

Seguidor de Rostro

Proyecto Final: Sistemas de Control II

Integrantes: Cruz, Enrique Luis Fernando
Perez Perera, Eduardo André
Recalde, Santiago
Villar, Federico Ignacio

Profesor: Pailos, Hugo Nicolás

Fecha de entrega: 21 de junio de 2023
Córdoba, Argentina

Resumen

En el siguiente informe se detallan los procedimientos referidos al desarrollo e implementación del sistema seguidor de rostros desarrollado por el grupo **El senado galáctico** como proyecto final de la asignatura Sistemas de Control II.

Se adjuntarán imágenes, explicaciones, fórmulas y códigos, de forma de ser lo más claro posibles, y poder lograr un aproximado a un proyecto de la vida profesional.

Índice de Contenidos

1.	Posibles trabajos, y selección	1
2.	Planificación y diagrama de Gantt	2
3.	Análisis y rúbrica	5
4.	Motor paso a paso	6
4.1.	Clasificación	6
4.1.1.	Tipos de motores de imán permanente	6
4.2.	Características mecánicas	7
4.3.	Motor 28BYJ-48	7
4.3.1.	Caja reductora	8
4.3.2.	Secuencia de pasos	8
4.3.3.	Driver ULN2003	9
4.3.4.	Conexionado del motor	10
4.3.5.	Práctica con microcontrolador Arduino	11
4.4.	Driver Pololu A4988	13
4.5.	Pruebas de código de Raspberry Pi	15
5.	Face tracking	19
6.	Harware utilizado	22
6.1.	Motor Nema 17	22
6.2.	Raspberry Pi 4B	23
6.3.	Raspberry Pi Camera v1.3	24
7.	Herramientas de Software Utilizadas	25
7.1.	Visual Studio Code	25
7.2.	Thonny IDE	25
7.3.	Micropython	25
7.4.	Raspbian (Raspberry OS)	26
7.5.	OpenCV	27
7.6.	Dlib	27
8.	Conjunción face tracking y motores (ideas y utilizado)	29
8.1.	Métodos de ejecución del programa (alternativas)	29
8.1.1.	Software y hardware	29
8.1.2.	Concurrencia por software	29
8.1.3.	Secuencial	30
9.	Modelo inicial función de transferencia para seguidor de rostro	32
9.1.	Cambio de variable en el manejo del error	32
9.2.	Definición de las constantes de operación del sistema	33
10.	Implementación	37

10.1. Conexiones del hardware	37
10.2. Código de programación utilizado	38
10.3. Implementación física	41
11. Lecciones aprendidas	44
11.1. Contratiempos y acciones implementadas	44
12. Conclusiones del proyecto	46
12.1. Recomendaciones finales	46

Índice de Figuras

1. Diagrama de Gantt para la organización.	3
2. Paso completo o doble.	8
3. Paso simple.	8
4. Medio paso.	9
6. Circuito con transistores en configuración D'Arlington.	9
7. Conexión del driver ULN2003.	10
8. Conexión del driver con el motor.	10
9. Conexión de arduino, driver y motor.	11
10. Esquema interno Pololu A4988.	14
11. Conexión de driver con motor paso a paso.	15
12. Reconocimiento facial y trackeo.	19
13. Motor Nema 17.	22
14. Raspberry Pi 4B.	23
15. Raspberry Pi Camera v1.3.	24
16. Diagrama de flujo para la alternativa concurrente.	30
17. Análisis del error.	32
18. Respuesta del sistema a lazo abierto.	35
19. Respuesta del sistema a lazo cerrado.	36
20. Circuito topológico.	37
21. Circuito esquemático.	37
22. Base del prototipo.	41
23. Soporte de la cámara.	41
24. Soporte de la PCB.	42
25. Soporte del motor paso a paso.	42
26. Base del prototipo.	42
27. Plaqueta PCB.	43
28. Prototipo ensamblado.	43

Índice de Tablas

1. Comparación motores paso a paso a imán permanente.	7
---------------------------------------------------------------	---

Índice de Códigos

1.	Diagrama de Gantt	3
2.	Código de Arduino para medio paso	11
3.	Código Arduino para paso simple	12
4.	Prueba de L298N en Raspberry Pi 4	15
5.	Simplificación del código para L298N	16
6.	Prueba Pololu A4988 en Raspberry Pi	17
7.	Código para reconocimiento facial y trackeo	19
8.	Simulación a lazo abierto	34
9.	Simulación a lazo cerrado	35
10.	Código de Micropython final	38

1. Posibles trabajos, y selección

Al inicio de la cursada, se barajaban distintas opciones de trabajos a realizar, estas eran:

- **Control PID de balanza con motores brushless:** de trata de un control de motor sin escobillas PID con Arduino para un eje, la cual consiste en mantener su posición fija a pesar de las perturbaciones.
- **Proyecto Grúa usando control PID y control P:** para este proyecto se tiene en cuenta que control PID se encargará del control del movimiento vertical mientras que el control P se encargará del movimiento horizontal.
- **Proyecto serpiente animatrónica seguidora de rostro:** para este proyecto se contempla el uso de (por lo menos) cuatro motores para controlar el movimiento de una serpiente que se asoma por un muro. Un motor regula el movimiento de flexión hacia arriba y abajo, y otro regula el movimiento de flexión hacia izquierda y derecha. Por último dos motores se encargan de los mismos movimientos, pero de la cabeza de la serpiente por separado. El objetivo del proyecto es generar un control PID que regule los movimientos del conjunto para que estos sean los más rápidos posibles sin tener grandes oscilaciones, además de que tenga la capacidad de seguir rostros con un gran fiabilidad.

El proyecto elegido resultó ser el del seguidor de rostro, pero simplificando su aspecto mecánico a un solo grado de libertad: un movimiento horizontal. Debido a esto, también se modificó el concepto de serpiente, y se lo sustituyó por el de un dinosaurio el cuál realiza este movimiento en horizontal.

2. Planificación y diagrama de Gantt

El diagrama de Gantt es un método de organización, el cual consiste en la división de una tarea compleja en tareas menores que sean más sencillas de realizar. Posterior a esta “simplificación” se procede a determinar fechas y plazos en los que las tareas deben realizarse. Esta información se introduce en un diagrama (justamente el diagrama de Gantt) y permite tener una imagen clara de nuestro proyecto.

De esta manera uno se asegura de mantener un avance constante en el desarrollo de nuestra idea y de que no van a quedar actividades sin realizar.

Las tareas a realizar en este proyecto fueron las siguientes:

1. Búsqueda de ideas
2. Documentación
3. Definición de objetivos
4. Investigación motor stepper
5. Elaboración de código de Python para Face Tracking
6. Elaboración de código motor stepper (demo)
7. Control de stepper con L298N
8. Implementación de cámara en Raspberry Pi
9. Completar diagrama de Gantt
10. Mejora de conocimiento grupal
11. Prueba de driver Pololu A4988
12. Investigación de programación concurrente
13. Conclusiones selección de driver
14. Diseño de programa de unión entre motor y seguidor de rostro
15. Prueba de funcionamiento
16. Diseño de controlador PID
17. Simulación en Matlab
18. Diseño de cabeza de animatrónico
19. Elaboración de estructura de animatrónico
20. Montaje del proyecto
21. Conclusiones y prueba final

Con las anteriores tareas, y, sus fechas de inicio y fin, se obtiene el siguiente diagrama de Gantt.

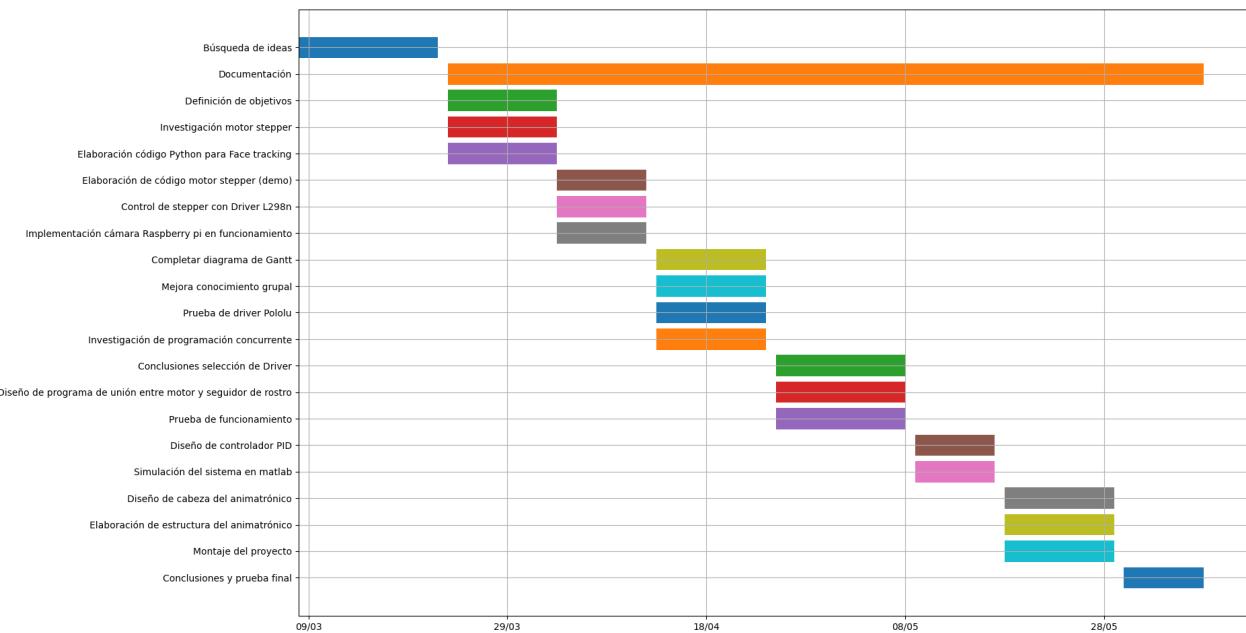


Figura 1: Diagrama de Gantt para la organización.

El anterior esquema se ploteó con Python. A modo ilustrativo, se adunta el código utilizado:

Código 1: Diagrama de Gantt

```

1 import matplotlib.pyplot as plt
2 from matplotlib.dates import DateFormatter
3 import pandas as pd
4
5 actividades = [
6     {'nombre': 'Búsqueda de ideas', 'inicio': '08/03', 'fin': '22/03'},
7     {'nombre': 'Documentación', 'inicio': '23/03', 'fin': '07/06'},
8     {'nombre': 'Definición de objetivos', 'inicio': '23/03', 'fin': '03/04'},
9     {'nombre': 'Investigación motor stepper', 'inicio': '23/03', 'fin': '03/04'},
10    {'nombre': 'Elaboración código Python para Face tracking',
11      'inicio': '23/03', 'fin': '03/04'},
12    {'nombre': 'Elaboración de código motor stepper (demo)',
13      'inicio': '03/04', 'fin': '12/04'},
14    {'nombre': 'Control de stepper con Driver L298n',
15      'inicio': '03/04', 'fin': '12/04'},
16    {'nombre': 'Implementación cámara Raspberry pi en funcionamiento',
17      'inicio': '03/04', 'fin': '12/04'},
18    {'nombre': 'Completar diagrama de Gantt', 'inicio': '13/04', 'fin': '24/04'},
19    {'nombre': 'Mejora conocimiento grupal', 'inicio': '13/04', 'fin': '24/04'},
20    {'nombre': 'Prueba de driver Pololu', 'inicio': '13/04', 'fin': '24/04'},
21    {'nombre': 'Investigación de programación concurrente',
22      'inicio': '13/04', 'fin': '24/04'},

```

```
23 {'nombre': 'Conclusiones selección de Driver',  
24     'inicio': '25/04', 'fin': '08/05'},  
25 {'nombre': 'Diseño de programa de unión entre motor y seguidor de rostro',  
26     'inicio': '25/04', 'fin': '08/05'},  
27 {'nombre': 'Prueba de funcionamiento', 'inicio': '25/04', 'fin': '08/05'},  
28 {'nombre': 'Diseño de controlador PID', 'inicio': '09/05', 'fin': '17/05'},  
29 {'nombre': 'Simulación del sistema en matlab',  
30     'inicio': '09/05', 'fin': '17/05'},  
31 {'nombre': 'Diseño de cabeza del animatrónico',  
32     'inicio': '18/05', 'fin': '29/05'},  
33 {'nombre': 'Elaboración de estructura del animatrónico',  
34     'inicio': '18/05', 'fin': '29/05'},  
35 {'nombre': 'Montaje del proyecto', 'inicio': '18/05', 'fin': '29/05'},  
36 {'nombre': 'Conclusiones y prueba final', 'inicio': '30/05', 'fin': '07/06'}  
37 ]  
38  
39 date_format = "%d/ %m"  
40 for actividad in actividades:  
41     actividad['inicio'] = pd.to_datetime(  
42         actividad['inicio'], format=date_format)  
43     actividad['fin'] = pd.to_datetime(actividad['fin'], format=date_format)  
44  
45 fig, ax = plt.subplots()  
46 y_labels = [actividad['nombre'] for actividad in actividades]  
47 y_positions = range(len(y_labels))  
48 ax.set_yticks(y_positions)  
49 ax.set_yticklabels(y_labels)  
50 ax.invert_yaxis()  
51  
52 date_formatter = DateFormatter(date_format)  
53 ax.xaxis.set_major_formatter(date_formatter)  
54  
55 for i, actividad in enumerate(actividades):  
56     inicio = actividad['inicio']  
57     fin = actividad['fin']  
58     ax.barh(i, fin - inicio, left=inicio)  
59  
60 plt.tight_layout()  
61 plt.grid()  
62 plt.show()
```

3. Análisis y rúbrica

Para el análisis y la rúbrica del proyecto grupal, se partirá de la referencia que se puede encontrar en el aula virtual de la materia. Allí, se detallan los procedimientos en 4 diferentes categorías, en donde las diferentes situaciones alcanzables en cada una de ellas representa una ponderación en una hipotética nota final del trabajo.

- Definición del problema de control: se entiende que el problema fue bien identificado, formalmente completo (en base a las investigaciones realizadas en cuanto a hardware y posibles implementaciones mediante software), es inherentemente no lineal, novedoso y difícil de resolver.
- Objetivos del proyecto: los objetivos fueron claros, alcanzables y alcanzados durante el tiempo en que se dictó la asignatura, pertinentes a la resolución del problema planteado.
- Metodología: la metodología fue adecuada para lograr los objetivos y la solución del problema planteado. A medida que se iba dictando la materia, se intentó relacionar los conceptos nuevos con lo que se estaba aplicando, de modo de poder mejorar los resultados obtenidos. Se incorporaron temáticas de otras asignaturas también (sobre todo en cuanto a programación y conceptos de métodos numéricos).
- Resultados esperados del proyecto: los resultados obtenidos están asociados a la resolución del problema propuesto, así como también se realizaron conexiones con proyectos de cátedras previas (como Sistemas de Control I). Se entiende que se lograron las competencias planteadas por la cátedra.

Se entiende que en cada una de las cuatro categorías se obtuvo un puntaje de 2,5 puntos.

4. Motor paso a paso

El motor paso a paso es un tipo de motor de corriente continua que convierte pulsos eléctricos en desplazamientos angulares de un eje. Está constituido por:

- Estator: Parte estacionaria del motor PaP Se montan en el bobinas para la creación se electroimanes.
- Rotor: Montado en el eje que gira en función de los electroimanes.

4.1. Clasificación

- Motor de reluctancia variable: constan de un estator con devanados de excitación y un rotor magnético, el rotor se alinea con la onda de flujo magnético producida por el estator, cuando se excita con corriente continua una de las bobinas de fase el rotor se alinea con la misma en la posición de mínima reluctancia.
- Motor de imán permanente: el rotor es un imán permanente funciona con el principio básico del magnetismo: polos de igual signo se repelen y polos signos opuestos de atraen, si el estator tiene una sola fase al invertir la corriente no se puede predeterminar el sentido de giro en cambio si el estator es de dos fases ya se elimina esa incertidumbre. Se suele mecanizar un número de dientes limitado por su estructura, el hecho de ser dentado hace que su posicionamiento no varía aun desapareciendo la excitación produciendo mayor precisión.
- Híbridos: combinación de ambos motores, el rotor suele estar constituido por anillos de acero dulce dentado en un número distinto al estator y dichos anillos están montados sobre un imán permanente.

4.1.1. Tipos de motores de imán permanente

- Unipolar: se trata de un motor que utiliza bobinas con centro en toma (center-tapped) y se conecta directamente a la fuente de alimentación a través de transistores de conmutación. Cada bobina tiene tres terminales: una conexión central y dos conexiones extremas. La polaridad de la corriente se invierte en cada fase para activar los polos magnéticos del motor. Aunque es fácil de controlar, este tipo de motor es menos eficiente debido a la necesidad de utilizar conexiones adicionales y a la dispersión de flujo magnético en las conexiones no utilizadas.
- Bipolar: este tipo de motor utiliza bobinas sin centro en toma y se conecta en serie para formar dos fases. Para su control se requieren transistores H o puentes H que permiten invertir la polaridad de la corriente en cada bobina de forma independiente. Al eliminar las conexiones no utilizadas y aprovechar mejor el flujo magnético, los motores paso a paso bipolares son más eficientes en comparación con los unipolares. Además, ofrecen un mayor torque gracias a su diseño y a la posibilidad de ajustar la corriente de forma más precisa.

Tabla 1: Comparación motores paso a paso a imán permanente.

	Motor paso a paso unipolar	Motor paso a paso bipolar
Conexión de bobinas	Centro en toma	Sin centro en toma
Control	Transistores de conmutación	Transistores H o puentes H
Eficiencia	Menor, conexiones adicionales	Mayor, conexiones optimizadas
Torque	Menor, dispersión de Φ_m	Mayor, mejor aprovechamiento
Controlador requerido	Sencillo	Possiblemente complejo

4.2. Características mecánicas

- Ángulo de paso (step angle): avance angular producido bajo un impulso de excitación.
- Número de pasos por vuelta: cantidad de paso a efectuar el rotor al realizar una revolución completa.
- Frecuencia máxima de paso: máximo número de pasos por segundo que el rotor puede efectuar.
- Paso completo: Consiste en activar cada una de las bobinas de forma independiente, lo que provoca que el eje del motor se oriente hacia la bobina activa. En algunos motores esto brinda un funcionamiento más suave.
- Medio Paso: En este caso se tiene que el paso común del motor se divide en dos. Se requiere energía alterna para las dos fases (bobinas). Esto da como resultado un funcionamiento "más suave" del motor y la duplicación de la resolución angular.

4.3. Motor 28BYJ-48

Es un motor de tipo unipolar ampliamente utilizado en aplicaciones donde se requiere un control de movimiento simple y preciso, como en impresoras, robots pequeños, sistemas de automatización y proyectos de electrónica en general. Es conocido por su bajo costo y facilidad de uso.

Su limitación radica en su diseño. Al ser de tipo unipolar, posee una eficiencia relativamente baja y un torque menor si se compara con su homólogo motor bipolar.

Entre sus características más importantes, se tiene:

- Tensión nominal de 5V (cinco cables)
- 4 fases
- Reductor de velocidad 1/64
- Ángulo de paso de $5.625^\circ/64$
- Frecuencia de 100 Hz
- Resistencia en CC $50\Omega \pm 7\%(25^\circ C)$
- Torque con tracción $> 34,3 mN.m$
- Arrastre en torque: $300gf.cm$

4.3.1. Caja reductora

La caja reductora se encarga de aumentar la fuerza o el par a velocidades muy bajas, disminuyendo el giro del motor, con el objetivo de aumentar la fuerza de empuje o torque.

Su explicación puede abordarse suponiendo dos ejes conectados entre sí: el principal y el eje externo, donde el primero debe dar sesenta y cuatro vueltas completas para representar en el eje externo una vuelta completa. De allí se deducen los siguientes cálculos con base a las especificaciones técnicas del motor:

- La cantidad de grados por paso que realiza el motor en el eje externo es $5,625^\circ / 64 = 0,088^\circ$.
- La cantidad de medios pasos necesarios para que el motor gire una vuelta completa es $64 \text{ (medios) pasos} * 64 \text{ vueltas} = 4096 \text{ medios pasos}$.
- Se requieren de 32 pasos simples, o completos, multiplicado por las 64 vueltas para dar un total de 2048 pasos en el eje externo.

4.3.2. Secuencia de pasos

Para definir la secuencia de pasos con la que girará el motor, es imprescindible tener conocimiento del orden en que deben encenderse los bobinados del motor. Se muestran las diferentes disposiciones a continuación.

PASO	BOBINA A	BOBINA B	BOBINA C	BOBINA D	
1	ON	ON	OFF	OFF	
2	OFF	ON	ON	OFF	
3	OFF	OFF	ON	ON	
4	ON	OFF	OFF	ON	

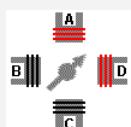


Figura 2: Paso completo o doble.

PASO	BOBINA A	BOBINA B	BOBINA C	BOBINA D	
1	ON	OFF	OFF	OFF	
2	OFF	ON	OFF	OFF	
3	OFF	OFF	ON	OFF	
4	OFF	OFF	OFF	ON	

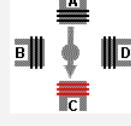


Figura 3: Paso simple.

PASO	BOBINA A	BOBINA B	BOBINA C	BOBINA D	
1	ON	OFF	OFF	OFF	
2	ON	ON	OFF	OFF	
3	OFF	ON	OFF	OFF	
4	OFF	ON	ON	OFF	
5	OFF	OFF	ON	OFF	
6	OFF	OFF	ON	ON	
7	OFF	OFF	OFF	ON	
8	ON	OFF	OFF	ON	

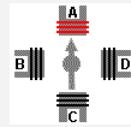


Figura 4: Medio paso.

4.3.3. Driver ULN2003

Como ya se expuso anteriormente, para controlar el motor 28BYJ-48, se requiere de un circuito de control que alterne la secuencia de activación de las bobinas en función del sentido de giro deseado. Esto se logra mediante el uso de transistores de conmutación, como transistores Darlington o transistores de potencia. Que es precisamente lo que es el dispositivo driver ULN2003.

Ampliando su definición, el driver ULN2003 es un circuito integrado utilizado para controlar cargas inductivas, como motores paso a paso o relés, a través de señales de bajo voltaje. Su principal objetivo es suministrar la corriente suficiente para mover el motor, y ello lo consigue con un conjunto de 7 pares de transistores en configuración Darlington, con diodo de protección de contracorriente. Cada salida es capaz de manejar 500 mA y hasta 50V en sus salidas.

Figura 5: Diagrama interno.

Los circuitos de transistores Darlington tienen un resistor en la entrada, de modo que se los puede conectar directamente a la salida de un microcontrolador. También tienen protección con un diodo para evitar daños en las entradas, si accidentalmente se les aplica un voltaje negativo.

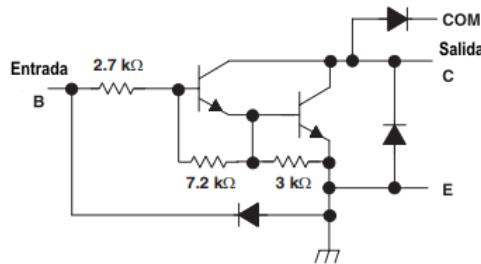


Figura 6: Circuito con transistores en configuración D'Arlington.

4.3.4. Conexión del motor

La conexión entre el módulo y el motor no requiere mucha atención ya que tiene un conector con ranuras para guiar la unión entre los dos dispositivos.



Figura 7: Conexión del driver ULN2003.

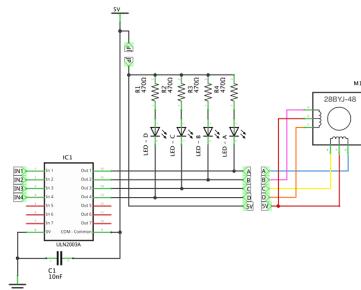


Figura 8: Conexión del driver con el motor.

4.3.5. Práctica con microcontrolador Arduino

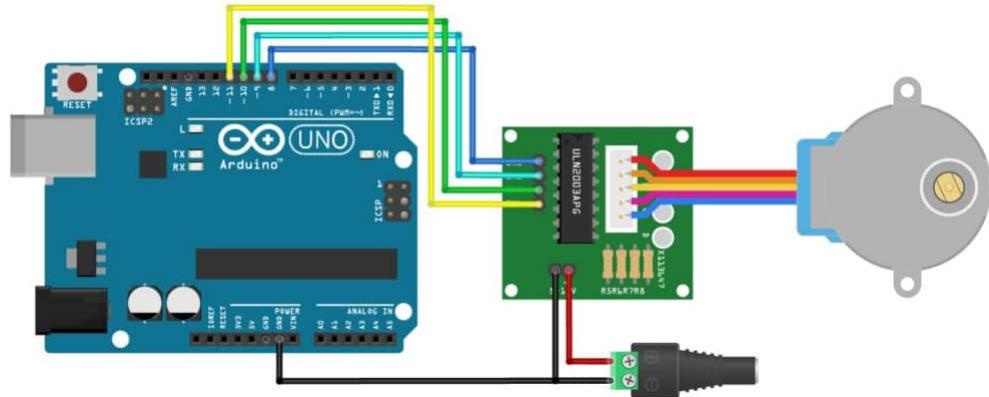


Figura 9: Conexión de arduino, driver y motor.

Una de las precauciones importantes (que un principio se pasó por alto) cuando se realizó el trabajo práctico fue que, para el funcionamiento correcto y seguro de los equipos, la alimentación del motor a 5v debe ser externa a la fuente con que se está alimentando el arduino, siempre y cuando se esté usando este último conectado a la computadora mediante el cable USB; que es lo que normalmente se suele hacer para tener mayor entre cargar el código del programa y su inmediata ejecución en el hardware de la placa.

Puesto la corriente necesaria aproximada para encender la bobina del motor es de 100 mA, la salida de 5V de la placa Arduino es suficiente para un motor, pero existe el peligro de exceder la corriente que es capaz de suministrar el microcontrolador (y en este caso la computadora), generando posibles daños a los equipos implicados.

Se emplea el entorno de programación Arduino IDE (Integrated Development Environment), el cual es utilizado para escribir, compilar y cargar código en las placas Arduino. Es una aplicación de software que proporciona una interfaz amigable para desarrollar proyectos con Arduino. Además está diseñado para ser fácil de usar, incluso para aquellos que no tienen experiencia previa en programación. Proporciona una amplia variedad de funciones y librerías predefinidas que simplifican el desarrollo de proyectos, permitiendo a los usuarios controlar sensores, actuadores y otros componentes electrónicos de manera sencilla.

A continuación, se presentan los códigos utilizados para controlar el movimiento del Stepper en dos modalidades:

Código 2: Código de Arduino para medio paso

```

1 int IN1= 8;
2 int IN2= 9;
3 int IN3= 10;
4 int IN4= 11;
5 int delaym=12;
6
7 int paso[8][4] ={ 
8 {1, 0, 0 ,0},
9 {1, 1, 0 ,0},
```

```

10  {0, 1, 0 ,0},
11  {0, 1, 1 ,0},
12  {0, 0, 1 ,0},
13  {0, 0, 1 ,1},
14  {0, 0, 0 ,1},
15  {1, 0, 0 ,1}
16 };
17 void setup() {
18   pinMode (IN1,OUTPUT);
19   pinMode (IN2,OUTPUT);
20   pinMode (IN3,OUTPUT);
21   pinMode (IN4,OUTPUT);
22 }
23
24 void loop() {
25   for (int i=0; i<512;i++){
26     for (int i=0; i<8;i++){
27       digitalWrite(IN1,paso[i][0]);
28       digitalWrite(IN2,paso[i][1]);
29       digitalWrite(IN3,paso[i][2]);
30       digitalWrite(IN4,paso[i][3]);
31       delay(delaym);
32     }
33   }
34 }
```

Código 3: Código Arduino para paso simple

```

1 int IN1= 8;
2 int IN2= 9;
3 int IN3= 10;
4 int IN4= 11;
5 int delaym=15;
6
7 void setup() {
8   pinMode (IN1,OUTPUT);
9   pinMode (IN2,OUTPUT);
10  pinMode (IN3,OUTPUT);
11  pinMode (IN4,OUTPUT);
12 }
13
14 void loop() {
15   for (int i=0; i<512;i++){
16     digitalWrite(IN1,HIGH);
17     digitalWrite(IN2,LOW);
18     digitalWrite(IN3,LOW);
19     digitalWrite(IN4,LOW);
20     delay(delaym);
21
22     digitalWrite(IN1,LOW);
23     digitalWrite(IN2,HIGH);
```

```

24   digitalWrite(IN3,LOW);
25   digitalWrite(IN4,LOW);
26   delay(delaym);

27
28   digitalWrite(IN1,LOW);
29   digitalWrite(IN2,LOW);
30   digitalWrite(IN3,HIGH);
31   digitalWrite(IN4,LOW);
32   delay(delaym);

33
34   digitalWrite(IN1,LOW);
35   digitalWrite(IN2,LOW);
36   digitalWrite(IN3,LOW);
37   digitalWrite(IN4,HIGH);
38   delay(delaym);
39 }
40 }
```

4.4. Driver Pololu A4988

El driver Pololu A4988, o su hermano, el DRV8825 son controladores que simplifican el manejo de un motor paso a paso para hacerlo desde un microcontrolador. Para el control propiamente dicho, requieren de 2 salidas digitales:

- Una para el sentido de giro
- Otra para el avance (paso a paso)

Además, estos controladores permiten el uso de microstepping, de modo de obtener más resolución que la que normalmente brinda el motor a utilizar. Las siguientes son las especificaciones técnicas.

- Color: rojo o verde
- Intensidad máxima: $2A$
- Tensión máxima: $35V$
- Microsteps: 16
- R_s típico: 0.05, 0.1, 0.2
- Fórmulas: $I_{max} = \frac{V_{ref}}{8R_s}$, $V_{ref} = 8I_{max}R_s$

Estos dispositivos cuentan con protecciones contra sobrecorriente, cortocircuito, sobretensión y sobretemperatura. Normalmente son utilizados para dispositivos CNC, plotters, impresoras 3D, escáneres 3D.

Estos controladores funcionan haciendo uso de un doble puente H constituidos por transistores MOSFET.

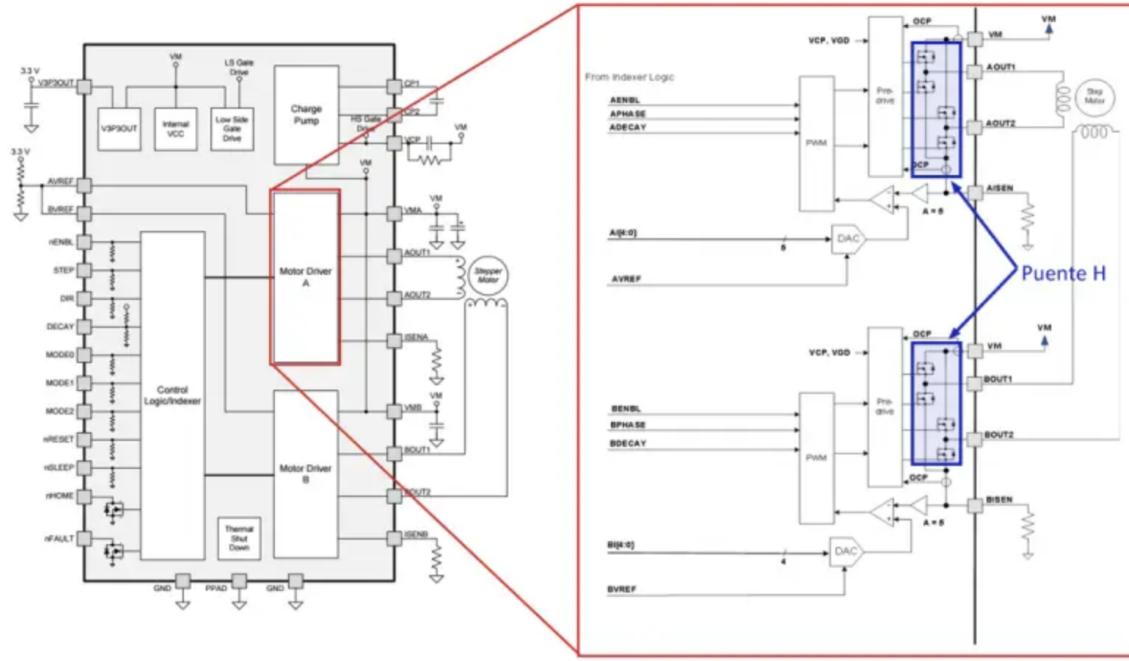


Figura 10: Esquema interno Pololu A4988.

El controlador dispone de reguladores de intensidad de modo de evitar la circulación de altas corrientes por las bobinas de los motores tipo NEMA 17 o NEMA 23, que son los motores para los que fueron diseñados en un inicio. Los motores paso a paso del estilo NEMA 17, como el que se utilizará en el proyecto requieren de tensiones alrededor de los 12V, y, teniendo en cuenta la resistencia de los bobinados (de aproximadamente 1.5Ω), circularán corrientes que superan por mucho a la nominal. El limitador interrumpe la señal proporcionando una señal PWM de forma que el valor promedio de la intensidad que atraviesa la bobina es la intensidad nominal del motor. Para poder ajustar el valor de corriente que proporciona el limitador es necesario ajustar un potenciómetro, y, para lograr el ajuste deseado, mediante una fórmula se obtiene un valor de Vref medido entre masa y el centro del trimmer.

El esquema de montaje general para este controlador es el siguiente:

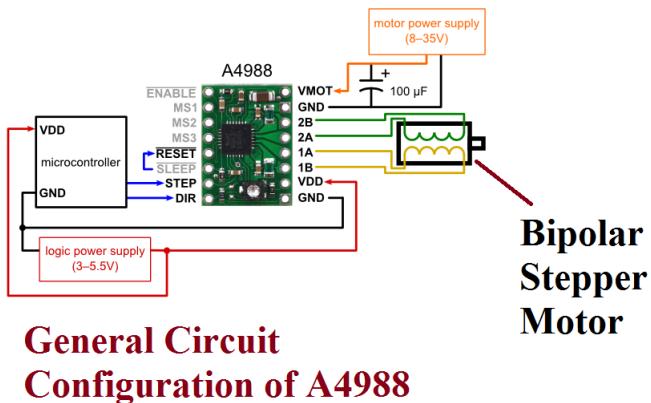


Figura 11: Conexión de driver con motor paso a paso.

4.5. Pruebas de código de Raspberry Pi

El primer código, probado en el campo, es el siguiente:

Código 4: Prueba de L298N en Raspberry Pi 4

```

1 #!/usr/bin/python3
2 import RPi.GPIO as GPIO
3 import time
4
5 # Se definen los pines a utilizar
6 out1 = 17
7 out2 = 18
8 out3 = 27
9 out4 = 22
10
11 # Tiempo de espera entre los pasos, hay que tener cuidado con este valor porque si es muy chico
12 # se puede estar sobrepasando un límite mecánico del motor
13 stepSleep = 0.01
14 # Número de pasos a realizar, para una vuelta entera se requieren 200 pasos
15 stepCount = 2000
16
17 # Se setean los diferentes pines (análogo a void setup de arduino)
18 GPIO.setmode(GPIO.BCM)
19 GPIO.setup(out1, GPIO.OUT)
20 GPIO.setup(out2, GPIO.OUT)
21 GPIO.setup(out3, GPIO.OUT)
22 GPIO.setup(out4, GPIO.OUT)
23
24 # Se inicializan todas las salidas en 0 (low)
25 GPIO.output(out1, GPIO.LOW)
26 GPIO.output(out2, GPIO.LOW)
27 GPIO.output(out3, GPIO.LOW)

```

```

28 GPIO.output(out4, GPIO.LOW)
29
30
31 # Se limpian las salidas
32 def cleanup():
33     GPIO.output(out1, GPIO.LOW)
34     GPIO.output(out2, GPIO.LOW)
35     GPIO.output(out3, GPIO.LOW)
36     GPIO.output(out4, GPIO.LOW)
37     GPIO.cleanup()
38
39
40 # Movimiento del stepper
41 try:
42     i = 0
43     for i in range(stepCount):
44         if i % 4 == 0:
45             GPIO.output(out4, GPIO.HIGH)
46             GPIO.output(out3, GPIO.LOW)
47             GPIO.output(out2, GPIO.LOW)
48             GPIO.output(out1, GPIO.LOW)
49         elif i % 4 == 1:
50             GPIO.output(out4, GPIO.LOW)
51             GPIO.output(out3, GPIO.LOW)
52             GPIO.output(out2, GPIO.HIGH)
53             GPIO.output(out1, GPIO.LOW)
54         elif i % 4 == 2:
55             GPIO.output(out4, GPIO.LOW)
56             GPIO.output(out3, GPIO.HIGH)
57             GPIO.output(out2, GPIO.LOW)
58             GPIO.output(out1, GPIO.LOW)
59         elif i % 4 == 3:
60             GPIO.output(out4, GPIO.LOW)
61             GPIO.output(out3, GPIO.LOW)
62             GPIO.output(out2, GPIO.LOW)
63             GPIO.output(out1, GPIO.HIGH)
64
65     time.sleep(stepSleep)
66
67 except KeyboardInterrupt:
68     cleanup()
69     exit(1)
70
71 cleanup()
72 exit(0)

```

Una alternativa al código anterior, y más simple, es la siguiente:

Código 5: Simplificación del código para L298N

```

1 Qm = 0 # (posición en grados)
2 Q = pi*32.5/180 # (grado máximo de visión de la cámara en una dirección)

```

```

3 To = 0.06 # (periodo) (periodo mínimo para velocidad máxima)
4 b1 = 0 # (pin 1)
5 b2 = 0 # (pin 2)
6 b3 = 0 # (pin 3)
7 b4 = 0 # (pin 4)
8 ciclo = [1, 0, 0, 0] # (array que determina que pin tiene un alto y cual tiene un bajo)
9
10 while E>=pi/200 or E<=-pi/200:
11     e=E;
12     if e<pi/200 and e>-pi/200:
13         break
14     T=Q*To/abs(e);
15     #(se asocia un pin a cada posición del array)
16     b1 = ciclo[0];
17     b2 = ciclo[1];
18     b3 = ciclo[2];
19     b4 = ciclo[3];
20     #(el siguiente código rota el array para variar donde hay un 1 en los pines)
21     #(rota en una dirección u otra si el signo de la entrada es negativo o positivo [invierte el giro del
22         ↵ stepper])
23     if e>0:
24         Qm = Qm+pi/100;
25         ciclo = ciclo[3:] + ciclo[0:3:];
26     elif e<0:
27         Qm = Qm-pi/100;
28         ciclo = ciclo[1:] + ciclo[0:1];
29     time.sleep(To/4)

```

Finalmente, y a modo de probar el funcionamiento del driver A4988, se tiene el siguiente código, que recibe por consola un ángulo y hace rotar al motor en esa dirección.

Código 6: Prueba Pololu A4988 en Raspberry Pi

```

1 import RPi.GPIO as GPIO
2 import time
3
4 # Configuración de los pines GPIO
5 DIR = 21
6 STEP = 20
7 CW = 1
8 CCW = 0
9 SPR = 200 # Pasos por vuelta del motor
10
11 # Inicialización de la Raspberry Pi
12 GPIO.setmode(GPIO.BCM)
13 GPIO.setup(DIR, GPIO.OUT)
14 GPIO.setup(STEP, GPIO.OUT)
15
16 # Función para controlar el motor
17 def step_motor(angle):

```

```
18 steps = int(angle / 360 * SPR) # Cálculo de los pasos necesarios
19 if steps < 0:
20     direction = CCW
21 else:
22     direction = CW
23
24 # Configuración de la dirección del motor
25 GPIO.output(DIR, direction)
26
27 # Generación de los pulsos para mover el motor
28 for _ in range(abs(steps)):
29     GPIO.output(STEP, GPIO.HIGH)
30     time.sleep(0.001) # Retardo entre pulsos
31     GPIO.output(STEP, GPIO.LOW)
32     time.sleep(0.001) # Retardo entre pulsos
33
34 # Obtención del ángulo desde la consola
35 angle = float(input("Ingrese el ángulo de rotación en grados: "))
36
37 # Llamada a la función para rotar el motor
38 step_motor(angle)
39
40 # Limpiar los pines GPIO y finalizar el programa
41 GPIO.cleanup()
```

5. Face tracking

Para la detección del rostro, se realizó un script de Python que utiliza las librerías siguientes:

- dlib
- OpenCV

Una vez importadas las librerías, se realiza una comprobación constante de la imagen que brinda la cámara conectada a la computadora en donde corra. Con el fin de realizar pruebas, este script fue probado en una computadora MacBook Air M1, pero, en el momento en que se deba embeber el comprobador de rostros en el sistema a implementar, se utilizará el módulo de cámara conectado a la Raspberry Pi.

Este script de detección de rostros muestra en pantalla la posición relativa al centro horizontal de la imagen de la webcam, de modo que esta sea la señal de error a utilizar en el script que controla el motor paso a paso. A modo de ilustración, a continuación, se muestra una figura en donde puede apreciarse el reconocimiento de rostro logrado con este programa.



Figura 12: Reconocimiento facial y trackeo.

El código de Python que logró el resultado anterior es el siguiente:

Código 7: Código para reconocimiento facial y trackeo

```
1 import cv2
2 import dlib
3
4 # Inicializo el cascade classifier de imágenes para OpenCV
5 faceCascade = cv2.CascadeClassifier(
6     '/Users/federicovillar/miniforge3/pkgs/libopencv-4.6.0-py310h5ab14b7_6/share/opencv4/
7         haarcascades/haarcascade_frontalface_default.xml'
8 )
9 # Dimensiones de la imagen de salida
```

```
10 OUTPUT_SIZE_WIDTH = 1280
11 OUTPUT_SIZE_HEIGHT = 720
12
13 def detectFace():
14     # Se inicializa la webcam de la posición 0, es decir, la que está por defecto
15     sampleVideo = cv2.VideoCapture(0)
16     # Se crea una ventana llamada "Reconocimiento" que mostrará la imagen con la cara remarcada
17     cv2.namedWindow("Reconocimiento", cv2.WINDOW_AUTOSIZE)
18     # Este es el faceTracker del paquete dlib
19     tracker = dlib.correlation_tracker()
20     # Se inicializa la variable a trackear en 0, posición inicial
21     trackingFace = 0
22     trackerColor = (0, 165, 255)
23     xValueString = ""
24     # Inicia el proceso de reconocimiento propiamente dicho
25     try:
26         """Bucle infinito para que la imagen sea un stream, la excepción se lanza en el momento en
27         que se aprieta Ctrl+C en terminal, y con eso salgo del bucle"""
28         while True:
29             # Se lee el último frame del video
30             rc, fullImage = sampleVideo.read()
31             # Para procesar de la mejor manera y optimizar recursos, se reescalda la imagen
32             originalImage = cv2.resize(fullImage, (480, 320))
33             # Si se presiona Q se cierran todas las ventanas que estén abiertas
34             # No logré que funcione sin esto, me parece que está demás
35             pressedKey = cv2.waitKey(2)
36             if pressedKey == ord('Q'):
37                 cv2.destroyAllWindows()
38                 exit(0)
39             # Se crea una copia del último frame para dibujarle el rectángulo
40             auxImage = originalImage.copy()
41             # Si no se está trackeando ninguna cara, entonces se trata de buscar alguna
42             if (not trackingFace):
43                 # Se pasa la imagen a escala de grises para el análisis
44                 grayImage = cv2.cvtColor(originalImage, cv2.COLOR_BGR2GRAY)
45                 # Se usa haarcascade para reconocer los rostros
46                 faces = faceCascade.detectMultiScale(grayImage, 1.3, 5)
47                 # Se inicializan las variables necesarias en 0
48                 maxArea = 0
49                 x = 0
50                 y = 0
51                 w = 0
52                 h = 0
53                 # Se calculan las áreas de las caras
54                 for (_x, _y, _w, _h) in faces:
55                     if _w * _h > maxArea:
56                         x = int(_x)
57                         y = int(_y)
58                         w = int(_w)
59                         h = int(_h)
60                         maxArea = w * h
```

```

61     # Se elige a la cara de mayor área, ver tema de minimo reconocible
62     if maxArea > 0:
63         # Se inicializa el tracker
64         tracker.start_track(
65             originalImage,
66             dlib.rectangle((x - 10), (y - 20), (x + w + 10),
67                           (y + h + 20)))
68         # Se pone el tracker en True, para saber que se está trackeando una cara
69         trackingFace = 1
70     # Se obtiene la posición de la cara
71     Value = ((x + (w / 2)) - 640) * (1280 / 480)
72     """xValueString = 'X: ' + str(xValue)"""
73     # Se comprueba si realmente se está trackeando una cara
74     if trackingFace:
75         # Se actualiza el tracker
76         trackingQuality = tracker.update(originalImage)
77         # Si la calidad del tracker es suficiente se obtienen las posiciones
78         if trackingQuality >= 8.75:
79             tracked_position = tracker.get_position()
80             t_x = int(tracked_position.left())
81             t_y = int(tracked_position.top())
82             t_w = int(tracked_position.width())
83             t_h = int(tracked_position.height())
84             # Obtenidas las posiciones, se puede imprimir la posición en el eje x
85             xValue = ((t_x + t_w / 2) * (1280 / 420) - 725
86             ) # convertir a radianes!
87             xValueString = 'X: ' + str(xValue.__round__(2))
88             cv2.rectangle(auxImage, (t_x, t_y), (t_x + t_w, t_y + t_h),
89                           trackerColor, 2)
90     else:
91         # Si la calidad no es suficiente, se vuelve a leer la imagen e intentar trackear
92         trackingFace = 0
93     # Se reescala la imagen nuevamente
94     largeResult = cv2.resize(auxImage,
95                             (OUTPUT_SIZE_WIDTH, OUTPUT_SIZE_HEIGHT))
96     printImage = cv2.putText(largeResult, xValueString, (200, 200),
97                             cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 255))
98     # Se abre una ventana para mostrar la imagen con el rectángulo
99     cv2.imshow("Reconocimiento", printImage)
100
101    # El comando Ctrl+C termina el proceso
102 except KeyboardInterrupt as e:
103     cv2.destroyAllWindows()
104     exit(0)
105
106 # Programa principal
107 if __name__ == '__main__':
108     detectFace()

```

6. Hardware utilizado

Para el desarrollo del proyecto se utilizaron los siguientes componentes:

- Motor Nema 17
- Raspberry Pi 4B
- Driver Pololu A4988
- Raspberry Pi Camera v1.3

6.1. Motor Nema 17

El motor NEMA 17 es un motor paso a paso ampliamente utilizado en aplicaciones de control de movimiento y robótica. Su nombre, NEMA 17, proviene de su tamaño de montaje estándar, que corresponde a una brida de 1.7 x 1.7 pulgadas. Este motor es conocido por su tamaño compacto y su alta precisión de posicionamiento.

El motor NEMA 17 encuentra aplicaciones en una variedad de campos, incluyendo impresoras 3D, máquinas CNC, robots, equipos médicos y automatización industrial. Su capacidad de generar movimientos precisos y reproducibles lo hace ideal para tareas que requieren posicionamiento exacto y control de velocidad.

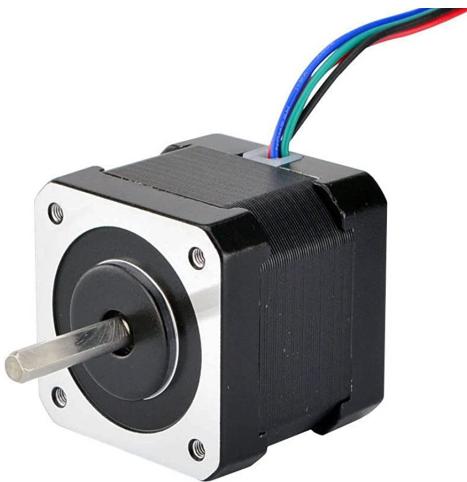


Figura 13: Motor Nema 17.

Este motor se caracteriza por tener una resolución alta, lo que significa que puede girar en pequeños pasos angulares con gran precisión. El NEMA 17 típicamente tiene una resolución de 1.8 grados por paso, lo que permite un control fino del movimiento. Además, algunos modelos pueden incluir opciones de engranajes para aumentar aún más la precisión y el torque.

Existen varios productos similares en el mercado, como el motor NEMA 23 o NEMA 34, que son de mayor tamaño y ofrecen mayor torque. Sin embargo, el NEMA 17 es especialmente popular debido a su tamaño compacto y a su relación equilibrada entre tamaño y rendimiento.

El funcionamiento del motor NEMA 17 se basa en la generación de pulsos eléctricos que hacen que el rotor se mueva en incrementos angulares. Estos pulsos se generan mediante un controlador, como

el Pololu A4988, que proporciona la secuencia de voltajes necesaria para controlar los bobinados del motor. Al cambiar el orden y el tiempo de los pulsos, es posible controlar la dirección y la velocidad del motor.

El motor NEMA 17 es compatible con una variedad de plataformas y controladores, como Arduino, Raspberry Pi y controladores específicos de motor paso a paso. Esto brinda a los usuarios la flexibilidad de elegir la plataforma y el entorno de desarrollo que mejor se adapten a sus necesidades y conocimientos. Además, existen bibliotecas y librerías disponibles que facilitan la programación y el control del motor en estas plataformas.

6.2. Raspberry Pi 4B

La Raspberry Pi 4B es una placa de desarrollo de bajo costo y tamaño reducido que ha ganado popularidad en el ámbito de la electrónica y la informática. Diseñada por la Fundación Raspberry Pi, esta placa ofrece una potente capacidad de procesamiento y una amplia gama de aplicaciones.

La Raspberry Pi 4B se ha utilizado en una variedad de proyectos, desde sistemas de automatización del hogar y servidores web hasta estaciones meteorológicas y sistemas de vigilancia. Su flexibilidad y capacidades la hacen adecuada para entusiastas, estudiantes y profesionales de diversas disciplinas.

La placa Raspberry Pi 4B cuenta con un procesador ARM Cortex-A72 de cuatro núcleos que ofrece un rendimiento significativamente mejorado en comparación con sus predecesoras. Además, ofrece opciones de memoria RAM de hasta 8 GB, lo que la hace adecuada para aplicaciones que requieren un mayor poder de procesamiento.

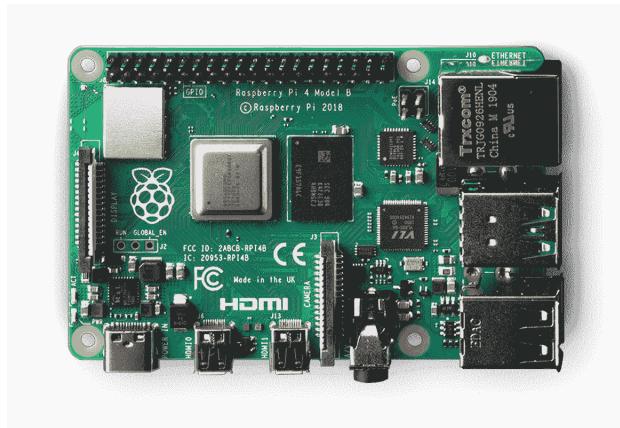


Figura 14: Raspberry Pi 4B.

Aunque existen otros productos similares en el mercado, como Arduino, la Raspberry Pi se destaca por su mayor capacidad de procesamiento y flexibilidad. Mientras que Arduino se enfoca principalmente en proyectos de electrónica y controladores, la Raspberry Pi se puede utilizar como una computadora completa, con capacidades de red, almacenamiento y periféricos.

El funcionamiento de la Raspberry Pi 4B se basa en un sistema operativo, como Raspbian (una distribución de Linux), que se instala en una tarjeta microSD. La placa incluye una variedad de puertos, como HDMI, USB, Ethernet y GPIO (General Purpose Input/Output), que permiten la conexión de periféricos y la interacción con otros dispositivos electrónicos.

En términos de plataformas soportadas, la Raspberry Pi 4B es compatible con una amplia gama de sistemas operativos, incluyendo Raspbian, Ubuntu, Windows 10 IoT Core y muchos más. Esto proporciona a los usuarios opciones flexibles para desarrollar y ejecutar sus proyectos según sus

necesidades y preferencias. La comunidad de Raspberry Pi es activa y ofrece un amplio soporte en forma de tutoriales, foros y proyectos de código abierto, lo que facilita el aprendizaje y el desarrollo en esta plataforma.

6.3. Raspberry Pi Camera v1.3

La cámara Raspberry Pi v1.3 es un módulo de cámara diseñado específicamente para las placas Raspberry Pi. Este módulo proporciona capacidades de captura de imágenes y grabación de video de alta calidad en un formato compacto y de fácil integración.



Figura 15: Raspberry Pi Camera v1.3.

La cámara v1.3 cuenta con un sensor de imagen de 5 megapíxeles que permite capturar fotografías nítidas y detalladas. También es capaz de grabar videos en resoluciones de hasta 1080p a 30 cuadros por segundo, lo que brinda una calidad visual bastante aceptable para una variedad de aplicaciones.

El módulo de cámara se conecta a la placa Raspberry Pi a través de un conector específico llamado CSI (Camera Serial Interface). Esto permite una conexión rápida y estable, garantizando una transferencia de datos eficiente entre la cámara y la placa.

La cámara v1.3 es compatible con una amplia gama de aplicaciones, como fotografía, videovigilancia, detección de movimiento, reconocimiento facial y proyectos de visión artificial. Su tamaño compacto y su fácil integración la hacen ideal para proyectos de bricolaje y prototipado rápido.

Para utilizar la cámara, la Raspberry Pi ofrece una interfaz de programación sencilla y poderosa. A través de lenguajes de programación como Python y librerías específicas, los usuarios pueden controlar la cámara, capturar imágenes y grabar videos, así como realizar operaciones de procesamiento de imágenes en tiempo real.

Es importante tener en cuenta que la cámara v1.3 es una versión anterior de la cámara Raspberry Pi y ha sido reemplazada por modelos más recientes. Sin embargo, sigue siendo una opción popular debido a su rendimiento confiable y su compatibilidad con las placas Raspberry Pi más antiguas.

7. Herramientas de Software Utilizadas

7.1. Visual Studio Code

Visual Studio Code es un entorno de desarrollo integrado (IDE) desarrollado por Microsoft que ha ganado popularidad en la comunidad de programadores. Este potente IDE se destaca por su interfaz intuitiva y fácil de usar, lo que facilita la navegación y el acceso a las herramientas necesarias para el desarrollo de software. Además, ofrece una amplia gama de extensiones y complementos que amplían su funcionalidad, permitiendo a los desarrolladores personalizar su experiencia de desarrollo según sus necesidades específicas.

Una de las características notables de Visual Studio Code es su editor de código, que proporciona resaltado de sintaxis, sugerencias inteligentes y herramientas de edición y refactorización. Estas características ayudan a los programadores a escribir código eficientemente y a detectar errores de manera más rápida. Asimismo, la función de depuración integrada ofrece un entorno sólido para identificar y solucionar problemas en el código.

Visual Studio Code es altamente compatible y admite una amplia gama de lenguajes de programación y frameworks, lo que lo convierte en una opción versátil para los desarrolladores. Desde JavaScript, Python y C hasta HTML, CSS y TypeScript, este IDE ofrece soporte y herramientas de desarrollo para diversas tecnologías.

7.2. Thonny IDE

Thonny IDE es un entorno de desarrollo integrado (IDE) diseñado específicamente para principiantes en la programación. Este IDE ofrece una interfaz simple y amigable que facilita el proceso de aprendizaje y desarrollo de software. Thonny IDE se destaca por su enfoque en la simplicidad y la facilidad de uso, lo que lo convierte en una opción popular para aquellos que están dando sus primeros pasos en la programación.

Una de las características destacadas de Thonny IDE es su diseño minimalista. La interfaz de usuario limpia y sin distracciones permite a los principiantes concentrarse en el código y las tareas de programación sin sentirse abrumados por una gran cantidad de opciones y herramientas complejas. Esto hace que Thonny IDE sea accesible y menos intimidante para aquellos que se están familiarizando con los conceptos básicos de la programación.

Thonny IDE ofrece un editor de código sencillo pero funcional que proporciona resaltado de sintaxis, sugerencias automáticas y otras características esenciales para facilitar la escritura y edición del código. Además, cuenta con una función de depuración integrada que ayuda a los programadores a identificar y corregir errores en su código paso a paso, lo cual resulta especialmente útil para aquellos que están aprendiendo a programar.

Otra ventaja de Thonny IDE es su capacidad para interactuar directamente con la Raspberry Pi. Esto permite a los usuarios desarrollar y ejecutar programas en la Raspberry Pi de manera más sencilla, sin tener que alternar entre diferentes entornos o herramientas de programación. Thonny IDE ofrece una experiencia integrada que facilita el desarrollo de proyectos para la Raspberry Pi.

7.3. Micropython

MicroPython es una implementación del lenguaje de programación Python diseñada específicamente para dispositivos de recursos limitados, como microcontroladores y sistemas embebidos. Se basa en el lenguaje Python estándar, pero ha sido optimizado para ejecutarse en entornos con me-

memoria y capacidad de procesamiento reducidas.

Una de las características destacadas de MicroPython es su simplicidad y facilidad de uso. Proporciona una sintaxis similar a Python, lo que facilita la transición para aquellos que ya están familiarizados con el lenguaje. Los desarrolladores pueden aprovechar la potencia y la flexibilidad del lenguaje Python para programar dispositivos embebidos de manera más sencilla y rápida.

MicroPython ofrece un entorno de programación interactivo que permite a los usuarios escribir y ejecutar código directamente en el dispositivo. Esto facilita el proceso de desarrollo y depuración, ya que los programadores pueden probar su código en tiempo real y obtener retroalimentación inmediata. Además, MicroPython admite la escritura de scripts y programas completos que se pueden ejecutar de forma autónoma en el dispositivo.

La implementación de MicroPython es altamente eficiente en términos de recursos. El intérprete y las bibliotecas están diseñados para utilizar de manera óptima la memoria y el procesador disponibles en los microcontroladores. Esto permite ejecutar programas Python en dispositivos con limitaciones de memoria y potencia, lo que los hace adecuados para una amplia gama de aplicaciones, como sistemas de sensores, dispositivos IoT y proyectos de robótica.

MicroPython también es altamente portable y es compatible con una amplia variedad de plataformas de hardware, incluyendo Arduino, ESP8266, ESP32 y muchas más. Esto permite a los desarrolladores utilizar MicroPython en diferentes dispositivos y aprovechar sus características en diversos proyectos.

7.4. Raspbian (Raspberry OS)

Raspbian es un sistema operativo basado en Linux y específicamente diseñado para las populares placas Raspberry Pi. Es una distribución oficial respaldada por la Fundación Raspberry Pi y está optimizada para ofrecer un entorno de desarrollo y uso amigable para usuarios de todas las edades y niveles de experiencia.

Una de las características principales de Raspbian es su fácil instalación y configuración. Se proporciona como una imagen descargable que se puede escribir en una tarjeta SD, la cual luego se inserta en la Raspberry Pi para iniciar el sistema operativo. Esto permite a los usuarios comenzar rápidamente con sus proyectos sin tener que pasar por complicados procesos de instalación.

Raspbian se basa en el sistema operativo Debian, una distribución popular y estable de Linux. Ofrece una amplia gama de software y herramientas, incluyendo el entorno de escritorio LXDE (Light-weight X11 Desktop Environment), que proporciona una interfaz gráfica intuitiva para la Raspberry Pi. Además, Raspbian es compatible con una gran cantidad de aplicaciones y bibliotecas específicas para el desarrollo en la plataforma Raspberry Pi.

Una de las ventajas clave de Raspbian es su estrecha integración con el hardware de la Raspberry Pi. Esto permite un acceso y control completo a los puertos GPIO (General Purpose Input/Output) de la placa, así como a otros componentes como la cámara, el Wi-Fi y el Bluetooth. Los usuarios pueden aprovechar estas capacidades para desarrollar una amplia variedad de proyectos, desde automatización del hogar hasta robótica y aplicaciones de IoT.

Raspbian también cuenta con una activa comunidad de usuarios y desarrolladores, lo que brinda un amplio soporte y recursos adicionales. Hay numerosos tutoriales, documentación y foros disponibles para ayudar a los usuarios a resolver problemas y aprender nuevas habilidades.

7.5. OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de código abierto ampliamente utilizada y altamente popular para aplicaciones de visión por computadora y procesamiento de imágenes. Proporciona una amplia gama de algoritmos y funciones que permiten a los desarrolladores procesar, analizar y manipular imágenes y videos de manera eficiente.

Una de las características destacadas de OpenCV es su capacidad para trabajar con una amplia variedad de formatos de imagen y video. Puede leer y escribir archivos en diferentes formatos, como JPEG, PNG, TIFF y MPEG, lo que permite a los usuarios trabajar con diferentes fuentes de datos. Además, OpenCV puede capturar y procesar secuencias de video en tiempo real desde cámaras y otros dispositivos de captura.

OpenCV ofrece un conjunto diverso de funciones y algoritmos que cubren una amplia gama de tareas relacionadas con la visión por computadora. Esto incluye operaciones básicas de manipulación de imágenes, como recorte, escalado y rotación, así como técnicas más avanzadas, como detección de objetos, seguimiento de movimiento, reconocimiento facial, segmentación de imágenes y más. Estas capacidades hacen de OpenCV una herramienta esencial para aplicaciones de visión artificial en áreas como robótica, seguridad, detección de objetos, análisis médico, entre otros.

OpenCV es compatible con varios lenguajes de programación, incluyendo C++, Python y Java, lo que facilita su integración en diferentes entornos de desarrollo. Además, está disponible en múltiples plataformas, como Windows, Linux, macOS y dispositivos móviles, lo que permite a los desarrolladores crear aplicaciones de visión por computadora en una amplia gama de dispositivos y sistemas operativos.

Otra ventaja de OpenCV es su comunidad activa de desarrolladores y su amplia documentación. La biblioteca cuenta con una gran cantidad de recursos, tutoriales y ejemplos que ayudan a los usuarios a comprender y aprovechar al máximo sus características y funcionalidades. Además, la comunidad de OpenCV está constantemente trabajando en el desarrollo y mejora de la biblioteca, asegurando su actualización y relevancia continuas.

7.6. Dlib

Dlib es una biblioteca de software de código abierto que ofrece un conjunto de herramientas y algoritmos avanzados para la visión por computadora y el aprendizaje automático. Destacándose por su versatilidad y eficiencia, Dlib proporciona una amplia gama de capacidades, desde detección de objetos y seguimiento facial hasta reconocimiento de emociones y modelado de forma facial.

Una de las características más destacadas de Dlib es su capacidad para trabajar con imágenes y videos en tiempo real. Esto se logra mediante la optimización de algoritmos y el aprovechamiento de la capacidad de procesamiento paralelo de las GPU. Como resultado, Dlib permite la detección y el seguimiento de objetos en tiempo real, lo que lo convierte en una herramienta valiosa para aplicaciones en tiempo real, como sistemas de vigilancia y detección de movimiento.

Dlib ofrece un conjunto de algoritmos altamente precisos y eficientes para el reconocimiento facial. Sus algoritmos avanzados permiten la detección y localización precisa de rostros en imágenes y videos, así como el reconocimiento de características faciales y la estimación de emociones. Esto lo convierte en una herramienta poderosa para aplicaciones de seguridad, análisis de imágenes y reconocimiento facial.

Además de la visión por computadora, Dlib también proporciona herramientas para el aprendizaje automático. Incluye implementaciones eficientes de algoritmos populares, como máquinas de vectores de soporte (SVM), clasificadores bayesianos ingenuos (Naive Bayes) y redes neuronales convolucionales.

les (CNN). Esto permite a los desarrolladores entrenar y aplicar modelos de aprendizaje automático en una amplia gama de aplicaciones, como clasificación de imágenes y detección de objetos.

Dlib es una biblioteca multiplataforma y es compatible con varios lenguajes de programación, incluyendo C++, Python y Java. Además, su comunidad activa de usuarios y desarrolladores brinda soporte y recursos adicionales, incluyendo documentación detallada y ejemplos de código.

8. Conjunción face tracking y motores (ideas y utilizado)

Luego de desarrollar ambas partes del programa (face tracking y movimiento del stepper), debemos unificarlas para poder ejecutar el sistema en su totalidad.

Debemos considerar los efectos que van a tener los programas entre sí. El hecho de que comparten una variable (posición del rostro), nos lleva a la situación de que cada uno la maneje por separado, ya que una variable no puede ser escrita (face tracking) y leída (stepper movement) a la vez, por lo que debemos tener en cuenta esto a la hora de unir ambos programas para así evitar errores. Otro factor a considerar es que el tiempo de ejecución de un programa no afecte notoriamente los tiempos de sleep que determinan la frecuencia de giro de los steppers.

Para resolver estos problemas se determinaron tres alternativas diferentes, las cuales se definirán a continuación.

8.1. Métodos de ejecución del programa (alternativas)

8.1.1. Software y hardware

En este caso la forma de establecer un paralelismo entre programas radica en que corramos ambos códigos en hilos diferentes de la raspberry y que la transmisión de esta variable común se haga a través de un pin de salida y otro de entrada mediante una conexión física. Es decir, el programa de face tracking imprimirá el valor de posición del rostro en (por ejemplo) el pin 1 y este estará conectado de alguna forma al pin 2, el cual funciona como input del código del stepper movement, siendo leído por el mismo. De esta manera ambos códigos se ejecutan por separado, sin interferirse entre sí en tiempos de ejecución, y nos resuelve el problema de tener una variable compartida entre ambos, dado que esta no está definida por software, sino por hardware.

8.1.2. Concurrencia por software

En este caso, al igual que en el anterior, se ejecutan ambos programas en paralelo, en hilos distintos de la raspberry. Pero en este caso, sí se determina por software una variable que será compartida.

Para evitar conflictos de escritura y lectura, se determinan procedimientos de lockeo de las variables cada vez que alguno de los hilos que ejecutan las tareas tienen que acceder a las mismas. En este caso, se corre en un hilo el programa de face tracking y en otro hilo el programa de stepper movement. De esta forma ambas tareas se ejecutan por separado, sin interferir entre ellas en términos de tiempo de ejecución, y no presentan conflicto de escritura y lectura debido a que se establecen los sistemas de lockeo comentados anteriormente. A continuación se puede ver un diagrama de flujo explicando superficialmente lo anterior:

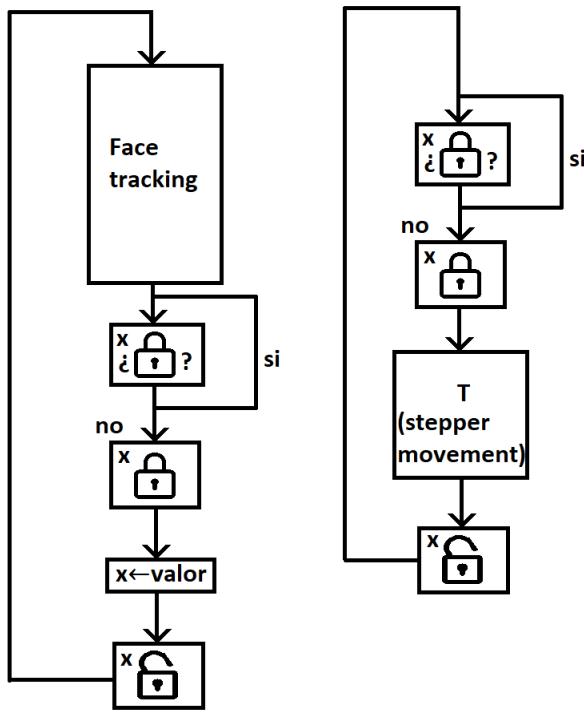


Figura 16: Diagrama de flujo para la alternativa concurrente.

Este método es el mejor de todos, debido a su consistencia, velocidad y funcionalidad.

8.1.3. Secuencial

Consiste en ejecutar los programas de forma secuencial, es decir, primero ejecutar el face tracking y a continuación ejecutar el stepper movement con el dato obtenido anteriormente, para luego volver a ejecutar el face tracking y seguir así.

En este caso no se presentan conflictos de lectura y escritura, pero no tenemos la ejecución en paralelo, por lo que la ejecución del código de stepper movement debe esperar a la finalización del face tracking para ejecutarse, perdiendo tiempo y ralentizando al sistema.

Como bien se mencionó anteriormente en el informe, tenemos que la frecuencia de señal que alimenta los steppers depende de un valor T que se calcula en función de la ubicación del rostro, y dado que el stepper presenta una variación angular fija (un paso), es justamente mediante la variación de esta T que podemos obtener las diferentes velocidades angulares del stepper:

$$\omega = \frac{\text{paso}}{T}$$

Sin embargo, debido a que la ejecución del face tracking conlleva un tiempo σ , al tiempo T entre paso y paso se le adicionará ese tiempo, el cual es fijo, por lo que la velocidad máxima del stepper quedará limitada por esta constante de tiempo:

$$\omega = \frac{\text{paso}}{T + \sigma}$$

Si T resulta ser cero (situación no posible, pero refiere a cuando la velocidad angular debería ser infinita), la velocidad angular resulta ser:

$$\omega = \frac{paso}{\sigma}$$

Como notamos, esta velocidad es finita, y genera una ralentización inevitable en nuestro sistema. Dependiendo de la velocidad de nuestro sistema, por ejemplo si la velocidad máxima del mismo estuviese una década por debajo de este límite, este último podría ser imperceptible, pero debemos tener en cuenta sus posibles efectos a la hora de evaluar el correcto funcionamiento de nuestro sistema ya que podría introducir errores significativos.

Esta última forma fue la implementada debido a su simpleza y basándonos en que, como mencionamos, el tiempo σ es una década más pequeño que el periodo T mínimo con el que trabajaremos.

9. Modelo inicial función de transferencia para seguidor de rostro

9.1. Cambio de variable en el manejo del error

Debido a que el sistema de reconocimiento facial entrega la posición del rostro en función de los píxeles que separan el centro del rostro con el centro de la pantalla, debemos convertir ese dato en una equivalencia angular en radianes, donde el centro de la imagen representa los 0° y, debido a que el ángulo de visión de la cámara es de 65° , el lateral izquierdo de la imagen representa $-32,5^\circ$ y el lateral derecho $32,5^\circ$. En radianes $32,5^\circ$ es $0,57$ aproximadamente.

Este cambio de variables lo realizamos dado que el movimiento del motor paso a paso es en radianes, y lo que requiere la función para girar el motor a una posición en radianes específica es justamente la posición en radianes hacia donde tiene que moverse. Esta conversión la realizamos de la siguiente forma:

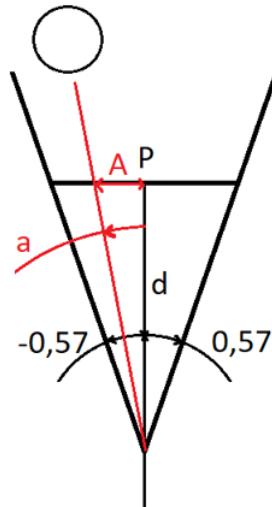


Figura 17: Análisis del error.

Definimos P como la medida en píxeles de la pantalla, d como la distancia ficticia entre la lente de la cámara y la pantalla (planteamos los cálculos como si de un proyector sobre una pared se tratara), A será la distancia en píxeles entre el centro de la pantalla y la cara (es el dato de salida del reconocedor facial) y a será el ángulo en radianes que deseamos obtener.

Se tiene por trigonometría que:

$$\operatorname{tg}(0.57) = \frac{P}{2d} \Rightarrow d = \frac{P}{2\operatorname{tg}(0.57)}$$

Y también:

$$\operatorname{tg}(a) = \frac{A}{d} \Rightarrow d = \frac{A}{\operatorname{tg}(a)}$$

Entonces se pueden igualar ambas:

$$\frac{P}{2tg(0.57)} = \frac{A}{tg(a)} \Rightarrow tg(a) = \frac{2Atg(0.57)}{P}$$

Despejando a :

$$a = tg^{-1}\left(\frac{2Atg(0.57)}{P}\right)$$

Sustituyendo $P = 480$ y resolviendo la tangente de 0.57:

$$a = tg^{-1}\left(\frac{2A(0.637)}{480}\right)$$

Resolviendo, resulta finalmente que:

$$a = tg^{-1}(0.00265A)$$

Utilizando esta equivalencia podemos convertir los valores de píxeles a radianes, y utilizar este valor en radianes como nuestra señal de error (R-Y), la cual es la entrada a nuestro controlador y posterior función de transferencia que determina el giro del motor.

9.2. Definición de las constantes de operación del sistema

El motor paso a paso con el que se trabaja presenta unos 200 pasos en una vuelta completa. Debido al funcionamiento de este tipo de motores, sabemos que su giro lo producen mediante una alimentación senoidal alterna a cada uno de sus bobinas (en este caso dos). Esta alimentación debe estar desfasada 90° entre ambas bobinas, esto para producir justamente el giro del campo magnético del estator y que el rotor pueda seguirlo. La frecuencia de estas señales sinusoidales determina la frecuencia de giro del motor. En este caso debido a que son dos bobinas tenemos que, por ciclo de la señal de la alimentación, el rotor girará unos cuatro pasos. De esta forma podemos con una simple fórmula determinar cual será la relación entre la frecuencia de la señal de alimentación y la velocidad angular del rotor.

$$\omega_m = \frac{2\pi f}{50} \Rightarrow f = \frac{25\omega_m}{\pi}$$

Se determina la velocidad angular como $\omega_m = \frac{2\pi}{3s}$, la frecuencia de alimentación resulta:

$$f = \frac{16.67}{s} \Rightarrow T = 0.06s$$

Por otro lado tenemos que la amplitud angular de visión de la cámara es de 65°. Dado que el sistema está centrado, y planteamos el centro como los 0°, esto nos determina que existirán 32° y -32° (cada uno para cada dirección), que en radianes son aproximadamente 0.57 (valor absoluto), y lo llamaremos . Desde acá podemos notar que una cara no podrá ser captada fuera de este ángulo de visión, y con la intención de que la velocidad máxima planteada más arriba se dé cuando el rostro esté en los límites de la visión de la cámara, normalizamos dentro de la señal de alimentación a la frecuencia en función de cual es la posición del rostro en la pantalla. Esto quiere decir que el argumento del seno será:

$$\text{argumento} = u \frac{2\pi f}{\theta}$$

En donde:

- u : es la entrada del sistema medida en radianes (posición del rostro).
- θ : ángulo máximo de visión de la cámara en radianes.

Adicionado a esto, debido a la relación planteada previamente, la función de transferencia entre la entrada del sistema (posición del rostro en radianes) y la salida (velocidad angular del rotor en radianes por segundo) resulta ser:

$$\frac{\omega}{u} = \frac{2\pi}{3\theta} \left[\frac{1}{s} \right]$$

Para obtener la salida en radianes debemos integrar la salida actual, lo que significa multiplicar por $\frac{1}{s}$ a la función de transferencia anterior:

$$\frac{y}{u} = \frac{2\pi}{3\theta s}$$

A continuación hacemos una simulación del sistema y definimos una realimentación unitaria para poder observar cómo se comporta el mismo si la acción proporcional es con $K_P = 1$.

Primero, a lazo abierto se tiene:

Código 8: Simulación a lazo abierto

```

1 s = tf('s');
2 wm = 2*pi/3;
3 theta = pi*32.5/180;
4 FDT = wm/(theta*s)
5 step(FDT)
```

Se obtiene:

$$G(s) = \frac{2.094}{0.5672s}$$

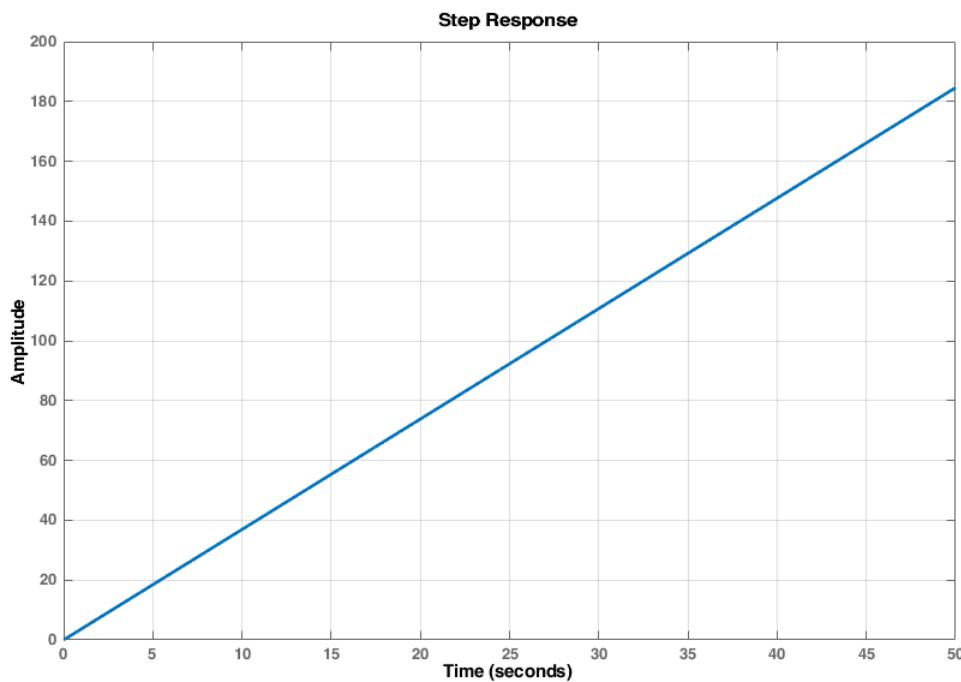


Figura 18: Respuesta del sistema a lazo abierto.

Podemos notar que a lazo abierto el ángulo de giro en radianes crece ilimitadamente debido a que no hay control alguno que determine que el sistema deje de girar: el sistema no es "consciente" de sus posición.

A continuación simulamos el sistema a lazo cerrado con realimentación unitaria y . Tomamos una entrada escalón de 0.28 de amplitud, lo que simula una distancia de 0.28 radianes del centro de visión al rostro:

Código 9: Simulación a lazo cerrado

```

1 FDTLC = feedback(FDT,1)
2 step(0.28*FDTLC)
```

Se obtiene:

$$G(s) = \frac{2.094}{0.5672s + 2.094}$$

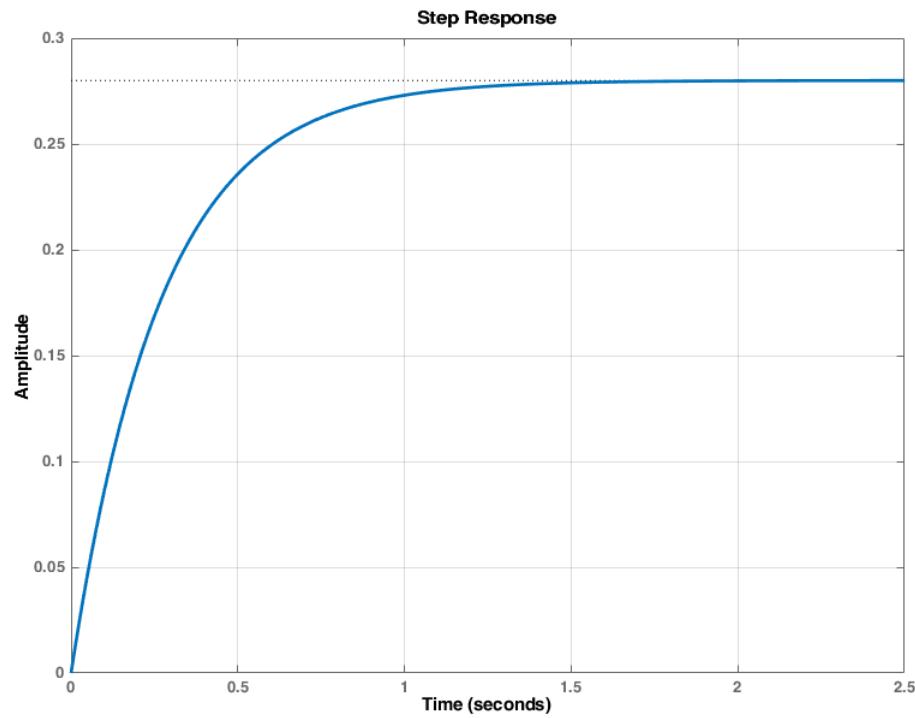


Figura 19: Respuesta del sistema a lazo cerrado.

Podemos notar como ahora la posición converge al valor de la entrada, es decir, como la cámara se mueve hacia la posición del rostro.

Lo próximo a esto sería aplicar una acción de control superior a la aplicada para mejorar la respuesta del sistema.

10. Implementación

10.1. Conexiones del hardware

El hardware utilizado sigue la siguiente conexión en forma topológica.

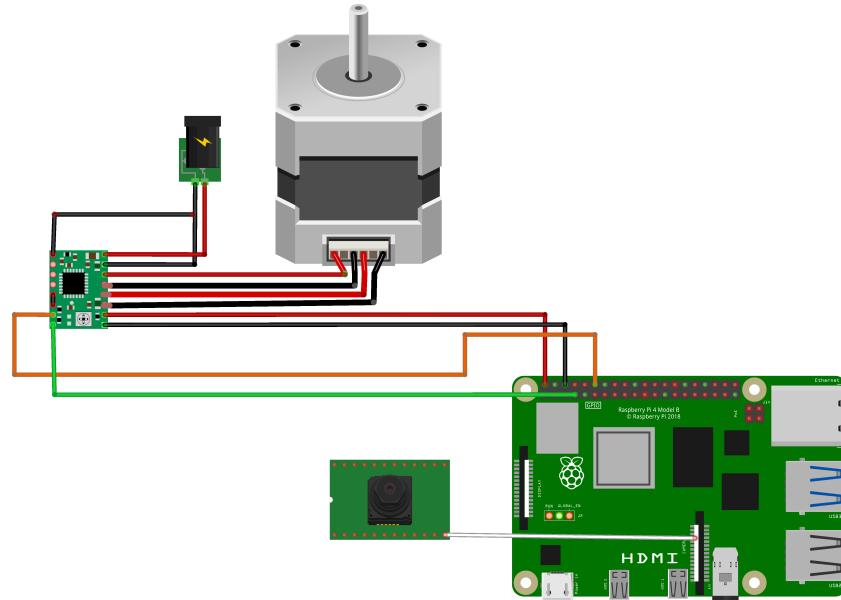


Figura 20: Circuito topológico.

Ahora, la conexión de la Raspberry Pi con el driver A4988 y el motor Nema 17 es la siguiente.

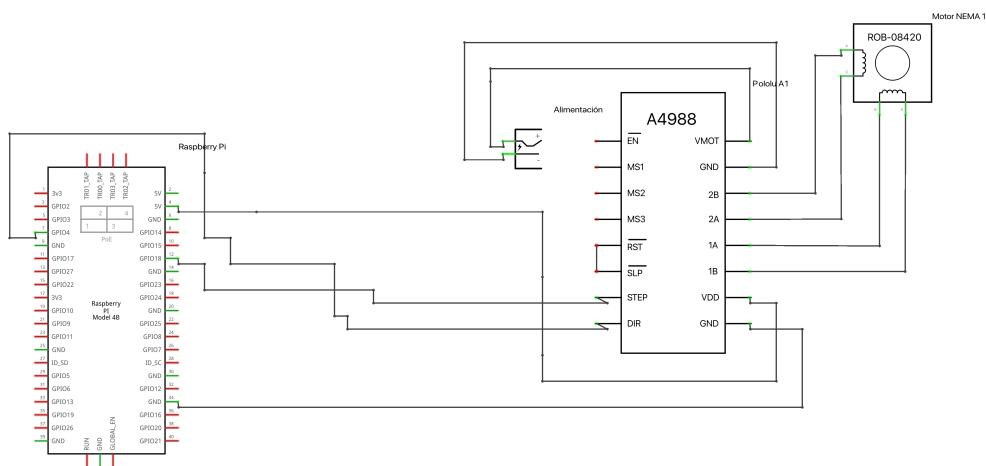


Figura 21: Circuito esquemático.

Como alimentación del sistema se utilizó una fuente ATX genérica de PC, de 500W.

10.2. Código de programación utilizado

El siguiente es el código que corre la Raspberry Pi para ejecutar el sistema de control desarrollado, se pueden ver todos los comentarios que explican cada una de las líneas.

Código 10: Código de Micropython final

```

1 import cv2
2 import dlib
3 import RPi.GPIO as GPIO
4 from time import sleep
5 import math as m
6
7 # Configurar los pines GPIO para el control del motor
8 ENA = 23
9 STEP = 18
10 DIR = 4
11
12 # Setup de GPIO
13 GPIO.setmode(GPIO.BCM)
14 GPIO.setup(DIR, GPIO.OUT)
15 GPIO.setup(STEP, GPIO.OUT)
16 GPIO.setup(ENA, GPIO.OUT)
17 GPIO.output(ENA, GPIO.LOW)
18
19 # Variables iniciales
20 xValue = 2
21 T0 = 0.06
22 delay = 0.002
23
24 # Inicializo el cascade classifier de imagenes para OpenCV
25 faceCascade = cv2.CascadeClassifier(
26     '/usr/local/lib/python3.7/dist-packages/cv2/data/haarcascade_frontalface_default.xml'
27 )
28
29 # Dimensiones de la imagen de salida
30 OUTPUT_SIZE_WIDTH = 1280
31 OUTPUT_SIZE_HEIGHT = 720
32
33
34 # Función que detecta rostros y mueve el Nema 17
35 def detectFace():
36     # Se inicializa la webcam de la posición 0, es decir, la que está por defecto
37     sampleVideo = cv2.VideoCapture(0)
38     # Se crea una ventana llamada "Reconocimiento" que mostrará la imagen con la cara remarcada
39     cv2.namedWindow("Reconocimiento", cv2.WINDOW_AUTOSIZE)
40     # Este es el faceTracker del paquete dlib
41     tracker = dlib.correlation_tracker()
42     # Se inicializa la variable a trackear en 0, posición inicial
43     trackingFace = 0
44     trackerColor = (0, 165, 255)
45     xValueString =

```

```
46 # Inicia el proceso de reconocimiento propiamente dicho
47 try:
48     """Bucle infinito para que la imagen sea un stream, la excepción se lanza en el momento en
49     que se aprieta Ctrl+C en terminal, y con eso salgo del bucle"""
50     while True:
51         # Se lee el último frame del video
52         rc, fullImage = sampleVideo.read()
53         #fullImage = cv2.rotate(fullImage, cv2.ROTATE_180)
54         # Para procesar de la mejor manera y optimizar recursos, se reescalda la imagen
55         originalImage = cv2.resize(fullImage, (480, 320))
56         # Si se presiona Q se cierran todas las ventanas que estén abiertas
57         # No logré que funcione sin esto, me parece que está demás
58         pressedKey = cv2.waitKey(2)
59         if pressedKey == ord('Q'):
60             cv2.destroyAllWindows()
61             exit(0)
62         # Se crea una copia del último frame para dibujarle el rectángulo
63         auxImage = originalImage.copy()
64         # Si no se está trackeando ninguna cara, entonces se trata de buscar alguna
65         if (not trackingFace):
66             # Se pasa la imagen a escala de grises para el análisis
67             grayImage = cv2.cvtColor(originalImage, cv2.COLOR_BGR2GRAY)
68             # Se usa haarcascade para reconocer los rostros
69             faces = faceCascade.detectMultiScale(grayImage, 1.3, 5)
70             # Se inicializan las variables necesarias en 0
71             maxArea = 0
72             x = 0
73             y = 0
74             w = 0
75             h = 0
76             # Se calculan las áreas de las caras
77             for (_x, _y, _w, _h) in faces:
78                 if _w * _h > maxArea:
79                     x = int(_x)
80                     y = int(_y)
81                     w = int(_w)
82                     h = int(_h)
83                     maxArea = w * h
84             # Se elige a la cara de mayor área, ver tema de minimo reconocible
85             if maxArea > 0:
86                 # Se inicializa el tracker
87                 tracker.start_track(
88                     originalImage,
89                     dlib.rectangle((x - 10), (y - 20), (x + w + 10),
90                                 (y + h + 20)))
91                 # Se pone el tracker en True, para saber que se está trackeando una cara
92                 trackingFace = 1
93                 # Se obtiene la posición de la cara
94                 '''xValue = ((x+(w/2))-640)*(1280/480)
95                 xValueString = 'X: ' + str(xValue)'''
96                 # Se comprueba si realmente se está trackeando una cara
```

```

97     if trackingFace:
98         # Se actualiza el tracker
99         trackingQuality = tracker.update(originalImage)
100        # Si la calidad del tracker es suficiente se obtienen las posiciones
101        if trackingQuality >= 4:
102            tracked_position = tracker.get_position()
103            t_x = int(tracked_position.left())
104            t_y = int(tracked_position.top())
105            t_w = int(tracked_position.width())
106            t_h = int(tracked_position.height())
107            # Obtenidas las posiciones, se puede imprimir la posición en el eje x
108            #xValue = ((t_x+t_w/2)*(1280/480)-640) # convertir a radianes!
109            xValue = ((t_x + t_w / 2) - 240)
110            xValueString = 'X: ' + str(xValue.__round__(2))
111            cv2.rectangle(auxImage, (t_x, t_y), (t_x + t_w, t_y + t_h),
112                          trackerColor, 2)
113        else:
114            # Si la calidad no es suficiente, se vuelve a leer la imagen e intentar trackear
115            trackingFace = 0
116            theta = 32.5 * m.pi / 180
117            error = m.atan(0.00265 * xValue)
118            if xValue != 0:
119                dela = theta * T0 / (8 * error)
120                delay = abs(dela)
121            else:
122                dela = theta * T0 / (8 * m.atan(0.00265 * 2))
123                delay = abs(dela)
124            if xValue < -20:
125                GPIO.output(DIR, GPIO.HIGH)
126                GPIO.output(STEP, GPIO.HIGH)
127                sleep(delay)
128                GPIO.output(STEP, GPIO.LOW)
129                #sleep(delay)
130            elif xValue > 20:
131                GPIO.output(DIR, GPIO.LOW)
132                GPIO.output(STEP, GPIO.HIGH)
133                sleep(delay)
134                GPIO.output(STEP, GPIO.LOW)
135                # sleep(delay)
136            # Se reescala la imagen nuevamente
137            largeResult = cv2.resize(auxImage,
138                                    (OUTPUT_SIZE_WIDTH, OUTPUT_SIZE_HEIGHT))
139            printImage = cv2.putText(largeResult, xValueString, (200, 200),
140                                    cv2.FONT_HERSHEY_SIMPLEX, 3,
141                                    (0, 255, 255))
142            # Se abre una ventana para mostrar la imagen con el rectángulo
143            cv2.imshow("Reconocimiento", printImage)
144
145        # El comando Ctrl+C termina el proceso
146        except KeyboardInterrupt as e:
147            cv2.destroyAllWindows()

```

```
148     exit(0)
149
150
151 # Programa principal
152 if __name__ == '__main__':
153     detectFace()
```

10.3. Implementación física

Para la implementación física se utilizaron varias piezas impresas en 3D, diseñadas por el grupo. Así como una cabeza de dinosaurio que puso a disposición el profesor, de modo de agregarle un carácter un poco más animado al seguidor de rostro. Las siguientes son las piezas impresas en 3D.

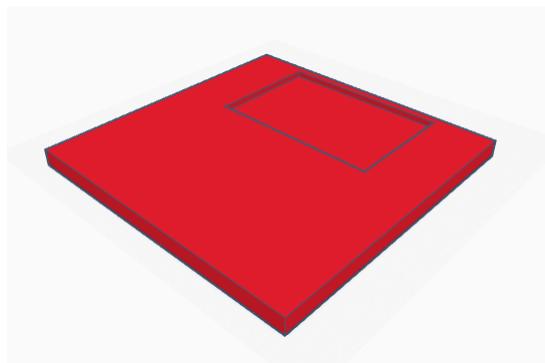


Figura 22: Base del prototipo.

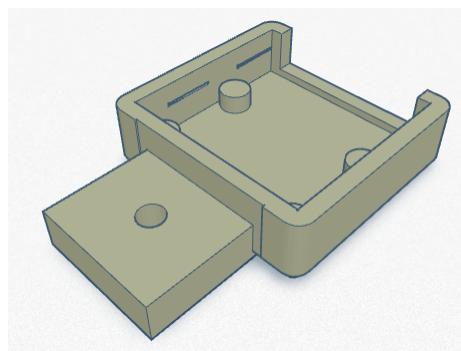


Figura 23: Soporte de la cámara.

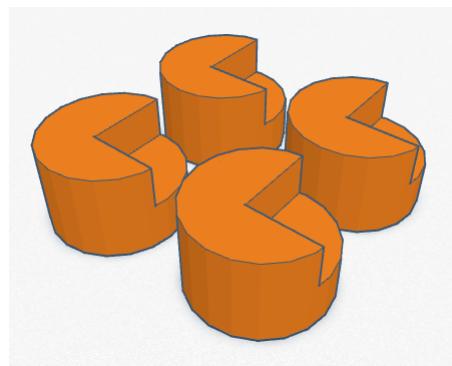


Figura 24: Soporte de la PCB.

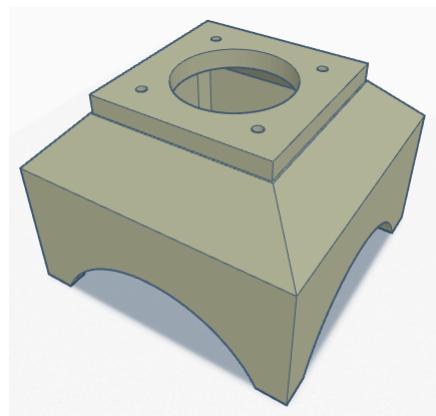


Figura 25: Soporte del motor paso a paso.

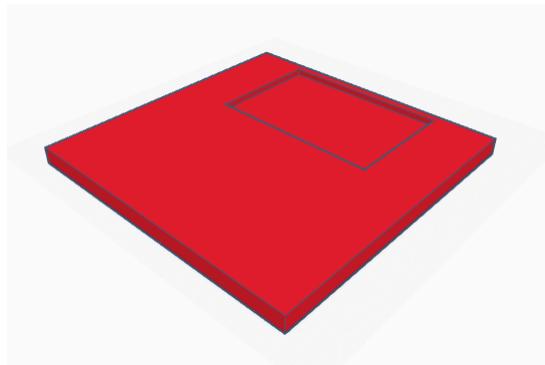


Figura 26: Base del prototipo.

La implementación real, quedó como a continuación:

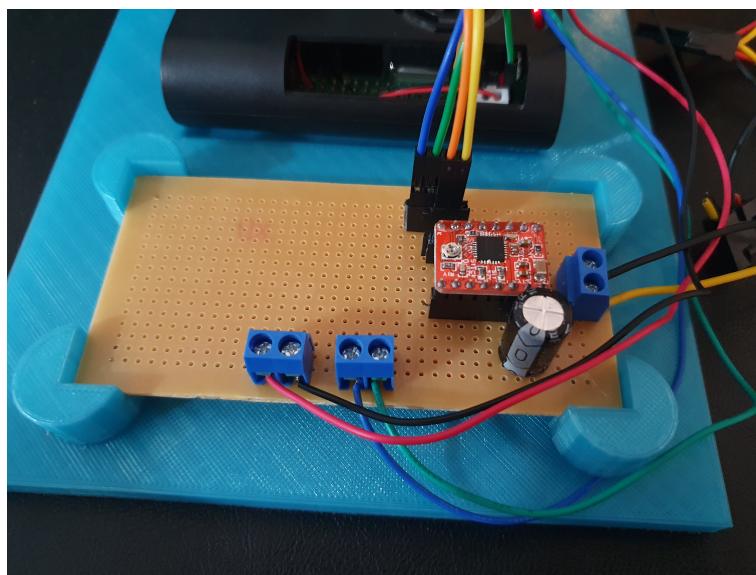


Figura 27: Plaqueta PCB.

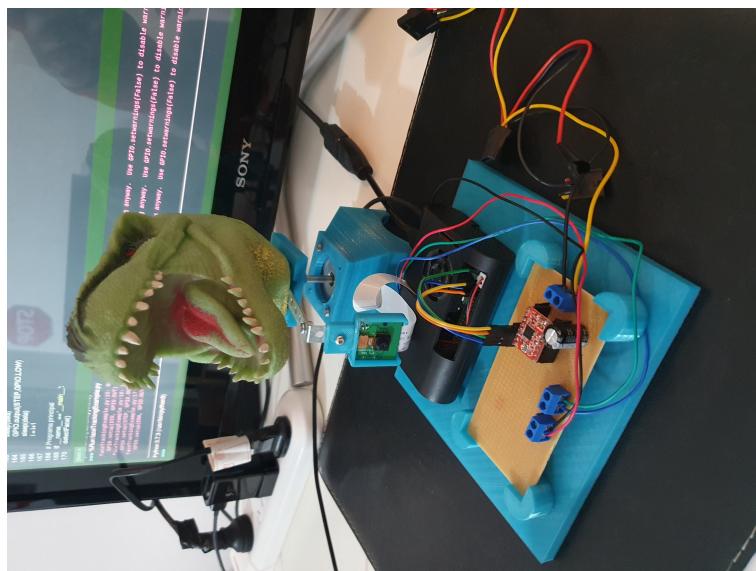


Figura 28: Prototipo ensamblado.

11. Lecciones aprendidas

En esta sección, se enumeran las diferentes lecciones aprendidas a partir de la investigación y desarrollo para la ejecución del proyecto planteado.

- Nombre del Proyecto: Seguidor de rostro.
- Fecha inicio del Proyecto y fecha de finalización del Proyecto: se inició el 8/03/2023 y finalizó el 21/06 del mismo año.
- Nombre del equipo: El Senado Galáctico (ESG).
- Carreras de ingeniería involucradas: Ing. Electrónica e Ing. Computación.
- Financiador del Proyecto: miembros del equipo en conjunto con el profesor Hugo Pailos.
- Finalidad del Proyecto: Fabricar prototipo para evaluación de la materia Sistemas de Control 2.
- Tema: control de posición de un motor Paso a Paso bipolar a través de dispositivo seguidor de rostro
- Descripción del Proyecto: con el objetivo de cumplir con la evaluación propuesta por el docente a cargo, el equipo se vió en la tarea de investigar la factibilidad de desarrollar diferentes propuestas de proyectos de sistemas de control realimentados, que estuvieran vinculados a los controladores PID (controlador proporcional, integral y derivativo) para que, una vez seleccionada la propuesta, y aprobada por el docente, se procedió a desarrollar el presente proyecto.
- Fase del Proyecto: La principal decisión que representó un cambio en la programación del proyecto se presentó entre el 03/04 y el 25/04. Fecha en la que el proyecto se encontraba en una fecha de evaluación constructiva tanto a nivel de software como hardware.

11.1. Contratiempos y acciones implementadas

- Software: mediante una evaluación exhaustiva del controlador que necesitaba el seguidor de rostro se llegó a la conclusión de que era innecesario diseñar un PID para su correcto funcionamiento, debido a las características del marco funcional que envuelve al motor seleccionado. Por lo cual, se tuvo que diseñar un propio controlador inspirado en la física de los motores Paso a Paso y la teoría de control realimentado. Para ello fue necesario, probar distintas técnicas de programación para poder integrar el programa que reconoce rostros con el mencionado programa que controla el giro del paso a paso. Las alternativas barajadas fueron: por hardware, por concurrencia de software y secuencial, siendo esta última la elegida. La única consideración particular sobre esta última es la existencia de un tiempo σ (debido al tiempo de ejecución del face tracking) que interfiere con la velocidad del stepper y ralentiza al sistema.
- Hardware: primero tuvo que ver con la selección del driver para controlar el motor. Al principio se empleó el L298 cumpliendo la función de puente H, con el que se elaboró el primer prototipo para mover el motor. Sin embargo, el docente a cargo sugirió emplear el driver Pololu A4988 por sus protecciones expuestas anteriormente, específicamente la de regular la intensidad de modo de evitar la circulación de altas corrientes por las bobinas de los motores tipo NEMA 17

para los cuales fue diseñado en un principio. Por último, se escogió y diseñó las características de forma que tendría la cabeza del seguidor de rostro.

- Gestión de grupo: se tuvieron unos pocos llamados de atención por parte del docente evaluador, debido a la desigualdad de conocimientos a la hora de exponer los contenidos del proyecto en las tutorías específicas en horario de clases, y a no presentar una sinergía grupal con los avances del proyecto. Motivo principal, por el cual el docente designó un trabajo práctico extra a la mitad de los integrantes con el objetivo de llenar esas lagunas de conocimiento generadas por la falta de integración en el trabajo en equipo. Por suerte, se cumplió la tarea con éxito y se trabajó en la comunicación grupal para establecer una ética de trabajo favorable que permitió desarrollar el resto del trabajo.
- Evaluación del funcionamiento: en el momento de hacer funcionar el proyecto, se le hicieron ciertas modificaciones al código (las cuales no fue necesario mencionar debido a que no afectan al funcionamiento del mismo) y al hardware para poder introducir el microstepping y obtener un giro más suavizado a la hora de buscar rostros. Sin embargo hacer esto conlleva un impacto más grande de la constante de tiempo σ (debido a que para hacer el microstepping se tuvieron que reducir los períodos de tiempo a la mitad para de esa forma conservar la velocidad), generando una velocidad un poco menor, disminuyendo un poco la notoriedad de la curva de velocidad y produciendo un sobrepasamiento en el sistema a pesar de haber sido modelado como de primer orden.
- Sobrepasamiento por microstepping: este sobrepasamiento se debe al hecho de que cuando el período era suficientemente alto, este se encontraba a más de una década del resto de polos del sistema y se lo podía considerar dominante. Sin embargo, cuando comenzó a disminuir su valor, este se acercó lo suficiente a otro polo del sistema, generando que el mismo sistema comenzara a comportarse como un sistema de segundo orden, el cual se evidencia como subamortiguado debido a la presencia de este sobrepasamiento.

12. Conclusiones del proyecto

A pesar de los contratiempos se puede concluir que el camino transitado en el desarrollo del seguidor de rostro fue todo menos turbulento. Los objetivos se modificaron y se cumplieron en tiempo y forma. Se solucionaron los problemas presentados a nivel de software y hardware, lo que permitió elaborar una solución satisfactoria. Entonces es posible argumentar que el proceso de aprendizaje y de desempeño en la elaboración del proyecto fue lo suficientemente bueno, y se vio representado en el correcto funcionamiento del dispositivo.

Por último, se agradece la disposición del docente para evaluar positivamente las cosas que funcionaban, corregir las cosas que no y siempre estar abierto a la comunicación tanto presencial como de forma remota. También agradecer a los miembros del equipo por tener la capacidad de adaptarse a los cambios para superar los obstáculos y por mantenerse unidos como grupo.

12.1. Recomendaciones finales

- Trabajar en la comunicación del equipo de trabajo desde el comienzo para asentar las bases con las que se va a trabajar.
- Rellenar las lagunas de conocimientos antes de avanzar con la realización del proyecto, sino se generan disparidades que después afectan al desarrollo del mismo.
- Plantear objetivos de acuerdo al tiempo disponible, para evitar incompletitudes o exigir mucho en poco tiempo al acercarse fechas definitorias.
- Hacer análisis en tiempo continuo de los sistemas a implementar, para luego llevarlos a su análogo de la vida real, que es el discreto, partiendo del muestreo implementado en los micro-controladores.
- Elegir desde un inicio un único lenguaje de programación, para poder luego juntar código que se haya hecho por separado, y permitir la división de tareas que refieran al desarrollo de un software específico.