Face Tracking Control (Proyecto Final Sistemas de Control II)

Introducción

En el siguiente reporte se muesstran los avances logrados en el proyecto de seguidor de rostro, propuesto por el Profesor Hugo Pailos para la materia Sistemas de Control II. Se añadirán descripciones, imágenes, desarrollos y códigos logrados en el transcurso de la última semana.

Se pretende, además, adjuntar links de importancia en el desarrollo y comprensión del sistema que se desea implementar y controlar.

Objetivos

Generales

Se pretende diseñar e implementar un controlador proporcional, integral y derivativo (PID) haciendo uso de un microcontrolador Raspberry Pi 4 para el control de posición de una cabeza animatrónica con sistema de reconocimiento facial.

Específicos

- Programar Sistema de reconocimiento facial en módulo de cámara Raspberry Pi a través de código Python y librerías OpenCV.
- Diseñar controlador PID para controlar la posición de un motor paso a paso por medio de dispositivo controlador L298N y microcontrolador Raspberry PI.
- Ensamblar el diseño de las dos partes para formar un solo sistema realimentado.
- Elaborar la estructura de la cabeza animatrónica.
- Implementar el seguidor de rostro basado en controlador PID y sistema de reconocimiento facial.

Materiales y Métodos

Hardware a utilizar

Desde el lado del hardware del sistema a implementar, se trabajará con los siguientes materiales:

- Raspberry Pi 4
- Raspberry Pi Camera 1.3
- Motor Nema 17
- Driver Puente H L298N
- Fuente de alimentación de PC

Para lograr obtener las piezas estructurales necesarias se hará uso de una impresora 3D.

Herramientas de software

Para el desarrollo de los diferentes programas para este hardware es necesaria la utilización de las siguientes herramientas de software:

- Visual Studio Code
- Thonny IDE

- Matlab
- Fritzing

Métodos

Para el desarrollo del presente proyecto se sigue una cierta línea de trabajo que consiste en la investigación de la disponibilidad de hardware para luego explotar las capacidades que estos brindan.

Partiendo de la lista de hardware disponible, y, entendiendo el concepto de la idea por implementar, se investiga primero qué elementos permiten el desarrollo deseado. Una vez definidos estos materiales a utilizar, se investiga su funcionamiento, y por consiguiente su modo de utilización. Una vez recabada toda la información necesaria se procede a desarrollar el software necesario para la implementación física del sistema, así como diferentes esquemas y programas de simulación para explicar mejor el cumplimiento de cada uno de los objetivos específicos planteados.

Desarrollo

Algoritmo de funcionamiento del sistema

El algoritmo de funcionamiento del sistema se basa en los siguientes pasos:

1. La cámara conectada a la Raspberry Pi detecta un rostro y obtiene la posición relativa del mismo con respecto al centro horizontal del campo de visión de la cámara.
2. La señal de la posición relativa se procesa de forma que se decida para qué dirección mover la cabeza.
3. Si la señal de posición se encuentra dentro de un margen de error predefinido, la cabeza no se moverá, caso contrario, se deberá seguir la posición.
4. El seguimiento de la posición se hace mediante la comprobación de la posición a adoptar mediante una estructura de decisión programada.
5. Dependiendo de la dirección que deba seguir el movimiento de la cabeza, se activarán las salidas digitales correspondientes para seleccionar el sentido de giro del motor.
6. El ángulo que debe rotar el eje del motor paso a paso será controlado mediante el uso de una señal senoidal muestreada 4 veces en un período.
7. Si al realizar el movimiento, aún existe un error superior al mínimo predefinido, entonces se vuelve a la instrucción 4.

Software desarrollado

Hasta el momento, el software desarrollado es el que se muestra a continuación.

Detección y seguimiento de rostros

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

- OpenCV
- dlib

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, 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 1. Ejemplo de reconocimiento facial logrado.

El código del script implementado es:

```
 1. import cv2
 2. import dlib
 3.
 4. # Inicializo el cascade classifier de imagenes para OpenCV
 5. faceCascade = cv2.CascadeClassifier('/Users/federicovillar/miniforge3/pkgs/libopencv-4.6.0-
py310h5ab14b7_6/share/opencv4/haarcascades/haarcascade_frontalface_default.xml')
 6.
 7. # Dimensiones de la imagen de salida
 8. OUTPUT_SIZE_WIDTH = 1280
 9. OUTPUT_SIZE_HEIGHT = 720
10.
11. def detectFace():
12.     # Se inicializa la webcam de la posición 0, es decir, la que está por defecto
13.     sampleVideo = cv2.VideoCapture(0)
14.     # Se crea una ventana llamada "Reconocimiento" que mostrará la imagen con la cara remarcada
15.     cv2.namedWindow("Reconocimiento", cv2.WINDOW_AUTOSIZE)
16.     # Este es el faceTracker del paquete dlib
17.     tracker = dlib.correlation_tracker()
18.     # Se inicializa la variable a trackear en 0, posición inicial
19.     trackingFace = 0
20.     trackerColor = (0,165,255)
21.     xValueString = ''
22.     # Inicia el proceso de reconocimiento propiamente dicho
23.     try:
24.         '''Bucle infinito para que la imagen sea un stream, la excepción se lanza en el momento en
25.         que se aprieta Ctrl+C en terminal, y con eso salgo del bucle'''
26.         while True:
27.             # Se lee el último frame del video
28.             rc,fullImage = sampleVideo.read()
29.             # Para procesar de la mejor manera y optimizar recursos, se reescala la imagen
```

```
            originalImage = cv2.resize(fullImage,(480,320))
            # Si se presiona Q se cierran todas las ventanas que estén abiertas
            # No logré que funcione sin esto, me parece que está demás
            pressedKey = cv2.waitKey(2)
            if pressedKey == ord('Q'):
                cv2.destroyAllWindows()
                exit(0)
            # Se crea una copia del último frame para dibujarle el rectángulo
            auxImage = originalImage.copy()
            # Si no se está trackeando ninguna cara, entonces se trata de buscar alguna
            if (not trackingFace):
                # Se pasa la imagen a escala de grises para el análisis
                grayImage = cv2.cvtColor(originalImage, cv2.COLOR_BGR2GRAY)
                # Se usa haarcascade para reconocer los rostros
                faces = faceCascade.detectMultiScale(grayImage, 1.3, 5)
                # Se inicializan las variables necesarias en 0
                maxArea = 0
                x = 0
                y = 0
                w = 0
                h = 0
                # Se calculan las áreas de las caras
                for (_x,_y,_w,_h) in faces:
                    if  _w*_h > maxArea:
                        x = int(_x)
                        y = int(_y)
                        w = int(_w)
                        h = int(_h)
                        maxArea = w*h
                # Se elige a la cara de mayor área, ver tema de minimo reconocible
                if maxArea > 0 :
                    # Se inicializa el tracker
                    tracker.start_track(originalImage,dlib.rectangle((x-10),(y-20),(x+w+10),(y+h+20)))
                    # Se pone el tracker en True, para saber que se está trackeando una cara
                    trackingFace = 1
            # Se obtiene la posición de la cara
            '''xValue = ((x+(w/2))-640)*(1280/480)
            xValueString = 'X: ' + str(xValue)'''
            # Se comprueba si realmente se está trackeando una cara
            if trackingFace:
                # Se actualiza el tracker
                trackingQuality = tracker.update(originalImage)
                # Si la calidad del tracker es suficiente se obtienen las posiciones
                if trackingQuality >= 8.75:
                    tracked_position =  tracker.get_position()
                    t_x = int(tracked_position.left())
                    t_y = int(tracked_position.top())
                    t_w = int(tracked_position.width())
                    t_h = int(tracked_position.height())
                    # Obtenidas las posiciones, se puede imprimir la posición en el eje x
                    xValue = ((t_x+t_w/2)*(1280/420)-725) # convertir a radianes!
                    xValueString = 'X: ' + str(xValue.__round__(2))
                    cv2.rectangle(auxImage,(t_x, t_y),(t_x + t_w , t_y + t_h),trackerColor,2)
                else:
                    # Si la calidad no es suficiente, se vuelve a leer la imagen e intentar trackear
                    trackingFace = 0
            # Se reescala la imagen nuevamente
            largeResult = cv2.resize(auxImage,(OUTPUT_SIZE_WIDTH,OUTPUT_SIZE_HEIGHT))
            printImage = cv2.putText(largeResult, xValueString, (200, 200), cv2.FONT_HERSHEY_SIMPLEX,
3, (0, 255, 255))
            # Se abre una ventana para mostrar la imagen con el rectángulo
            cv2.imshow("Reconocimiento", printImage)

    # El comando Ctrl+C termina el proceso
    except KeyboardInterrupt as e:
```

```
94.         cv2.destroyAllWindows()
95.         exit(0)
96.
97. # Programa principal
98. if __name__ == '__main__':
99.     detectFace()
100.
```

Test de Servomotor para Raspberry Pi

Para este código, se intentó emular una estructura de código similar a la de Arduino, en donde se cuenta con dos funciones principales: setup y loop. Este pequeño script a modo de ilustración permite mover un servo de 0º a 180º y de 180º a 0º por un tiempo indefinido.

A continuación, se adjunta lo logrado.

```
1. import RPi.GPIO as GPIO
2. import time
3.
4. # Se definen los parametros del pwm a utilizar
5. SERVO_MIN_PULSE = 500
6. SERVO_MAX_PULSE = 2500
7. # Se define el pin al que ira conectado el servo
8. servo = 23
9.
10. def map(value, inMin, inMax, outMin, outMax):
11.     return (outMax - outMin) * (value - inMin) / (inMax - inMin) + outMin
12.
13. def setup():
14.     global p
15.     GPIO.setmode(GPIO.BCM)
16.     GPIO.setup(servo, GPIO.OUT)
17.     GPIO.output(servo, GPIO.LOW)
18.     p = GPIO.PWM(servo, 50)
19.     p.start(0)
20.
21. def setAngle(angle):
22.     angle = max(0, min(180, angle))
23.     pulse_width = map(angle, 0, 180, SERVO_MIN_PULSE, SERVO_MAX_PULSE)
24.     pwm = map(pulse_width, 0, 20000, 0, 100)
25.     p.ChangeDutyCycle(pwm)
26.
27. def loop():
28.     while True:
29.         for i in range(0, 91, 5):
30.             setAngle(i)
31.             time.sleep(0.002)
32.         time.sleep(1)
33.         for i in range(90, -1, -5):
34.             setAngle(i)
35.             time.sleep(0.001)
36.         time.sleep(1)
37.
38. def destroy():
39.     p.stop()
40.     GPIO.cleanup()
41.
42. if __name__ == '__main__':
43.     setup()
44.     try:
45.         loop()
46.     except KeyboardInterrupt:
47.         destroy()
```

Test de motor paso a paso

A diferencia del anterior código, para probar el funcionamiento del motor paso a paso haciendo uso de la Raspberry Pi 4, no se utilizó una estructura de loop y setup, sino que se optó por una programación en forma secuencial. Se muestra entonces el siguiente código:

```
#!/usr/bin/python3
import RPi.GPIO as GPIO
import time

# Se definen los pines a utilizar
out1 = 17
out2 = 18
out3 = 27
out4 = 22

# Tiempo de espera entre los pasos, hay que tener cuidado con este valor porque si es muy chico
# se puede estar sobrepasando un límite mecánico del motor
stepSleep = 0.01
# Número de pasos a realizar, para una vuelta entera se requieren 200 pasos
stepCount = 2000

# Se setean los diferentes pines (analogo a void setup de arduino)
GPIO.setmode(GPIO.BCM)
GPIO.setup(out1, GPIO.OUT)
GPIO.setup(out2, GPIO.OUT)
GPIO.setup(out3, GPIO.OUT)
GPIO.setup(out4, GPIO.OUT)

# Se inicializan todas las salidas en 0 (low)
GPIO.output(out1, GPIO.LOW)
GPIO.output(out2, GPIO.LOW)
GPIO.output(out3, GPIO.LOW)
GPIO.output(out4, GPIO.LOW)

# Se limpian las salidas
def cleanup():
    GPIO.output(out1, GPIO.LOW)
    GPIO.output(out2, GPIO.LOW)
    GPIO.output(out3, GPIO.LOW)
    GPIO.output(out4, GPIO.LOW)
    GPIO.cleanup()

# Movimiento del stepper
try:
    i = 0
    for i in range(stepCount):
        if i % 4 == 0:
            GPIO.output(out4, GPIO.HIGH)
            GPIO.output(out3, GPIO.LOW)
            GPIO.output(out2, GPIO.LOW)
            GPIO.output(out1, GPIO.LOW)
        elif i % 4 == 1:
            GPIO.output(out4, GPIO.LOW)
            GPIO.output(out3, GPIO.LOW)
            GPIO.output(out2, GPIO.HIGH)
            GPIO.output(out1, GPIO.LOW)
        elif i % 4 == 2:
            GPIO.output(out4, GPIO.LOW)
            GPIO.output(out3, GPIO.HIGH)
            GPIO.output(out2, GPIO.LOW)
            GPIO.output(out1, GPIO.LOW)
        elif i % 4 == 3:
            GPIO.output(out4, GPIO.LOW)
```

```
            GPIO.output(out3, GPIO.LOW)
            GPIO.output(out2, GPIO.LOW)
            GPIO.output(out1, GPIO.HIGH)

        time.sleep(stepSleep)

except KeyboardInterrupt:
    cleanup()
    exit(1)

cleanup()
exit(0)
```

Control del motor paso a paso

Determinación de la relación entre la frecuencia de alimentación y la velocidad de giro del motor

Se sabe que el motor NEMA 17 a disposición es bipolar y presenta unos 200 pasos en una vuelta, esto implica 200 electroimanes que se activan en una secuencia de cuatro pasos, dos por cada bobinado, y cada bobinado con polarización directa e inversa.

De esta forma, y utilizando las señales de trabajo de un stepper bipolar analizando su full step sequence, es posible notar que la alimentación de las bobinas A y B se hace mediante la utilización de un seno muestreado con un retentor de orden cero, y desfasado 90° entre una bobina y la otra. La cantidad de muestreos depende de la cantidad de pasos (steps) que hay en la secuencia de cuatro pasos mencionada anteriormente (justamente son cuatro), y depende de si se aplica microstepping o no (generación de pasos intermedios entre bobinas A y B mediante la activación parcial de ambas a la vez para generar una composición de fuerzas que produzca una bobina ficticia AB entre las otras).

En este proyecto no se utilizará microstepping, por lo que el muestreo será de cuatro veces dentro del seno, es decir a una frecuencia cuatro veces mayor.

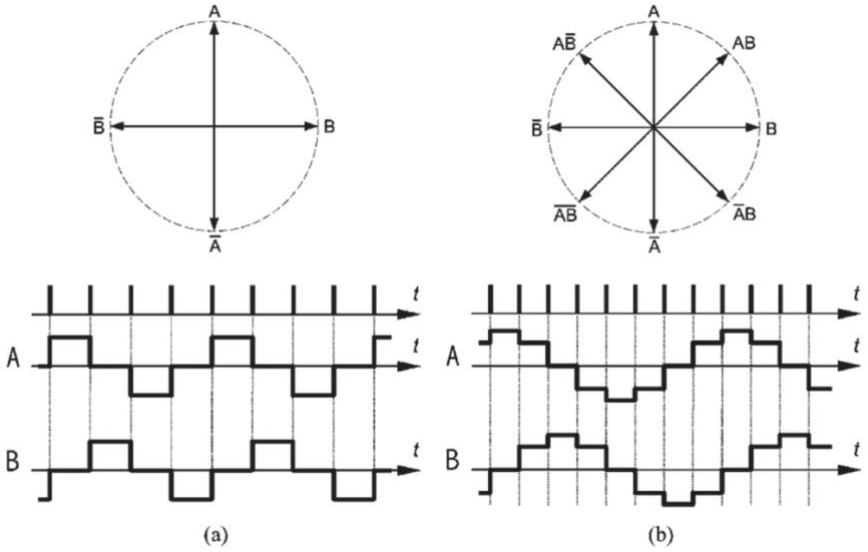

Figura 2. a) Secuencia full step, b) Microstepping

Cada vez que alguna de las señales alcanza un 1 o -1 (es decir cuatro veces por ciclo) se avanza un paso en el motor. Debido a que este tiene 200 pasos, para que el motor haga una vuelta completa se necesitarán 50 ciclos en la alimentación, por lo que:

$$fs = \frac{50 \times \omega_m}{2\pi}$$

Tomando una velocidad de giro del motor de una vuelta cada 3 segundos, podemos despejar la frecuencia de la señal de alimentación de las bobinas:

$$fs = \frac{50 \times 2\pi}{3s \times 2\pi} = \frac{50}{3s} \Rightarrow fs = \frac{16{,}67}{s} \Rightarrow Ts = 0{,}06s$$

Para invertir el giro del motor se debe desfasar la señal de alimentación de alguna de las bobinas unos 180°, o lo que resulta igual, alimentar al motor con las mismas señales, pero con ellas siendo rebobinadas, es decir, leyéndolas hacia atrás (teniendo todo el sentido del mundo: si alimento con una señal y gira en una dirección, si lo alimento con la señal espejada/rotada, el motor girará en sentido inverso). Este último es el método que implementamos para invertir el giro.

Determinación del modelo y diagrama de cajas del sistema

El objetivo del motor es moverse para ubicar un rostro con la cámara. Por lo tanto, la información principal es la distancia angular entre el centro de la cámara y el rostro. Esa es la entrada. Véase que para plantear el modelo a lazo abierto primero se debe considerar a la medida absoluta, es decir, no se debe variar la velocidad del motor a medida que se esté acercando a la cara, el motor no debe frenarse al centrarla tampoco, simplemente debe moverse en función de la ubicación que esta tenga en el plano. En otras palabras, la velocidad angular permanece constante y solo es función de la ubicación angular del rostro:

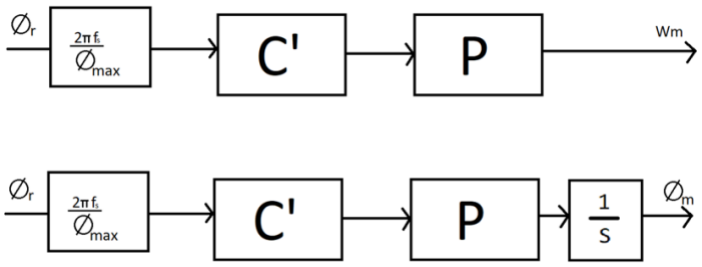

Figura 3. Diagrama de bloques genérico.

El bloque integrador aparece debido a que el ángulo de rotación del motor resulta de integrar temporalmente la velocidad angular.

- **P**: es la planta, es decir, el motor paso a paso.
- **C'**: es el controlador definido anteriormente, el que determina la velocidad del motor según la frecuencia de entrada.
- **Φmax**: es el ángulo de visión máxima de la cámara. Está dispuesto de esta forma para que la velocidad máxima del motor se alcance cuando la cara esté lo más lejos posible del centro de la imagen.

Agrupando los bloques resulta:

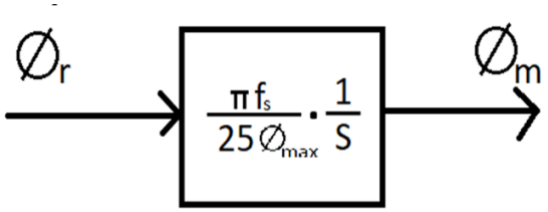

Figura 4. Diagrama de bloques resumido.

El 25 surge de la relación de 50 entre frecuencia de alimentación y velocidad angular del motor.

Entonces desde aquí se puede definir el sistema realimentado para aplicarle la acción de control PID y manejar las curvas mediante las cuales llega enfocar directamente al rostro, centrado en la pantalla:

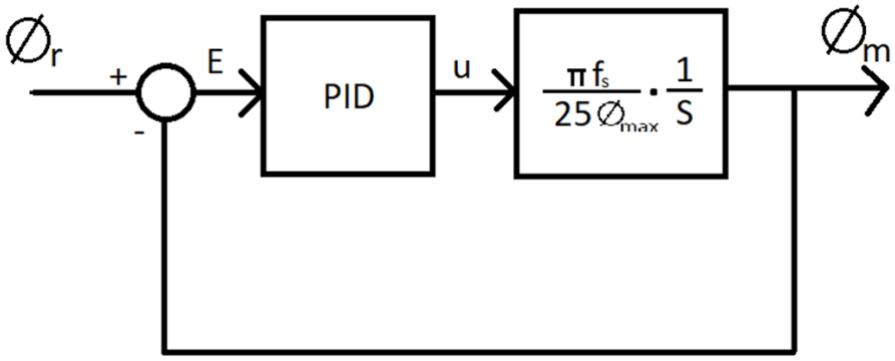

Figura 5. Diagrama de bloques objetivo.

Vale la pena destacar que el algoritmo de reconocimiento de rostros realmente mide E (el error), y que la realimentación, por lo menos unitaria, es necesaria para que el sistema funcione y permita el centrado del rostro en la pantalla.

Organización de las tareas

Para la organización de las tareas, se tomaron las siguientes como referencia:

- Búsqueda de ideas
- Búsqueda de información
- Desarrollo del seguidor de rostros
- Desarrollo del controlador para el motor paso a paso
- Desarrollo del control PID del sistema
- Implementación en físico
- Diseño del animatrónico
- Realización de la estructura del animatrónico
- Implementación del animatrónico en conjunto con los circuitos
- Correcciones

Cabe destacar que cada una de las actividades mencionadas anteriormente está sujeta a cambios, es decir, que se podrán ir añadiendo más a medida que se lo considere necesario. Es por ello que se crearán reportes individuales para cada una de las clases con el profesor a cargo.

A continuación, se adjunta el diagrama de Gantt que representa el desarrollo de las actividades de este proyecto.

	8/3	9/3	10/3	11/3	12/3	13/3	14/3	15/3	16/3	17/3	18/3	19/3	20/3	21/3	22/3	23/3	24/3	25/3	26/3	27/3	28/3	29/3	30/3	1/4
Búsqueda de ideas																								
Búsqueda de información																								
Desarrollo del seguidor de rostros																								
Desarrollo del controlador para el motor paso a paso																								
Desarrollo del control PID del sistema																								
Implementación en físico																								
Diseño del animatrónico																								
Realización de la estructura del animatrónico																								
Implementación del animatrónico en conjunto con los circuitos																								
Correcciones																								

Circuitos implementados

Prueba del motor paso a paso

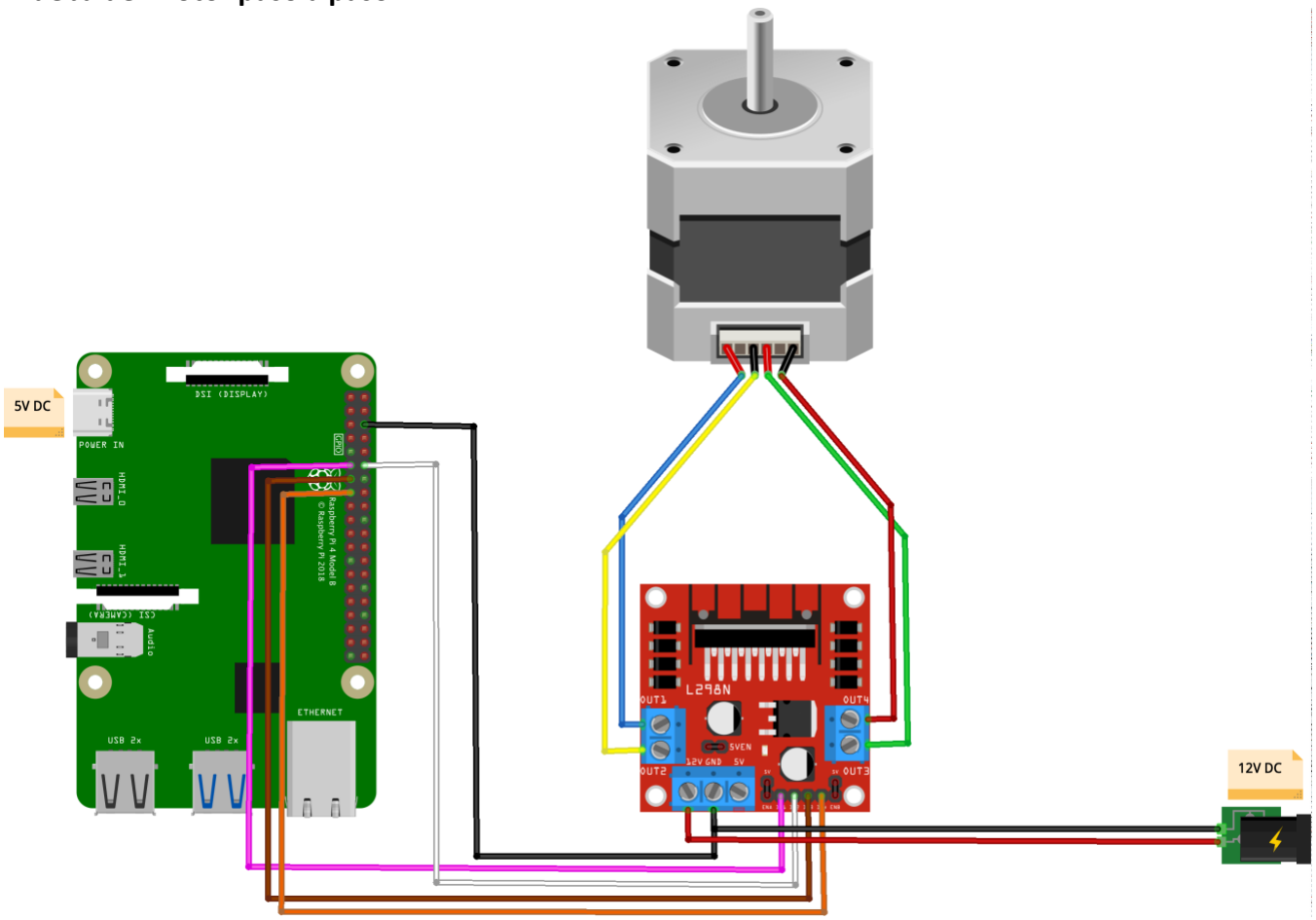

Ilustración 1. Circuito implementado para la prueba del motor paso a paso con L298N.

Prueba del servomotor

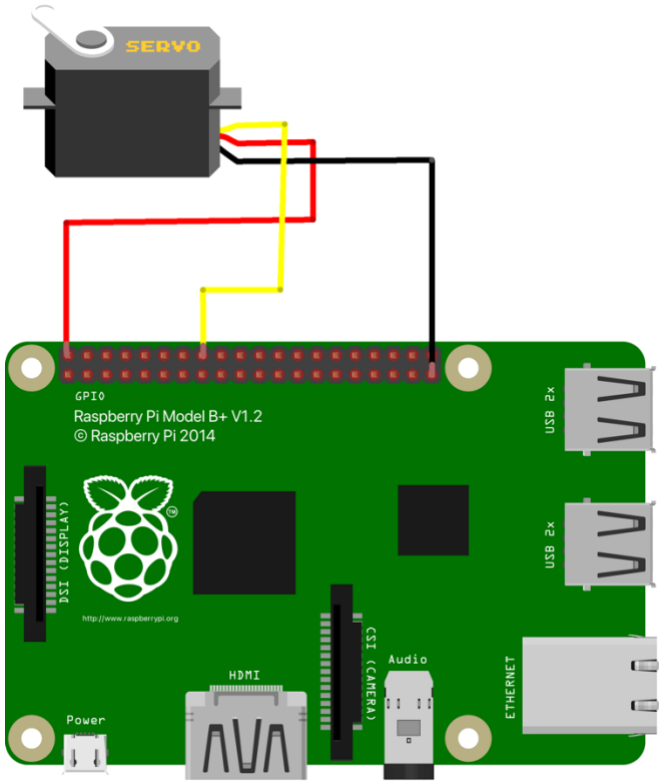

Ilustración 2. Circuito implementado para la prueba del servomotor.

Links de utilidad

Los siguientes son links a los que se recomienda ingresar para complementar la información de este pequeño reporte:

- Repositorio GitHub del proyecto
- Características Nema 17
- Datasheet L298N
- Página principal Raspberry Pi 4B
- Documentación Cámara Raspberry Pi
- Generación de PWM con Raspberry Pi