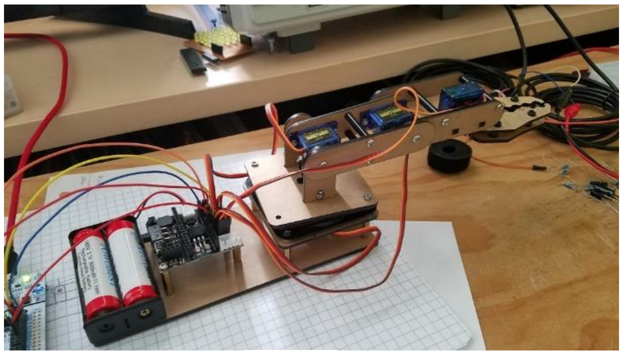

State 2

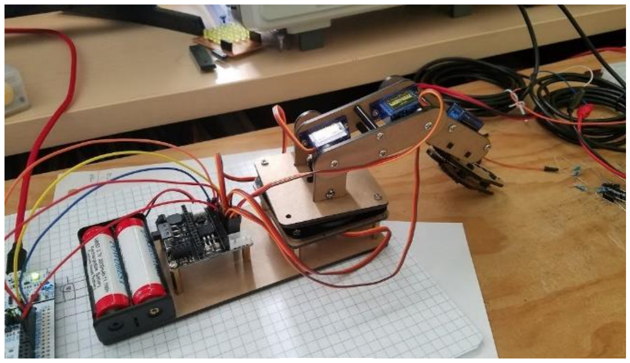

State 3

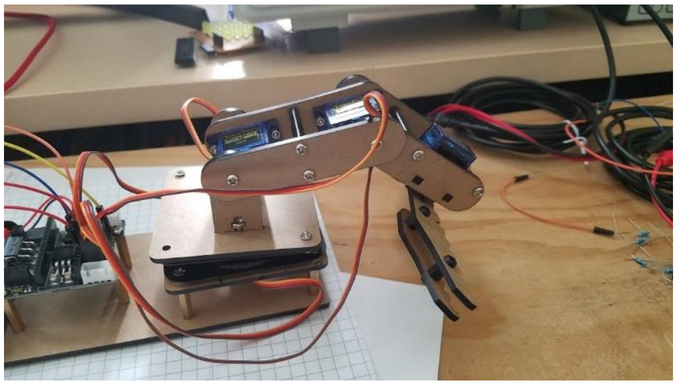

State 4

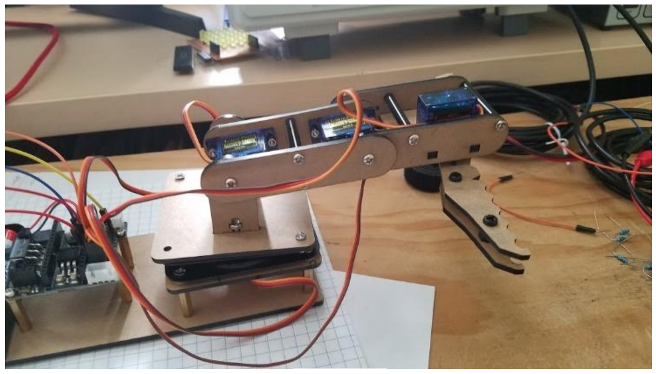

State 5


State 6


State 7


State 8


State 9


State 10

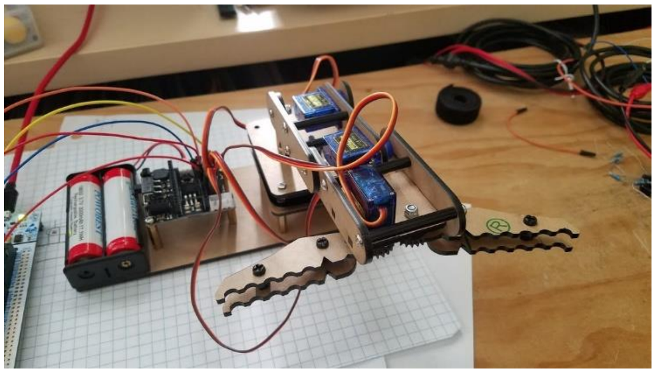

State 1

Figure 13. Pictures of arm going to each position as stored in array, goes from 1-10, back to 1

Distance of Position in centimeters									
State	Expected			Measured			Difference		
	x	y	z	x	y	z	x	y	z
1	17	0	3	17	0	2	0	0	-1
2/3	14	0	-4	14	0	-4	0	0	0
4	11	0	9	11	0	8	0	0	-1
5	7.7	7.7	9	7.5	7	8	-0.2	-0.7	-1
6	0	11	9	0	11	8	0	0	-1
7	0	17	3	0.5	17	2	0.5	0	-1
8/3	0	14	-4	0.5	14	-4	0.5	0	0

From looking at the table we can see that there are some errors, the greatest difference in error is the distance in the z-axis, looking at our robot and our model the reason is obvious, when creating the model, the offset of the claw was overlooked, the model assumes that the frame of the claw was in line with the z_2 axis. Measuring the offset between the axis and claw accounts for the one-centimeter difference. The other offsets are in the x and y direction, the reason for this error can be explained by the servo's inaccuracies since we have validated our inverse kinematic equation, looking at our picture for State 10 the arm does not line up exactly at 90°.

Conclusion

So, in conclusion we were able to fully model our robot by following the procedures for D-H and by using Matlab we were able to easily find the forward kinematic equation and thus the inverse kinematic equation. We were able to program our controller which was a Nucleo Board using MatLab Simulink and creating a block diagram representation of our program. If I were to continue with this project, I would program the Nucleo Board using the STM32 Cube software, which would mean using C coding language, even though Simulink provides a simpler way by using block diagrams, I believe Simulink uses the resources on the Nucleo Board are inefficiently which limits the controller's potential, for example when using multiple blocks with timing capabilities the model fails when running on the Nucleo Board. Using another software, I can program the resources more efficiently.

References

[1] Saeed B. Niku, “Introduction to Robotics Analysis, Control, Applications”, *Wiley*, 2011

[2] Daehyung Park, Yuuna Hoshi, Harshal P. Mahajan, Ho Keun Kim, Zackory Erickson, Wendy A. Rogers, Charles C. Kemp, “Active robot-assisted feeding with a general-purpose mobile manipulator: Design, evaluation, and lessons learned”, *Science Direct,* 2019

[3] Amogh Patwardhan, Aditya Prakash, Rajeevlochana G. Chittawadigi, “Kinematic Analysis and Development of Simulation Software for Nex Dexter Robotic Manipulator”, *Science Direct*, 2018

[4] Sheng Lin, Bi Cong Li, “A Wire-Driven Soft Manipulator Based on Flexible Curved Beam Joints”, *Science Direct*, 2020

[5] Mohammad Hosein Kazemi, Mohammad Bagher Abolhasani Jabali, “State-feedback control of robot manipulators using polytopic LPV modelling with fuzzy-clustering”, *Science Direct*, 2018

[6] Xuemei Liua, Chengrong Qiu, Qingfei Zeng, Aiping Li, “Kinematics Analysis and Trajectory Planning of collaborative welding robot with multiple manipulators”, *Science Direct,* 2019

[7] A. Rhiat, A. Aggoun, R. Lachere “Combining Mobile Robotics and Packing for Optimal deliveries”, *Science Direct,* 2019

[8] Sudarshan K.Valluru, Madhusudan Singh, “Optimization Strategy of Bio-Inspired Metaheuristic Algorithms Tuned PID Controller for PMBDC Actuated Robotic Manipulator”, *Science Direct*, 2020