



UNIVERSIDAD NACIONAL
AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Robótica

Open-Puma UNAM

Profesor: Erick Peña Medina
Semestre 2024 – 2
Grupo: 01

Alumnos:

Colima Flores Mauricio Gibran
Pascual Hernández Daniel
Alfonso
Rivera Moreno Javier
Vilchis Torres Itzel





Contenido

INTRODUCCIÓN	3
OBJETIVO	4
RUBRICA	4
EVALUACIÓN.....	4
DESARROLLO	4
LISTA DE PARTES	5
BASE DEL ROBOT	11
• Comprar a base:	11
• Rediseñar la base:	12
• Modelo CAD obtenido por terceros:	12
CONTROL DEL ROBOT	17
Tipos de movimiento.....	17
Driver de los servomotores	18
Conexión de los servomotores.....	19
Desarrollo del código	21
Restricción del movimiento de los servomotores	29
MANUFACTURA Y ENSAMBLE DE LAS PIEZAS	32
CONCLUSIÓN.....	39
COMENTARIOS	39
REFERENCIAS	40



INTRODUCCIÓN

A lo largo del curso de Robótica se adquirieron herramientas que son de gran utilidad para analizar, diseñar y controlar un robot serial, desde los análisis de posición, hasta el cálculo de los modelos cinemáticos directo e inverso, todos estos indispensables para describir a nuestro objeto de estudio.

Sin embargo, bajo la premisa de que, como futuros profesionistas en el campo de la ingeniería, no sólo es necesario el conocimiento de un área, sino también la capacidad de implementarlo, es que un proyecto como Open-Puma se vuelve fundamental para nuestra formación ya que ofrece distintos retos y aprendizajes que amplían nuestra visión y complementan nuestro perfil académico.

Open-Puma es un proyecto en el que se busca diseñar, analizar, construir y evaluar un robot serial con 5 grados de libertad. Este robot en el que se trabajó está basado en el OpenMANIPULATOR-X [1], el cual fue modificado para la implementación del motor AX-18A con su controlador DYNAMIXEL shield.

Este proyecto fue trabajado previamente por alumnos del curso de robótica impartido en el semestre 2024-I, en donde se realizaron modificaciones de las piezas de ensamble, los análisis cinemáticos y dinámicos del robot, así como la documentación requerida para la implementación del controlador DYNAMIXEL shield.

En el caso de este equipo, se seguirá trabajando con las etapas de la manufactura, ensamble, control y evaluación del proyecto propuesto anteriormente, no sin antes realizar una evaluación general de la condición actual del proyecto para realizar ajustes o modificaciones en caso de ser necesarias con respecto al trabajo realizado el curso anterior.



OBJETIVO

- Hacer una revisión de los componentes con los que se cuentan para la construcción del robot.
- Realizar la manufactura de las piezas del robot y realizar pruebas de ensamble.
- Rediseñar la base del robot para que el robot sea estable y garantice el correcto funcionamiento de su actuador.
- Trabajar en una interfaz de control de los motores en Matlab para que el robot realice una tarea específica.

RUBRICA

La evaluación del siguiente proyecto será definida por el profesor con respecto a los avances entregados en este reporte y documentos agregados al apartado de GitHub.

EVALUACIÓN

Se deben desarrollar los siguientes puntos:

1. Lista de partes.
2. Base del robot.
3. Control del robot.
4. Manufactura y ensamble de las piezas.

DESARROLLO



LISTA DE PARTES

Para el desarrollo de esta parte se contempló realizar una lista de partes, la cual nos diera una guía de todos los elementos que se mandarían a imprimir. En primera instancia, se pensaba partir del trabajo realizado anteriormente y considerar las partes creadas por el equipo que trabajó en este proyecto durante el semestre 2024-I. De tal manera que, nuestra primera lista de componentes se constituyó de la siguiente manera:




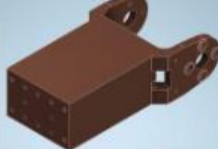
Parte	Imagen	Descripción	# de partes a imprimir
FP04-F2		Parte de la articulación	1
FP04-F3		Parte de la base	3
FP04-F2_3		Parte del brazo	1
FP04-F2_1		Parte del antebrazo	1

Tabla 1 Lista de partes versión 1

A primera vista el ensamble del robot parecía tener todo en orden, sin embargo, la brigada tuvo la observación por parte del profesor de la falta de métodos de unión entre las piezas y los servomotores, por lo que se tuvo que realizar un análisis tanto de los eslabones como del ensamble para ver si estas eran adecuadas para su implementación o era necesario realizar ajustes o rediseños en todos los componentes propuestos.



Figura 1. Vista del ensamble primera versión

Después del análisis del ensamble de esta primera lista de componentes generados en el semestre anterior 2024-I, se propuso una nueva lista de componentes, realizando modificaciones a las piezas tanto en su geometría, como añadiendo elementos que permitieran unir el eslabón con el servomotor. Es así que la segunda versión de lista de componentes contó con estas modificaciones además de un caso en particular en donde se optó por sustituir la pieza “FP04-F2_3” por una segunda pieza “FP04-F2_1”, además se eliminó la pieza “FP04-F3” y crear otra que cumpliera con su misma función a partir de modificaciones de “FP04-F2”.

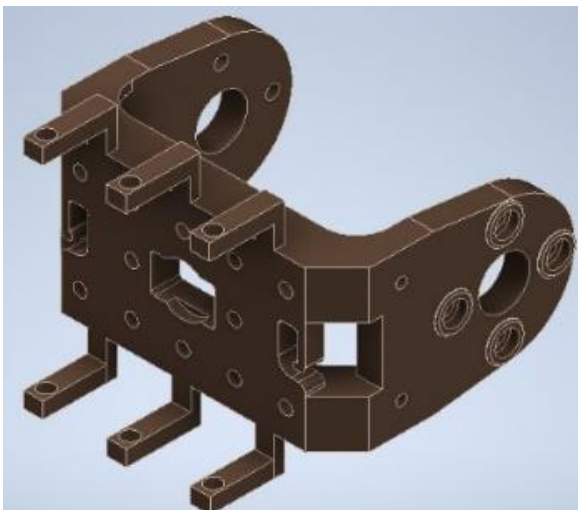


Figura 2. Modificación 1 FP04-F2



Figura 3. Modificación 2 FP04-F2

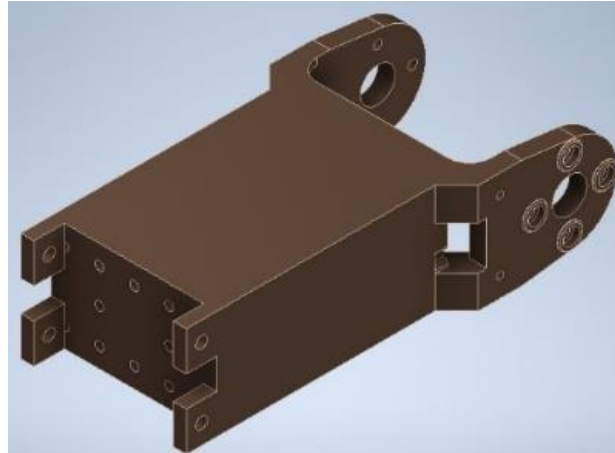


Figura 4. Modificación FP04-F2_1

Con estas modificaciones se realizó la segunda versión de la lista de componentes, en esta se un método de unión roscada, esta lista también presentó errores debido a parámetros que no fueron tomados en cuenta tales como las dimensiones del servomotor o la distancia entre los barrenos de sus guías, por lo que se procedió a realizar nuevamente modificaciones a las piezas.

Parte	Imagen	Descripción	# de partes a imprimir
FP04-F2_1		Partes constitutivas del brazo	2
FP04-F2_2		Unión del motor en control del gripper	1
FP04-F2_3		Unión del motor de la base con el primer motor del brazo	1

Tabla 2 Lista de componentes versión 2

En esta iteración, se renombraron las piezas para poder ubicarlas de mejor manera. Las correcciones que se tuvieron que hacer fueron sobre las piezas “FP04-F2_3” este rediseño se puede apreciar a continuación.



Figura 5. Modificación FP04-F2_3

La tercera versión de la lista de componentes quedó de la siguiente manera, sin embargo nuevamente se tuvieron que hacer ajustes en las piezas “FP04-F2_3” y “FP04-F2_2”

Parte	Imagen	Descripción	# de partes a imprimir
FP04-F2_1		Partes constitutivas del brazo	2
FP04-F2_2		Unión del motor en control del <u>gripper</u>	1
FP04-F2_3		Unión del motor de la base con el primer motor del brazo	1

Tabla 3. Lista de componentes versión 3

Se realizaron modificaciones a las piezas mencionadas anteriormente y también se volvieron a renombrar las piezas, esto con el objetivo no sólo de identificarlas sino también con la intención de evidenciar las iteraciones de diseño que se tuvieron que aplicar a cada pieza que fungirá como eslabón dentro del robot. Es así que las piezas “FP04-F2_1”, “FP04-F2_2” y “FP04-F2_3” serán renombradas como “FP04_2V1”, “FP04_3V2” y “FP04_1V3” respectivamente, en donde la terminación “Vx” significa versión seguida por el número que corresponde a las iteraciones que se implementaron en el rediseño de la pieza.



Figura 6. Pieza FP04_3V2



Figura 7. Pieza FP04_1V3

Antes de dar por finalizada esta etapa es necesario realizar una prueba de colisiones en el ensamble para poder estar seguros de que estas piezas pueden cumplir con su función y no darán problemas a la hora de la construcción del robot, para esto nos apoyamos del software Fusion 360. Una vez que revisamos que no hay colisiones podemos pasar a la siguiente etapa del proyecto.

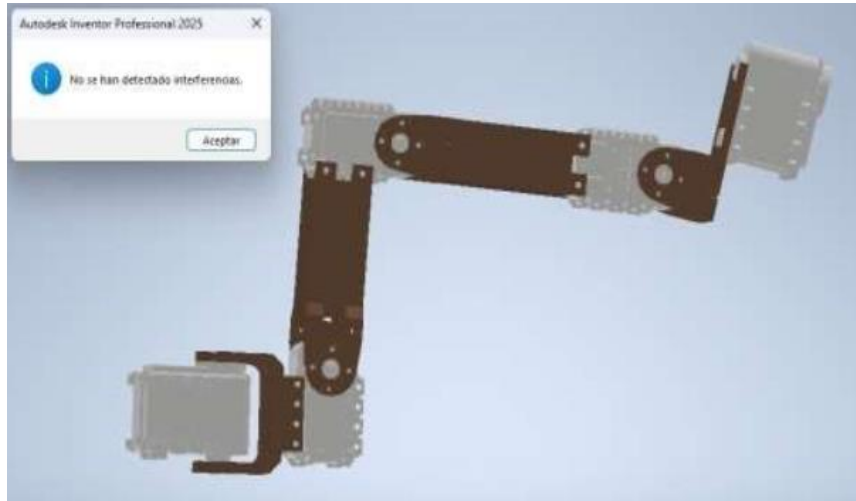


Figura 8. Prueba de colisiones del ensamble

Después de revisar que no existieran colisiones en nuestro ensamble podemos pasar a la siguiente etapa del proyecto, la versión final de la lista de componentes se muestra a continuación.




Parte	Imagen	Descripción	# de partes a imprimir
FP04_2V1		Partes constitutivas del brazo	2
FP04_3V2		Unión del motor en control del gripper	1
FP04_1V3		Unión del motor de la base con el primer motor del brazo	1

Tabla 4. Lista de componentes versión 4

BASE DEL ROBOT

Dado que no se contaba con un diseño previo de la base se realizó la valoración de diversas opciones y así saber la opción más viable para continuar con el análisis del proyecto, para lo mismo se plantearon 3 opciones: Comprar la base, rediseñar la base, obtener un modelo CAD libre de un tercero. Tras el análisis de las 3 opciones los resultados fueron los siguientes:

- **Comprar a base:**
Se consultó con dos tiendas diferentes; Sandorobotics [2] y Génération Robots [3], sin embargo, el producto estaba fuera de stock en una de ellas y en la otra el precio era demasiado elevado, por lo que la opción de comprar la base quedó descartada.

SNAPPER ROBOT ARM



Figura 9. Base Sandorobotics agotada



Figura 10. Base Génération Robots presupuesto

- **Rediseñar la base:**
Teniendo el modelo físico que se encuentra en el laboratorio, se pueden sacar medidas para obtener un modelo en CAD y realizar un prototipo rápido con el que se puedan realizar pruebas y hacer ajustes si es necesario en cuanto a medidas o a materiales.
- **Modelo CAD obtenido por terceros:**
Se encontraron diversos modelos ya diseñados, algunos de ellos de uso libre y otro más con costo (Todos los modelos se encuentran en el documento “Revisión para base de robot open-puma” [4]), sin embargo se decidió utilizar una propuesta del modelo InterbotiX PhantomX, obtenida directamente de la página de GRABCAD COMMUNITY [5]. Esta propuesta fue la elegida por la facilidad de manufacturar la base por placas a través de la corte láser e impresión 3D.



Figura 11. Modelo robot InterbotiX PhantomX

Posteriormente de elegir el CAD se tomó el archivo del ensamble completo “bases_juntas”, ara poder obtener las distintas placas que componen la base del robot se tuvo que duplicar ese archivo e ir eliminando todos los componentes excepto la placa requerida para poder tener una sola pieza y poder trabajar de forma adecuada cada una. Este proceso se realizó para cada una de las piezas de la placa.







 base_enmedio.ipt	29/05/2024 09:24 a. m.	Pieza de Autodesk...
 base_inferior.ipt	29/05/2024 09:23 a. m.	Pieza de Autodesk...
 base_sup.ipt	29/05/2024 09:25 a. m.	Pieza de Autodesk...
 base_sup_mov.ipt	29/05/2024 09:47 a. m.	Pieza de Autodesk...
 base_sup_mov_2.ipt	29/05/2024 09:48 a. m.	Pieza de Autodesk...
 bases iuntas.ipt	29/05/2024 09:17 a. m.	Pieza de Autodesk...

Figura 12. Lista de componentes de la base

Posteriormente se realizó un análisis del ensamble, con el cual se logró determinar que las piezas serían fabricadas por medio de corte láser dado que poseen el mismo grosor de 3 [mm], siendo más fácil y rápido que imprimirlas. Las piezas que se cortan son las siguientes:

- Base_enmedio.ipt
- Base_inferior.ipt
- Base_sup.ipt
- Base_sup_mov.ipt
- Base_sup_mov_2.ipt
- Tapa_base_sup.ipt

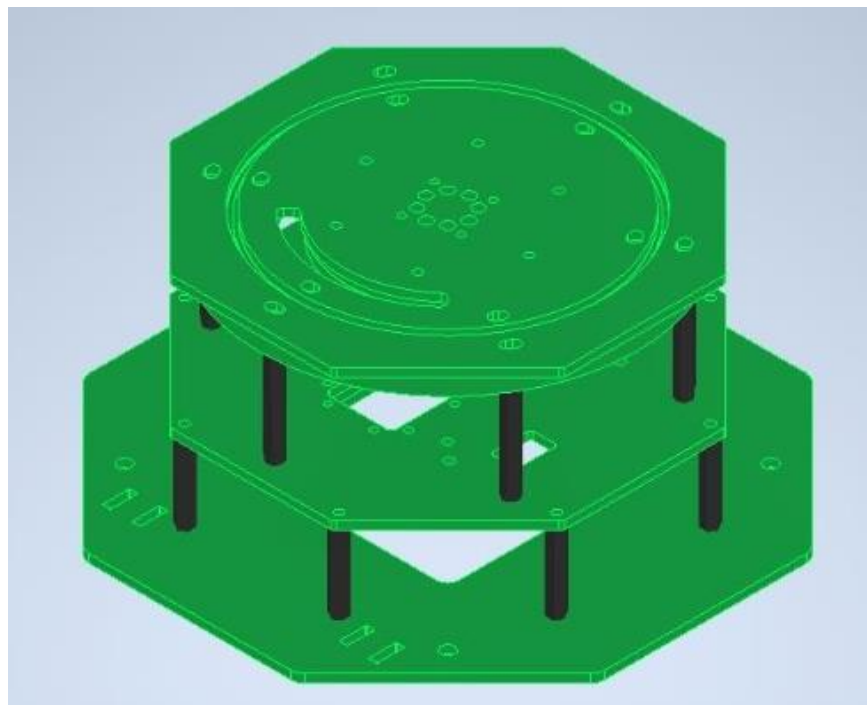


Figura 13. Ensamble de la base del robot

Para poder comenzar el proceso de corte, primero se plasmó en inventor las proyecciones de las placas y se exportaron como archivos con extensión “.dwg” en la versión 2013, este archivo generado se debe de pasar a Fusión 360 creando un nuevo proyecto y cargando el archivo para proyectar las formas en el “dibujo”, es necesario quitar el margen y cajetín para finalmente convertirlo a un archivo tipo “.dxf” que podrá ser ingresado al software de la cortadora.

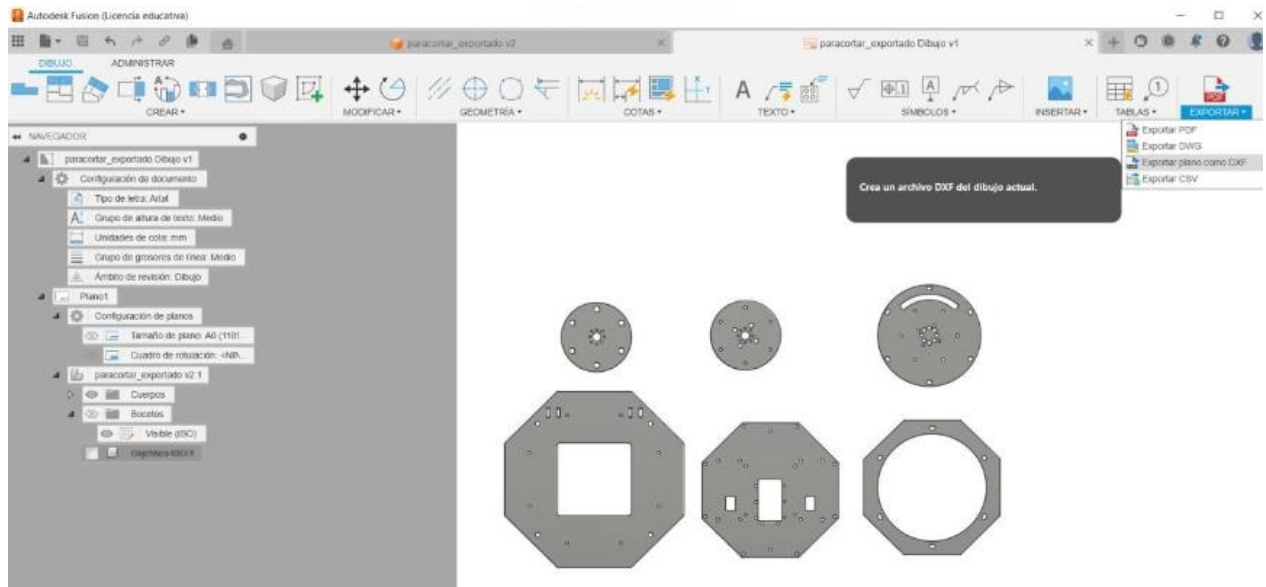


Figura 14. Manejo de la base en Fusion

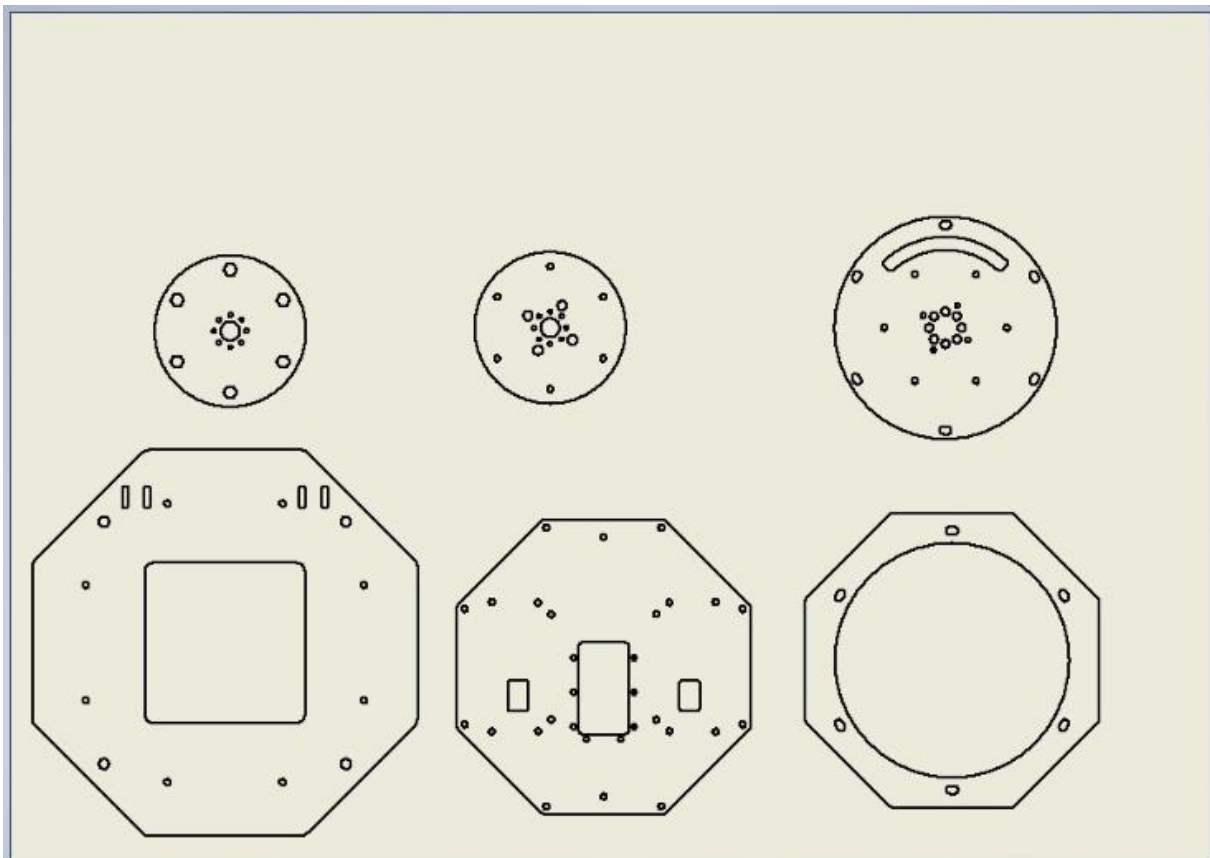


Figura 15. Proyección de las placas de la base en el plano

Durante la exportación del documento a la computadora de la cortadora, se debe verificar el mandar el archivo a la máquina, dado que el software utilizado no puede leer en primera instancia los archivos “.dxf” es necesario agregarlos desde el apartado de exportación. Una vez el archivo fue cargado se puede comenzar con el corte.

Una pieza de la base se fabricó por medio de manufactura aditiva (3d_1.ipt) para luego solo determinar el valor de los balines adecuados para su construcción. El manejo de las piezas a manufacturar por medio de impresión 3D se muestran en el apartado de manufactura y ensamble.



Figura 16. Pieza de la base impresa en 3D

Se le retiraron los soportes y dimensionalmente cumple con lo establecido. Para este caso los separadores entre las distintas placas son de 3[mm] de diámetro y los balines de la parte superior de la base son de 8[mm] de diámetro.

CONTROL DEL ROBOT

Para el control del robot tomaremos como apoyo el uso de motores, que en este caso serán los AX-18A, y los AX-12A. Estos servomotores cuentan con; una gran precisión, durabilidad y una alta facilidad de control, dichas características serán de gran beneficio para el desarrollo de nuestro robot. Recordemos que este robot trabaja a 12 [V], y cuenta con un par de torsión de 1.8 [Nm], una velocidad de hasta 97 rpm . Su rango de movimiento va desde los 0° hasta los 300° {Ref}, sin embargo, también puede operar de manera continua. [6]

Tipos de movimiento

A continuación, tenemos un par de ejemplos de las maneras mediante las cuales se puede operar el servomotor.

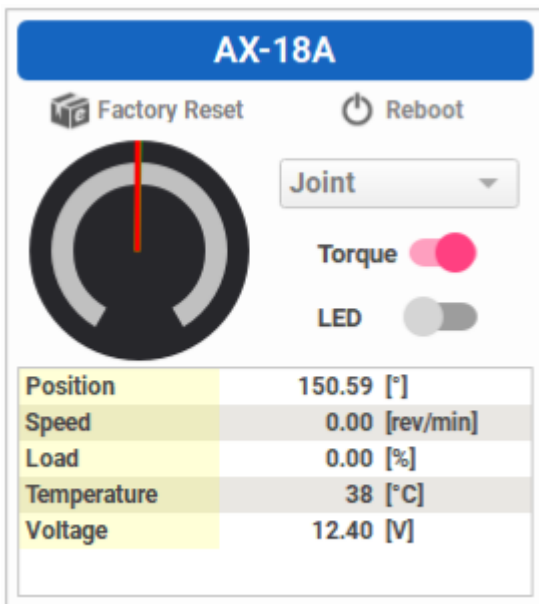


Figura 17. Movimiento controlado

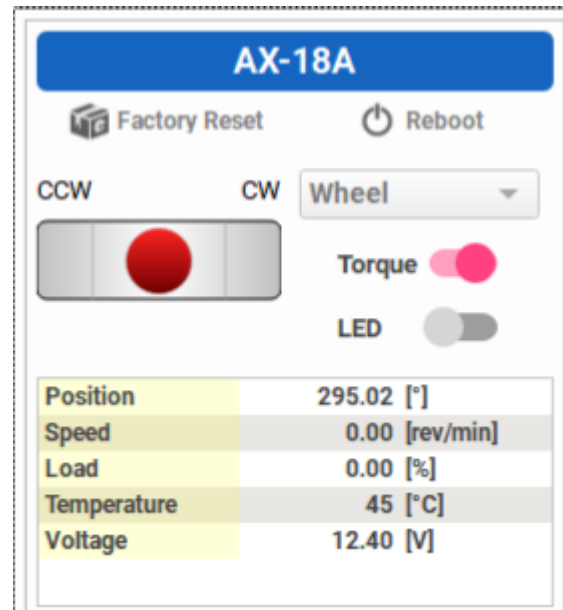


Figura 18. Movimiento de rueda

En el caso de movimiento controlado, podemos elegir la posición exacta en la que queremos que se mueva el servomotor, mientras que, para el movimiento de rueda, el servomotor semeja su comportamiento más al de una llanta en movimiento.

Driver de los servomotores

Para controlar los servomotores tomamos como apoyo DINAMIXEL shield y Wizard 2.0.

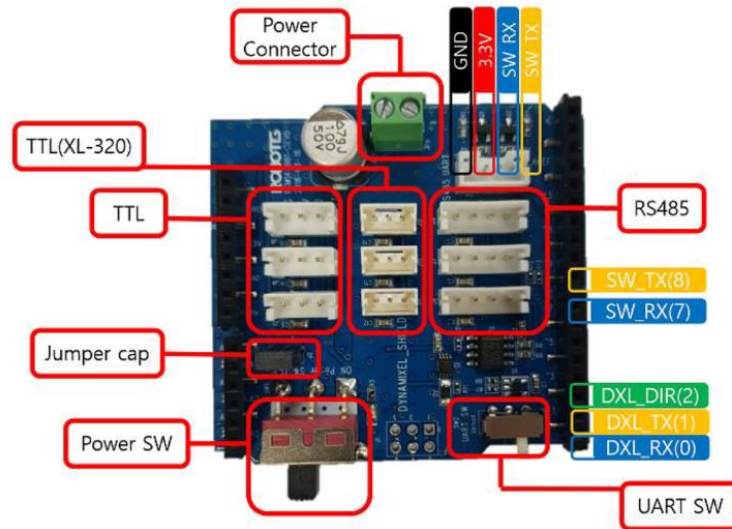


Figura 19. Placa controladora

Conectamos el servomotor al driver, a su vez esta placa estaba conectada a alimentación y a la laptop. Tal y como se puede apreciar en la siguiente ilustración:

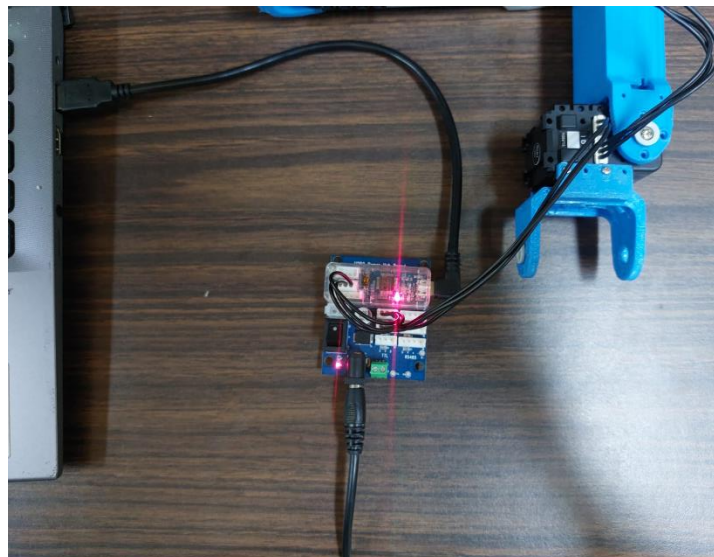


Figura 20. Conexión del servo



Una vez que el servomotor estaba conectado a la placa y a la laptop, haciendo uso de la herramienta Wizard 2.0, obtuvimos los datos necesarios. Para emplear esta herramienta era necesario buscar y seleccionar el puerto al que se encontraba conectada la placa, en nuestro caso COM5.

Address	Item	Decimal	Hex	Actual
0	Model Number	18	0x0012	AX-18A
2	Firmware Version	27	0x1B	
3	ID	1	0x01	ID 1
4	Baud Rate (Bus)	1	0x01	1M bps = 2M / (1 + 1)
5	Return Delay Time	250	0xFA	500 [psec]
6	CW Angle Limit	0	0x0000	0.00 [°]
8	CCW Angle Limit	0	0x0000	0.00 [°]
11	Temperature Limit	75	0x4B	75 [°C]
12	Min Voltage Limit	60	0x3C	6.00 [V]
13	Max Voltage Limit	140	0x8C	14.00 [V]
14	Max Torque	1023	0x03FF	100.00 [%]
16	Status Return Level	2	0x02	Return for all
17	Alarm LED	36	0x24	
18	Shutdown	36	0x24	
24	Torque Enable	1	0x01	ON
25	LED	0	0x00	OFF
26	CW Compliance Margin	1	0x01	0.29 [°]
27	CCW Compliance Margin	1	0x01	0.29 [°]
28	CW Compliance Slope	32	0x20	
29	CCW Compliance Slope	32	0x20	
30	Goal Position	830	0x033E	243.16 [°]
32	Moving Speed	0	0x0000	0.00 [rev/min]
34	Torque Limit	1023	0x03FF	100.00 [%]
36	Present Position	259	0x0103	75.88 [°]
38	Present Speed	0	0x0000	0.00 [rev/min]
40	Present Load	0	0x0000	0.00 [%]
42	Present Input Voltage	124	0x7C	12.40 [V]
43	Present Temperature	53	0x35	53 [°C]
44	Registered	0	0x00	No commands transmitted by REG_WRITE
46	Moving	0	0x00	Idle
47	Lock	0	0x00	Unlock EEPROM
48	Punch	0	0x0000	

Figura 21. Datos del servomotor

Conexión de los servomotores

De manera unitaria

Para conectar un solo servomotor era necesario seguir el diagrama de la Ilustración 17, y de esta manera obteníamos la siguiente conexión:



Figura 22. Conexión unitaria

En serie

Para poder conectar todos los servomotores que se necesitarían para el proyecto y obtener sus respectivas tablas de datos en Wizard 2.0. Fue necesario conectarlos en serie, un contratiempo contra tiempo con el cual nos enfrentamos es que los servos tienen que conectarse de una manera específica. La primera entrada va conectada a la segunda del siguiente servomotor y al finalizar con la conexión en serie, del ultimo servomotor cometamos la primera entrada a la placa.



Figura 23. Conexión en serie

De esta manera nuestro Wizard 2.0, va a registrar de un solo puerto todos los servomotores que se encuentren conectados en serie.

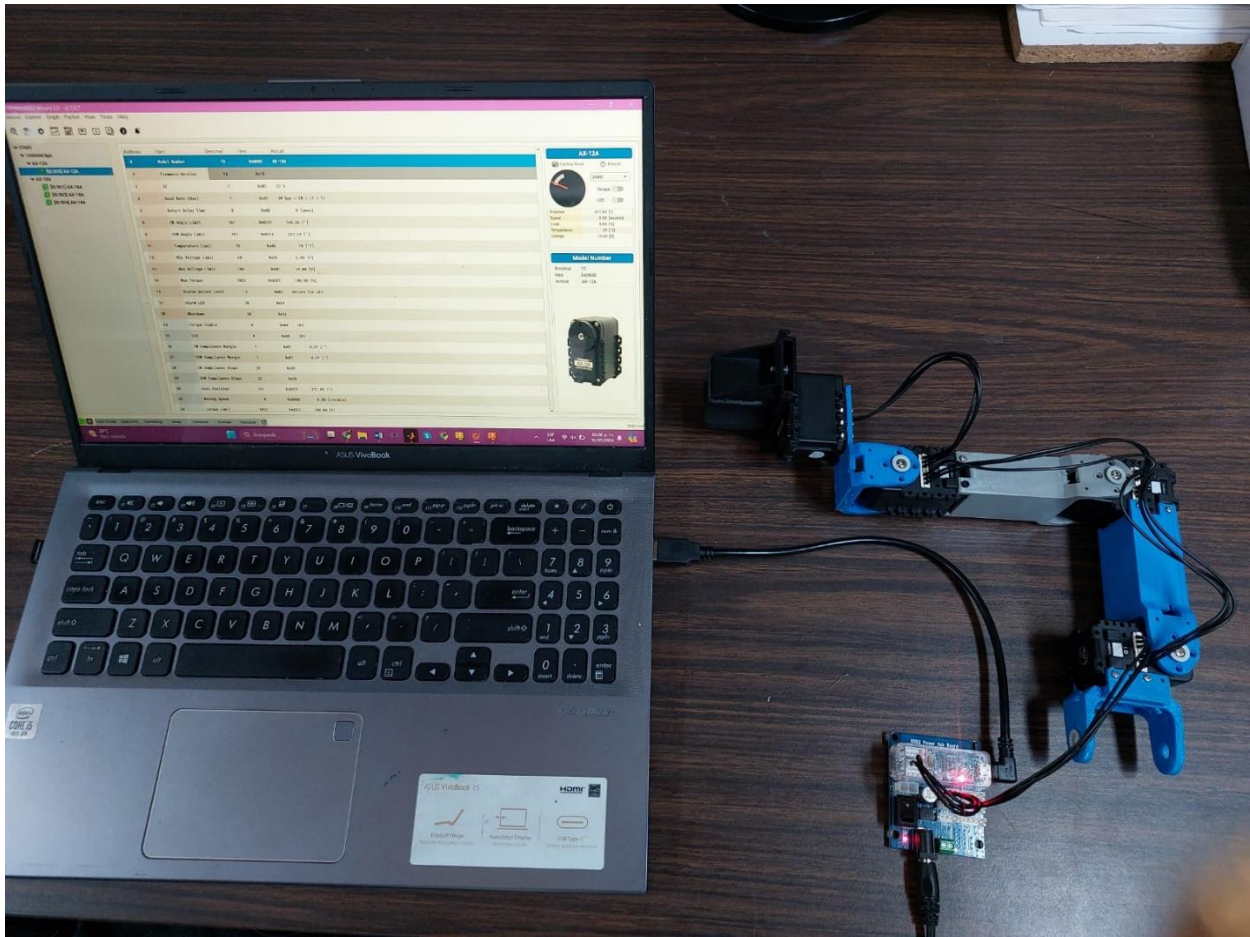


Figura 24. Registro de los 4 servomotores

Desarrollo del código

Para el desarrollo de este código tomamos como base el que se encontraba en la carpeta del GitHub, con el nombre de “DynamixelSDK-3.7.3”. Este código se trabajó en MATLAB.

Para poder acceder a archivo, dentro de la carpeta DynamixelSDK-3.7.31, teníamos una subcarpeta llamada “Matlab”, de la cual seleccionamos el protocolo 2.0. Una vez dentro abrimos el archivo “read_write”.



```
% Control table address
ADDR_PRO_TORQUE_ENABLE    = 562;      % Control table address is different in Dynamixel model
ADDR_PRO_GOAL_POSITION    = 596;
ADDR_PRO_PRESENT_POSITION = 611;

% Protocol version
PROTOCOL_VERSION          = 2.0;      % See which protocol version is used in the Dynamixel

% Default setting
DXL_ID                    = 1;         % Dynamixel ID: 1
BAUDRATE                  = 57600;
DEVICENAME                 = 'COM1';   % Check which port is being used on your controller
                                % ex) Windows: 'COM1' Linux: '/dev/ttyUSB0' Mac: '/dev/tty.usbserial-*'

TORQUE_ENABLE              = 1;        % Value for enabling the torque
TORQUE_DISABLE             = 0;        % Value for disabling the torque
DXL_MINIMUM_POSITION_VALUE = -150000; % Dynamixel will rotate between this value
DXL_MAXIMUM_POSITION_VALUE = 150000;  % and this value (note that the Dynamixel would not move when the position value is out
DXL_MOVING_STATUS_THRESHOLD = 20;     % Dynamixel moving status threshold

ESC_CHARACTER              = 'e';      % Key for escaping loop

COMM_SUCCESS               = 0;        % Communication Success result value
COMM_TX_FAIL               = -1001;    % Communication Tx Failed
```

Figura 25. Parte del código "read_write" sin editar

Comprensión del código

Para poder comprender este código fue necesario tomar como apoyo una playlist de tutoriales. Los cuales se pueden encontrar en el siguiente enlace:

https://youtube.com/playlist?list=PLEf1s0tzVSqatNEj_TMwnv86f0IuvVa9&si=d5fGBUv68ZcJ-ezf

Dentro de esta playlist, nos encontramos con el video número 12, el cual nos lleva de la mano paso a paso para entender el código y cambiar el puerto, que en este caso fue de COM1 a COM5. Sin embargo, esto no era todo lo que se le debía modificar al código, ya que los valores y datos asignados que podemos ver en la ilustración 25, debían ser modificados por los del servomotor conectado, ilustración 21.

Este código nos permitía inicializar y comunicarnos con los servomotores.

Al ser un código considerablemente grande me tomé la libertad de comentar cada una de sus líneas de código explicando a detalle que hace cada parte, tal como se puede mostrar a continuación.



```
1  klc; % Limpia la ventana de comandos.
2  clear all; % Limpia todas las variables del espacio de trabajo.
3
4  lib_name = ''; % Inicializa la variable lib_name como una cadena vacía.
5
6  % Dependiendo del sistema operativo, se asigna el nombre de la biblioteca correspondiente a lib_name.
7  if strcmp(computer, 'PCWIN')
8      lib_name = 'dxl_x86_c';
9  elseif strcmp(computer, 'PCWIN64')
10     lib_name = 'dxl_x64_c';
11  elseif strcmp(computer, 'GLNX86')
12     lib_name = 'libdxl_x86_c';
13  elseif strcmp(computer, 'GLNXA64')
14     lib_name = 'libdxl_x64_c';
15  elseif strcmp(computer, 'MACI64')
16     lib_name = 'libdxl_mac_c';
17  end
18
19 % Carga las bibliotecas si no están ya cargadas.
20 if ~libisloaded(lib_name)
21     [notfound, warnings] = loadlibrary(lib_name, 'dynamixel_sdk.h', 'addheader', 'port_handler.h', 'addheader', 'packet_handler.h');
22 end
23
24 % Versión del protocolo
25 PROTOCOL_VERSION = 2.0; % Establece la versión del protocolo a usar en el Dynamixel.
26
27 % Configuración predeterminada
28 BAUDRATE = 57600; % Establece la tasa de baudios.
29 DEVICENAME = 'COM5'; % Establece el nombre del dispositivo (puerto COM).
30
31 MAX_ID = 252; % Valor máximo de ID para los dispositivos.
32 COMM_SUCCESS = 0; % Valor que indica éxito en la comunicación.
33 COMM_TX_FAIL = -1001; % Valor que indica fallo en la transmisión de comunicación.
34
35 % Inicializa las estructuras PortHandler.
36 % Establece la ruta del puerto.
37 % Obtiene métodos y miembros de PortHandlerLinux o PortHandlerWindows.
38 port_num = portHandler(DEVICENAME);
39
40 % Inicializa las estructuras PacketHandler.
41 packetHandler();
42
43 dxl_comm_result = COMM_TX_FAIL; % Inicializa la variable de resultado de comunicación con fallo.
44
45 % Abre el puerto.
46 if (openPort(port_num))
47     fprintf('Succeeded to open the port!\n'); % Muestra un mensaje si el puerto se abre con éxito.
48 else
49     unloadlibrary(lib_name); % Descarga la biblioteca si no se puede abrir el puerto.
50     fprintf('Failed to open the port!\n'); % Muestra un mensaje si falla al abrir el puerto.
51     input('Press any key to terminate...\n'); % Espera una entrada del usuario para terminar.
52     return;
53 end
54
55 % Establece la tasa de baudios del puerto.
56 if (setBaudRate(port_num, BAUDRATE))
57     fprintf('Succeeded to change the baudrate!\n'); % Muestra un mensaje si se cambia la tasa de baudios con éxito.
58 else
59     unloadlibrary(lib_name); % Descarga la biblioteca si no se puede cambiar la tasa de baudios.
60     fprintf('Failed to change the baudrate!\n'); % Muestra un mensaje si falla al cambiar la tasa de baudios.
61     input('Press any key to terminate...\n'); % Espera una entrada del usuario para terminar.
62     return;
63 end
64
65 % Intenta hacer un ping de difusión al Dynamixel.
66 broadcastPing(port_num, PROTOCOL_VERSION);
67 dxl_comm_result = getLastRxResult(port_num, PROTOCOL_VERSION);
68 if dxl_comm_result ~= COMM_SUCCESS
69     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result)); % Muestra el resultado del ping si falla.
70 end
71
72 fprintf('Detected Dynamixel : \n'); % Muestra un mensaje indicando que se han detectado Dynamixel.
73 for id = 0 : MAX_ID
74     if getBroadcastPingResult(port_num, PROTOCOL_VERSION, id)
75         fprintf('[ID:%03d]\n', id); % Muestra el ID de los Dynamixel detectados.
76     end
77 end
78
79 % Cierra el puerto.
80 closePort(port_num);
81
82 % Descarga la biblioteca.
83 unloadlibrary(lib_name);
84
85 close all; % Cierra todas las figuras.
86 clear all; % Limpia todas las variables del espacio de trabajo.
87
```

Figura 26. Código explicado



Primera iteración

Como se mencionó anteriormente, no bastaba con cambiar solo el puerto, sino que también fue necesario cambiar todos los datos y valores correspondientes:

```
% Control table address
ADDR_PRO_TORQUE_ENABLE   = 562;      % Control table address is different in Dynamixel model
ADDR_PRO_GOAL_POSITION   = 596;      %596
ADDR_PRO_PRESENT_POSITION = 611;

% Protocol version
PROTOCOL_VERSION         = 2.0;      % See which protocol version is used in the Dynamixel

% Default setting
DXL_ID                   = 1;         % Dynamixel ID: 1
BAUDRATE                 = 57600;

DEVICENAME                = 'COM5';   % Check which port is being used on your controller
                                % ex) Windows: 'COM1'   Linux: '/dev/ttyUSB0' Mac: '/dev/tty.usbserial-*'

TORQUE_ENABLE             = 1;        % Value for enabling the torque
TORQUE_DISABLE            = 0;        % Value for disabling the torque
DXL_MINIMUM_POSITION_VALUE = 1;       % Dynamixel will rotate between this value
DXL_MAXIMUM_POSITION_VALUE = 4095;    % and this value (note that the Dynamixel would not move when the position value is out o
DXL_MOVING_STATUS_THRESHOLD = 20;     % Dynamixel moving status threshold 20

ESC_CHARACTER             = 'e';      % Key for escaping loop

COMM_SUCCESS              = 0;        % Communication Success result value
COMM_TX_FAIL              = -1001;    % Communication Tx Failed
```

Figura 27. Primera iteración

Fue en esta parte en donde se presentaron más problemas. El primero de nuestros problemas fue que para poder ejecutar el código en MATLAB era necesario desconectar el servomotor del Wizard 2.0. El segundo problema se presentó porque al ejecutar una vez el archivo en MATLAB, este no se podía ejecutar directamente por segunda vez, sino que requería reiniciar el entorno de programación. Debido a estos detalles era muy tardado poder realizar las pruebas correspondientes y calibrar los servomotores.

Es necesario mencionar que estos problemas no fueron los únicos que se presentaron, ya que el código, no permitía cambiar posición en la que se encontraba el servomotor, sino que tomaba directamente la posición que se encontraba escrita en las variables DXL_MINIMUM_POSITION_VALUE y en DXL_MAXIMUM_POSITION_VALUE.

Además, no movía el servomotor.

Segunda iteración

Para la segunda iteración se agregó una interacción con el usuario en la interfaz, esto para poder solicitar una posición específica que entrara dentro del rango de movimiento de nuestro servomotor, el cual se encontraba entre 2 y 1023:

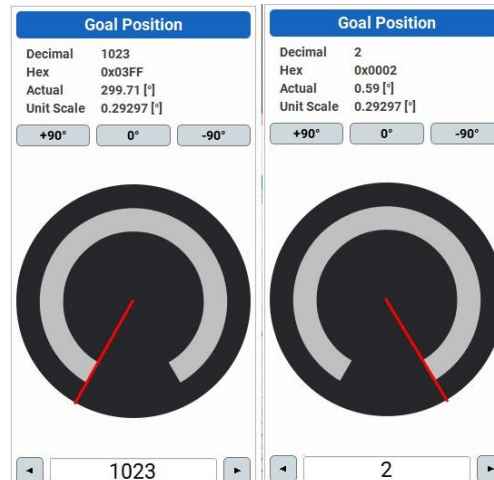


Ilustración 28. Rango de movimiento

```
% Control table address
ADDR_PRO_TORQUE_ENABLE      = 1023;
ADDR_PRO_GOAL_POSITION      = 548;
ADDR_PRO_PRESENT_POSITION   = 521;

% Protocol version
PROTOCOL_VERSION            = 2.0;

% Default setting
DXL_ID                      = 5;
BAUDRATE                    = 57600;
DEVICENAME                   = 'COM5';

TORQUE_ENABLE                = 1;
TORQUE_DISABLE               = 0;
DXL_MINIMUM_POSITION_VALUE   = 2;
DXL_MAXIMUM_POSITION_VALUE   = 1023;
DXL_MOVING_STATUS_THRESHOLD = 20;

ESC_CHARACTER                 = 'e';

COMM_SUCCESS                  = 0;
COMM_TX_FAIL                  = -1001;
```

Figura 29. Cambios para segunda iteración

Además, para este código se agregaron líneas que permitieran interactuar directamente con el usuario y darle la información del rango en el que se puede mover el servo:



```
1  clc;
2  clear all;
3
4  lib_name = '';
5
6  if strcmp(computer, 'PCWIN')
7      lib_name = 'dxl_x86_c';
8  elseif strcmp(computer, 'PCWIN64')
9      lib_name = 'dxl_x64_c';
10 elseif strcmp(computer, 'GLNX86')
11     lib_name = 'libdxl_x86_c';
12 elseif strcmp(computer, 'GLNXA64')
13     lib_name = 'libdxl_x64_c';
14 elseif strcmp(computer, 'MACI64')
15     lib_name = 'libdxl_mac_c';
16 end
17
18 % Load Libraries
19 if ~libisloaded(lib_name)
20     [notfound, warnings] = loadlibrary(lib_name, 'dynamixel_sdk.h', 'addheader', 'port_handler.h', 'addheader', 'packet_handler.h')
21 end
22
23 % Control table address
24 ADDR_PRO_TORQUE_ENABLE      = 1023;
25 ADDR_PRO_GOAL_POSITION     = 548;
26 ADDR_PRO_PRESENT_POSITION  = 521;
27
28 % Protocol version
29 PROTOCOL_VERSION           = 2.0;
30
31 % Default setting
32 DXL_ID                      = 5;
33 BAUDRATE                    = 57600;
34 DEVICENAME                  = 'COM5';
35
36 TORQUE_ENABLE               = 1;
37 TORQUE_DISABLE              = 0;
38 DXL_MINIMUM_POSITION_VALUE  = 2;
39 DXL_MAXIMUM_POSITION_VALUE  = 1023;
40 DXL_MOVING_STATUS_THRESHOLD = 20;
41
42 ESC_CHARACTER                = 'e';
43
44 COMM_SUCCESS                 = 0;
45 COMM_TX_FAIL                 = -1001;
46
47 % Initialize PortHandler Structs
48 port_num = portHandler(DEVICENAME);
49
50 % Initialize PacketHandler Structs
51 packetHandler();
52
53 dxl_comm_result = COMM_TX_FAIL;
54 dxl_goal_position = 0;
55
56 dxl_error = 0;
57 dxl_present_position = 0;
58
59 % Open port
60 if (openPort(port_num))
61     fprintf('Succeeded to open the port!\n');
62 else
63     unloadlibrary(lib_name);
64     fprintf('Failed to open the port!\n');
65     input('Press any key to terminate...\n');
66     return;
67 end
68
69 % Set port baudrate
70 if (setBaudRate(port_num, BAUDRATE))
71     fprintf('Succeeded to change the baudrate!\n');
72 else
73     unloadlibrary(lib_name);
74     fprintf('Failed to change the baudrate!\n');
75     input('Press any key to terminate...\n');
76     return;
77 end
78
79 % Enable Dynamixel Torque
80 writeByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID, ADDR_PRO_TORQUE_ENABLE, TORQUE_ENABLE);
81 dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
82 dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
83 if dxl_comm_result ~= COMM_SUCCESS
84     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
85 elseif dxl_error ~= 0
86     fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
87 else
88     fprintf('Dynamixel has been successfully connected \n');
89 end
90
```



```
91 while 1
92     user_input = input('Enter goal position (2-1023) or e to quit: ', 's');
93     if user_input == ESC_CHARACTER
94         break;
95     end
96
97     dxl_goal_position = str2double(user_input);
98
99     if isnan(dxl_goal_position) || dxl_goal_position < DXL_MINIMUM_POSITION_VALUE || dxl_goal_position > DXL_MAXIMUM_POSITION_VALUE
100         fprintf('Invalid position value. Please enter a value between 2 and 1023.\n');
101         continue;
102     end
103
104     % Write goal position
105     writeByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID, ADDR_PRO_GOAL_POSITION, typecast(int32(dxl_goal_position), 'uint32'));
106     dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
107     dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
108     if dxl_comm_result ~= COMM_SUCCESS
109         fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
110     elseif dxl_error ~= 0
111         fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
112     end
113
114     while 1
115         % Read present position
116         dxl_present_position = read4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID, ADDR_PRO_PRESENT_POSITION);
117         dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
118         dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
119         if dxl_comm_result ~= COMM_SUCCESS
120             fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
121         elseif dxl_error ~= 0
122             fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
123         end
124
125         fprintf('[ID:%03d] GoalPos:%03d PresPos:%03d\n', DXL_ID, dxl_goal_position, typecast(uint32(dxl_present_position), 'int32'));
126
127         if ~(abs(dxl_goal_position - typecast(uint32(dxl_present_position), 'int32')) > DXL_MOVING_STATUS_THRESHOLD)
128             break;
129         end
130     end
131 end
132
133 % Disable Dynamixel Torque
134 writeByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID, ADDR_PRO_TORQUE_ENABLE, TORQUE_DISABLE);
135 dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
136 dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
137 if dxl_comm_result ~= COMM_SUCCESS
138     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
139 elseif dxl_error ~= 0
140     fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
141 end
142
143 % Close port
144 closePort(port_num);
145
146 % Unload Library
147 unloadLibrary(lib_name);
148
149 close all;
150 clear all;
```

Figura 30. Código de la segunda iteración

Pese a que se comprobó la funcionalidad del código, seguía sin poder mover el servomotor. Conectaba el código con la posición del servomotor, sin embargo, no cambiaba a este de posición.

Para intentar solucionarlo se probó con varios rangos, ya que nuestra teoría suponía que la conexión no era posible porque el rango no era correcto. Pero aun cambiando el rango, el servomotor seguía sin moverse.

Tercera iteración

En la tercera iteración volví a modificar el código para que este pudiera comunicarse con 2 servomotores al mismo tiempo. Con el propósito de no tener que volver a hacer todo el código otra vez y adaptarlo a la última versión de servomotores se optó por trabajar con la misma versión de servos (AX-18A).



```
1  clc; % Limpia la ventana de comandos.
2  clear all; % Limpia todas las variables del espacio de trabajo.
3
4  lib_name = ''; % Inicializa la variable lib_name como una cadena vacía.
5
6  % Dependiendo del sistema operativo, se asigna el nombre de la biblioteca correspondiente a lib_name.
7  if strcmp(computer, 'PCWIN')
8      lib_name = 'dxl_x86_c';
9  elseif strcmp(computer, 'PCWIN64')
10     lib_name = 'dxl_x64_c';
11  elseif strcmp(computer, 'GLIX86')
12     lib_name = 'libdxl_x86_c';
13  elseif strcmp(computer, 'GLIX64')
14     lib_name = 'libdxl_x64_c';
15  elseif strcmp(computer, 'MACI64')
16     lib_name = 'libdxl_mac_c';
17  end
18
19  % Carga las bibliotecas si no están ya cargadas.
20  if ~libisloaded(lib_name)
21      [notfound, warnings] = loadlibrary(lib_name, 'dynamixel_sdk.h', 'addheader', 'port_handler.h', 'addheader', 'packet_handler.h')
22  end
23
24  % Versión del protocolo
25  PROTOCOL_VERSION = 2.0; % Establece la versión del protocolo a usar en el Dynamixel.
26
27  % Configuración predeterminada
28  BAUDRATE = 57600; % Establece la tasa de baudios.
29  DEVICENAME = 'COM1'; % Establece el nombre del dispositivo (puerto COM).
30
31  MAX_ID = 252; % Valor máximo de ID para los dispositivos.
32  COMM_SUCCESS = 0; % Valor que indica éxito en la comunicación.
33  COMM_TX_FAIL = -1001; % Valor que indica fallo en la transmisión de comunicación.
34
35  % IDs de los motores
36  DXL1_ID = 1; % ID del primer motor
37  DXL2_ID = 2; % ID del segundo motor
38
39  % Inicializa las estructuras PortHandler.
40  % Establece la ruta del puerto.
41  % Obtiene métodos y miembros de PortHandlerLinux o PortHandlerWindows.
42  port_num = portHandler(DEVICENAME);
43
44  % Inicializa las estructuras PacketHandler.
45  packetHandler();
46
47  dxl_comm_result = COMM_TX_FAIL; % Inicializa la variable de resultado de comunicación con fallo.
48
49  % Abre el puerto.
50  if (openPort(port_num))
51      fprintf('Succeeded to open the port!\n'); % Muestra un mensaje si el puerto se abre con éxito.
52  else
53      unloadlibrary(lib_name); % Descarga la biblioteca si no se puede abrir el puerto.
54      fprintf('Failed to open the port!\n'); % Muestra un mensaje si falla al abrir el puerto.
55      input('Press any key to terminate...\n'); % Espera una entrada del usuario para terminar.
56      return;
57  end
58
59  % Establece la tasa de baudios del puerto.
60  if (setBaudRate(port_num, BAUDRATE))
61      fprintf('Succeeded to change the baudrate!\n'); % Muestra un mensaje si se cambia la tasa de baudios con éxito.
62  else
63      unloadlibrary(lib_name); % Descarga la biblioteca si no se puede cambiar la tasa de baudios.
64      fprintf('Failed to change the baudrate!\n'); % Muestra un mensaje si falla al cambiar la tasa de baudios.
65      input('Press any key to terminate...\n'); % Espera una entrada del usuario para terminar.
66      return;
67  end
68
69  % Intenta hacer un ping a los dos motores.
70  for dxl_id = [DXL1_ID, DXL2_ID]
71      ping(port_num, PROTOCOL_VERSION, dxl_id);
72      dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
73      if dxl_comm_result ~= COMM_SUCCESS
74          fprintf('Motor ID %d: %s\n', dxl_id, getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
75      else
76          fprintf('Motor ID %d ping succeeded.\n', dxl_id);
77      end
78  end
79
80  % Intenta hacer un ping de difusión al Dynamixel.
81  broadcastPing(port_num, PROTOCOL_VERSION);
82  dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
83  if dxl_comm_result ~= COMM_SUCCESS
84      fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result)); % Muestra el resultado del ping si falla.
85  end
86  |
87  fprintf('Detected Dynamixel : \n'); % Muestra un mensaje indicando que se han detectado Dynamixel.
88  for id = 0 : MAX_ID
89      if getBroadcastPingResult(port_num, PROTOCOL_VERSION, id)
```



```
90     fprintf('[ID:%03d]\n', id); % Muestra el ID de los Dynamixel detectados.
91 end
92 end
93
94 % Cierra el puerto.
95 closePort(port_num);
96
97 % Descarga la biblioteca.
98 unloadlibrary(lib_name);
99
100 close all; % Cierra todas las figuras.
101 clear all; % Limpia todas las variables del espacio de trabajo.
102 He añadido las siguientes líneas clave para trabajar con dos motores:
103
104 IDs de los motores:
105
106 matlab
107 Copiar código
108 DXL1_ID = 1; % ID del primer motor
109 DXL2_ID = 2; % ID del segundo motor
```

Figura 31. Código de la tercera iteración

Aunque se intentó de muchas maneras que la interfaz de MATLAB lograra mover los servomotores esto no fue posible. Además, la plataforma complicó mucho las pruebas, ya que para cada intento se tenía que reiniciar todo, de lo contrario no se ejecutaba el código.

Por las iteraciones realizadas puedo sugerir que es mejor cambiar la plataforma MATLAB, por una que sea mucho más fácil de usar; como lo es Arduino o en su defecto ROS que, aunque requiera de buenos conocimiento en Linux puede llegar a ser mucho más rápido que MATLAB.

Restricción del movimiento de los servomotores

Para el movimiento de Open-Puma, fue necesario establecer un rango de posiciones entre cada servomotor que articulaba la pieza. Esto es fundamental debido a que sin estas restricciones los componentes pueden llegar a chocar entre sí o a forzar demasiado la pinza y romperla.

Por ello desde Wizard 2.0, asignamos las restricciones a cada uno de los servomotores, de acuerdo a los movimientos que debía hacer cada uno de ellos:

Restricción del segundo motor a la base:

Cuenta con una posición máxima de 211 y una posición mínima de 638:

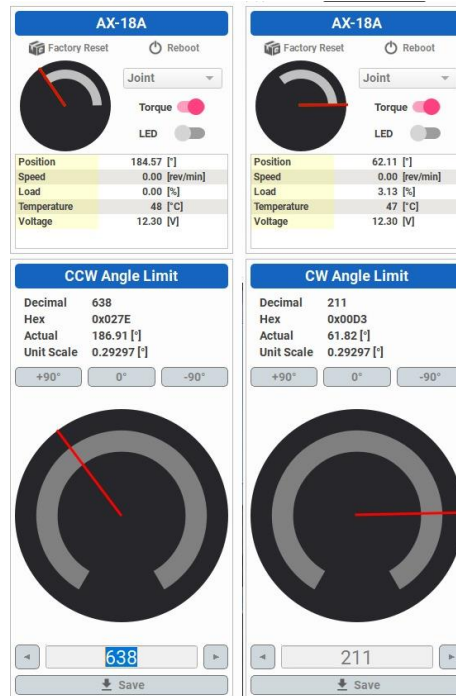


Figura 32. Restricción del segundo motor a la base

Restricción del tercer motor a la base:

Cuenta con una posición máxima de 338 y una posición mínima de 846:

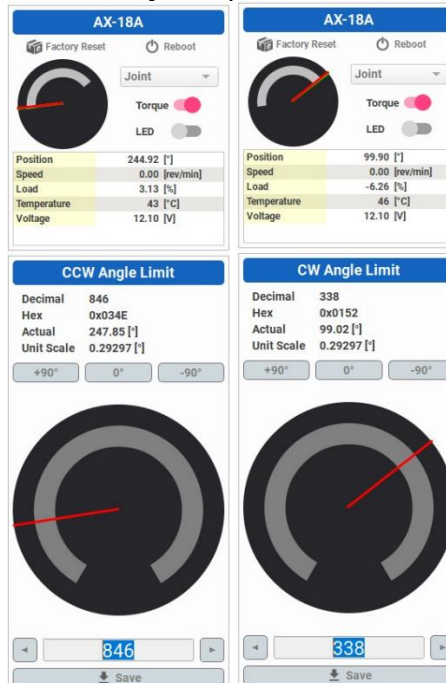


Figura 33. Restricción del tercer motor a la base

Restricción del cuarto motor a la base:

Cuenta con una posición máxima de 209 y una posición mínima de 638:

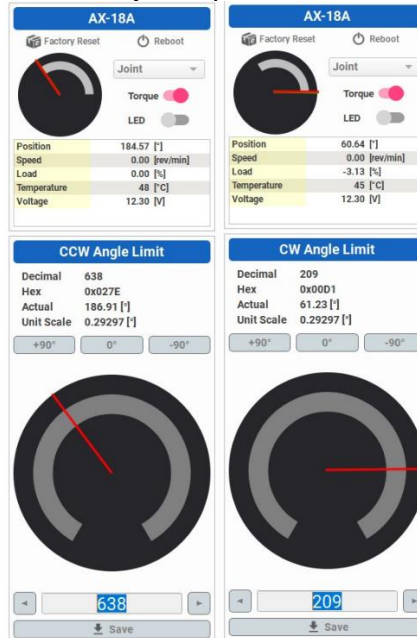


Figura 34. Restricción del cuarto motor a la base

Restricción del motor que controla la pinza:

Cuenta con una posición máxima de 561 y una posición mínima de 762:

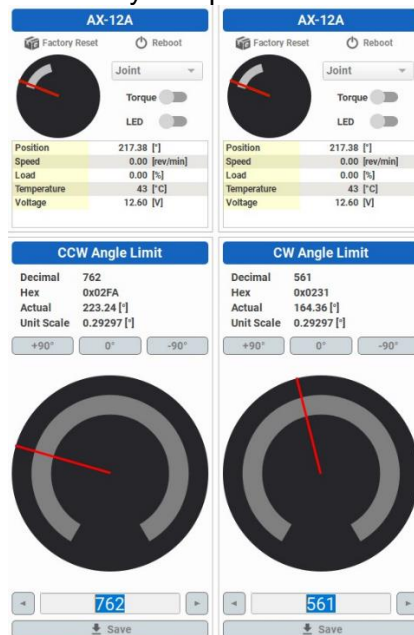


Figura 35. Restricción del motor que controla la pinza

Video de las restricciones de los servos:

En el siguiente enlace se podrá encontrar el movimiento de los servomotores con sus respectivas restricciones:

<https://www.youtube.com/watch?v=Bx20I3kAxpY>

MANUFACTURA Y ENSAMBLE DE LAS PIEZAS

Para la fabricación de las piezas se optó por el método de manufactura aditiva, es decir por medio de impresión 3D. A lo largo de las distintas pruebas se utilizaron dos impresoras de la marca CREALITY; una Ender-3 V2 con un filamento de PLA y una Ender-3 V2 Neo con un filamento de PETG.

A lo largo de esta etapa hubo diversas iteraciones debido a diferentes causas, entre ellas, la nula experiencia por parte del equipo con este método de manufactura, los errores de diseño que hubo en las distintas piezas y las fallas recurrentes por parte de la impresora Ender-3 V2. Todas estas iteraciones se ven reflejadas en el documento de “Pruebas de manufactura aditiva” [7].

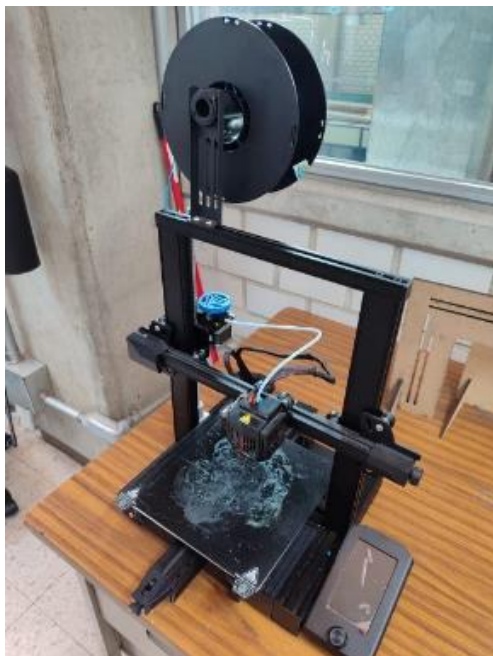


Figura 36. Ender-3 V2 Neo

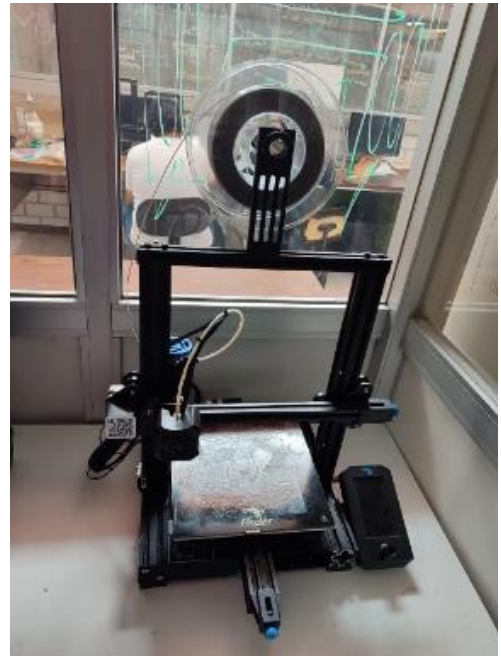


Figura 37. Ender-3 V2

Como primer paso, es necesario generar los archivos que la máquina pueda leer para realizar la impresión a partir de los modelos de las piezas realizados en CAD. Dado que la impresora tiene distintos parámetros a tomar en cuenta como el material de impresión, los soportes de la pieza, la densidad de relleno o el diámetro de la boquilla del extrusor, es necesario utilizar un software que nos permita configurar todos estos parámetros, ya que nuestro CAD no tiene estas herramientas.

El programa que vamos a utilizar es UltiMaker Cura (Versión 5.7.1), un software de uso libre en donde podemos realizar toda la configuración necesaria de las piezas y generar un archivo que la impresora pueda leer para realizar su trabajo. Cabe mencionar que con este programa es posible exportar dos piezas en la misma área de trabajo para que se puedan imprimir ambas al mismo tiempo sin necesidad de generar dos archivos diferentes.



Figura 38. Pantalla de carga UltiMaker Cura

Para poder empezar a trabajar con este programa debemos de generar los archivos de las piezas con extensión “.stl”, estos pueden ser generados con el CAD “Autodesk Fusion” a partir del modelo con extensión “.ipt” de la pieza. Una vez que se tiene la pieza en archivo “.stl” se puede importar desde la interfaz de UltiMaker Cura para agregarla al espacio de trabajo, este software nos permite importar varias piezas en el mismo espacio de trabajo e imprimir más de una al mismo tiempo.

Dentro de la interfaz podemos posicionar y orientar cada elemento de manera que nos resulte más conveniente, la postura de las piezas es de gran importancia a la hora de la manufactura, ya que los tiempos de impresión, la calidad del acabado superficial y la cantidad de material utilizado pueden variar. También se deben de considerar los soportes auxiliares que se impriman junto con la pieza, ya que igualmente la cantidad de soportes, y por tanto de material utilizado, pueden ser mayores o menores.

Una vez que tenemos nuestros modelos con la postura adecuada, debemos de modificar los distintos parámetros que se seguirán durante la impresión. En la parte superior se despliegan cuadros en donde uno puede modificar el material, la densidad de relleno, el diámetro de la boquilla del extrusor y también podemos activar la opción “Supported”, esta última debe de estar activa en cada pieza o de lo contrario no se van a poder imprimir.

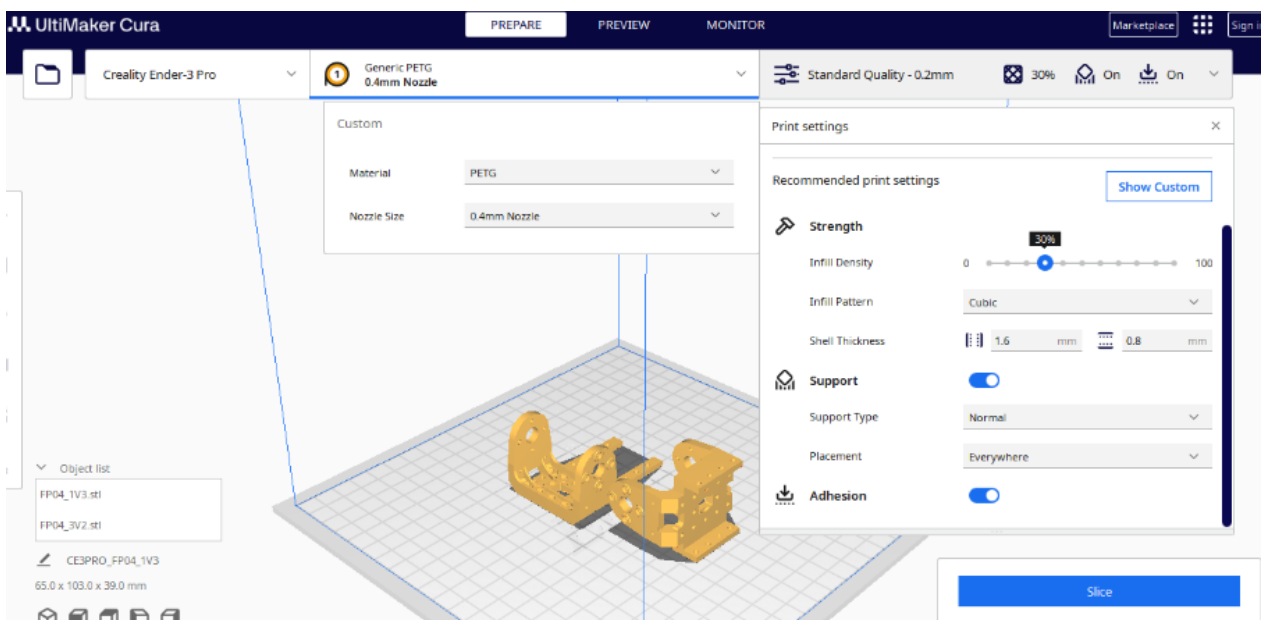


Figura 39. Espacio de trabajo UltiMaker Cura

En este caso se utilizó una densidad del 30% y una boquilla de 0.4[mm] de diámetro, como se mencionó anteriormente, se utilizaron dos máquinas con diferente material, por lo que este parámetro va a variar dependiendo de cada pieza. Una vez que se tienen los modelos con la postura necesaria y todos los parámetros configurados, se selecciona la opción “Slice” para que el programa genere el laminado capa por capa de la pieza, dado que se tiene

un Shell que únicamente considera la superficie exterior del cuerpo, se rellena el interior tomando interpolaciones de diversos puntos que describen la geometría de la pieza, siguiendo un mallado que rellena el Shell.

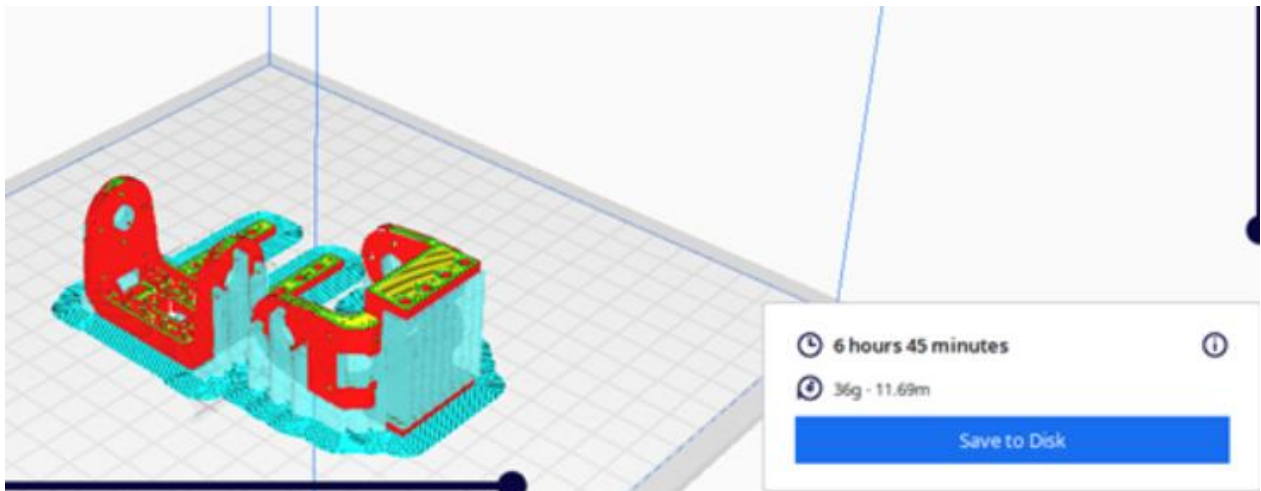


Figura 40. Preview de las piezas

Una vez que termina el proceso de Slice, el programa nos deja ver una vista previa de los componentes que se van a imprimir, indicando un tiempo aproximado de impresión y una cantidad de material a utilizar. El Slice también se encarga de generar los soportes que se necesitan para que pueda llevarse a cabo la manufactura.

Una vez que terminamos Ultimaker Cura genera un archivo con extensión “.gcode” que podemos guardar en una memoria micro SD, este archivo ya puede ser leído por nuestra máquina y será el que le ingresemos por medio de la unidad de memoria portátil para poder avanzar al siguiente paso en la fabricación.

Antes de dar inicio a la manufactura fue necesario aprender algunos parámetros de configuración para que la impresora pudiera cumplir con su tarea, es de suma importancia este paso ya que de lo contrario ninguna pieza va a poder ser fabricada. Cada impresora se ajustó de manera distinta sin embargo para ambas se utilizaron los mismos parámetros.

Para la calibración de la primera máquina con filamento de PLA fue necesario nivelar la cama caliente, se ubicó el extrusor en el “Home” y después se utilizó una hoja blanca de papel para medir la distancia en el extrusor y la cama, la cual debe de ser de aproximadamente 0.4[mm] (La medida del grosor de la hoja). Se ajustó la distancia por medio de los tornillos



puestos en la base para lograr la nivelación correcta al poner la hoja entre la cama y la boquilla del extrusor y se apretaban o aflojaban los tornillos hasta que se presionara la hoja sin sujetarla completamente, esta operación se llevó a cabo dos veces en diferentes puntos de la cama.

Para la máquina con filamento de PETG, dado que esta cuenta con un sensor para detectar la distancia, no fue necesario hacer la prueba con la hoja de papel, al igual que en el caso anterior se llevó el extrusor a la posición de “Home” y simplemente se niveló la cama caliente utilizando un calibrador vernier, midiendo que fuera la misma distancia de la cabeza de los cuatro tornillos a la cama. Después se utilizó la función de autonivelación que tiene la impresora en donde, de manera automática, se lleva la boquilla del extrusor a distintos puntos de la cama y por medio del sensor se ajusta la distancia entre ambas partes.

Una vez calibradas las impresoras podemos comenzar nuestro proceso de manufactura, la interfaz de ambas es idéntica lo que facilita el manejo. Dentro del menú se debe de seleccionar la opción Print, posteriormente se selecciona el archivo con las piezas que se desean imprimir y listo.

La temperatura que se seleccionó para la cama caliente fue de 60°C, la del extrusor varió un poco dependiendo del material pero se manejaron temperaturas de 210°C para el PLA y 215° para el PETG. Antes de que comience la impresión también fue necesario rociar la cama caliente con un fijador en aerosol, esto con la intención de que las piezas no se despeguen durante el proceso y nuestra impresión falle.

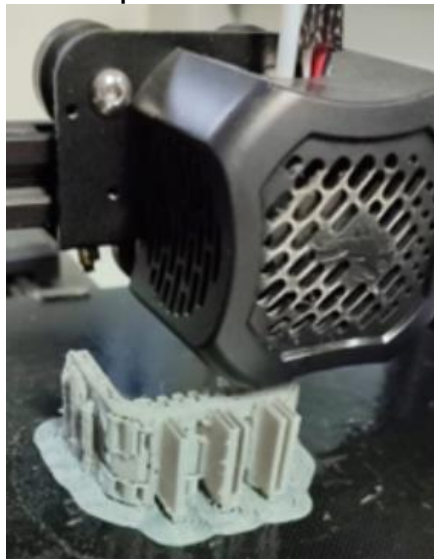


Figura 41. Proceso de impresión con PLA

Durante la etapa de diseño de las piezas se realizaron distintas iteraciones debido a la falta de experiencia con este proceso de manufactura y la falta de la prueba de colisiones en el ensamble durante las primeras 3 versiones de la lista de componentes. A continuación se muestran algunos de los modelos resultantes de estas iteraciones, en donde algunas de estas piezas pudieron cumplir con las dimensiones y requerimientos para el ensamble físico.

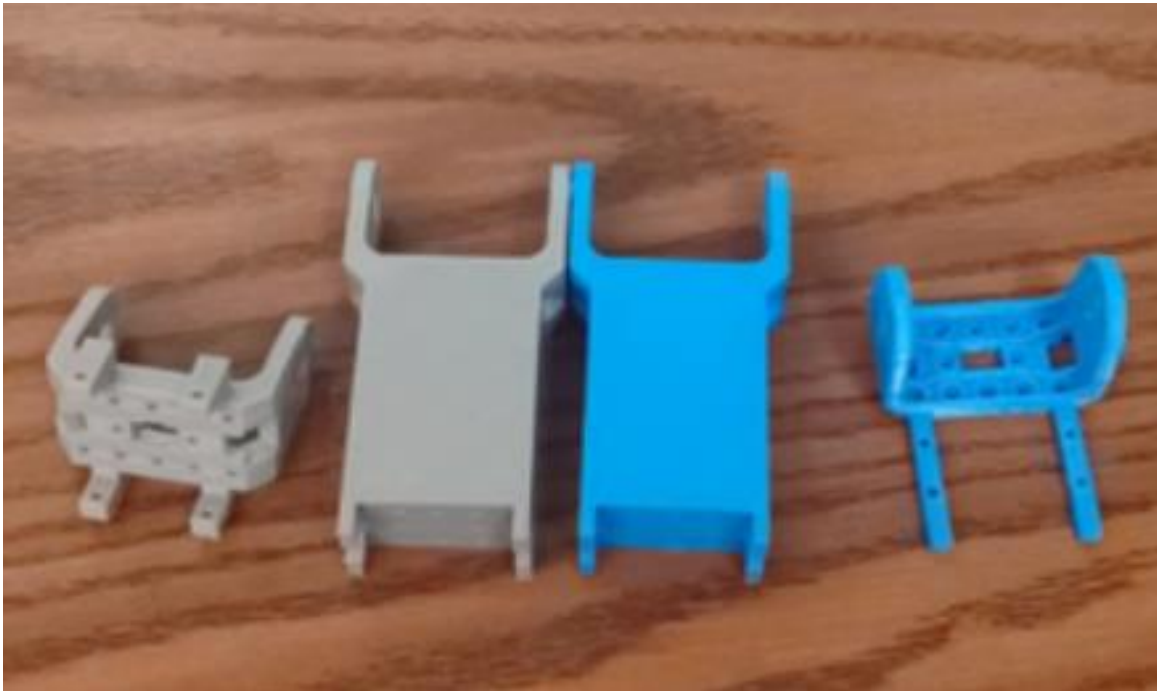


Figura 42. Prototipos de las piezas a ensamblar

Después de que todas las piezas fueran fabricadas se procedió al ensamble del brazo robótico. Como se mencionó al inicio, el brazo robótico al contar con 5 grados de libertad, necesitará el mismo número de actuadores que nos permitan realizar los diferentes movimientos, por lo que se utilizaron de 5 servomotores AX-18A. Además de los servos es necesaria una pinza que pueda abrir y cerrar con el objetivo de poder manipular objetos.

Para poder ensamblar se utilizaron tornillos M3 con sus respectivas tuercas, los actuadores cuentan con barrenos y guías para poder atornillar las piezas, hay que tener mucho cuidado de utilizar estos espacios designados para el ensamble y no dañar el motor. Fue necesaria la utilización de un desarmador con punta de cruz y una llave Allen para poder llevar a cabo el ensamble ya que la cabeza de algunos tornillos era diferente.



Figura 43. Proceso de ensamble

Primero se procedió a ensamblar las piezas a las guías laterales e inferiores de los servos antes de atornillar en los ejes, esto con el objetivo de facilitar la visualización del orden de los eslabones con base al modelo hecho en el CAD Fusion. Posteriormente se conectaron los servos en serie siguiendo el orden correcto teniendo cuidado de conectar los cables en el puerto correcto, cada servo tiene un puerto de entrada que se debe conectar con el actuador anterior y otro de salida en donde se deberá conectar con el actuador siguiente.

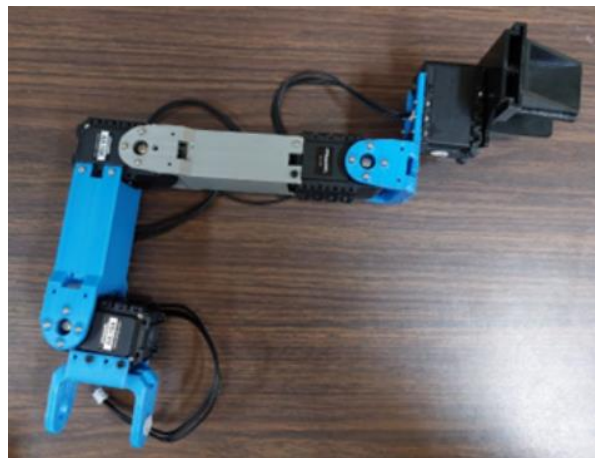


Figura 44. Ensamble del brazo vista lateral

Como al momento de realizar el ensamble aún no se contaba con la base del robot, sólo se pudieron ensamblar 4 servomotores, ya que el faltante es el que se ensamblaba con la base faltante, sin embargo, con el modelo realizado se pudieron realizar las pruebas de funcionamiento.



CONCLUSIÓN

Con base a lo desarrollado en este proyecto podemos afirmar que cuando se trabaja en un proyecto grande cada parte es importante, y es indispensable verificar hasta los detalles más pequeños. Tal es el caso de las medidas y características de los diseños, ya que si estas están mal la impresión en 3D no será la mejor, causando contratiempos o incluso errores que solo pueden ser corregidos volviendo a diseñar e imprimir, generando una gran pérdida de tiempo y material. A su vez aprendimos que en ocasiones el problema puede ser ocasionado por la plataforma que empleamos, tal fue nuestro caso con MATLAB que presentaba problemas para comunicarse con los servomotores.

Pese a la falta de tiempo con la que trabajamos logramos hacer una revisión y corrección de las piezas para poder manufacturarlas mediante el uso de una impresora 3D, comprendimos e implementamos la interfaz de los motores y logramos obtener un avance considerable a pesar de las adversidades que se nos presentaron en este semestre. Por lo que podemos concluir que se alcanzaron la mayoría de los objetivos planteados al comienzo del proyecto.

Se pudieron cumplir con tres de los cuatro objetivos planteados al inicio del proyecto, el objetivo particular de “Trabajar en una interfaz de control de los motores en Matlab para que el robot realice una tarea específica.” no se pudo Completar al 100% debido a los recursos computacionales con los que contaba la brigada y el software de Matlab presentó diversos problemas. Sin embargo, con el trabajo realizado por el equipo se logró cumplir con la premisa de que como futuros profesionistas en el campo de la ingeniería, pudimos aplicar los conocimientos aprendidos a lo largo del curso en una implementación práctica.

COMENTARIOS

Nos gustaría sugerir que se preste una mayor atención a los diseños, ya que en la computadora puede parecer que están bien, pero al momento de imprimirlos en 3D y hacerle pruebas, tienden a fallar por errores mínimos.



También nos gustaría sugerir que se trabaje con otra plataforma como Arduino o ROS para el desarrollo de la interfaz ya que MATLAB genera bastantes contratiempos.

REFERENCIAS

- [1] R. e-manual, «Open Manipulador-X,» 2017. [En línea]. Available: https://manual.robotis.com/docs/en/platform/openmanipulator_x/overview/. [Último acceso: 04 junio 2024].
- [2] S. SanDoRobotics, «SNAPPER ROBOT ARM,» 2024. [En línea]. Available: <https://sandorobotics.com.mx/producto/rg-snapper/>. [Último acceso: 04 junio 2024].
- [3] G. Robots, «Base robot,» 2024. [En línea]. Available: <https://www.generationrobots.com/en/401792-widowx-robot-arm-without-servomotors.html> . [Último acceso: 04 junio 2024].
- [4] Pascual, «Revisión para base de robot open-puma,» 2024. [En línea]. Available: https://github.com/arrg-mx/Open-UNAM/blob/main/Carpeta_2024_2/Base_robot/analisis%20de%20base%20robotica.pdf. [Último acceso: 04 junio 2024].
- [5] GRABCAD, «Grabcad Community,» 2024. [En línea]. Available: <https://grabcad.com/library/interbotix-phantomx-reactor-robot-arm-kit-1> . [Último acceso: 04 junio 2024].
- [6] R. e. -. Manual, «AX-18F, AX-18A,» 2024. [En línea]. Available: <https://manual.robotis.com/docs/en/dxl/ax/ax-18a/>. [Último acceso: 04 junio 2024].
- [7] Gibran, «Pruebas de manufactura aditiva,» 2024. [En línea]. Available: https://github.com/arrg-mx/Open-UNAM/blob/main/Carpeta_2024_2/Pruebas%20de%20manufactura%20aditiva.pdf. [Último acceso: 04 junio 2024].