
Diseño de un sistema de motores inteligentes en conjunto con un sistema de control del brazo robótico asistencial para cirugías estereotácticas.

Peter Alejandro Yau Pan



UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería



**Diseño de un sistema de motores inteligentes en conjunto con
un sistema de control del brazo robótico asistencial para
cirugías estereotácticas.**

Trabajo de graduación presentado por Peter Alejandro Yau Pan para
optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería



**Diseño de un sistema de motores inteligentes en conjunto con
un sistema de control del brazo robótico asistencial para
cirugías estereotácticas.**

Trabajo de graduación presentado por Peter Alejandro Yau Pan para
optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,


2022


Vo.Bo.:

(f) 
Ing. Kurt Kellner

Tribunal Examinador:

(f) 
Ing. Kurt Kellner

(f) 
MSc. Carlos Esquit

(f) 
Dr. Juan Carlos Lara

Fecha de aprobación: Guatemala, 8 de enero de 2022.

Inspiración

El desarrollo y uso de manipuladores seriales es un área en la robótica que ha sido implementada muy comúnmente en la industria automotriz, al igual que en otras industrias como la de ensamblaje de electrónicos y semiconductores. Sin embargo, es reciente que el uso de manipuladores seriales, más específicamente brazos robóticos, se han popularizado en el campo médico con la invención del sistema de cirugía DaVinci. El sistema de cirugía Da Vinci es un sistema compuesto de 3 o 4 brazos robóticos que son manipulados por el médico a través de una consola. Estos brazos pueden sostener diferentes instrumentos médicos, desde bisturís, pinzas y tijeras. Esto significa que el nivel de precisión que tanto el sistema como el médico requieren tener son considerablemente altos. El brazo que estamos trabajando intenta reducir lo mayor posible el componente humano en el sistema, lo que permite que el sistema trabaje solamente con componentes electrónicos. Sin embargo, debido a que se redujo el componente humano, el brazo no trabaja en operaciones directamente sino que provee apoyo a dichas cirugías, aún ejercidas por el médico. Es debido a esto que tomamos como referencia mayormente el sistema de cirugía ROSA, el cual es un sistema mucho más sencillo, relativo al sistema Da Vinci, que aún ejerce un nivel de apoyo considerable al médico al poder indicarle en dónde colocar tanto los electrodos como los catéteres.

Prefacio	v
Lista de figuras	ix
Lista de cuadros	x
Resumen	xi
Abstract	xii
1. Introducción	1
2. Antecedentes	2
2.1. ROSA	2
2.2. Aplicación de ROSA en epilepsia y neurooncología	3
2.3. Primera etapa	4
2.4. Segunda etapa	5
3. Justificación	9
4. Objetivos	10
4.1. Objetivo general	10
4.2. Objetivos específicos	10
5. Alcance	11
6. Marco teórico	12
6.1. Epilepsia	12
6.2. Neurocirugía	13
6.3. Cinemática inversa	13
6.4. Controlador PID de tipo fuzzy gain scheduling	14
6.5. Método de daisy chain para control tolerante a fallas basado en asignación de control	16
6.6. Motores paso a paso (stepper)	16

6.7. Modelado matemático de un motor stepper	18
6.8. VarioGuide de Brainlab	19
6.9. Protothreads	19
6.10. Dynamixel XL-320	20
6.11. Monitoreando diferentes sensores con microprocesador Atmega328	21
6.12. Condiciones de uso para los GPIO tolerantes de 5-V de los microcontroladores Tiva series TM4C123x	24
6.13. EGCL: Un Lenguaje de G-Code Extendido con Control de Flujo, Funciones y Variables Mnemotécnicas	25
7. Desarrollo de los motores inteligentes	27
7.1. Selección de microcontroladores para cada motor	27
7.2. Selección de drivers para los motores	30
7.3. Calibración preliminar con motor Nema 17HS4401S para pruebas	31
7.4. Desarrollo de un protocolo de comunicación para los actuadores	33
7.5. Listado de comandos programados para los motores revisado	38
7.6. Selección de IDE y lenguaje para el microcontrolador maestro	43
8. Sistema de control del brazo robótico asistencial	44
8.1. Comunicación entre los dispositivos del brazo robótico asistencial	44
8.2. Cálculos para determinar el número de pasos en base a sus reducciones de engranajes	45
8.3. Pruebas con simulación de sistema de reconocimiento óptico de caracteres (OCR) en Python	46
8.4. Incorporación del sistema OCR al movimiento de los actuadores	48
8.5. Pruebas con simulación del mando de control manual	49
8.6. Integración del mando de control físico al movimiento de los actuadores	51
8.7. Implementación simultánea del sistema de reconocimiento de texto con el mando de control	52
9. Conclusiones	58
10.Recomendaciones	59
11.Bibliografía	60
12.Anexos	63
12.1. Repositorio en Github con los programas	63

Lista de figuras

1.	ROSA One Brain para neurocirugías	3
2.	Servomotor AX-12A	4
3.	Servomotor MX-106	5
4.	Microcontrolador Tiva C TM4C123GXL	6
5.	Motores Nema 23	6
6.	Motores Nema 17	7
7.	Drivers que se utilizaron	8
8.	Cinemática inversa de un brazo de 6 grados de libertad [9].	14
9.	Diagrama de un sistema usando fuzzy gain scheduling	15
10.	Conexiones tipo paralelo vs. daisy-chain.	16
11.	Sistema interno de un motor paso a paso [12].	17
12.	Variedades de motores paso a paso[14].	18
13.	Sistema de ubicación y alineación VarioGuide [17]	19
14.	Dynamixel XL-320	21
15.	Microcontroladores de la familia AVR [22].	22
16.	Diagrama de entradas y salidas del ATmega328	23
17.	Esquema de conexión del driver A4988. [25]	32
18.	Conexión de los componentes en una protoboard.	33
19.	Pantalla de inicio y de configuración de PuTTY	34
20.	Interfaz de la aplicación de utilidad Hércules.	34
21.	Sistema original de nombrado de microcontroladores.	35
22.	Sistema actualizado de nombrado de microcontroladores.	35
23.	Comando de pasos y el número de bits total del comando.	36
24.	Diagrama de comunicación entre los dispositivos del brazo robótico	44
25.	Conexión entre el sistema OCR, y sistema de control de actuadores (azul y verde)	48
26.	Imagen con error de 0.2° utilizado para las pruebas.	49
27.	Sosteniendo el servo para verificar movimiento.	49
28.	Circuito armado para simular el encoder utilizado por mi compañero.	50
29.	Conexión del mando de control al sistema de actuadores.	51

30.	Ventana serial mostrando salida de datos de la Tiva C con la conversión de “ticks” a pasos	52
31.	Esquema de conexión del driver DRV8825. [26]	53
32.	Esquema de conexión del driver TB67S249FTG. [27]	53
33.	Esquema de conexión del driver TB67S128FTG. [28]	53
34.	Brazo robótico con los motores y ejes flexibles instalados.	54
35.	Conexión de los microcontroladores a los motores del brazo.	55
36.	Reenvío de datos al microcontrolador correcto.	55
37.	Resolución de movimiento del mando convertido a pasos.	56
38.	Error angular del OCR convertido a pasos.	57

Lista de cuadros

1.	Comparación de drivers	7
2.	Comparación de motores y drivers	7
3.	Especificaciones del ATmega328	23
4.	Condiciones de uso de los pines GPIO dependiendo de su configuración. [23] .	25
5.	Especificaciones de varios microcontroladores	29
6.	Parámetros de comparación del trades study de microcontroladores	30
7.	Trade study de microcontroladores	30
8.	Especificaciones de los drivers de motores paso a paso	31
9.	Comparación del sistema de comandos anterior y actualizado.	38
10.	Manual de comandos para los actuadores	43
11.	Relaciones y reducciones de cada junta	45

El presente trabajo tiene como fin el de desarrollar un sistema de actuadores que puedan implementarse para convertir motores paso a paso comunes en motores paso a paso “inteligentes” que le permitan al usuario utilizar un listado de comandos para ejecutar su movimiento. En este caso, esto va a ir orientado a un brazo robótico asistencial para procedimientos de cirugía de epilepsia. Para lograr esto, se dividió el proyecto en dos grupos: el desarrollo de los motores y el sistema de control.

Para el desarrollo de los motores se hizo una selección de los microcontroladores que se van a utilizar para formar parte de cada uno de los actuadores al igual que el microcontrolador principal que funcionaría como el maestro. Adicionalmente, se hizo la selección de los drivers que se consideraron adecuados para los motores stepper, tomando en cuenta el uso que se le quiere dar (aunque también se consideraron las características adicionales que el driver pueda proveer al sistema). Ambas selecciones se hicieron a través de trade studies para poder comparar las especificaciones entre los dispositivos. Según los resultados obtenidos por el primer trade study, se seleccionó la Tiva C (TM4C123GH6PM) como el controlador maestro y el ATmega328P para ir adjunto a cada motor. Para la comunicación entre dispositivos se empleó un sistema daisy chain, en donde el control maestro envía un comando, el cual es transmitido a través de cada uno de los microcontroladores para luego ser finalmente recibido por el maestro para cerrar el bucle. Este retorno de datos al microcontrolador maestro permite que el usuario pueda tener retroalimentación sobre el funcionamiento correcto de los motores. Utilizando este sistema se logró incorporar un listado de comandos que tiene las siguientes funciones: le permite al usuario asignarle “direcciones” o “nombres” a cada uno de los microcontroladores que le permite luego asignarle comandos específicos a cada actuador individual; le permite al usuario establecer el número de pasos que desea que el motor stepper dé y si se desea utilizar el microstepping; permite establecer un cero relativo para que el usuario pueda determinar el número de giros que el motor ha dado.

The present work aims to develop a system of actuators that can be implemented to convert common stepper motors into “ smart ” stepper motors that allow the user to use a list of commands to carry out their movement. In this case, this will be oriented to a robotic arm for epilepsy surgery procedures. To achieve this, the project was divided into two groups: the development of the actuators and the control system.

For the development of the motors, a selection was made between a variety of microcontrollers that will be used to form part of each of the actuators as well as the microcontroller that would function as the master. Additionally, a selection of the drivers that were considered suitable for the stepper motors was made, taking into account their intended use (although the additional features that the driver can provide to the system were also considered). Both selections were made through trade studies in order to be able to compare the different specifications between the devices. Based on the results obtained from the first trade study, the Tiva C (TM4C123GH6PM) was selected as the master controller and the ATmega328P to be attached to each motor. For communication between devices, a daisy chain system was implemented, where the master controller sends a command, which is then transmitted through each of the actuators, finally returning to the master controller in order to close the loop. This return of data to the master microcontroller allows the user to have feedback on the proper operation of the motors. Using this system, it was possible to incorporate a list of commands that has the following functions: it allows the user to assign “ addresses ” or “ names ” to each of the microcontrollers, which then allows them to assign specific commands to each individual actuator; it allows users to set the number of steps that they want the stepper motor to take and whether they would like to enable microstepping; it allows setting a relative zero so that the user can determine the number of turns the motor has made based on the angle at which the motor ultimately stops at.

CAPÍTULO 1

Introducción

Actualmente se está trabajando en la fase 3 del desarrollo de dicho brazo, aunque debido a cambios y ajustes al enfoque original, se hicieron modificaciones a la finalidad del proyecto. El fin general es el mismo, el de brindar apoyo al médico para facilitar las operaciones neuroquirúrgicas, pero la metodología original para desarrollar el brazo se tuvo que adaptar. Tomando esto en cuenta, el objetivo general de esta fase del proyecto es el del desarrollo de los motores inteligentes que se van a emplear para el sistema de control del brazo. Dicho sistema de motores se incorporó en conjunto con lecturas de error de posición para permitir un ajuste automatizado al igual que un control remoto para permitir ajustes manuales donde fuera requerido. Para lograrlo se creó un protocolo de comunicación, al igual que un listado de comandos que le permite al usuario fácilmente controlar los motores para el propósito que requiera.

2.1. ROSA

ROSA es un brazo robótico creado por Medtech que utiliza tecnología a base de imagen para guiarse y ofrecer asistencia en cuanto a procedimientos neuroquirúrgicos poco invasivos. El sistema en sí incorpora cuatro componentes:

- Una base móvil.
- Un brazo telescópico retraible.
- Una pantalla táctil.
- El brazo robótico.

Para obtener una imagen preoperativa de la cabeza del paciente, ROSA utiliza escaneo laser para generar una figura de tres dimensiones utilizando software propio, para luego establecer una trayectoria de movimiento utilizando ese mismo software.

El interés en la implementación de ROSA se ha visto mucho más en el área de neurocirugía de niños, principalmente debido a que como que el cerebro de los niños sigue en proceso de desarrollo, para prevenir efectos secundarios, utilizar un sistema guiado a base de imagen considerablemente preciso podría reducir el riesgo a estos efectos y a daño colateral a otras partes del cerebro. [1]



Figura 1: ROSA One Brain para neurocirugías

2.2. Aplicación de ROSA en epilepsia y neurooncología

Uno de los requerimientos que tiene el proceso de ablación láser es el de poder posicionar la fibra de láser de manera segura y con alta precisión en el área requerida. Debido al nivel de precisión ofrecido por ROSA, combinado con su habilidad de generar imágenes de resonancia magnética intraoperativa (durante el procedimiento), se ha utilizado en una variedad de operaciones neuroquirúrgicas.

Para el proceso de epilepsia, se analizó el posicionamiento de los electrodos, el tiempo requerido para ponerlos, el tiempo de la cirugía. Para poder poner a prueba la precisión de los electrodos colocados, se hizo una comparación con escaneos CT que contienen trayectorias previamente establecidas por el programa. El brazo llevó a cabo un ratreo de dichas trayectorias y cualquier trayectoria que tuviera una desviación de más de 3 mm fue considerada imprecisa.

Durante la fase de planeado, el equipo médico se encarga de utilizar combinaciones de escaneos CT y MRI para poder establecer diferentes trayectorias con el programa de ROSA. Se toman en cuenta la vasculatura intracranial del paciente, su anatomía y hasta la complejidad del proceso que se tiene que llevar a cabo. Durante la selección de trayectoria, los médicos se aseguran de elegir trayectorias con el menor contacto posible con el paciente, especialmente en áreas del cráneo en donde se localizaran puntos delicados como vasos corticales superficiales. El método de escaneo que utilizar ROSA para obtener los detalles de la estructura craneal del paciente es el uso de un láser para establecer el contorno del cráneo y agregarle los detalles necesarios a los escaneos tridimensionales anteriormente hechos ya con 3D CT scanning. ROSA permite ajuste manual en el eje de la trayectoria para poder

hacer cambios en cuanto a la profundidad a la que el brazo se encuentre en el momento de colocar el electrodo o el cateter. [2]

2.3. Primera etapa

Para la primera etapa, el brazo robótico constó de 3 prototipos, todos utilizando motores de tipo servo. El primer prototipo estuvo compuesto de 6 servomotores Dynamixel AX-12A proporcionados por la Universidad del Valle de Guatemala, los cuales se utilizaron para generar el movimiento rotacional de 6 juntas.



Figura 2: Servomotor AX-12A

Sin embargo, se descartó el primer prototipo debido a que su diseño no ofrecía la robustez y estabilidad necesaria debido a su asimetría. En el caso del segundo prototipo, el diseño del brazo fue sustituido por uno simétrico, lo que resultaba en mayor estabilidad. Sin embargo, la problemática que se encontró para esta segunda iteración fue la de la inhabilidad de los primeros tres motores AX-12A de las juntas más cercanas a la base de soportar el peso del brazo. Debido a esto, los motores activaban sus sistemas de seguridad, bloqueando su movimiento, por lo que el brazo quedaba estático en su lugar. Debido a esto, se descartó el prototipo 2 y se implementó un tercer prototipo en donde se sustituyeron los servomotores AX-12A por MX-106, también de Dynamixel.



Figura 3: Servomotor MX-106

Para el tercer prototipo, se sustituyeron los motores AX-12A más cercanos a la base por los motores MX-106 para incrementar la cantidad de torque disponible, por lo que también incrementó el peso que puede soportar.

Para la programación de los motores, se consideraron varias alternativas, desde las placas propias Robotis (la empresa que diseña y distribuye los motores Dynamixel) al igual que microcontroladores como el PIC o el Arduino. Finalmente, se decidió implementar el uso de un Arduino Mega, principalmente por su alto número de GPIO, además de su abundante cantidad de librerías para poder programar los actuadores.

2.4. Segunda etapa

En la segunda etapa, a diferencia de la primera etapa, se descartó el uso de los motores AX-12A y MX-106 y en su lugar se implementaron motores paso a paso. Para comenzar, se llevó a cabo un análisis para determinar la mejor opción de microcontrolador para el control de los motores, el cual, después de comparaciones entre diversos dispositivos, se decidió utilizar el microcontrolador Tiva C (TM4C123GXL).

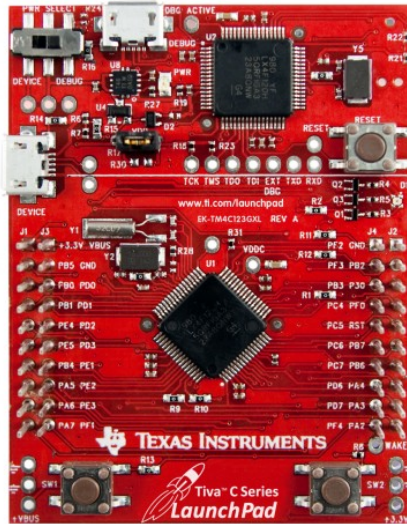
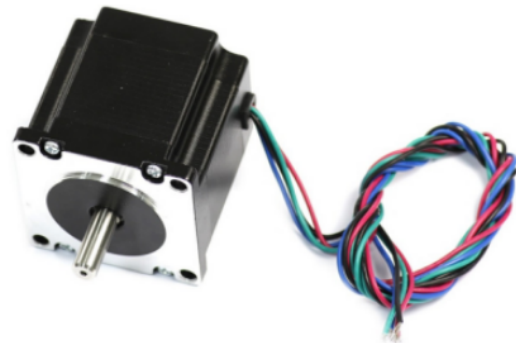


Figura 4: Microcontrolador Tiva C TM4C123GXL

Para la selección de motores, se mencionó anteriormente que se suplantaron los servomotores por motores de paso a paso debido a la necesidad de que el sistema tuviera suficiente torque para mantener estabilidad. Esto también se traslada a mayor peso que el sistema pueda soportar. Para seleccionar los motores de paso a paso a utilizar, se hicieron cálculos utilizando el peso de los nuevos componentes del brazo para encontrar el torque requerido para soportar dichos pesos. Habiendo hecho esto, se seleccionaron los motores paso a paso Nema 23, para las juntas principales y Nema 17 para las juntas ubicadas más cerca de la herramienta.



(a) Motor Nema 23 de la primera junta



(b) Motor Nema 23 de la cuarta junta

Figura 5: Motores Nema 23



(a) Motor Nema 17 con reducción de engranajes para las juntas 2 y 3



(b) Motor Nema 17 para las juntas 5 y 6

Figura 6: Motores Nema 17

Para la selección de drivers para conducir los motores paso a paso se llevó a cabo una comparación de especificaciones. Entre estos parámetros se encuentran el voltaje y corriente requeridos por los motores, el tipo de control que utilizan (pasos/dirección) y el protocolo de comunicación que soportan. Para ello se hizo una tabla comparativa en donde se fueron comparando estas características entre una variedad de *drivers*.

	Driver A4988	Driver DRV8825	Driver TB67S249FTG Full Breakout	Driver TB67S128FTG	TMC2226-SA	TMC2209-LA	TMC2660-PA
Voltaje mínimo	8.2 V	8.2 V	10 V	6.5 V	4.75 V	4.75 V	9 V
Voltaje máximo	35 V	45 V	47 V	44 V	29 V	29 V	30 V
Corriente continua por fase	1 A	1.5 A	1.7 A	2.1 A	2 A	2 A	2.8 A
Corriente máxima por fase	2 A	2.2 A	4.5 A	5 A	-	-	-
Step/dir	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Microstep	1/16	1/32	1/32	1/128	1/256	1/256	1/256
Programación SPI	No	No	No	No	No	No	Sí
Programación UART	No	No	No	No	Sí	Sí	No
Precio	\$5.95	\$8.95	\$11.95	\$13.95	\$15	\$15	\$15

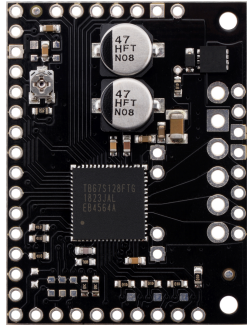
Cuadro 1: Comparación de drivers

Habiendo comparado los *drivers* entre ellos, lo que se prosiguió a hacer fue entrelazar los parámetros requeridos por los motores con los que pueden otorgar los *drivers*.

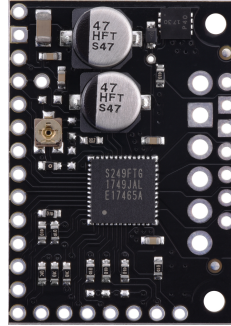
	Motor Nema 23 Junta 1	Driver TB67S128FTG	Motor Nema 23 Junta 4	Driver TB67S249FTG Full Breakout	Motores Nema 17 Juntas 2 y 3	Motores Nema 17 Juntas 5 y 6	Driver DRV8825
Voltaje nominal	3.78 V	-	-	-	-	3.4 V	-
Corriente requerida	4.2 A	5.0 A	2.8 A	4.5 A	1.7 A	1.7 A	2.2 A

Cuadro 2: Comparación de motores y drivers

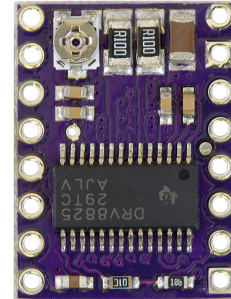
Como se puede ver en el Cuadro 2, el motor Nema 23 que ubicará en la base va a ir adjunto al *driver* TB67S128FTG, dado que es el *driver* que puede otorgar la mayor cantidad de corriente al motor. En cuanto al segundo motor Nema 23, el cual forma la junta 4, va a ir adjunto al *driver* TB67S249FTG. El resto de los motores Nema 17 que conforman el resto de las juntas del brazo (juntas 2, 3, 5, y 6) van a ir adjuntos a *drivers* DRV8825.



(a) TB67S128FTG Stepper Motor Driver Carrier (Vista Superior).



(b) TB67S249FTG Stepper Motor Driver Carrier (Vista Superior).



(c) DRV8825 stepper motor driver carrier.

Figura 7: Drivers que se utilizaron

La cirugía de epilepsia es un procedimiento largo y arduo que requiere extrema precisión de parte del médico encargado. Debido a esto, tener un dispositivo asistencial que desarrolle el posicionamiento inicial y que sea lo suficientemente robusto para mantener su posición podría ser de gran ayuda para facilitar el trabajo del cirujano.

En Guatemala, se estima que alrededor de 325,000 personas padecen de epilepsia, en donde el 30 % sufre de un caso de difícil control. La institución que ofrece ayuda contra enfermedades como la epilepsia, el Parkinson y los tumores cerebrales es el Centro de Epilepsia y Neurocirugía Funcional HUMANA. Para poder brindar apoyo en el esfuerzo de ayudar a las personas que sufren de esta condición, se está diseñando un brazo robótico que pueda asistir al neurocirujano para facilitar el proceso de operación durante la cirugía. Actualmente, se quiere implementar un sistema casi automático en donde el médico no tenga que hacer nada más que establecer un punto inicial para insertar el electrodo y usar un mando de control para afinar la posición del efector final. De esta manera, no es necesario que el médico tenga que dividir su concentración en mantener una posición adecuada y en la inserción del electrodo al mismo tiempo.

El enfoque de este trabajo es en el área de control del brazo robótico asistencial. La razón por ello es debido a que se quiere poder desarrollar un sistema de motores que permita modularidad y flexibilidad, de manera que se pueda implementar en el diseño del brazo asistencial, al igual que otros proyectos futuros en donde se haga uso de motores stepper. El enfoque principal de este sistema es del brazo asistencial, por lo que los ajustes y los controladores que se van a implementar van a ir configurados al funcionamiento correcto y estable del brazo. Cabe mencionar que a pesar de que anteriormente se trabajó en este proyecto, debido a los cambios que se tuvieron que hacer en cuanto a la manera de implementación del proyecto, algunos de los avances hechos anteriormente se tuvieron que modificar y otros descartar.

4.1. Objetivo general

Diseñar un sistema de motores inteligentes que incorpore motores stepper, drivers, sensores y microcontroladores para luego emplearlo en el sistema de control del brazo robótico asistencial.

4.2. Objetivos específicos

- Diseñar un protocolo de comunicación que permita entrelazar los módulos de reconocimiento de imágenes y el control remoto con el sistema de control.
- Diseñar un sistema de control manual (lazo abierto) para los diferentes grados de libertad individuales del brazo asistencial.
- Diseñar un sistema de control de lazo cerrado para los últimos grados de libertad para ajuste fino del brazo asistencial.
- Diseñar un sistema de motores inteligentes que combine motores paso a paso, drivers, sensores y microcontroladores con comunicación tipo "daisy-chaining".

Este proyecto consta de un sistema de motores stepper inteligentes, los cuales se van a utilizar para el desarrollo del movimiento de un brazo robótico asistencial para operaciones neurológicas. Desde un inicio, el proyecto tuvo como fin el implementar dicho brazo robótico como apoyo para operaciones desarrolladas en el Centro de Epilepsia y Neurocirugía Funcional HUMANA. Este fin no ha cambiado, aunque el proceso y metodología de desarrollo de dicho brazo sí ha cambiado conforme el proyecto ha ido evolucionando.

Para la fase actual, se desarrolló el sistema de motores inteligentes haciendo pruebas con los drivers y los motores que se tuvieran disponibles, pero no se quiere limitar su uso exclusivamente para el brazo robótico asistencial, sino que también se espera poder implementar este sistema en proyectos futuros, ya sea en proyectos de exhibición o inclusive en proyectos de clase. Como se mencionó anteriormente, se inició el desarrollo de los motores utilizando solamente los drivers y motores que se tenían disponibles, pero con esto ya se logró desarrollar un protocolo de comunicación propio. Adicionalmente, a comparación de sistemas existentes que utilizan métodos de conexión paralelos, para este proyecto se utilizó una conexión estilo "daisy-chain" en donde los datos son transmitidos de motor a motor para finalizar con una retroalimentación al controlador maestro.

6.1. Epilepsia

La epilepsia es una enfermedad cerebral que afecta a más de 50 millones de personas mundialmente. Se caracteriza por movimientos involuntarios hechos durante breves episodios de convulsión, donde la persona pierde control de partes del cuerpo (debido a que solamente un lado del cerebro fue afectado) o incluso de todo el cuerpo (debido a que ambos lados del cerebro fueron afectados). En casos más severos, la persona también puede llegar hasta a perder control de la función intestinal, y incluso perder la conciencia. [3]

Las convulsiones asociadas a esta enfermedad ocurren debido a la saturación de señales en las células cerebrales, las cuales pueden ocurrir en cualquier parte del cerebro, y debido a su naturaleza aleatoria, las convulsiones resultantes de dicha saturación neuronal pueden ser desde cortos episodios hasta episodios largos y graves. [3]

Entre los métodos de tratamiento se encuentran los siguientes:

1. Medicación: existe un relativamente alto número de medicamentos anticonvulsivos que tienen como fin limitar la propagación de las convulsiones en el cerebro. Dicho esto, es necesario, como todo medicamento, ajustar el número y volumen de medicamentos que el paciente tendrá que ingerir, y en algunos casos, cambiar la receta para incrementar la efectividad. Es imperativo tomar en cuenta que, dependiendo del medicamento prescrito, es posible que el paciente sufra de efectos secundarios. [3]
2. Cirugía: en casos donde las convulsiones ocurran debido a epilepsia focal o parcial, una de las alternativas al uso de medicamentos es el de llevar a cabo cirugía cerebral para extirpar la sección del cerebro culpable de generar dichas convulsiones. Sin embargo, esto es solo si dicha sección es considerada lo suficientemente pequeña de manera que no afecte otras funciones vitales del cerebro. Estas cirugías pueden llegar a eliminar la posible recurrencia de convulsiones futuras y en otros casos, reducir la magnitud de las convulsiones de manera que se puedan controlar con medicamentos. [3]

3. Terapia: el método de terapias consecuentemente se divide bajo 3 terapias diferentes:

- Estimulación del nervio vago: se implanta un dispositivo llamado estimulador de nervio vago por debajo de la piel del pecho, similar a la instalación de un marcapasos, la cual se conecta al nervio vago del cuello. Esta luego envía pulsos eléctricos a través del nervio hacia el cerebro. [4]
- Dieta cetogénica: dietas altas en grasas y bajas en carbohidratos. Con el tiempo, este método puede reducir e incluso eliminar las convulsiones, pero presenta efectos secundarios como deshidratación, deficiencia nutricional e incluso acumulación de ácido úrico en la sangre. [4]
- Estimulación cerebral: se implanta un electrodo en el área del cerebro que genera las convulsiones y se adjunta a un generador instalado en el pecho, el cual envía impulsos eléctricos al cerebro. [4]

6.2. Neurocirugía

La neurocirugía es el área médica especializada en la diagnosis y tratamiento de daño o enfermedades del sistema nervioso, incluyendo el cerebro, la columna, y nervios periféricos relacionados a estos. Para poder desempeñarse como neurocirujano, uno debe terminar los estudios y el entrenamiento necesario para practicar como doctor. Después, debe terminar entrenamiento especializado en neurocirugía y hasta puede llegar a tomar una subespecialización en áreas mucho más específicas como en cirugía de niños, cirugía de columna o incluso cirugía de tumores y cáncer cerebral. [5] [6]

El rol de llevar a cabo el diagnóstico y los tratamientos cae sobre el neurólogo. Ellos tienden a tratar enfermedades como esclerosis múltiple, Parkinson's, Alzheimer, trastorno de Lou Gehrig, enfermedades o infecciones del sistema nervioso perifero, entre otros. Sin embargo, cabe mencionar que el proceso de cirugía no es responsabilidad del neurólogo sino que del neurocirujano. [5]

6.3. Cinemática inversa

La cinemática inversa es una herramienta matemática empleada para la predicción y generación de movimiento de manipuladores seriales pero con énfasis en el efector final. Debido a esto, es una herramienta necesaria para poder desarrollar el movimiento del brazo robótico, especialmente la trayectoria del eslabón sosteniendo el electrodo. Uno de los métodos para desarrollar dicha cinemática inversa es con matrices de transformación al igual que emplear la matriz de Denavit-Hartenberg. Este es el método más popular para el desarrollo de la cinemática inversa, pero presenta un problema (o en este caso, se denomina singularidad), y eso es cuando los eslabones del manipulador serial se encuentran completamente en paralelo. [7]

Para poder resolver problemas de la cinemática inversa, existen una variedad de métodos, como los siguientes [7]:

- Traspuesta del Jacobiano
- Pseudoinversos
- Descenso de coordenadas cíclicas
- Levenberg-Marquardt
- Cuasi-Newton y gradientes conjugados
- Inteligencia artificial

La cinemática inversa ha tenido un uso extenso en aplicaciones gráficas, especialmente en el modelado tridimensional de personas y criaturas para el movimiento de sus extremidades (manos, cabezas, patas, colas, etc.). Un multicuerpo se modela a través de juntas y eslabones. Los tipos de junta pueden variar dependiendo del caso o el fin para el cual se quiera utilizar el multicuerpo, pero entre las juntas más comúnmente utilizadas están las juntas revolutas (movimiento rotacional descrito por un ángulo) y las prismáticas (movimiento traslacional descrito por una longitud escalar). [7] [8]

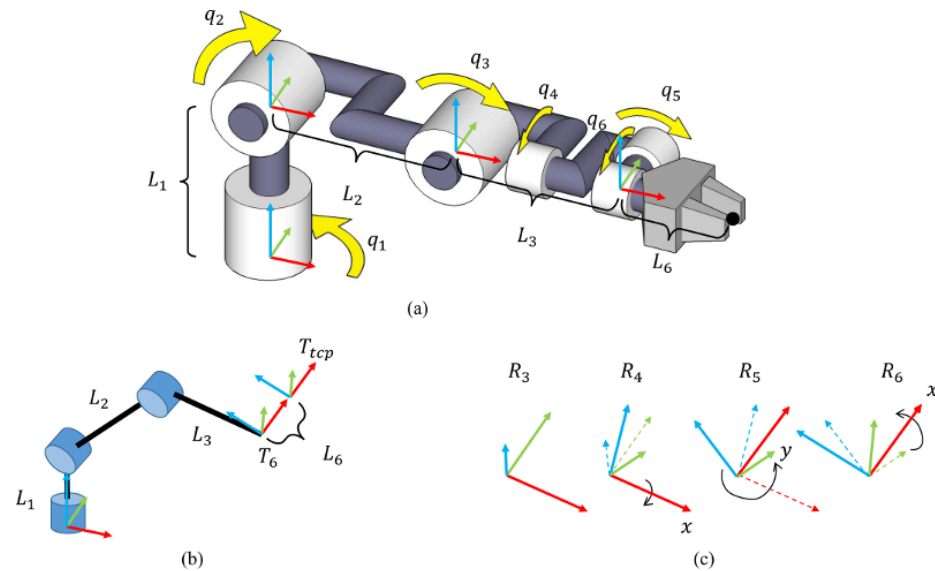


Figura 8: Cinemática inversa de un brazo de 6 grados de libertad [9].

6.4. Controlador PID de tipo fuzzy gain scheduling

El control PID es uno de los controladores más utilizados para la industria de control debido a su estructura simple y su habilidad de ser implementados en una gran variedad de situaciones. El controlador PID está compuesto de tres parámetros principales [10]:

- Ganancia proporcional
- Ganancia integral
- Ganancia derivada

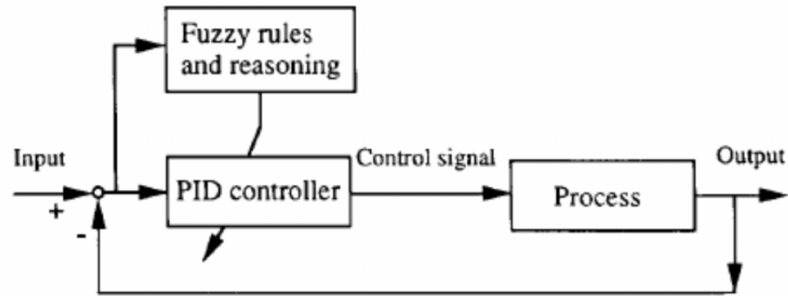


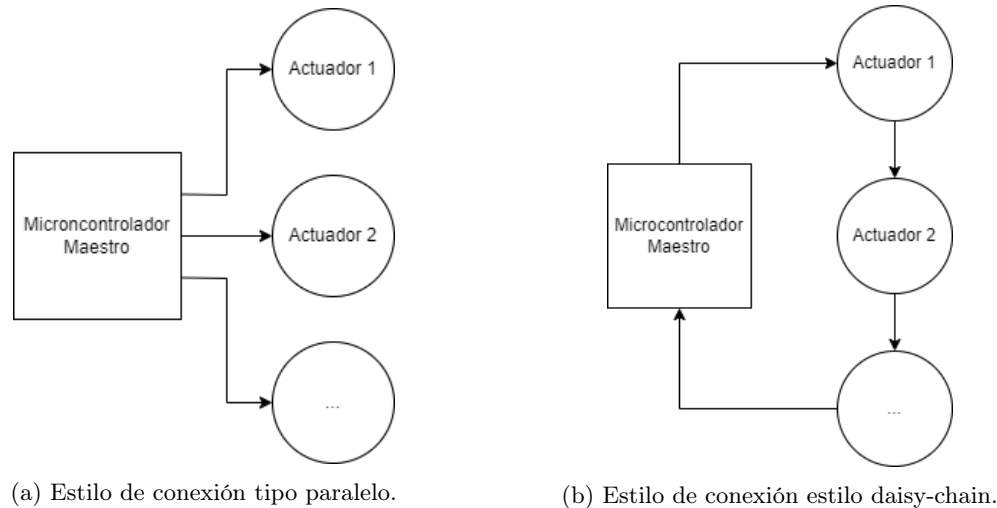
Figura 9: Diagrama de un sistema usando fuzzy gain scheduling

Un problema que existe con el controlador PID es que mucho del tiempo se invierte en decidir en los valores de ganancia adecuadas para que el controlador funcione de manera aceptable. Existen dos tipos de controladores PID [10]:

1. Control PID tradicional con los tres parámetros de ganancia establecidos de manera fija en el transcurso de la operación. Estos controladores son simples, pero debido a su naturaleza fija, su funcionamiento no es el mejor cuando la planta sufre de un cambio (requiere volver a ajustar sus ganancias para la nueva planta). [10]
2. Control PID adaptivo que tiene una estructura similar a la del PID tradicional pero cuyas ganancias K pueden irse modificando conforme ocurren cambios en la planta. Sin embargo, para que se pueda implementar este tipo de controlador PID, es necesario conocer la estructura de la planta. [10]

La implementación de un “fuzzy gain scheduled PID controller” involucra el uso del conocimiento de los estados del proceso en conjunto con un sistema de inferencia que le permite reaccionar de manera indicada a los cambios que surgen en el sistema. A pesar de que el “fuzzy gain” no es de por sí una estructura similar a la de los controladores PID, se les ve como controladores PID no lineales que, similar a un circuito cerrado, depende de la señal de error y de sus diferencias en el tiempo. [10]

6.5. Método de daisy chain para control tolerante a fallas basado en asignación de control



(a) Estilo de conexión tipo paralelo.

(b) Estilo de conexión estilo daisy-chain.

Figura 10: Conexiones tipo paralelo vs. daisy-chain.

Se propuso el uso de un método de daisy chain modificado para el control de vuelo de un vehículo aéreo. El método de “control allocation” (asignación de control) hace uso de redundancia para reemplazar actuadores fallidos. El método original de daisy chain era el de dividir la entrada del sistema en diferentes grupos, de manera que cada grupo estuviera compuesto de un conjunto de actuadores y el proceso se lleva a cabo en orden. Sin embargo, este método de daisy chain presenta problemas debido a que cuando el proceso llegue al último actuador, si este grupo contiene algún actuador que presente problemas, ya no hay algún grupo que le siga que pueda corregir los errores resultantes generados por este grupo defectuoso. [11]

Este método utiliza actuadores inteligentes, los cuales traen incorporados sistemas de diagnóstico local y compensación local, que permiten detección de fallos en los actuadores y además, comunicación bi-direccional. Considerando esto, la modificación propuesta fue la de utilizar los sistemas incorporados en los actuadores inteligentes y, en base a lo detectado, modificar el “control allocation” de manera que se calculen los valores de los actuadores con falla de primero, lo que permite que los actuadores que le sigan puedan ajustar el error que no logró ajustar el actuador con falla. [11]

6.6. Motores paso a paso (stepper)

Los motores stepper se pueden considerar un motor eléctrico sin conmutadores. Típicamente el bobinado forma parte del estator, mientras que el rotor está compuesto de un imán permanente adjunto a un eje.

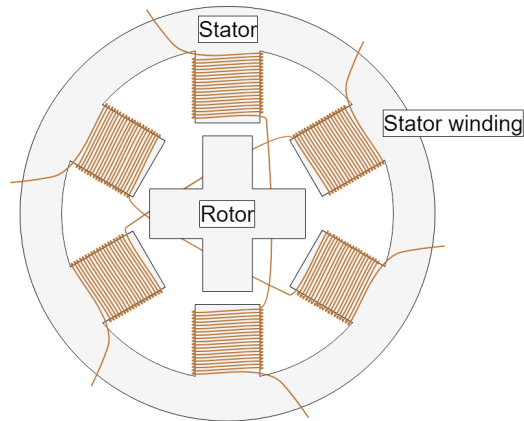


Figura 11: Sistema interno de un motor paso a paso [12].

Para utilizar el motor, comúnmente se utiliza un controlador, el cual se encarga de generar las conmutaciones de manera externa. Esto es lo que le permite al motor girar y mantener su posición fija en los ángulos deseados, al igual que cambiar dirección de rotación. Los motores paso a paso, a comparación de motores eléctricos convencionales que requieren una fuente constante de voltaje para poder funcionar, requieren una señal PWM, lo cual genera pulsos eléctricos a un periodo de tiempo constantes. Son estos pulsos, el motor ejecuta un movimiento de un ángulo exacto, denominado un “paso”. Es debido a esta habilidad por la que estos motores se utilizan para operaciones que requieren cierto nivel de precisión. Los motores más comunes son los que cada paso ejecutado genera un desplazamiento angular de 1.8° , aunque con la habilidad de establecer un “*microstepping*”, dicho desplazamiento puede ser menor y mucho más preciso. [13]

Los motores paso a paso generalmente se describen con las siguientes características eléctricas:

- Voltaje: todo motor paso a paso tiene establecido un voltaje de operación. Dicho valor es el valor recomendado de funcionamiento pero hay casos en donde es necesario sobrepasar dicho voltaje para poder incrementar el torque, pero a cambio de mayor calor generado o incluso una disminución en la vida útil del motor. [13]
- Resistencia: cada motor paso a paso tiene su propia resistencia por bobinado, la cual determina la corriente que va a aguantar o va a extraer de la fuente, al igual que la curva de torque y velocidades de operación del motor. [13]
- Grados por paso: esta característica es considerada comúnmente como una de las más importantes al seleccionar un motor paso a paso, dado que es la característica que determina la resolución de dicho motor. Determina el número de pasos requeridos para que el motor genere una revolución completa en modo “*full – step*”. Existen motores de 0.72 , 1.8 , 3.6 , 7.5 , 15 , e inclusive 90 grados por paso. [13]

Los motores paso a paso se pueden subdividir en tres diferentes tipos de motores: unipolares, bipolares y de reluctancia variable. [13]

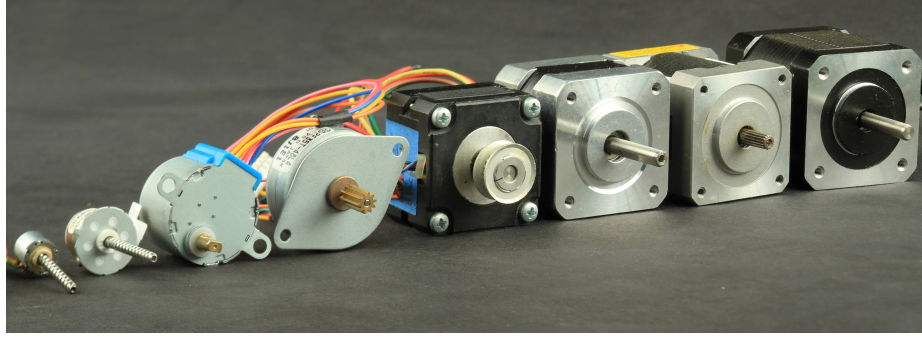


Figura 12: Variedades de motores paso a paso[14].

6.7. Modelado matemático de un motor stepper

El modelado matemático de un motor stepper se puede sub-dividir en dos categorías adicionales, que son los siguientes [15]:

- Modelado eléctrico
- Modelado mecánico

Cada método tiene su distinción en cuanto a las funciones y expresiones a utilizar. En el caso del modelo eléctrico, cada una de las fases del motor se puede representar como un circuito RL que adicionalmente contiene una fuerza electromotriz (emf). Las ecuaciones que representan cada fase (A y B) y la dinámica mecánica se ven a continuación [15]:

$$\frac{di_j(t)}{dt} = \frac{1}{L} * (u_j(t) - R * i_j(t) - e_j(t)) \quad (1)$$

Donde $j = a, b$, R es la resistencia de fase, L es la inductancia de fase, $i_j(t)$ es la corriente de fase, $u_j(t)$ es el voltaje en las terminales, y el elemento $e_j(t)$ se describe con las siguientes ecuaciones [15]:

$$e_a(t) = -K_m * \omega_m * \sin(p * \theta) \quad (2)$$

$$e_b(t) = K_m * \omega_m * \cos(p * \theta) \quad (3)$$

En donde K_m se conoce como la constante del motor, p es el número de parejas de polos (dentadura del rotor del stepper), ω es la velocidad rotacional del rotor y θ es el ángulo de posición del motor. La dinámica mecánica del motor se representa con la siguiente ecuación [15]:

$$J * \frac{d\omega_b}{dt} = (\tau_{em} - B * \omega_m - \tau_{dm} - \tau_l) \quad (4)$$

En donde J representa el momento de inercia del motor, B representa el coeficiente de viscosidad, ω_m representa la velocidad angular del motor y los valores de τ_{em} , τ_{dm} y τ_l son [15]:

$$\tau_{em} = K_m * (-i_a * \sin(p * \theta) + i_b * \cos(p * \theta)) \quad (5)$$

$$\tau_{dm} = T_{dm} * (\sin(2 * p * \theta + \phi)) \quad (6)$$

Donde τ_{em} es el torque electromagnético, K_m es una constante del motor, τ_{dm} es el torque de retención, T_{dm} es la amplitud del torque de retención, ϕ es un desfase asociado al torque de retención y τ_l es el torque generado por fuerzas externas. [15] [16]

6.8. VarioGuide de Brainlab

VarioGuide es un sistema de seguimiento de trayectorias en donde el médico puede fácilmente seguir una trayectoria previamente establecida con retroalimentación en cuanto a la posición en la que se encuentra el instrumento utilizado y un asistente informático que provee apoyo. Se le considera, por Brainlab, la alternativa que existe entre el uso del marco estereotáctico para la cabeza del paciente y la operación a mano alzada. Este instrumento ofrece flexibilidad en cuanto al tamaño de los instrumentos utilizados en caso de operaciones de biopsia, permite que el instrumento alcance rápidamente el punto final al cual se quiere llegar si se utiliza la aguja de biopsias precalibrado que incluye el dispositivo. [17]



Figura 13: Sistema de ubicación y alineación VarioGuide [17]

6.9. Protothreads

Los Protothreads son un sistema que utiliza hilos, o sus denominados “*threads*” para simular lo que en sistemas de mayor memoria y potencia computacional se define “*multithreading*”.

En lo general describe el uso de estos hilos para permitirle al microcontrolador ejecutar varias tareas “al mismo tiempo”, simulando el multithreading. Es una técnica creada por Adam Dunkels, que hace uso de programación en C pura (lo cual habilita su uso sin la necesidad de código de ensamble específico de la máquina) para poder permitirle al microcontrolador llevar a cabo diferentes “secciones” de código a la vez, lo cual remueve la limitación que tiene el estilo de ejecución lineal que tienen los microcontroladores. Dado que solamente requiere 2 bytes de memoria por protothread, a comparación de la cantidad de memoria que se tendría que apartar para otros métodos, es un método viable y práctico para programación en forma de bloques. [18]

Los protothreads se pueden ver como una combinación de eventos y de threads ya que contienen características de ambos. De los eventos deriva la habilidad de mantenerse liviano, lo cual, como se mencionó anteriormente, es un beneficio que ofrece para dispositivos que están limitados en cuanto a la memoria que tienen disponible. En cuanto a los threads, deriva la habilidad de generar “bloqueos”, un proceso convencional en la programación lineal en donde primero se termina de ejecutar una función antes de iniciar otra. [18]

Cabe mencionar que, a pesar de que esta técnica para ejecución de programas permite flexibilidad en cuanto a la programación de microcontroladores, es una técnica que tiene limitantes. Entre ellas se encuentran las siguientes [19] [18]:

- Se pueden utilizar dentro de funciones de C pero no puede llamar a funciones externas debido a que no podría ejercer su funcionalidad de “threading”.
- Las variables automáticas - variables locales que asignan y deasignan memoria cuando entran y salen de su función - no son compatibles con el uso del protothreading dado que como se inicializan cada vez que se entra al thread, es posible que estas variables no se inicialicen de manera correcta. Una alternativa de esto es crear y mantener dicha variable como estática fuera del protothread para asegurar que va a asignarse a un bloque de memoria de manera permanente.
- Los protothreads funcionan muy similar a interrupciones en programación con C, por lo que de la misma manera en la que no es conveniente y recomendado el agregar demasiados procesos dentro de una interrupción, lo mismo aplica con un protothread. Es debido a esto que, en momentos en donde se considere que el tiempo de procesamiento podría ser demasiado para completar una tarea, es recomendado acortar dicha tarea en varias tareas menores.

6.10. Dynamixel XL-320

El Dynamixel XL-320 es un actuador creado y diseñado por la compañía ROBOTIS. A diferencia de un servomotor común utilizado para proyectos de pasatiempo, sus especificaciones son superiores, al igual que su precisión en cada giro. Entre las características que ofrece el actuador están [20]:

- Resolución de 0.29°
- Dos modos de operación: modo de junta (0 - 300°) y modo de rueda (giro ilimitado)

- Relación de engranajes de 238 : 1
- Stall torque de 0.39Nm @ 7.4V, 1.1A
- Velocidad sin carga de 114rpm @ 7.4V, 0.18A
- Protocolo UART (Universal Asynchronous Receiver Transmitter)
- Control PID integrado
- Sensor de posición (encoder)

Para operarlo, el dispositivo hace uso de paquetes de datos, ya sea en modo lectura o modo escritura en donde el operador decide cuál de los dos quiere enviarle al dispositivo. La tabla de control se divide en dos áreas de memoria: EEPROM y RAM. Cada una de estas áreas contiene su conjunto de instrucciones en donde algunos está restringidos a solamente lectura (es decir, para obtener datos del dispositivo) mientras que otros permiten tanto lectura como escritura (la habilidad de editar el dato para cambiar el comportamiento del dispositivo). [20]



Figura 14: Dynamixel XL-320

6.11. Monitoreando diferentes sensores con microprocesador Atmega328

El microcontrolador AVR original fue desarrollada en un local ASIC localizado en Trondheim, Noruega, conocido como Nordic VLSI en esos momentos, aunque hoy en día se encuentra bajo el nombre de Nordic Semiconductor, por dos miembros, Alf-Egil Bogen y Vegard Wollan. Dicha serie AVR se conocía en ese entonces como μ RISC y era considera solamente un IP de silicón de la compañía. Cuando Atmel compró la esta tecnología, Goben y Vegard continuaron su desarrollo con Atmel Norway, un subsidiario de Atmel ubicado en Noruega.

El desarrolló combinó esfuerzos de escritores de compiladores, al igual que diseñadores para garantizar que el AVR pudiera compilar lenguajes de alto nivel de manera eficiente. [21]

Hoy en día, los AVRs se clasifican bajo lo siguientes dispositivos: [21]

- tinyAVR - La serie ATtiny
- megaAVR - La serie ATmega
- XMEGA - La serie ATxmega
- AVRs para aplicaciones específicas - dispositivos megaAVR que contiene características específicas no encontradas en otros miembros de la familia AVR como controladores LCD, controladores USB, PWM avanzados, entre otros.
- FPSLIC - dispositivos AVR con FPGA.
- AVRs de 32-bits.

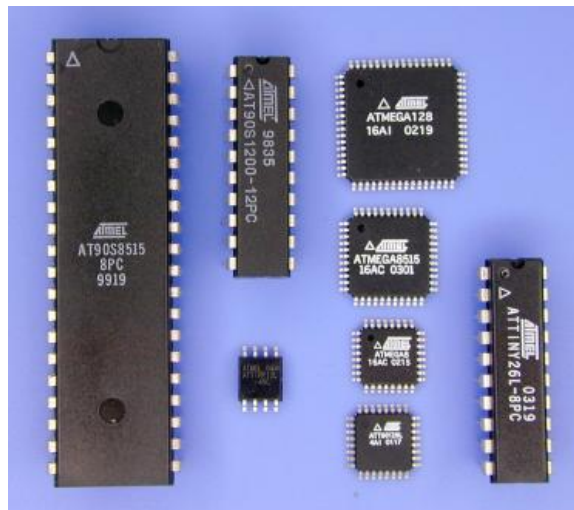


Figura 15: Microcontroladores de la familia AVR [22].

El ATmega328 es un microcontrolador de 8-bits de un solo chip, creado por Atmel dentro de la serie megaAVR que contiene la siguientes características: [21]

Parámetro	Valor
Tipo de CPU	8-bits AVR
Rendimiento	20 MIPS a 20 MHz
Memoria flash	32 kB
Memoria SRAM	2 kB
Memoria EEPROM	1 kB
Número de pines	28 pines PDIP, MLF, 32 pines TQFP, MLF
Frecuencia máxima de operación	20 MHz
Número máximo de pines I/O	26
Otras características	3 temporizadores/contadores con modos de comparación, interrupciones internas y externas, comunicación serial USART programable, puertos SPI, ADC de 10 bits, oscilador interno.

Cuadro 3: Especificaciones del ATmega328

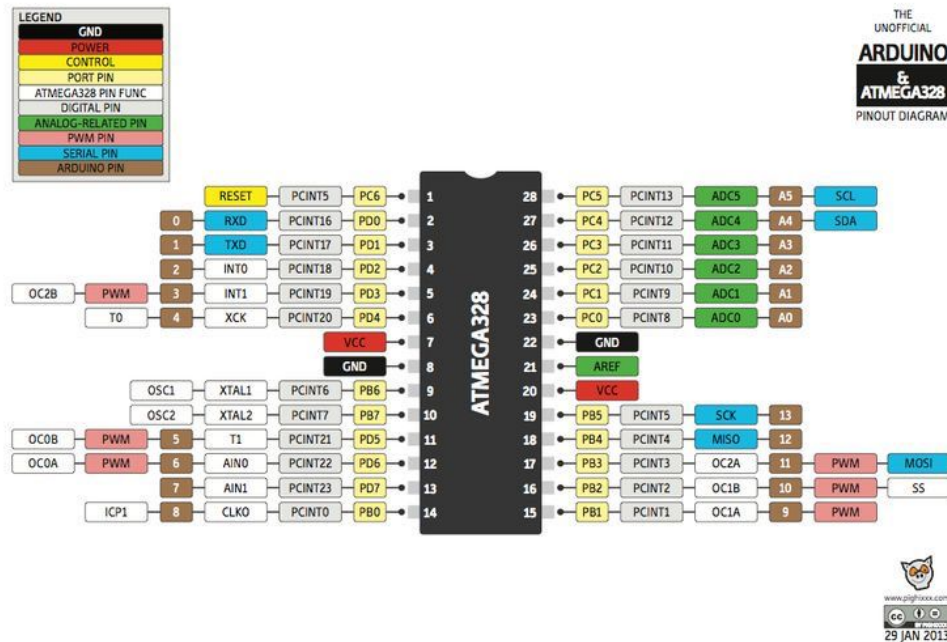


Figura 16: Diagrama de entradas y salidas del ATmega328

Uno de los tableros de microcontrolador más populares que integran al ATmega328 es el Arduino Uno. Este dispositivo contiene 14 pines de I/O digitales, 6 de los cuales proveen la habilidad de ser utilizados como salidas PWM, 6 entradas analógicas, un oscilador de 16 MHz interno, una entrada USB y una entrada de poder tipo barril. Para poder programar el microcontrolador, es necesario utilizar un *bootloader*, el cual es un software que permite que el la tabla del Arduino se comunique con el Arduino IDE y que se carguen los *sketches* de la IDE al Arduino. Por lo general, cuando se carga un programa al microcontrolador, un programador externo es necesario, como el Arduino ISP, sin embargo, el *bootloader* elimina la necesidad de dicho programador dado que el protocolo que permite que la computadora

programe la memoria flash del AVR se encuentra integrada dentro del *bootloader*. [21]

Entre los sensores que se van a utilizar se encuentran un acelerómetro, un sensor infrarrojo pasivo de (PIR) de movimiento, y un sensor ultrasónico. [21]

El acelerómetro es un dispositivo que mide la aceleración propia. Es importante no confundir esto con aceleración coordinada (tasa de cambio de la velocidad). Por ejemplo, si un acelerómetro se encuentra en reposo sobre la superficie de la Tierra, este va a medir una aceleración debido a la gravedad de la Tierra positiva (por definición), es decir, $g = 9.81m/s^2$. [21]

El sensor infrarrojo pasivo (PIR) de movimiento es sensible a la temperatura de la piel de una persona a través de radiación a longitudes de banda ubicados a mediados de la longitud de banda del infrarrojo, a comparación de objetos a temperatura ambiente. Este sensor no emite ningún tipo de energía, por lo que se denomina pasivo. Dicho de otra manera, el sensor infrarrojo detecta niveles de radiación infrarroja, lo que significa que mientras más caliente un objeto, más radiación emite. El sensor en un detecto de movimiento se encuentra dividido en dos, de manera que cuando uno detecta más radiación que otro, la salida del sensor se torna en *high* y si esa misma división detecta menor radiación, se torna en *low*. [21]

El sensor ultrasónico es un sensor que puede medir la distancia hacia un objeto utilizando ondas de sonido. Esto lo hace enviando una onda de sonido a una frecuencia específica y esperando recibirla de vuelta. Durante todo este trayecto, el sensor va midiendo el tiempo transcurrido desde emisión hasta recepción y utiliza este tiempo para calcular la distancia. [21]

6.12. Condiciones de uso para los GPIO tolerantes de 5-V de los microcontroladores Tiva series TM4C123x

Los microcontroladores TM4C123x contienen tres tipos de I/O en cuanto a sus características de protección de descargas electroestáticas (ESD) y fuga de corriente [23]:

- I/Os de potencia (VDD, VDDA, VDDC, VBAT, GND, GNDA, GNDX)
- I/Os con protección de descargas electroestáticas tipo fail-safe (GPIOs, pines XOSCn, USBD+, USBD-). Existen excepciones.
- I/Os con protección a ESD pero no de tipo fail-safe (VREFA+, VREFA-, pines que no sean de potencia, GPIO, y XOSCn).

Los GPIOs tolerantes a 5V del microcontrolador incluyen limitadores internos (los cuales restringen el voltaje a un rango determinado) y, como se estipuló anteriormente, protección de ESD tipo fail-safe. Es imperativo mencionar que estos pines al ser configurados como entradas son tolerantes a 5V, no diseñados para 5V, lo que significa que al conectar 5V al pin, este reduce el voltaje internamente al rango establecido por VDD. Esto significa que el pin se encuentra a 5V, pero el sistema interno del microcontrolador está trabajando con la reducción de voltaje. [23]

Dependiendo del modo en el cual se configure el pin, el sistema de limitación trabaja de maneras diferentes: [23]

Configuración de GPIO	Mecanismo de limitación	Descripción
Función digital	Modo de entrada	El sistema de protección y limitación integrado del microcontrolador reduce el voltaje inyectado al pin GPIO (V_{IN}) a un voltaje interno aceptable.
	Modo de salida	El voltaje de salida del pin GPIO (V_{OUT}) está limitado entre $0V - V_{DD}$.
	Modo open-drain	Un resistor externo conectado entre una fuente externa de poder y el GPIO le permite al pin GPIO trabajar con señales mucho más grandes, aunque en este modo, la fuente de poder debe estar limitada a $5.5V$.
Función analógica	Modo entrada/salida	El limitador interno protege el circuito interno, permitiendo que se empleen voltajes mucho mayores a esos de V_{DD} en los pines GPIO. Sin embargo, el funcionamiento de las especificaciones analógicas no se garantizan si dicho voltaje se encuentra fuera del rango entre $0V - V_{DD}$.

Cuadro 4: Condiciones de uso de los pines GPIO dependiendo de su configuración. [23]

6.13. EGCL: Un Lenguaje de G-Code Extendido con Control de Flujo, Funciones y Variables Mnemotécnicas

La interacción entre los programadores y las máquinas CNC se lleva a cabo a través de lenguajes de maquinado. El lenguaje más comúnmente usado es el G-code, a pesar de que existen otros lenguajes como el APT o el STEP-NC. El G-code fue definido en los años 1960s por el estándar ISO6983. Se puede considerar un lenguaje de programación de bajo nivel ya que carece de variables de implementación, expresiones aritméticas y booleanas, control de flujo de estructuras y llamado de funciones. Actualmente, hay un cierto número de formas de producir G-code [24]:

- Programación manual.
- Programación usando lenguaje macro.
- Sistemas CAD-CAM.

Programación manual, que cubre la mayoría de las aplicaciones de la industria, es una técnica basada en el Lenguaje ISO G-code (Síglas IGCL en inglés). Este lenguaje es uno en donde cada movimiento a través de una secuencia de coordenadas debe ser especificado por una instrucción. Adicionalmente, el lenguaje G-code tradicional no permite la programación simbólica de coordenadas. Esto significa que, conforme se hacen cambios de diseño a la pieza o a la parte que ejecuta el movimiento, se pueden generar errores si no se reprograman las

coordenadas. Debido a esto, cada vendedor de herramientas de maquinado extiende el IGCL, creando sus propios macro-lenguajes agregándole variables numéricas, ciclos de maquinado y definiciones de control de flujo específicos a sus herramientas. En sistemas CAD-CAM, las instrucciones de maquinado son obtenidos de un modelo geométrico tridimensional y estrategias de maquinao definidas por el usuario, lo que resulta en que se genere un G-code específico para el proceso establecido. [24]

Desarrollo de los motores inteligentes

Se inició con el desarrollo de los motores inteligentes dado que estos iban a ser uno de los dos componentes principales para el funcionamiento del brazo robótico asistencial, el segundo siendo el programa de los microcontroladores. Para ello fue imperativo dar a entender los estándares con los cuales se iba a definir un motor inteligente para este proyecto. En este caso, se estableció como un motor inteligente aquel que se encuentre equipado con dispositivos como sensores, drivers y microcontroladores que le permita funcionar y comunicarse con otros motores, además de ejecutar comandos enviados por un usuario o microcontrolador maestro. Con esta definición establecida, se prosiguió a hacer la selección de los dispositivos requeridos para construir el motor inteligente.

7.1. Selección de microcontroladores para cada motor

Para la selección del microcontrolador lo primero que se hizo fue verificar el tipo de microcontrolador a utilizar con base en su número de bits. En este caso, se seleccionaron varios microcontroladores de 8-bits para poder garantizar su disponibilidad dentro del país. Adicionalmente, reduce los costos requeridos tomnando en cuenta que se requieren 8 microcontroladores para los actuadores. Las diferentes marcas que se tomaron en cuenta fueron los siguientes:

- Texas Instruments
 - TM4C
 - MSP
- Microchip Technologies
 - AVR

- PIC

Para ello se seleccionaron microcontroladores de cada categoría que se consideraron que podrían cumplir con las siguientes necesidades del proyecto:

1. Debe haber disponibilidad del microcontrolador dentro del país para evitar la necesidad de compras del extranjero.
2. Debe tener un amplio entorno de desarrollo. Es decir, debe tener una documentación amplia para evitar la necesidad de buscar documentación de terceros no comprobado.
3. El microcontrolador requiere suficiente memoria EEPROM para guardar las ganancias totales de cada actuador.
4. I/O apto para el número de conexiones a implementar en cada actuador.
5. En el caso del master, tener un conjunto extenso de puertos seriales UART para tener acceso a tres puertos totales.

Entre estos hay tres que se han utilizado anteriormente por lo que, por razones de familiaridad, se incluyeron en el trade study. La principal razón por la selección de los microcontroladores para el trade study fue debido a que sus características cumplían de manera justa con los requerimientos (es decir, no se sobre-exceden con lo requerido).

En la siguiente tabla se pueden ver los microcontroladores que se utilizaron para el trade study, con sus respectivas especificaciones:

uC	Reloj (MHz)	I/O (pines)	COM	Memoria de programación (B)	EEPROM (B)	RAM (B)	Voltaje de suministro (V)
ATMega328P	16	23	SPI (1) UART (1) I2C (1)	32,000	1024	2048	2.7 - 5.5
ATMega1608	20	41	SPI (1) UART (3) I2C (1)	16,000	256	2048	4.5 - 5.5
ATTiny3216	20	18	SPI (1) UART (1) I2C (1)	32,000	256	2048	4.5 - 5.5
TM4C123GH6PM	80	64	SPI (4) UART (8) I2C (4)	256,000	2048	32,000	3.3
MSP432P401R	48	84	SPI (2) UART (1) I2C (1)	256,000	32,000	64,000	1.62 - 3.3
PIC16F887	8	35	SPI (1) UART (1) I2C (1)	8192	256	368	2.0 - 5.5
PIC16F19186	32	43	SPI (1) UART (2) I2C (1)	28,000	256	2048	2.3 - 5.5
PIC18F55K42	64	43	SPI (1) UART (2) I2C (2)	128,000	1024	8,000	2.3 - 5.5

Cuadro 5: Especificaciones de varios microcontroladores

Se puede ver que dentro del trade study se incluyó el microcontrolador TM4C123GH6PM a pesar de que este es de 32-bits. Esto se debe a que fue considerado en la fase anterior y se reconsideró para esta fase, pero no para los actuadores sino que para funcionar como el maestro. Para el trade study se establecieron los siguientes parámetros, con su peso y su porcentaje respectivo:

Criterios	Peso	Porcentaje	Descripción
Costo	2	0.0769	Para la comparación de costos, se utilizó DigiKey.
Disponibilidad	3	0.1154	Se refiere a si se puede conseguir localmente o el envío no tarda demasiado.
Familiaridad	4	0.1538	Se refiere a qué tanto conocimiento sobre el dispositivo se tiene para programarlo.
Conveniencia	4	0.1538	Contiene los requerimientos de características que se exige (asumiendo que viene dentro de un launchpad).
Capacidad	5	0.1923	Memoria EEPROM.
Puertos	5	0.1923	Puertos I/O que tiene disponibles.
Durabilidad	3	0.1154	Se refiere a cuál es más propenso a dañarse en caso de conexiones erróneas.
Total	26	1	

Cuadro 6: Parámetros de comparación del trades study de microcontroladores

La tabla de estudio de microcontroladores resultó siendo la siguiente:

uC	Costo	Disponibilidad	Familiaridad	Conveniencia	Capacidad	Puertos	Durabilidad	Promedio	Con peso
ATMega328P	3	3	5	3	5	5	3	3.8571	4.0769
ATMega1608	4	1	1	3	5	5	3	3.1429	3.3077
ATTiny3216	5	1	1	3	5	4	3	3.1429	3.1923
TM4C123GH6PM	3	1	4	3	5	5	3	3.4286	3.6923
MSP432P401R	1	1	1	3	5	5	3	2.7143	3.0769
PIC16F887	3	3	3	3	4	5	3	3.4286	3.5769
PIC16F19186	4	1	1	3	5	5	3	3.1428	3.3077
PIC18F55K42	4	1	1	3	5	5	3	3.1428	3.3077

Cuadro 7: Trade study de microcontroladores

El estudio del Cuadro 7 concluyó que el microcontrolador más apto para ir adjunto a los motores paso a paso es el ATMega328, el cual se encuentra instalado en el *launchpad* Arduino Uno. Entre los microcontroladores comparados para funcionar como el maestro, el que resultó siendo más viable utilizar fue el TM4C123GH6PM, el cual se encuentra instalado en el *launchpad* Tiva C.

7.2. Selección de drivers para los motores

Al igual que para la selección de los microcontroladores, para la selección de los drivers se hizo un trade study. Entre las diferentes marcas que se utilizaron están los siguientes:

- Texas Instruments (Serie DRV)
- Toshiba
- Trinamic
- Monolithic Power Systems

Drivers	Rango de voltaje (V)	Corriente continua/fase (A)	Corriente máx (A)	Microsteps
A4988	8 - 35	1	2	1/16
DRV8825	8.2 - 45	1.5	2.5	1/32
DRV8425	8.2 - 33	1.4	3.2	1/256
TMC2208	4.75 - 36	1.4	2	1/32
TMC2209	4.75 - 29	2	2.8	1/64
MP6500	4.5 - 35	1.5	2.5	1/8
TB67S279-FTG	10 - 47	1.1	2	1/32
TB67S249-FTG	10 - 47	1.6	4.5	1/32
TB67S128-FTG	6.5 - 44	2.1	5	1/128

Cuadro 8: Especificaciones de los drivers de motores paso a paso

La tabla anterior contiene datos obtenidos directamente de las hojas de datos de cada uno de los drivers, al igual que información procurada de proveedores como Pololu.

El trade study hecho para la selección de drivers utilizó los mismos aspectos para la comparación, pero se ajustó lo que representaba cada aspecto para los drivers. Se generó una tabla para explicar dicha representación y lo que conlleva cada una de las características utilizadas muy similar al Cuadro 7.

A pesar de que al final del trade study los drivers más flexibles para implementar en el proyecto fueron el TB67S128FTG (propuesto en una fase anterior) para los motores más grandes y el TMC2209 para los motores más pequeños, se utilizó el driver A4988 para hacer las pruebas preliminares con un solo motor y luego con dos. Esto se debe a la problemática que tenemos actualmente tanto por la disponibilidad local, como por la dificultad y tiempo requerido al obtenerlos importados de otro país. Sin embargo, cabe mencionar que el driver a utilizar provee las características necesarias para llevar a cabo las pruebas de movimiento de con el motor nema 17. Adicionalmente, es necesario aclarar que, a pesar de que se estableció el driver preferible a través del trade study, por las razones mencionadas anteriormente se van a utilizar los drivers que ya se habían obtenido en la fase anterior, que son los drivers DRV8825 para los motores más cercanos al efector final, y TB67S128FTG y TB67S249FTG para el motor en la junta 1 y el motor para la junta 4 respectivamente.

7.3. Calibración preliminar con motor Nema 17HS4401S para pruebas

Antes de trabajar con los motores se hicieron pruebas de calibración y de movimiento con un motor Nema 17HS4401S. Dado que estas son pruebas preliminares, se utilizó un driver con fácil acceso para experimentar con la calibración del motor, tanto de los voltajes y las corrientes a utilizar. El esquema que se utilizó se encuentra en la Figura 17:

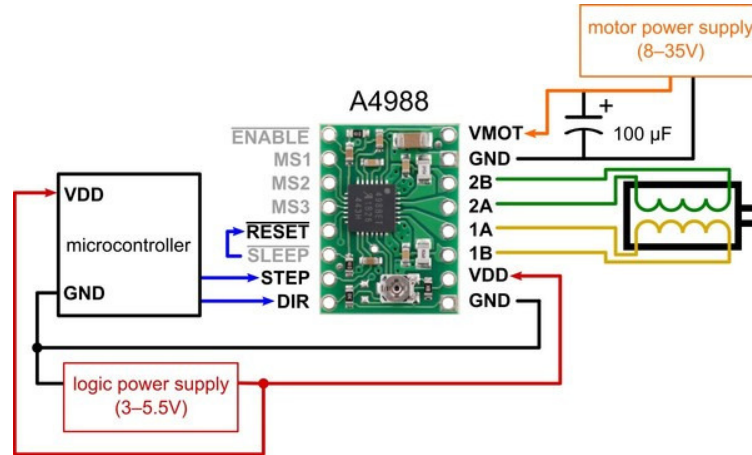


Figura 17: Esquema de conexión del driver A4988. [25]

Estas conexiones se hicieron en un “*protoboard*” (placa o tabla de prototipado), en donde la fuente de alimentación de 5V provino de la placa de desarrollo Arduino Uno y la alimentación de 12V provino de una fuente de poder de banco calibrado con una limitación de corriente de 0.1A. Se decidió que se iba a usar una corriente de 0.5A para conducir el motor stepper, el cual representa aproximadamente el 25% de la corriente que puede tolerar el driver. Conociendo la corriente con la cual se deseaba conducir el motor, se utilizó la siguiente fórmula para calcular el voltaje de referencia del driver, el cual se calibraría ajustando el potenciómetro en la placa:

$$V_{ref} = I * 8 * R_1 \quad (7)$$

Cabe mencionar que esta fórmula fue derivada exclusivamente para su aplicación con drivers A4988, por lo que si se llegara a utilizar un driver de diferente marca esta fórmula va a cambiar. El valor de R_1 representa la resistencia interna del driver. En este caso, con una corriente de 0.5A, y una resistencia interna de 0.1Ω , se obtuvo un voltaje de referencia de 0.4V, el cual se ajustó utilizando un multímetro.

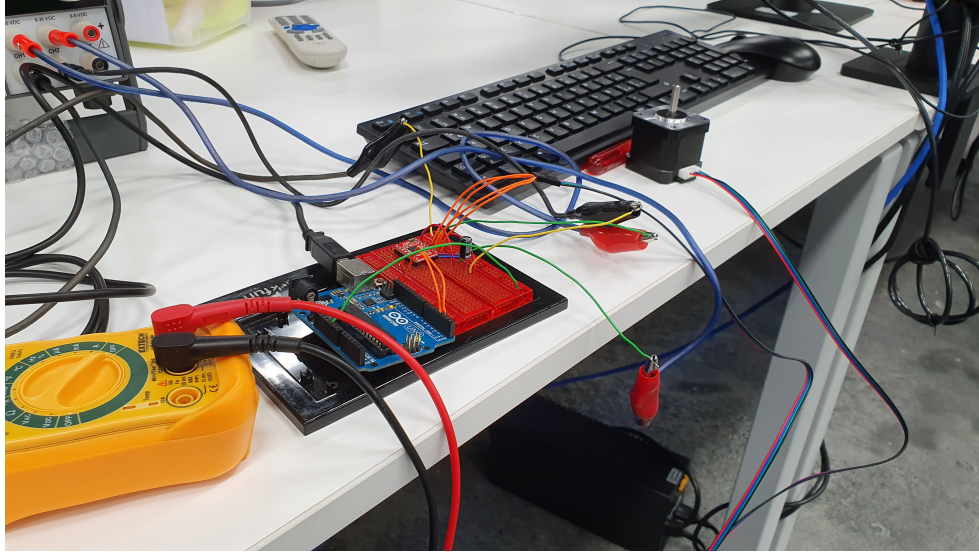


Figura 18: Conexión de los componentes en una protoboard.

Debido a razones de disponibilidad actual, se utilizó el motor Nema 17HS4401S para hacer las pruebas, pero no se recomienda utilizar este mismo motor para usarse en las juntas del brazo robótico asistencial debido a su poca capacidad de torque.

7.4. Desarrollo de un protocolo de comunicación para los actuadores

Para el desarrollo del protocolo de comunicación entre los actuadores, al igual que con la calibración del motor stepper, se hicieron pruebas preliminares para verificar la manera en la que los datos se estaban enviando a través de comunicación serial entre microcontroladores. Para ello se fue escribiendo un programa sencillo en Arduino en donde se comenzó con transmisión y recepción de un dato a la vez, para verificar que la comunicación entre microcontroladores fuera exitosa. De ahí, se fue incrementando la cantidad de información que se quería transmitir y a la vez se fueron agregando los comandos que se habían establecido para permitirle a los microcontroladores ejecutar las acciones deseadas. Para lograr visualizar este envío de datos, se utilizó el programa PuTTY, una ventana terminal que actúa como un cliente para el protocolo de comunicación serial que se está implementando en este proyecto.

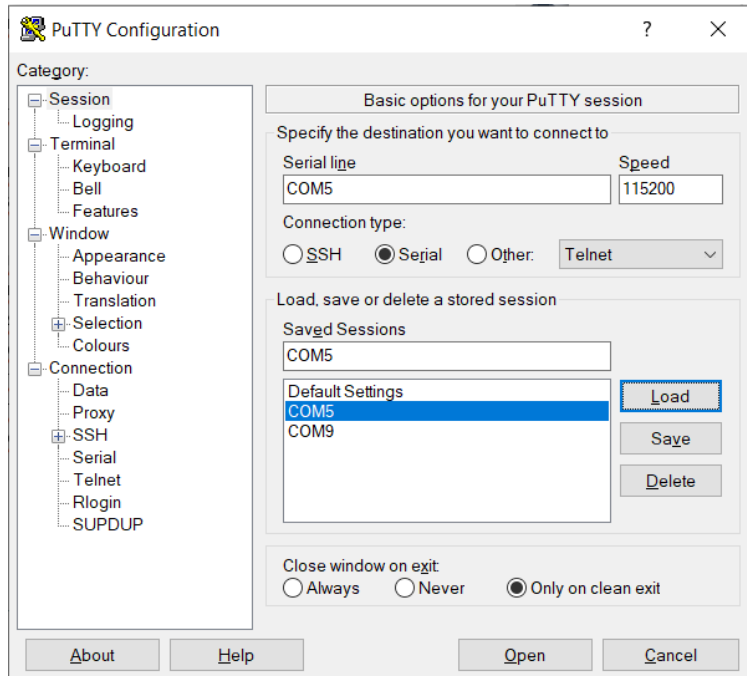


Figura 19: Pantalla de inicio y de configuración de PuTTY

Más adelante, se hizo el cambio de PuTTY a Hércules, debido a que este brindaba mucha mayor conveniencia en la conexión y desconexión de los puertos seriales (no se cerraba la ventana terminal al cortar la comunicación con el puerto serial).

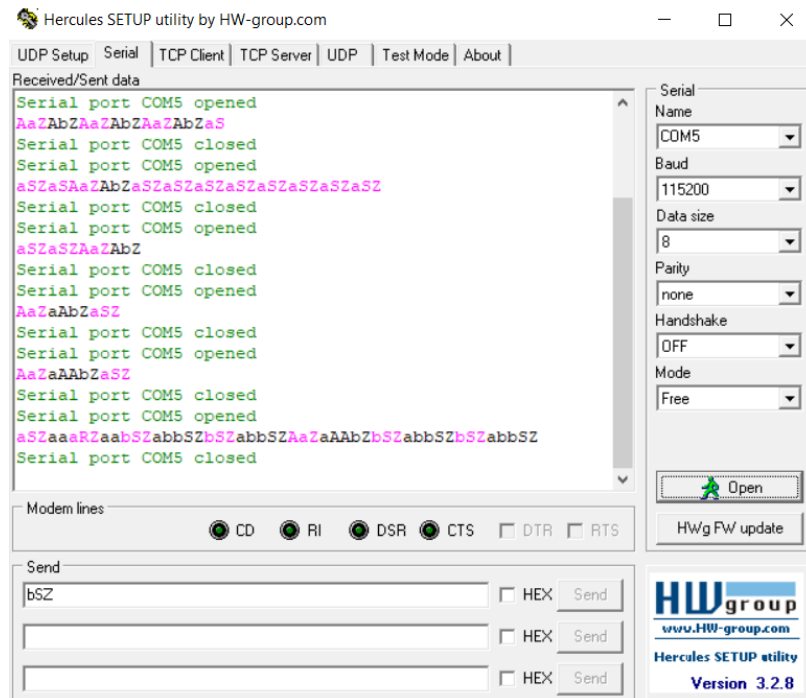


Figura 20: Interfaz de la aplicación de utilidad Hércules.

Se trabajó primero los comandos de activación de motores, el cual se implemento con un sistema de restado. Es decir, se utilizó un sistema en donde el usuario puede ingresar un número de motores que desea activar y ese número de motores va a ir decrementando conforme pase a través de cada uno de los microcontroladores de los motores. Cuando el número enviado por el usuario llegue a 1, el último microcontrolador va a enviar un 0 al microcontrolador maestro y este, al recibir el 0, va a saber que cada uno de los motores habrá sido activado. Se diseñó un esquema para explicar esto de manera visual.

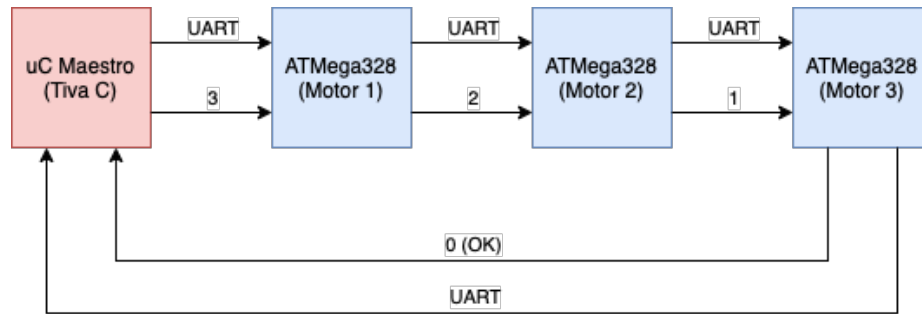


Figura 21: Sistema original de nombrado de microcontroladores.

Sin embargo, conforme se fueron haciendo las pruebas, se vio que el método numérico que se estaba implementando generaba confusión al ser implementado en conjunto con el sistema de movimiento de pasos. Adicionalmente, dado que solamente se permitía el uso de los números del 1 al 9, esto limitaba el número de motores que se puede implementar, por lo que si en un futuro se quisiera utilizar más sería imposible. Se consideró que en lugar de necesitar conocer el número total de motores empleados era mucho más sencillo para el usuario comenzar desde el número inicial (por lo general, 1 o, en caso del nuevo sistema, “a”). De esta manera se hizo la transición del sistema anteriormente establecido con números a un sistema nuevo utilizando las letras del alfabeto en minúscula, desde la “a” a la “z”, que concuerdan con los números 1 al 26 respectivamente. Es esquema de funcionamiento cambió al siguiente:

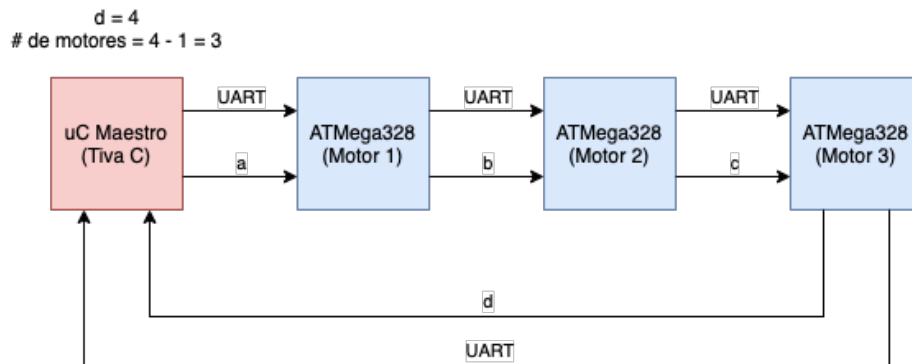


Figura 22: Sistema actualizado de nombrado de microcontroladores.

Ya habiendo establecido este nuevo sistema, se comenzó a implementar el guardado de

nombres en la EEPROM de cada uno de los arduinos utilizando la librería de EEPROM propio de Arduino.

Con el sistema de nombramiento establecido, se integró el código creado anteriormente de la calibración de motores para hacer pruebas. Cabe mencionar que, debido a los diferentes cambios en cuanto al método que se utilizó para seleccionar el microcontrolador con el que se quería comunicar, se tuvo que hacer ajustes al código original de movimiento de los motores paso a paso.

Por la función utilizada para el envío de datos, estos son transmitidos en forma de caracteres ASCII. Debido a esto, fue necesario reconfigurar dichos caracteres en números enteros para permitirle a los motores interpretar de manera adecuada el número de pasos que debe dar (para poder tener hasta 65535 pasos). Para ello se utilizó un método de separación, conversión y cálculo explicado a continuación:

1. Se extrajo cada uno de los dígitos obtenidos a través de la comunicación serial como bytes separados y guardarlos dentro de un array.
2. Se convirtió cada uno de los elementos de dicho array de un ASCII a un integer de manera que se pudieran trabajar con operaciones aritméticas.
3. Se ejecutó un bucle de tipo *for* en donde se multiplicó cada uno de los elementos por un múltiplo de 10 (hasta 10000) que se encontraba establecidos ya dentro de un array anteriormente inicializado.
4. Se sumaron todos los elementos convertidos y multiplicados para obtener un solo integer.
5. Por último, se asignó el integer resultante a una variable anteriormente inicializada, encargada de los pasos del actuador.

Para poder controlar la dirección de rotación del motor stepper, se aseguró de enviar un símbolo positivo o negativo para permitirle al microcontrolador determinar si el motor debe rotar en dirección de las agujas del reloj u opuestas a estas. Este símbolo se colocó a una posición antes del primer dígito. De esta manera, el código completo ser vería de manera siguiente: $aS + 12345Z$, para el caso del comando de asignación de número de pasos.

Comando	a	S	+	1	2	3	4	5	Z
Bits	X	0	1	2	3	4	5	6	X

Figura 23: Comando de pasos y el número de bits total del comando.

Conforme se fueron agregando comandos, surgieron dos complicaciones en el sistema:

1. El sistema era muy poco intuitivo debido a que solamente ofrecía una letra mayúscula para definir y generar los comandos. Las letras que se utilizaban representaban la primera letra del comando a efectuar ya fuera en inglés o español para poder mantener concordancia entre la letra utilizada y el comando. Sin embargo, debido a esto, surgieron situaciones en las que se vio la necesidad de repetir el uso de una letra conforme se agregaron comandos adicionales y debido a la cantidad limitada de letras,

esto resultaba en un entrecruce de comandos.

2. El número de pasos máximo todavía era muy pequeño tomando en consideración el sistema de relación de engranajes planetarios que se utilizó mecánicamente para el brazo robótico, en conjunto con la caja reductora de engranajes que va adjunto a dos de los motores. Esto resultaba en la necesidad de un número considerablemente grande de pasos a una velocidad considerablemente rápida para que el brazo pudiera generar algún movimiento visible.

Para resolver la primera problemática, se implementó un sistema de comandos diferentes, el cual consiste de un comando general y un comando específico. Esto permite el uso de las mismas letras mayúsculas para los comandos siempre y cuando sean asignadas bajo comandos generales diferentes. De esta manera, el usuario puede seleccionar entre tres diferentes categorías de comando general, y dentro de estas categorías seleccionar un comando específico:

1. Comando general de **Configuración**.
 - Asignación de dirección (address).
 - Establecimiento de un cero relativo.
 - Asignación de microstepping.
2. Comando general de **Movimiento**.
 - Ejecución de pasos (steps).
 - Parada de ejecución de pasos.
3. Comando general de **Diagnóstico**.
 - Obtención de pasos anteriormente dados para un actuador específico.
 - Obtención de la posición relativa actual del actuador deseado (habiendo previamente establecido un cero relativo propio o usando el cero del motor al momento de inicializar).

Para resolver la segunda problemática, se hizo un cambio de tipo de variable para guardar el número de pasos de un *unsigned int* a un *unsigned long*. Esto permite que el motor pueda ejecutar hasta 4,294,967,295 pasos. Las modificaciones hechas resultaron en los siguientes cambios en algunos de los comandos:

Sistema antiguo	Comando de pasos	aS+23000Z	El comando S le instruía al motor el moverse el número de pasos descritos después del signo.
	Comando de detener el motor	aP	Se había propuesto utilizar el comando P originalmente para no generar conflictos con el comando de pasos en inglés (Step [S] y Stop [S]).
	Comando de obtención de posición relativa del motor deseado	aPZ	El comando para posición, tanto en inglés como en español iban a utilizar el comando P, por lo que se generaría un conflicto.
Sistema actualizado	Comando de pasos	aMS+1200300400Z	Se establece que es comando de movimiento, seguido del comando S de ejecución de pasos.
	Comando de detener el motor	aMP	Se establece que es comando de movimiento, seguido del comando P de detener la ejecución de pasos.
	Comando para obtener posición relativa del motor	aDPZ	Se establece que es comando de diagnóstico, seguido del comando P de obtención de posición relativa del motor deseado, esta vez sin conflictos.

Cuadro 9: Comparación del sistema de comandos anterior y actualizado.

No se demostraron los cambios en todos los comandos, sino que se presentaron aquellos que podrían haber tenido o tuvieron conflicto por el uso de una misma letra para definir el comando.

7.5. Listado de comandos programados para los motores revisado

Para este proyecto, los motores se diseñaron principalmente para generar los movimientos de las juntas del brazo robótico, por lo que los comandos a utilizar son pocos. Sin embargo, dado que se programaron los motores para que pudieran utilizarse también para otros propósitos, ya sea para llevar a cabo proyectos diferentes o incluso para fines de exhibición, se les incorporó comandos adicionales que podrían ser de mucha utilidad en cuanto a su funcionamiento.

- Bajo la categoría de CONFIGURACIÓN, se encuentran los siguientes comandos:

- **Comando de asignación de dirección.** Este comando es para poder asignarle una dirección o “nombre” a cada uno de los actuadores para poder diferenciarlos y permitirle al usuario enviar comandos individuales a cada uno.
 - **Comando para establecer un cero relativo como referencia.** Este comando funciona para que el usuario pueda establecer un cero relativo, es decir, un cero de referencia sin importar en qué posición se encuentre el eje del actuador, de manera que desde ese punto de referencia pueda conocer cuántos grados ha girado el motor después de ejecutar un comando de pasos.
 - **Comando para establecer nivel de microstepping.** Este comando le permite al usuario establecer qué nivel de microstepping desea utilizar, desde paso completo (full-step) hasta un paso de 1/32 (utilizando drivers DRV).
- Bajo la categoría de MOVIMIENTO, se encuentran los siguientes comandos:
 - **Comando de pasos.** Este comando es el que se utiliza para que el usuario le pueda ordenar al actuador deseado el número de pasos y dirección a la cual desea que rote.
 - **Comando de paro de giro.** Este comando se utiliza para detener el movimiento del motor antes que termine de dar el número de pasos enviado.
 - Por último, bajo la categoría de DIAGNÓSTICO, se encuentran los siguientes comandos:
 - **Comando para obtener el número de pasos anteriormente asignado a un actuador específico.** Este comando es para que el usuario pueda saber cuál fue el número de pasos anteriormente asignado a un actuador específico. Cabe mencionar que este comando solo obtiene el primer valor de pasos que se le estableció al motor, es decir, si a medio movimiento el usuario le asigna un número diferente de pasos, a pesar de que el motor va a considerar este nuevo valor, no se verá reflejado con este comando.
 - **Comando para obtener la posición relativa del eje del actuador.** Este comando le permite al usuario obtener la posición relativa en la que se encuentra el actuador, calculado en base al microstepping establecido anteriormente (o en caso de no haber establecido un microstepping, se utiliza paso completo). Este comando se utiliza en conjunto con el comando de establecer un cero relativo.

Comando	Sintaxis	Uso	Descripción	Consideraciones
Asignación de dirección	CA[dir]Z	CAaZ	Comando para asignarle una dirección [dir] a cada actuador, desde la letra “a” hasta la letra “z” MINÚSCULAS.	El sistema so- lamente tiene soporte de letras “a” a “z”, por lo que solamente tiene soporte hasta 26 motores.

Comando	Sintaxis	Uso	Descripción	Consideraciones
Establecer cero relativo	[dir]CCZ	aCCZ	Comando para establecer un cero relativo en el actuador deseado. Reemplazar el campo de [dir] por la dirección del actuador.	Por defecto, el microstepping está puesto en full-step. Esto significa que en caso el usuario se le olvide establecer un microstepping al inicio, las siguientes operaciones van a utilizar el recorrido angular de un full-step.

Comando	Sintaxis	Uso	Descripción	Consideraciones
Establecer microsteps	[dir]CMA49[MS]Z [dir]CMDRV[MS]Z [dir]CM249[MS]Z [dir]CM128[MS]Z	aCMA4901Z	Comando para establecer el microstepping del actuador deseado con uno de los drivers . Para ello, ver el siguiente listado de valores para los cuales puede reemplazar [MS]: <ul style="list-style-type: none"> ▪ 01 = full-step. ▪ 02 = half-step. ▪ 04 = quarter-step. ▪ 08 = 1/8 step. ▪ 16 = 1/16 step. ▪ 32 = 1/32 step. Cambiar [dir] por la dirección del actuador deseado.	<p>A pesar de que el programa permite establecer hasta un microstepping de hasta 1/32, es posible que el driver no sea compatible o no permita dicha resolución. Adicionalmente, la conexión de los pines del microcontrolador al driver ya está establecida de la siguiente manera:</p> <ul style="list-style-type: none"> ▪ Pin 3 -> MS3 (MS2 en otros drivers) ▪ Pin 4 -> MS2 (MS1 en otros drivers) ▪ Pin 5 -> MS1 (MS0 en otros drivers) <p>Si el usuario lo desea, puede reconfigurarlos modificando el programa del microcontrolador directamente. Actualmente el sistema tiene soporte para hasta 4 drivers diferentes:</p> <ul style="list-style-type: none"> ▪ A4988 (código A49) ▪ DRV8825 (código DRV) ▪ TB67S249FTG (código 249) ▪ TB67S128FTG (código 128)

Comando	Sintaxis	Uso	Descripción	Consideraciones
Ejecución de pasos	[dir]MS[+/-] [4294967295]Z	aMS+ 1234567890Z	Comando para que el motor ejecute un número de pasos. Reemplazar los siguientes espacios por sus valores correspondientes: <ul style="list-style-type: none"> ▪ [dir] = dirección o “nombre” del actuador. ▪ [+/-] = el símbolo de + o - para la dirección de giro. ▪ [4294967295] = el número de pasos. 	El sistema permite un valor entre 0 hasta 4,294,967,295 dado que la variable que guarda este valor es un “unsigned long”.
Detener ejecución de pasos	[dir]MP	aMP	Comando para detener la ejecución de pasos. Reemplazar el campo de [dir] por la dirección del motor deseado.	Este comando se tiende a utilizar más en la ventana terminal directamente.
Obtener número de pasos anteriores	[dir]DGZ	aDGZ	Comando para retornar el número de pasos que el actuador ejecutó anteriormente. Reemplazar el campo de [dir] por la dirección del actuador deseado.	Si se ejecuta este comando justo después de encender el sistema, va a retornar 0. Adicionalmente, este comando está limitado al primer valor que se le fue enviado al motor de paso a paso. Esto significa que cualquier otro valor enviado será ignorado mientras el motor no haya terminado de girar.

Comando	Sintaxis	Uso	Descripción	Consideraciones
Obtener posición angular	[dir]DPZ	aDPZ	Comando para obtener la posición angular del actuador deseado. Reemplazar el campo de [dir] por la dirección del actuador deseado.	En varios casos, si el valor que se debería retornar es un entero, puede que lo que el sistema regrese sea un valor con decimal muy cercano al entero. Esto tiene una precisión de hasta dos decimales.

Cuadro 10: Manual de comandos para los actuadores

7.6. Selección de IDE y lenguaje para el microcontrolador maestro

Para programar el microcontrolador maestro, se contemplaron dos opciones en cuanto a la IDE que se iba a utilizar para programar la Tiva C, al igual que el lenguaje que se iba a utilizar:

- Eclipse IDE con TivaWare
- Energia con C++

Se consideraron las características de las dos opciones, como el lenguaje de programación que se podía utilizar en cada una de las interfaces, la dificultad de implementación de cada una y la conveniencia de utilizar cada uno (en este caso, el término de “conveniencia” se refiere a los requerimientos para que cada una de las dos interfaces pueda funcionar). Con respecto a esta característica de conveniencia, Eclipse IDE resultó necesitar elementos como el código de startup, modificación del PATH, múltiples modificaciones a la configuración del IDE y también aplicaciones externas como el LM Flash Programmer para poder descargar el programa al microcontrolador. Por el otro lado, Energia IDE proporciona una IDE muy similar a la de Arduino IDE, pero debido a que no se puede utilizar un API adicional, como la de TivaWare en Eclipse, activar ciertos periféricos resulta siendo más complejo. En este caso, dado que lo único que se quiere utilizar son los varios puertos de comunicación serial que tiene la Tiva, se seleccionó Energia IDE con C++.

Ya habiendo decidido esto, se establecieron los puertos seriales que iban a utilizar tanto para la recepción de datos del sistema de OCR como para la recepción de datos del mando de control manual, al igual que los baudios que se iban a implementar en cada uno de estos.

Sistema de control del brazo robótico asistencial

8.1. Comunicación entre los dispositivos del brazo robótico asistencial

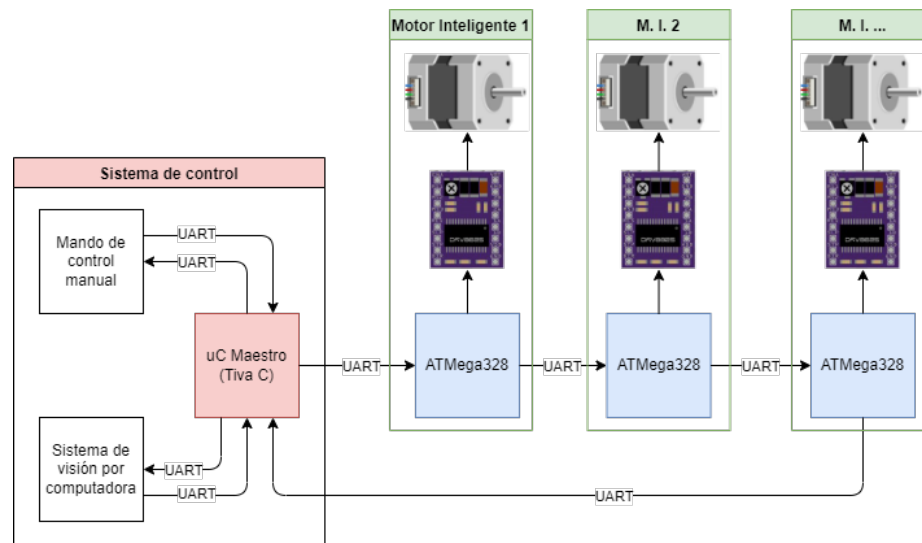


Figura 24: Diagrama de comunicación entre los dispositivos del brazo robótico

Un dato importante a mencionar es que la comunicación entre el mando, el OCR y los microcontroladores se está haciendo de manera paralela. De esta manera, el sistema mantiene un flujo constante de datos que le permite al microcontrolador maestro reaccionar y ejecutar comandos siempre y cuando detecte ciertos cambios en los datos que esté recibiendo. Es decir, cuando se obtenga el dato del mando de control que determine el cambio entre estados de

control manual y control automático, el sistema sabrá qué datos utilizar y qué datos ignorar.

8.2. Cálculos para determinar el número de pasos en base a sus reducciones de engranajes

Se tuvo que determinar el número adecuado de pasos a ejecutar en cada uno de los motores y para que exista concordancia entre el movimiento generado por el brazo tanto con el mando de control como con el programa de reconocimiento óptico de caracteres. Para ello fue necesario conocer la reducción generada por los engranajes planetarios de las juntas del brazo, al igual que la relación de engranajes de la caja reductora de algunos de los motores paso a paso. Para los cálculos de las juntas más cercanas a la base se van a utilizar los motores Nema 23 y Nema 17 con caja reductora propuestos en la fase anterior del proyecto. Esto es porque su torque ya fue analizado y comprobado teóricamente en dicha fase y experimentalmente en la fase actual. Sin embargo, los cuatro motores de las juntas finales, a pesar de haber tenido su capacidad de torque comprobada, fueron reemplazados por motores Nema 11 que fueron analizados teóricamente durante esta fase y se encontró que cumplen con las necesidades de torque para estas últimas juntas. Además, presentaban un costo y tamaño menor a los Nema 17 sin caja reductora que originalmente iban a generar el movimiento de estas juntas.

Junta (Motor)	Microstepping	Reducción por caja reductora	Reducción por sistema adicional de engranajes	Relación final
Junta 1 (Nema 23)	1:1	N/A	73:1	41 pasos : 1°
Junta 2 (Nema 17)	1:1	100:1	73:1	4056 pasos : 1°
Junta 3 (Nema 17)	1:1	100:1	73:1	4056 pasos : 1°
Junta 4 (Nema 23)	1:1	N/A	73:1	41 pasos : 1°
Junta 5 (Nema 11)	1:2	N/A	20:1	11 pasos : 1°
Junta 6 (Nema 11)	1:1	N/A	1:1	200 pasos : 1 mm
Junta 7 (Nema 11)	1:32	N/A	N/A	20 pasos : 1°
Junta 8 (Nema 11)	1:32	N/A	N/A	20 pasos : 1°

Cuadro 11: Relaciones y reducciones de cada junta

Uno de los motores que requirió la mayor consideración de las relaciones internas del sistema fue el Nema 17 que contiene una caja de engranajes para incrementar el torque de salida. Dicho motor es un motor de paso a paso bipolar que por cada paso que da se

ejecuta una rotación de 1.8° . Esto significa que el motor debe dar 200 pasos para completar una revolución. La caja de engranajes que contiene establece una relación de 99.51:1 en el motor. Para fines prácticos, se aproximó a una relación de 100:1. Entonces se sabe que para que la salida de la caja de engranajes complete una revolución, la entrada de dicha caja requiere dar 100 revoluciones. Conociendo esta información, se calculó que se requieren 20,000 pasos de parte del motor para que el eje de salida de la caja de engranajes diera una sola revolución. Ahora con la relación de los engranajes planetarios de 73:1, para generar 73 revoluciones, serían necesarios $20,000 * 73 = 1,460,000$ pasos. Esto significa que son necesarios 1,460,000 pasos para rotar el eje de salida 360° . Conociendo esto, se obtuvo una relación final de 4055.5 pasos/grado. Por la inhabilidad del motor de dar pasos parciales no se puede contar el valor de 0.5, se aproxima a 4056 pasos/grado. Este desarrollo matemático se encuentra a continuación:

$$\begin{aligned}
 &200 \text{ pasos del motor} = 1 \text{ revolución } (360^\circ) \text{ del eje} \\
 &200 * 100 = 20,000 \text{ pasos} = 1 \text{ revolución } (360^\circ) \text{ de la relación } 100:1 \\
 &20,000 * 73 = 1,460,000 \text{ pasos} = 360^\circ \text{ de la relación } 73:1 \\
 &\frac{1,460,000 \text{ pasos}}{360 \text{ grados}} = 4,056 \text{ pasos/grado}
 \end{aligned}$$

Este proceso se repitió para todas las juntas que tuvieran una caja de engranajes o una reducción de engranajes planetarios de entremedio. Hubo algunas excepciones:

- En el caso de la junta 6 del brazo que convierte movimiento rotacional en movimiento lineal. En este caso, el proceso requirió conocer el paso del tornillo sinfín (1 mm) para poder hacer la conversión. Sabiendo que el paso es la distancia que se mueve el tornillo al dar una revolución, se llegó a la conclusión que se requieren 200 pasos (1 revolución del motor) para mover 1 mm del tornillo.
- Los motores paso a paso que no contienen ni caja de engranajes ni reducción por engranajes planetarios. Estos son los último motores Nema 11 más cercanos al efector final del brazo. Para estos, se desarrollaron las relaciones con un microstepping de 1/32. Esto resulta en una relación de aproximadamente 20 pasos/grado.

Estos cálculos aplicaron principalmente para el sistema del mando de control, dado que el sistema de mando de control permite resoluciones de 0.05° , 0.1° , 1° , 10° . A diferencia del mando, el sistema de reconocimiento de caracteres no tiene acceso a la resolución de 0.05° . Adicionalmente, el sistema empleado para el mando de control fue el de establecer previamente un número determinado de pasos para cada uno de los casos requeridos. Es decir, si el mando establece movimientos de 0.05° , el motor ejecuta un número determinado de pasos y para 0.1° ejecuta un número diferente. A comparación, el sistema OCR utiliza un método algebraico para sumar el número de pasos requerido para reducir el error detectado.

8.3. Pruebas con simulación de sistema de reconocimiento óptico de caracteres (OCR) en Python

Se comenzaron a hacer pruebas con la Tiva C y el sistema de OCR, para el cual se requirió utilizar Python. Para iniciar las pruebas, se escribió un programa sencillo en Python que

le permitiera iniciar la comunicación serial con el microcontrolador Tiva C y verificar el envío de datos. Del lado de la Tiva, se escribió también un programa para poder recibir los datos extraídos por el reconocimiento de texto y verificar que los datos recibidos por el microcontrolador fueran los que la cámara realmente captó.

Sin embargo, se encontraron problemas durante el envío y lectura de los datos enviados. Para resolver este problema se hicieron varias modificaciones al código de simulación inicial, al igual que el código cargado la Tiva C para encontrar el problema. Dichas modificaciones incluyeron:

- La conexión inicial del puerto serial en Python empleando la función de `ser.Serial()` y desconexión del puerto serial con la función `ser.close()`.
- La verificación de la apertura del puerto serial en Python con la función `ser.isOpen()`, el cual devuelve un valor booleano.
- El envío de un solo *byte* que concuerda con el nombramiento de cada uno de los microcontroladores. Esto nos ayudó a verificar si el microcontrolador estaba detectando el dato recibido de manera correcta.
- Uso de indicadores como los LEDs integrados de la Tiva C y el Arduino para verificar si se estaban ejecutando las condicionales establecidas en el código.

Mientras se llevaban a cabo las modificaciones anteriormente mencionadas, se conectaron tanto la Tiva C con uno de los Arduino UNO para poder obtener retroalimentación de los valores enviados por comunicación serial. Se empleó tanto la ventana terminal del programa de *Spyder* (incluido en el paquete de Anaconda, el cual a su vez incluye Python) como el programa de “Hercules SETUP utility” (anteriormente utilizado para probar la conexión tipo daisy chain de los actuadores) para visualizar los datos enviados por Python y recibidos por el Arduino.

Conforme se fue reconstruyó el código, se fueron agregando y quitando algunas variables y líneas de código que posiblemente podrían haber tenido conflicto y al final se observó que el problema se encontraba en el orden en la cual se llevaban a cabo algunos comandos.

Habiendo resuelto el problema, surgió un problema adicional, el cual involucra la latencia del envío de datos. Al probar el movimiento de los motores paso a paso, fue posible ver que los datos se envían y el Arduino logra procesar los datos de manera correcta para generar el movimiento de los motores. El problema es que el número de pasos enviados y el número de pasos que ejecuta el motor es incongruente. Esto se pudo ver enviando un número variado de pasos, desde 50 pasos, hasta 400 pasos y el movimiento del motor es inconsistente, sin importar el número de pasos que se enviaba, además de que la función de reducción de pasos funcionaba, pero tenía problemas cuando la variable de pasos llegaba a cero. Para resolver este problema, en lugar de cambiar completamente el código original, se decidió hacer cambios en los tiempos de *delay* (demora) que se utilizaron tanto para el código de Python como para el código de los actuadores. Se fueron cambiando los tiempos hasta que se obtuvo una sincronización deseada en cuanto al tiempo de envío de datos del programa en Python, la recepción de datos y envío de comandos de la Tiva a los Arduinos y la recepción y ejecución de dichos comandos. Se logró ajustar de manera que, a simple vista, el motor ya se mueve el número de pasos correctos (200 pasos para una revolución en full-step). Esto

consecuentemente logró resolver la problemática con el número de pasos cuando este llegaba a cero pasos. Sin embargo, dado que esta es una solución obtenida en base a los tiempos de *delay*, es muy posible que se tenga que volver a ajustar una vez se hayan conectado el sistema de OCR y el mando de control manual.

8.4. Incorporación del sistema OCR al movimiento de los actuadores

Una vez terminado el sistema de reconocimiento de caracteres, se llevaron a cabo las pruebas de envío de datos y de movimiento de los motores. Es imperativo aclarar que el problema del movimiento retardado debido a la latencia que se tuvo anteriormente fue resuelto al implementar el sistema OCR original al sistema de control del brazo. Para ello, se conectó el sistema de OCR directamente al microcontrolador maestro, el cual va a transmitir los datos del OCR reconfigurados en forma de comandos a cada uno de los actuadores.

Las conexiones se pueden visualizar en la Figura 25, el cuadro rojo encerrando el sistema OCR, el cuadro azul y verde encerrando el sistema de control y los actuadores.

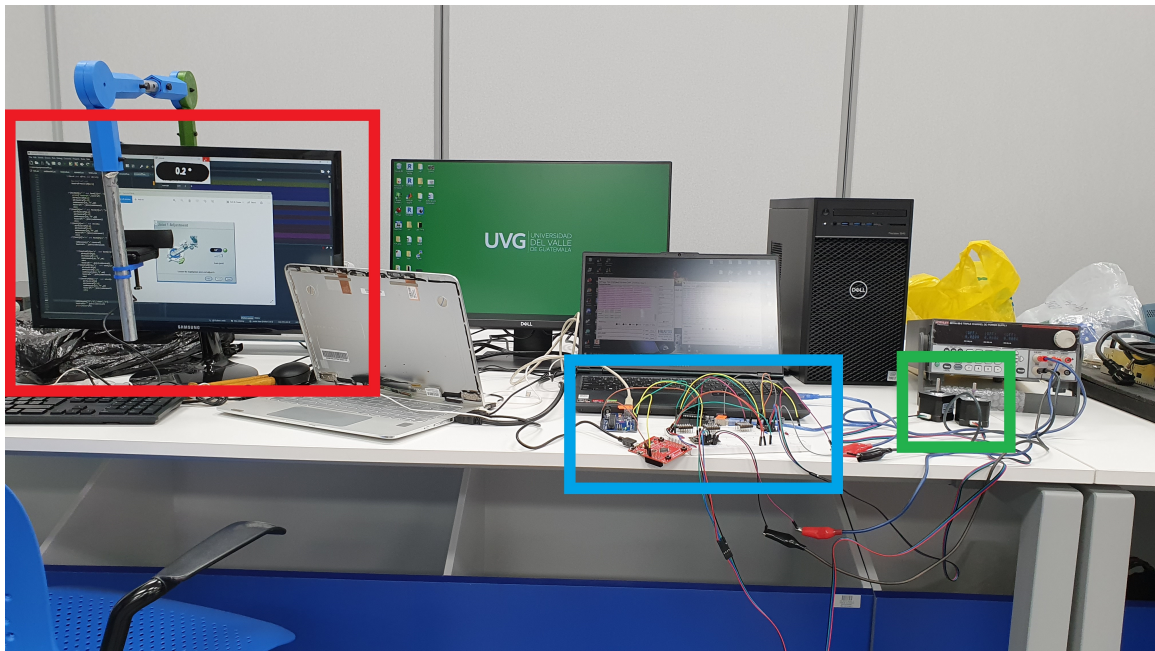


Figura 25: Conexión entre el sistema OCR, y sistema de control de actuadores (azul y verde)

Se iniciaron las pruebas con el programa de reconocimiento de caracteres original, el cual va a utilizar imágenes obtenidas del sistema de Varioguide para simular una situación dentro de la sala de operaciones. Esta imagen brinda un error angular, el cual los motores van a intentar corregir. Debido a que en estos momentos aún no se tiene disponible el brazo para las pruebas, se utilizaron motores Nema 17 para verificar la concordancia entre el error con el número de pasos requeridos. Dado que las imágenes utilizadas presentaban errores considerablemente pequeños, el máximo siendo de 0.2° , era de esperarse que el número de

pasos ejecutados por los motores fuera bajo. Dado que era muy difícil observar el movimiento de los motores, se tuvo que sostener su eje con la mano suavemente para poder llevar a cabo las pruebas experimentales y poder sentir el movimiento.

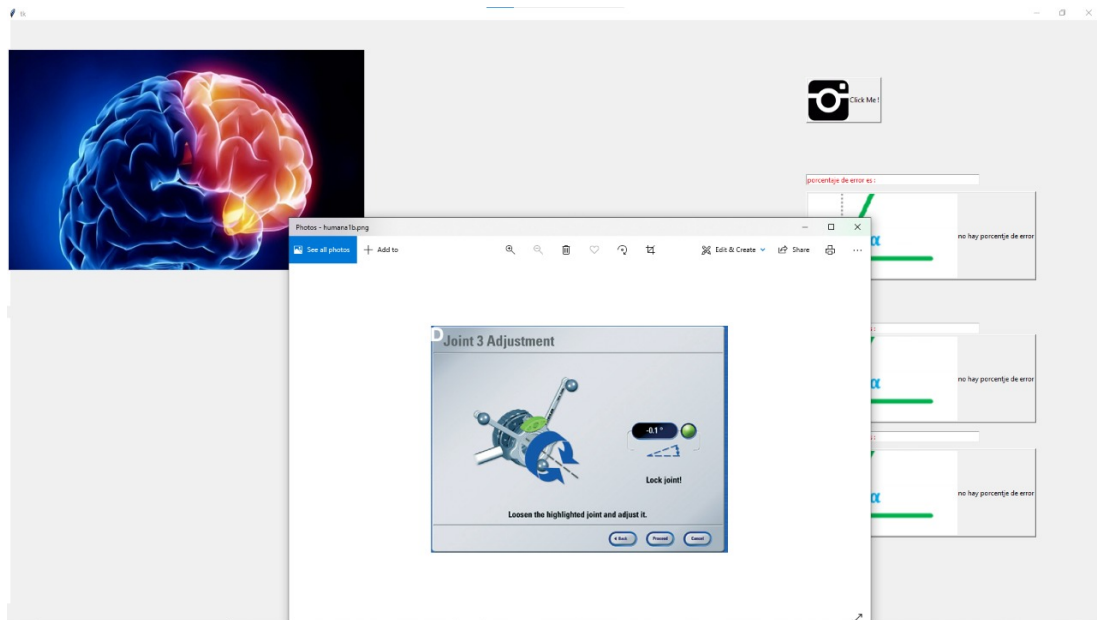


Figura 26: Imagen con error de 0.2° utilizado para las pruebas.

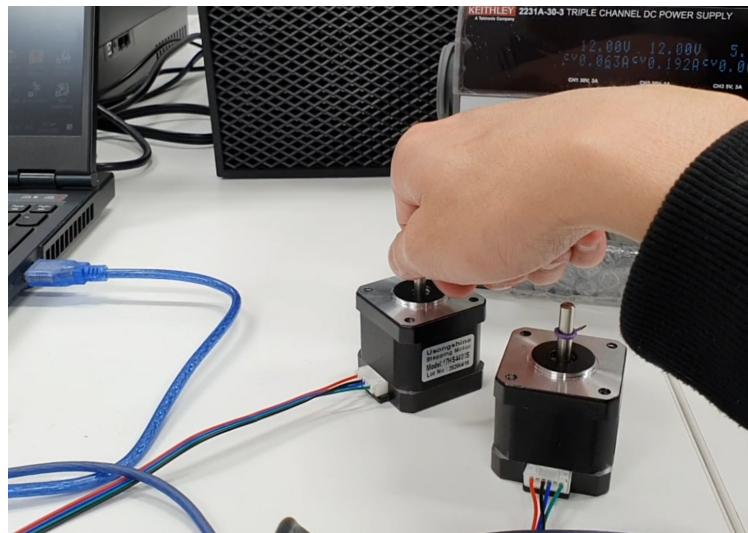


Figura 27: Sosteniendo el servo para verificar movimiento.

8.5. Pruebas con simulación del mando de control manual

Se prosiguió con el desarrollo del código para la recepción de datos para el mando de control manual. Para implementar la comunicación con el mando de control fue necesario

establecer un nuevo sistema dado que esta vez no se recibe un número de pasos, sino que se recibe un número de “ticks”, dado que el dispositivo empleado en el mando para el movimiento de los motores es un encoder. Para ello fue necesario establecer un nuevo conjunto de valores a recibir, diferente al de la comunicación con el sistema de OCR. En este caso, después de considerar diferentes métodos, para las pruebas preliminares, se estableció un conjunto compuesto de los siguientes tres valores:

- Modo de operación (ya sea en modo automático o modo manual)
- Número del motor deseado (con el cual se hace la correlación con su dirección asignada por los microcontroladores).
- Número de “ticks” con base en la resolución establecida por el mando de control.
- El signo de la revolución, es decir, si el usuario deseaba girar el motor en dirección a favor o en contra de las agujas del reloj.

Estos valores se enviaron desde el mando de control encerrados en marcadores de inicio y final para que el programa pueda detectar y saber qué datos extraer. El dato final que el microcontrolador maestro recibe es el siguiente:

< Modo, Motor, Ticks, Signo >

Para simular este sistema, se armó un circuito sencillo utilizando un potenciómetro, el cual intentaba acercarse lo más posible al funcionamiento real del encoder.

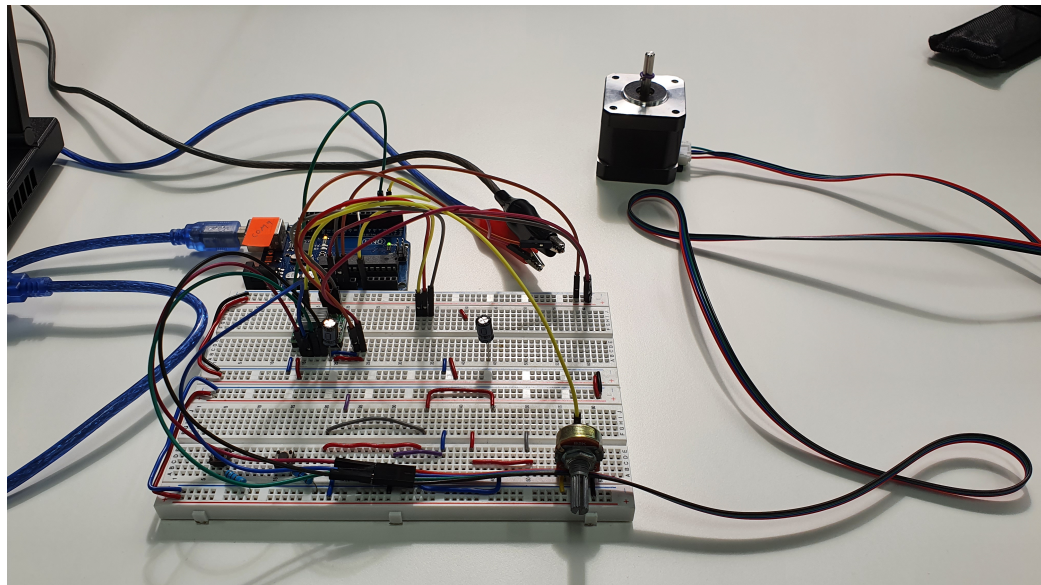


Figura 28: Circuito armado para simular el encoder utilizado por mi compañero.

Utilizando este circuito y el código creado, se pudo mover el motor de paso a paso de la manera esperada, pero solamente cuando el potenciómetro era rotado lentamente. Es decir, el sistema se desincronizaba al momento de incrementar la velocidad de rotación

del potenciómetro. Se concluyó que esto se podía deber al flujo de iteraciones utilizando la función “for” para llevar a cabo los pasos, en conjunto con el sistema de valores actuales y valores pasados implementado para evitar repeticiones de pasos. Debido a esto no fue posible continuar con las pruebas del sistema utilizando este circuito.

8.6. Integración del mando de control físico al movimiento de los actuadores

En lugar de seguir utilizando el potenciómetro como reemplazo del encoder para las pruebas, se utilizó el circuito original (cuadro verde), el cual forma parte del mando de control a utilizar. Se conectó dicho circuito por comunicación serial con el microcontrolador maestro (cuadro rojo) y se ejecutó el programa para efectuar el movimiento de los motores (cuadro azul).

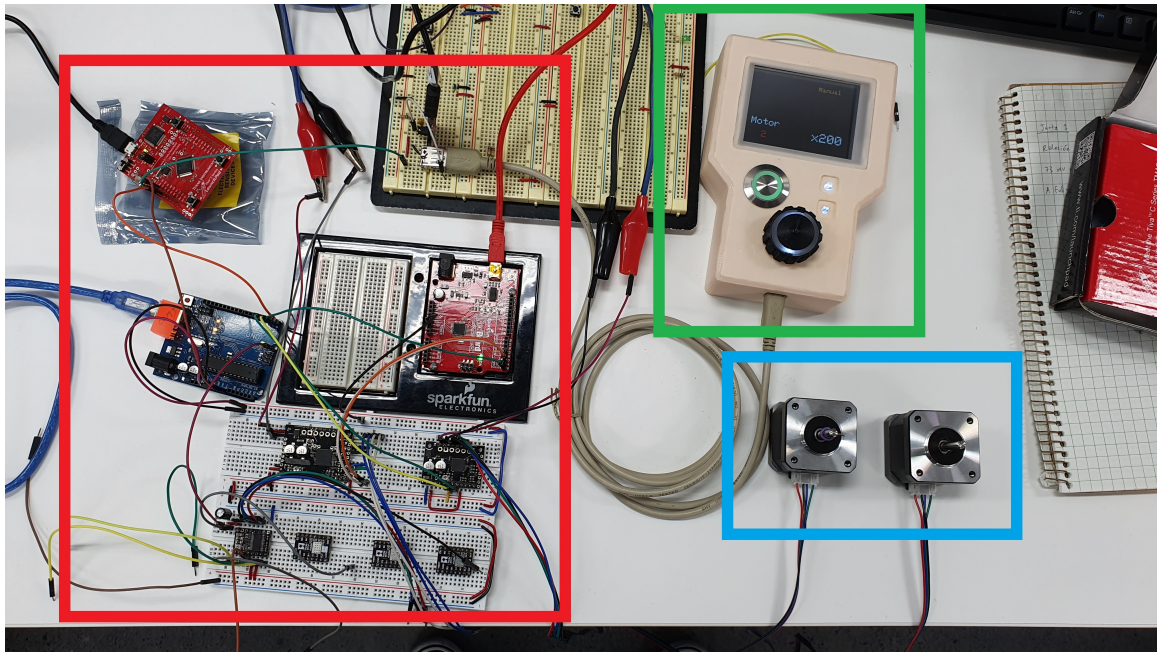


Figura 29: Conexión del mando de control al sistema de actuadores.

Dicho esto, fue necesario reconstruir el código de manera que pueda abarcar tanto el sistema del mando de control como el sistema de reconocimiento de texto. Sin embargo, para pruebas preliminares, se extrajo la sección requerida para el funcionamiento del mando de control y se hicieron pruebas de recepción de datos utilizando solamente la sección extraída.

Se pudo observar que los datos son recibidos de la manera esperada y se reorganizaron estos datos para que cumplieran con el formato de los comandos establecidos anteriormente para el movimiento de los actuadores.

Un problema que surgió llevando a cabo pruebas con las diferentes resoluciones fue que los datos recibidos y luego reconfigurados tendían a enviar un signo adicional de vez en

cuando. Esto resulta en problemas con el movimiento de los actuadores, principalmente en la resolución de “10” (equivalente a una resolución 0.1°) dado que resultaba en giros máximos. Esto causaba que el motor girara al número máximo de pasos disponibles, el cual es de 4,294,967,295 pasos. Para resolver este problema, se implementó una manera diferente de guardar el signo recibido a través del mando de control y se agregó una segunda verificación después de reconfigurar los datos pero antes de enviarlos. Esto resultó en que se eliminara el signo adicional al enviar los datos a los actuadores.

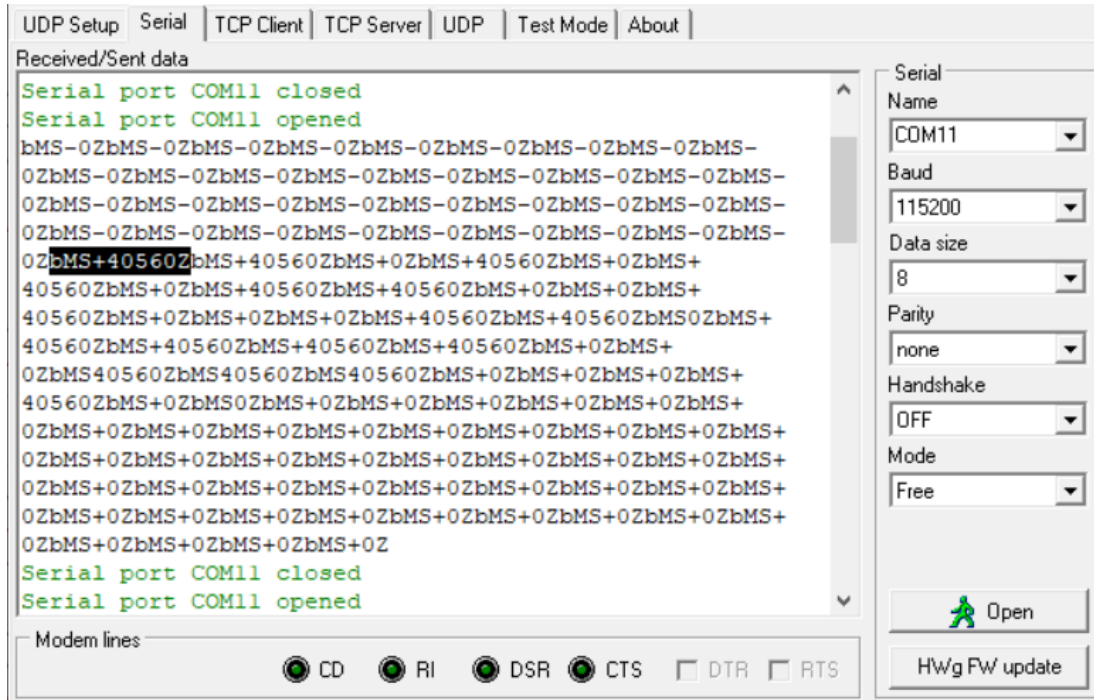


Figura 30: Ventana serial mostrando salida de datos de la Tiva C con la conversión de “ticks” a pasos

Como se puede ver en la figura anterior, el dato recibido del mando de control contuvo el motor a utilizar (motor “b”), la dirección de giro (el signo positivo) y la resolución de “200” (la cual se convierte en 40,560 pasos para el motor “b” específicamente).

8.7. Implementación simultánea del sistema de reconocimiento de texto con el mando de control

Para la implementación simultánea con el sistema de control se hizo la conexión de los drivers de los motores propuestos de la fase anterior en una tabla de prototipado. Con base en el driver que se utilizó, fue necesario seguir un esquema de conexión diferente:

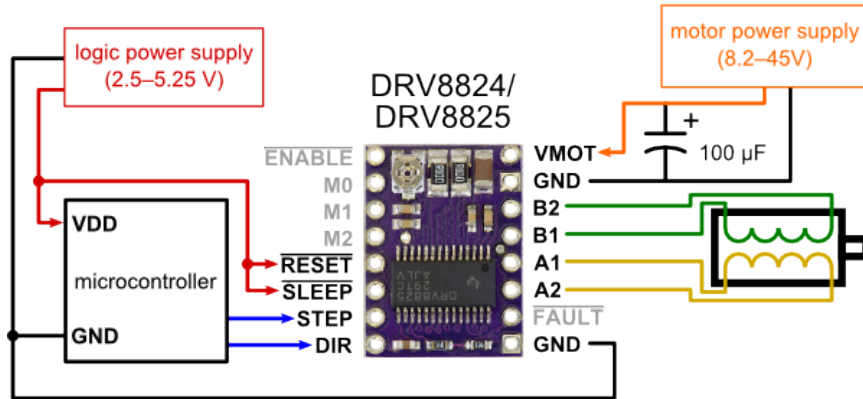


Figura 31: Esquema de conexión del driver DRV8825. [26]

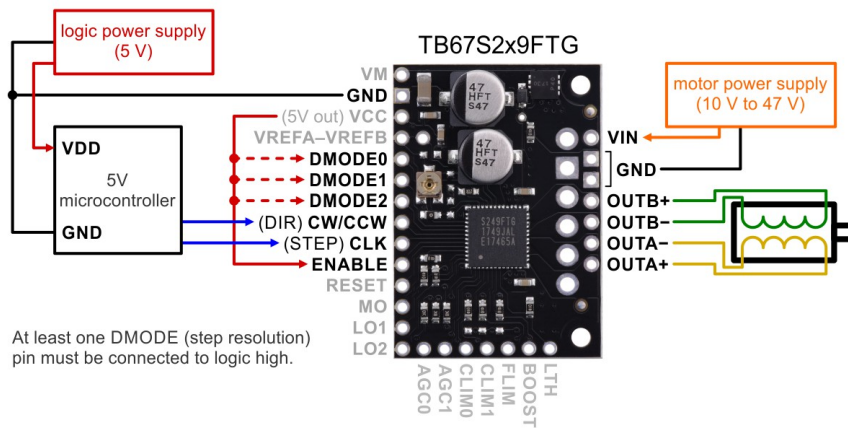


Figura 32: Esquema de conexión del driver TB67S249FTG. [27]

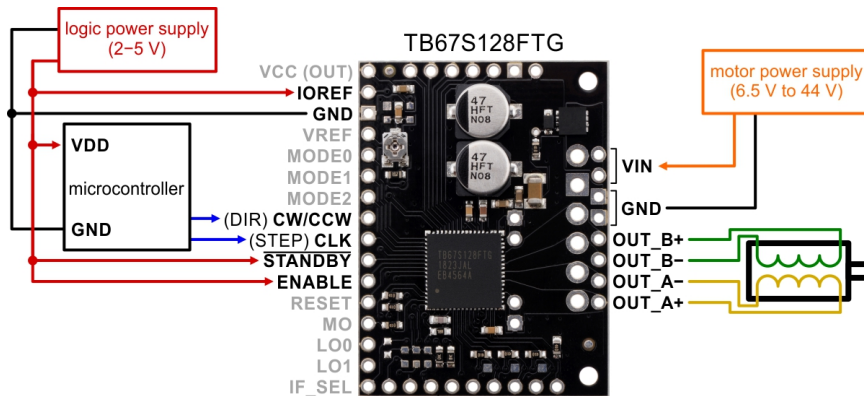


Figura 33: Esquema de conexión del driver TB67S128FTG. [28]

Estos *drivers* fueron luego conectados a los motores indicados en base a la corriente que iban a requerir. Es decir, los *drivers* con mayor capacidad de corriente (TB67S128FTG y TB67S249FTG) fueron conectados a los motores Nema 23 mientras que los *drivers* DRV8825 fueron conectados a los motores Nema 17. En este caso, debido a que no se tenían disponibles

los motores Nema 11, no hubo una manera adecuada de probar el funcionamiento de estos motores, por lo que se utilizaron motores Nema 17 adjuntas a eje flexibles (Figura 34) para generar el movimiento de los últimos actuadores.



Figura 34: Brazo robótico con los motores y ejes flexibles instalados.

Debido a la limitación con el número de microcontroladores disponibles, solamente se utilizaron 3 microcontroladores y en conjunto con estos, 3 motores paso a paso. Para las juntas más cercanas al efector final se fueron alternando dependiendo de la junta que se deseara mover. Se hicieron las conexiones necesarias entre los microcontroladores, los drivers y los motores paso a paso adjuntos al brazo y aquellos adjuntos a los ejes flexibles.

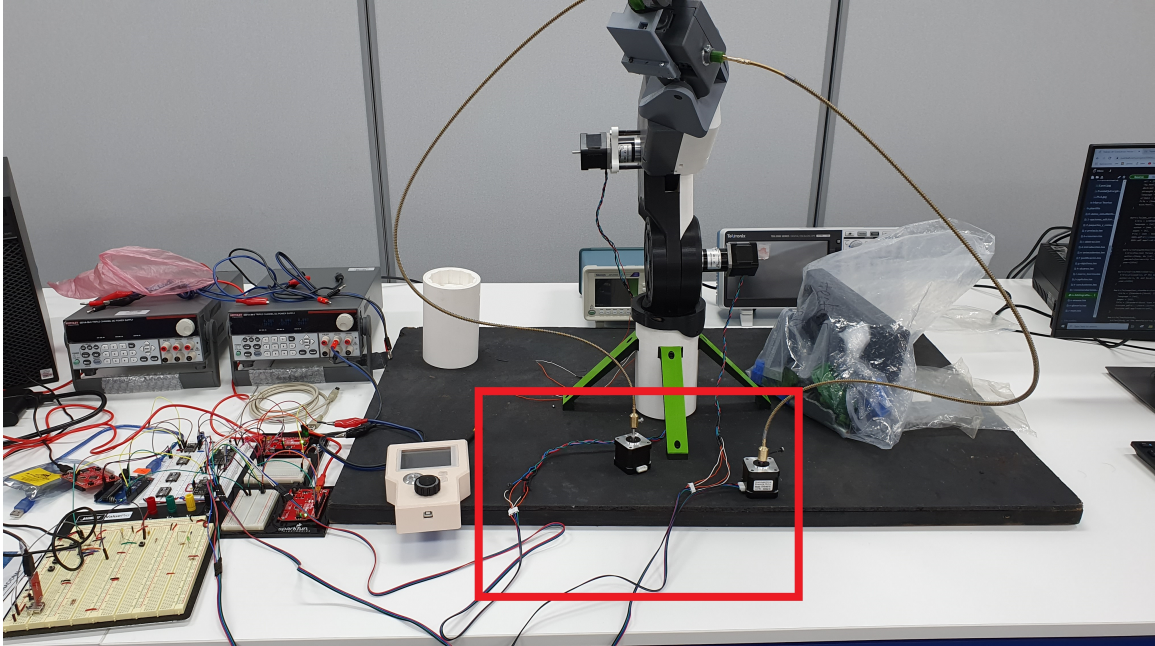
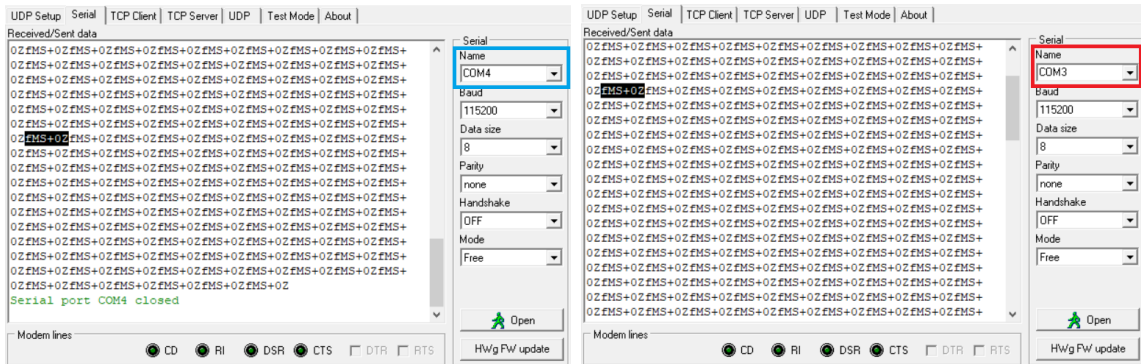


Figura 35: Conexión de los microcontroladores a los motores del brazo.

Al microcontrolador Tiva C se le cargó la última versión del código creado para la recepción de datos del mando de control y del sistema OCR y se conectó al sistema de actuadores (Figura 35). Lo primero que se quiso verificar era el funcionamiento de la función de selección de motores del mando. Para ello fue necesario enfocarse en las diferentes ventanas terminales de cada uno de los microcontroladores conectados al sistema.



(a) Reenvío del primer microcontrolador al segundo (b) Reenvío del segundo microcontrolador al tercer microcontrolador.

Figura 36: Reenvío de datos al microcontrolador correcto.

En la Figura 36 se pueden ver dos ventanas terminales en Hércules, en donde están resaltados los comandos de pasos transmitidos de una ventana terminal a otra. Esto representa el funcionamiento de reenvío de comandos en base a si el motor deseado por el usuario concuerda con la dirección guardada en la memoria del microcontrolador. En este caso, como no concuerda con ninguno de los dos motores, el sistema va a seguir reenviando el comando

hasta que encuentre al motor deseado y este será el que reciba y ejecute el comando.

Con esto, se corrió el programa de OCR, el cual, en la sala de operaciones, va a estar ejecutándose constantemente. Se pudo ver que, mientras el mando de control no cambie de modo de operación de manual a automático, el sistema de movimiento va a recibir datos del sistema OCR pero los va a ignorar. Para comenzar las pruebas de movimiento con el brazo, se utilizó el encoder del mando de control. Para verificar su funcionamiento adecuado, se comprobó que el motor que desplegaba la pantalla fuera el que iba conectado al microcontrolador correcto y que el número de pasos dados fuera el correcto en la terminal.



Figura 37: Resolución de movimiento del mando convertido a pasos.

Como se pudo ver en la Figura 37, el valor enviado por la Tiva C fue de 20 pasos, equivalente a un movimiento lineal de 0.1 mm en base a los cálculos anteriormente hechos.

Se prosiguió a activar el modo automático a través del mando de control y se pudo ver que el sistema ya no ignoraba la información proveniente del sistema de OCR y se convertían al número de pasos adecuado. En este caso, como el sistema de OCR estaba utilizando una imagen con un error angular de 0.2° , se requerirían 40 pasos.

- Por el nivel de modularidad y flexibilidad que se le desea imponer al sistema, este requiere de una cantidad extensa de programación defensiva para evitar conflictos en su futura implementación.
- Se implementaron direcciones y nombramientos específicos para cada motor inteligente en el protocolo de comunicación.
- Se diseñó un sistema de motores inteligentes que incluye un motor paso a paso, su driver y un microcontrolador.
- Se diseñó un protocolo de comunicación de formato daisy-chain entre microcontroladores utilizando comunicación UART para el control de los motores inteligentes.
- Se diseñó un protocolo de comunicación UART entre el microcontrolador maestro y los sistemas de reconocimiento de caracteres y mando de control manual.
- El movimiento de los primeros cuatro grados de libertad del brazo asistencial fue implementado utilizando un sistema de lazo abierto y funcionan solamente con control manual.
- Se implementó el uso del sistema OCR solamente para el afinado del movimiento de los últimos grados de libertad del brazo, los cuales funcionan tanto con control manual y control automático.
- El lazo cerrado de los últimos grados de libertad proviene del sistema OCR, el cual obtiene el error de posición y se lo retroalimenta al sistema de control.

- Cuando se trabaje con la comunicación serial, siempre verificar si lo que se está enviando y recibiendo en el buffer se va a interpretar en ASCII o como bytes. Esto principalmente para evitar conflictos al intentar utilizar condicionales o comparaciones en el código por la discrepancia de tipo de datos empleados.
- Para crear un motor mucho más completo, se puede agregar un encoder de manera que el sistema pueda tener un cero global y obtener un sistema de lazo cerrado. Esto logra que el usuario pueda conocer en qué posición se encuentran los actuadores, permitiéndole al emplear algún sistema de control que reduzca vibraciones, sobresaltos (overshoot), entre otros, si se llega a emplear un motor diferente.
- Para hasta mayor flexibilidad y modularidad, se recomienda reconfigurar el sistema para poder implementarse con motores DC o servomotores de manera que el usuario pueda utilizar el mismo sistema de control para una variedad de motores.
- Se recomienda analizar la implementación de un protocolo de comunicación tipo TLV (Type-Length-Value) el cual es un protocolo que le permite al usuario definir directamente con los comandos el tipo de dato que está transmitiendo, la longitud de dicho dato y por último, el dato en sí, respectivamente, ya permitiéndole al dispositivo saber cuántas veces debe leer el puerto serial para recibir los datos.
- Se recomienda reescribir el código fuente de forma que esta pueda ser implementada como una librería. Esto le otorga al usuario mayor facilidad al alterar secciones del código.
- Para remover el requerimiento de utilizar un launchpad para utilizar el microcontrolador de los actuadores, es posible el utilizar código en C para que de esta manera, solamente se requiera integrar el microcontrolador al dispositivo que se va a adjuntar a los motores.

-
-
- [1] J. H. Nelson, S. L. Brackett, C. O. Oluigbo y S. K. Reddy, «Robotic Stereotactic Assistance (ROSA) for Pediatric Epilepsy: A Single-Center Experience of 23 Consecutive Cases,» *Children*, vol. 7, n.º 8, 2020. dirección: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7465763/>.
 - [2] B. A. Miller, A. Salehi, D. D. Limbrick y M. D. Smyth, «Applications of a robotic stereotactic arm for pediatric epilepsy and neurooncology surgery,» *Journal of Neurosurgery: Pediatrics*, vol. 20, n.º 4, págs. 364-370, 2017. dirección: <https://thejns.org/pediatrics/view/journals/j-neurosurg-pediatr/20/4/article-p364.xml>.
 - [3] *Epilepsia*, Organización Mundial de la Salud, 2019. dirección: <https://www.who.int/es/news-room/fact-sheets/detail/epilepsy>.
 - [4] *Epilepsia*, Mayo Clinic, 2021. dirección: <https://www.mayoclinic.org/es-es/diseases-conditions/epilepsy/diagnosis-treatment/drc-20350098>.
 - [5] *What is a Neurosurgeon - Neurosurgery - Highland Hospital - University of Rochester Medical Center*. dirección: <https://www.urmc.rochester.edu/highland/departments-centers/neurosurgery/what-is-a-neurosurgeon.aspx>.
 - [6] *What is Neurosurgery? | OHSU*. dirección: <https://www.ohsu.edu/school-of-medicine/neurosurgery/what-neurosurgery>.
 - [7] R. Zhao, Z. Shi, Y. Guan, Z. Shao, Q. Zhang y G. Wang, «Inverse kinematic solution of 6R robot manipulators based on screw theory and the Paden–Kahan subproblem,» *International Journal of Advanced Robotic Systems*, vol. 15, n.º 6, 2018. dirección: <https://doi.org/10.1177/1729881418818297>.
 - [8] Z. Lu, C. Xu, Q. Pan, X. Zhao y X. Li, «Inverse Kinematic Analysis and Evaluation of a Robot for Nondestructive Testing Application,» *Journal of Robotics*, vol. 2015, págs. 1-7, 2015. dirección: <http://www.hindawi.com/journals/jr/2015/596327/>.
 - [9] *Chapter 6. Inverse Kinematics*. dirección: <http://motion.pratt.duke.edu/RoboticSystems/InverseKinematics.html>.

- [10] Z.-Y. Zhao, M. Tomizuka y S. Isaka, «Fuzzy gain scheduling of PID controllers,» *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 23, págs. 1392-1398, 1993. dirección: https://www.researchgate.net/publication/3114297_Fuzzy_gain_scheduling_of_PID_controllers.
- [11] J. Kim, I. Yang y D. Lee, «Daisy Chain Method for Control Allocation Based Fault-Tolerant Control,» *Journal of IEMEK*, vol. 8, 2013. dirección: https://www.researchgate.net/publication/263362978_Daisy_Chain_Method_for_Control_Allocation_Based_Fault-Tolerant_Control.
- [12] *Stepper Motors: Types, Uses and Working Principle | Article | MPS*. dirección: <https://www.monolithicpower.com/stepper-motors-basics-types-uses>.
- [13] T. S. Roy, H. Kabir y A. M. Chowdhury, «Simple Discussion on Stepper Motors for the Development of Electronic Device,» vol. 5, n.º 1, pág. 9, 2014. dirección: https://www.researchgate.net/publication/273448791_Simple_Discussion_on_Stepper_Motors_for_the_Development_of_Electronic_Device.
- [14] *All About Stepper Motors*. dirección: <https://learn.adafruit.com/all-about-stepper-motors/types-of-steppers>.
- [15] Department of Electronic Engineering, University of Nigeria, Nsukka, Nigeria, E. Onyeka, M. Chidiebere, Department of Electrical and Electronic Engineering, C. O. Ojukwu University, P.M.B.02 Uli, Nigeria, A. Nkiruka y Department of Electrical and Electronic Engineering, C. O. Ojukwu University, P.M.B.02 Uli, Nigeria, «Improved Response Performance of Two-Phase Hybrid Stepping Motor Control Using PID Tuned Outer and Inner Loop Compensators,» *Journal of Engineering Sciences*, vol. 6, n.º 1, págs. d1-d6, 2019. dirección: <http://jes.sumdu.edu.ua/improved-response-performance-of-two-phase-hybrid-stepping-motor-control-using-pid-tuned-outer-and-inner-loop-compensators/>.
- [16] R. P. Ruilope, «Modelling and Control of Stepper Motors for High Accuracy Positioning Systems Used in Radioactive Environments,» pág. 223, dirección: <https://core.ac.uk/download/pdf/148670797.pdf>.
- [17] *Cranial Navigation*, 2021. dirección: <https://www.brainlab.com/es/productos-de-cirugia/relacion-de-productos-de-neurocirugia/navegacion-craneal/>.
- [18] P. H. Schimpf, «Modified Protothreads for an Embedded Systems Course,» *The Journal of Computing Sciences in Colleges*, pág. 6, 2012. dirección: <https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbXNxc2NoaW1wZjk5fGd40j%20E3N2Q3MjFmMTFiMDkzMmE>.
- [19] A. Dunkels, O. Schmidt, T. Voigt y M. Ali, «Protothreads: simplifying event-driven programming of memory-constrained embedded systems,» en *SenSys '06*, 2006. dirección: <https://www.semanticscholar.org/paper/Protothreads%5C%3A-simplifying-event-driven-programming-Dunkels-Schmidt/7176b31ba891fe43556e16a295bbf%208f58981547f>.
- [20] *ROBOTIS e-Manual*, en. dirección: <https://emanual.robotis.com/docs/en/dxl/x/x1320/>.
- [21] Z. Khodzhaev, «Monitoring Different Sensors with ATmega328 Microprocessor,» 2016. dirección: https://www.researchgate.net/publication/325154131_Monitoring_Different_Sensors_with_ATmega328_Microprocessor.

- [22] *eXtreme AVR's: AVR Family Overview*, 2012. dirección: <http://extremeavr.blogspot.com/2012/08/avr-family-overview.html>.
- [23] G. Gomez y A. Ahuja, «Use Conditions for 5-V Tolerant GPIOs on Tiva™ C Series TM4C123x Microcontrollers,» pág. 9, 2013. dirección: <https://www.ti.com/lit/an/spma053/spma053.pdf?ts=1599329908802>.
- [24] O. Ruiz, S. Arroyave-Tobón y J. Cardona, «EGCL: An Extended G-Code Language with Flow Control, Functions and Mnemonic Variables,» *World Academy of Science, Engineering and Technology*, vol. 67, 2012. dirección: https://www.researchgate.net/publication/265532211_EGCL_An_Extended_G-Code_Language_with_Flow_Control_Functions_and_Mnemonic_Variables.
- [25] *A4988 Stepper Motor Driver Carrier*. dirección: <https://www.pololu.com/product/1182>.
- [26] *DRV8825 Stepper Motor Driver Carrier, High Current*. dirección: <https://www.pololu.com/product/2133>.
- [27] *TB67S249FTG Stepper Motor Driver Carrier - Full Breakout*. dirección: <https://www.pololu.com/product/2973>.
- [28] *TB67S128FTG Stepper Motor Driver Carrier*. dirección: <https://www.pololu.com/product/2998>.

12.1. Repositorio en Github con los programas

Presionar en el enlace: [Repositorio en Github](#)