Universidad Nacional de Asunción

Facultad de Ingeniería

Robótica II

Robots Móviles

Profesor:
Dr. Fernando Brunetti
Ing. Juanjo Rojas

Alumno:
Carlos Gaona
Hans Mersch

Índice

1. Introducción

En la catedra de Robotica II se nos asigno el disenho e implementacion de un robot movil que renga la capacidad de desplazarse en linea recta empleando PID. Este robot tambien debe ser capaz de maniobrar para evitar obstaculos, los cuales seran detectados mediante sensores ubicados en la parte frontal del robot. Mediante la informacion captada por los sensores y un algoritmo de campo potencial, el robot evitara los obstaculos del medio.

2. Objetivos

2.1. Objetivo General

Desarrolle un robot móvil que navegue aleatoriamente siguiendo un algoritmo de campo de potencial.

2.2. Objetivos Específicos

- Diseñar e implementar un controlador y sistema de actuación (locomoción) basado en ruedas para robots móviles.
- Utilizar la teoría de control para solucionar problemas de actuación
- Implementar y verificar los resultados y limitaciones de un algoritmo de navegación concreto para robots móviles.

3. Marco Teorico

3.1. Algoritmos de Control

3.1.1. PID

El control proporcional-integral-derivativo (PID) es sin duda la estrategia de control más utilizada en la actualidad. Se estima que más del 90 % de los lazos de control emplean control PID, muy a menudo con la ganancia derivada establecida en cero (control PI). Durante el último medio siglo, una gran cantidad de esfuerzos académicos e industriales se han centrado en mejorar el control de PID, principalmente en las áreas de reglas de sintonización, esquemas de identificación y técnicas de adaptación. Es apropiado en este momento considerar el estado del arte en el control PID así como los nuevos desarrollos en este enfoque de control[2].

La idea básica detrás de un controlador PID es leer un sensor, luego calcular la salida deseada del actuador, calculando las respuestas proporcional, integral y derivada y sumando esos tres componentes para calcular la salida[6]. A continuacion, se realizara una breve descripcion de los 3 controles a implementar.

3.1.1.1 Control Proporcional

El componente proporcional depende únicamente de la diferencia entre el punto de referencia y la variable del proceso. Esta diferencia se conoce como el término error. La ganancia proporcional (K_c) determina la relación entre la respuesta de salida y la señal de error. Por ejemplo, si el término de error tiene una magnitud de 10, una ganancia proporcional de 5 produciría una respuesta proporcional de 50.

En general, aumentar la ganancia proporcional aumentará la velocidad de respuesta del sistema de control. Sin embargo, si la ganancia proporcional es demasiado grande, la variable del proceso comenzará a oscilar. Si K_c se aumenta aún más, las oscilaciones serán mayores y el sistema se volverá inestable e incluso puede oscilar fuera de control.

3.1.1.2 Control Integral

El componente integral suma el término de error en el transcurso del tiempo. El resultado es que incluso un término de error pequeño hará que el componente integral aumente lentamente. La respuesta integral aumentará continuamente con el tiempo a menos que el error sea cero, por lo que el efecto es llevar el error de estado estable a cero. El error de estado estable es la diferencia final entre la variable del proceso y el punto de referencia. Se produce un fenómeno llamado windup integral cuando la acción integral satura un controlador sin que el controlador conduzca la señal de error hacia cero.

3.1.1.3 Control Derivado

El componente derivado hace que la salida disminuya si la variable del proceso aumenta rápidamente. La respuesta derivada es proporcional a la tasa de cambio de la variable del proceso. Aumentar el parámetro de tiempo derivado (T_d) hará que el sistema de control reaccione con más fuerza a los cambios en el término de error y aumentará la velocidad de respuesta del sistema de control en general. Los sistemas de control más prácticos utilizan un tiempo derivado muy pequeño (T_d), porque la respuesta derivada es muy sensible al ruido en la señal de la variable del proceso. Si la señal de retroalimentación del sensor es ruidosa o si la tasa del ciclo de control es demasiado lenta, la respuesta derivada puede hacer que el sistema de control sea inestable

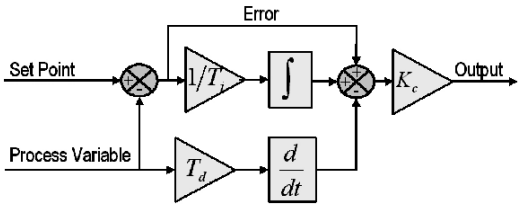


Figura 1: Diagrama de bloques de un algoritmo básico de control PID.

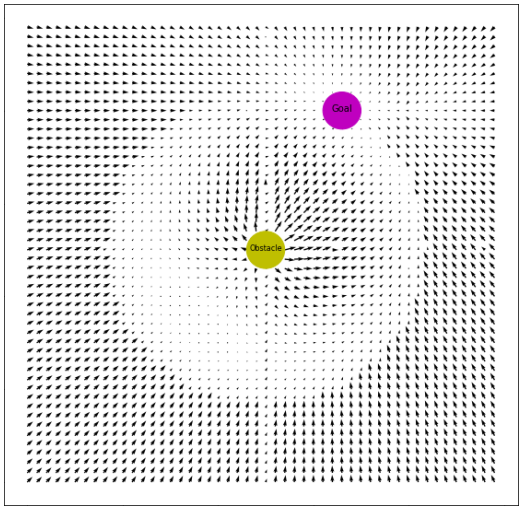
3.1.2. Campo Potencial

El campo potencial se puede utilizar para obtener una representación global del espacio para que la planificación general se pueda realizar a nivel global. Un campo de potencial continuo da una buena indicación de las distancias y las formas de los obstáculos para que los cambios necesarios en la posición y orientación del robot se puedan realizar de manera suave y continua. Al detectar colisiones, se evita la complejidad combinatoria de la detección de intersecciones realizada con representaciones geométricas.

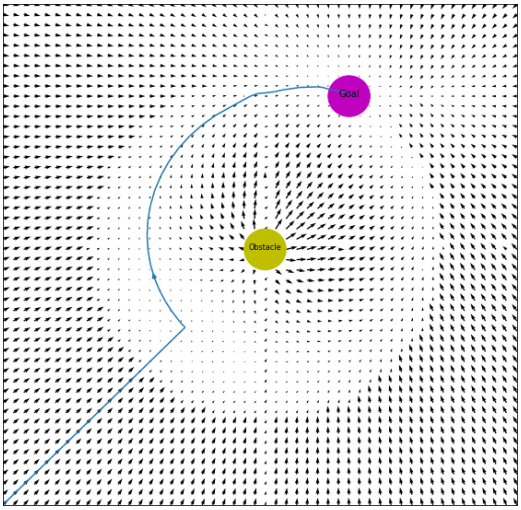
Esto se logra al eliminar la necesidad de realizar explícitamente la detección de intersecciones mediante el uso de un campo potencial que proporciona información sobre la distancia del objeto. Al usar una definición apropiada del potencial en un punto, se elimina la influencia de los obstáculos que no están en las proximidades del punto[1].

En robótica, podemos simular el mismo efecto creando un campo potencial artificial que atraerá al robot hacia la meta. Al diseñar un campo potencial adecuado, podemos hacer que el robot muestre comportamientos simples. Por ejemplo, supongamos que no hay ningún obstáculo en el entorno y que el robot debe buscar este objetivo. En la planificación convencional, se debe calcular la posición relativa del robot con respecto a la meta y luego aplicar las fuerzas adecuadas que conducirán al robot a la meta.

En el enfoque de campo potencial, creamos un campo atractivo que va dentro del objetivo. El campo potencial se define en todo el espacio libre y, en cada paso de tiempo, calculamos el campo potencial en la posición del robot y luego calculamos la fuerza inducida por este campo. Entonces, el robot debería moverse de acuerdo con esta fuerza[3].



(a) Potencial combinado cuando el obstáculo y el objetivo son diferentes



(b) Trayectoria definida por el robot para llegar al objetivo

Figura 2: Descripción de la implementación del campo potencial

3.2. Hardware a Utilizar

3.2.1. Raspberry PI

El Raspberry PI es un microprocesador muy versatil que posee una comunidad activa, el sistema operativo es Raspberry PI OS, el cual esta basado en una distribucion de Linux. Nuestro grupo contaba con un Raspberry PI 3 modelo B, el cual tiene las siguientes especificaciones:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 LAN inalambrico y Bluetooth Low Energy (BLE) en la placa
- 100 Ethernet de base
- 40-pin extendidos de GPIO
- 4 puertos USB
- 4 salida unipolar de stereo y puerto de video compuesto
- HDMI
- Puerto para camara CSI
- Puerto para display DSI
- Puerto Micro SD para cargarle el sistema operativo y los archivos
- Fuente de alimentación por Micro USB hasta 2.5A

Las principales ventajas del Raspberry sobre el Arduino es que el Raspberry ya trae incorporado un receptor WiFi, y el uso de una distribución de Linux como sistema operativo supondría ciertas facilidades mas adelante cuando sea necesario conectarse al ROS en Ubuntu.

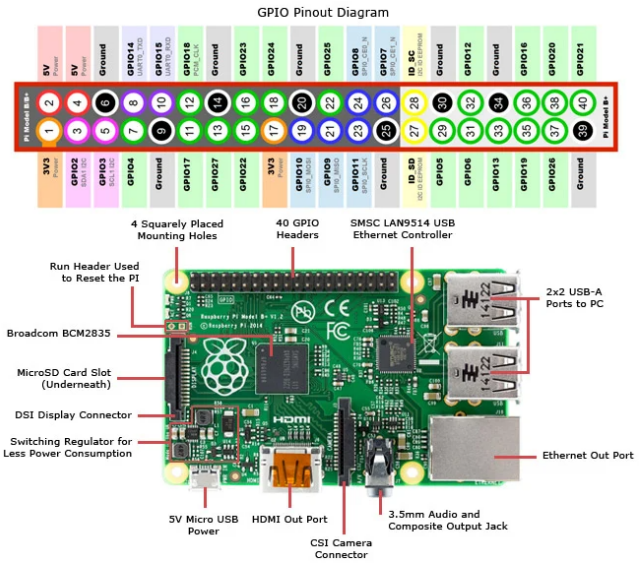


Figura 3: Representación física del Raspberry PI 3 modelo B y sus terminales

3.2.2. L298N

El módulo controlador de motores L298N H-bridge nos permite controlar la velocidad y la dirección de dos motores de corriente continua o un motor paso a paso de una forma muy sencilla, gracias a los 2 los dos H-bridge que monta. El puente-H o H-bridge es un componente formado por 4 transistores que nos permite invertir el sentido de la corriente, y de esta forma podemos invertir el sentido de giro del motor.

El rango de tensiones en el que trabaja este módulo va desde 3V hasta 35V, y una intensidad de hasta 2A. A la hora de alimentarlo hay que tener en cuenta que la electrónica del módulo consume unos 3V, así que los motores reciben 3V menos que la tensión con la que alimentemos el módulo.

Además el L298N incluye un regulador de tensión que nos permite obtener del módulo una tensión de 5V, perfecta para alimentar nuestro Arduino. Eso sí, este regulador sólo funciona si alimentamos el módulo con una tensión máxima de 12V. Es un módulo que se utiliza mucho en proyectos de robótica, por su facilidad de uso y su reducido precio[4].

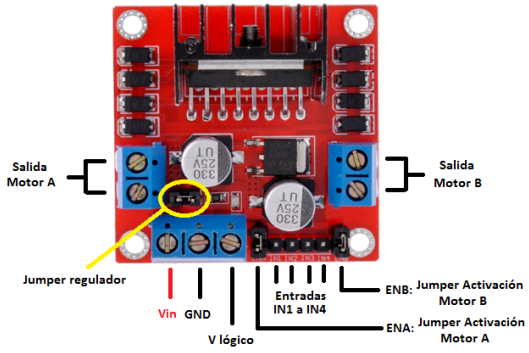


Figura 4: Descripción de los pines del L298N

3.2.3. Sensores Ultrasónicos

El sensor HC-SR04 es un sensor de distancia de bajo costo que utiliza ultrasonido para determinar la distancia de un objeto en un rango de 2 a 450 cm. Destaca por su pequeño tamaño, bajo consumo energético, buena precisión y excelente precio. El sensor HC-SR04 es el más utilizado dentro de los sensores de tipo ultrasonido, principalmente por la cantidad de información y proyectos disponibles en la web. De igual forma es el más empleado en proyectos de robótica como robots laberinto o sumo, y en proyectos de automatización como sistemas de medición de nivel o distancia.

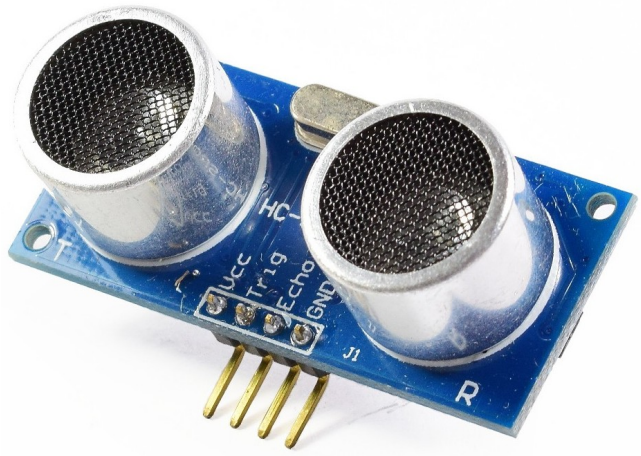


Figura 5: HC-SR04

El sensor HC-SR04 posee dos transductores: un emisor y un receptor piezoeléctricos, además de la electrónica necesaria para su operación. El funcionamiento del sensor es el siguiente: el emisor piezoeléctrico emite 8 pulsos de ultrasonido(40KHz) luego de recibir la orden en el pin TRIG, las ondas de sonido viajan en el aire y rebotan

al encontrar un objeto, el sonido de rebote es detectado por el receptor piezoeléctrico, luego el pin ECHO cambia a Alto (5V) por un tiempo igual al que demoró la onda desde que fue emitida hasta que fue detectada, el tiempo del pulso ECO es medido por el microcontrolador y así se puede calcular la distancia al objeto. El funcionamiento del sensor no se ve afectado por la luz solar o material de color negro (aunque los materiales blandos acústicamente como tela o lana pueden llegar a ser difíciles de detectar)[5].

La distancia se puede calcular utilizando la siguiente formula:

$$Distancia[m] = \frac{\text{Tiempo del pulso ECHO}[s] * \text{Velocidad del sonido}[\frac{m}{s}]}{2}$$

4. Diagramas de Bloques

4.1. Funcionamiento General

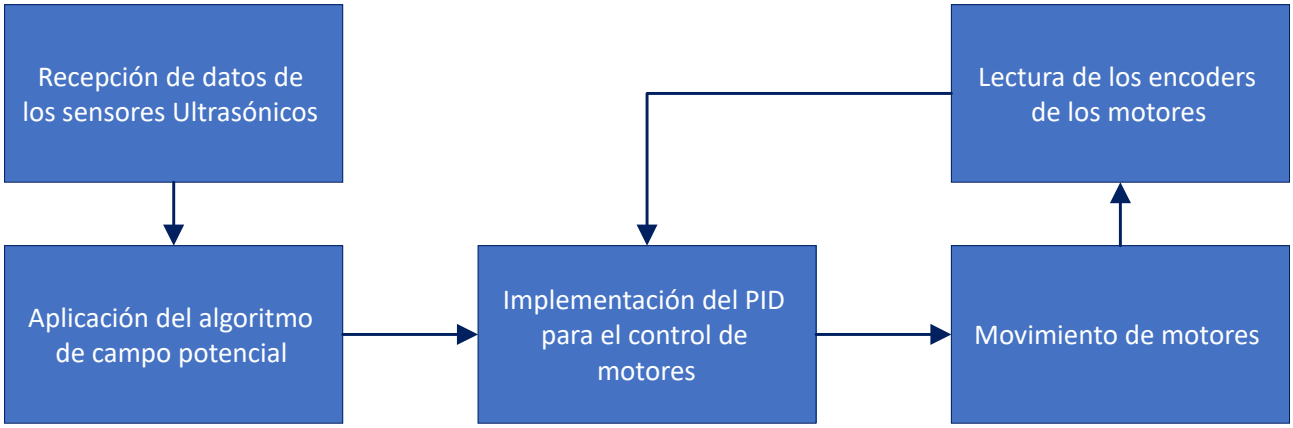


Figura 6: Diagrama general del funcionamiento autónomo

4.2. Campo Potencial

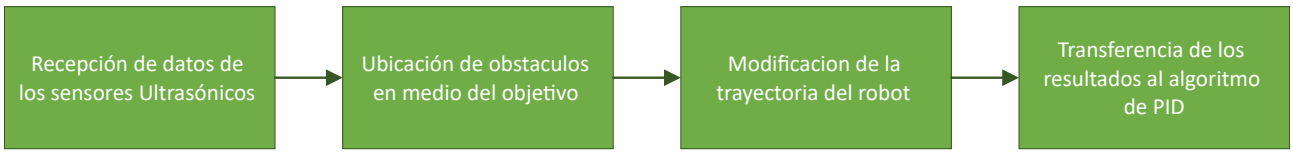


Figura 7: Diagrama aplicación del campo potencial

4.3. Aplicación del PID

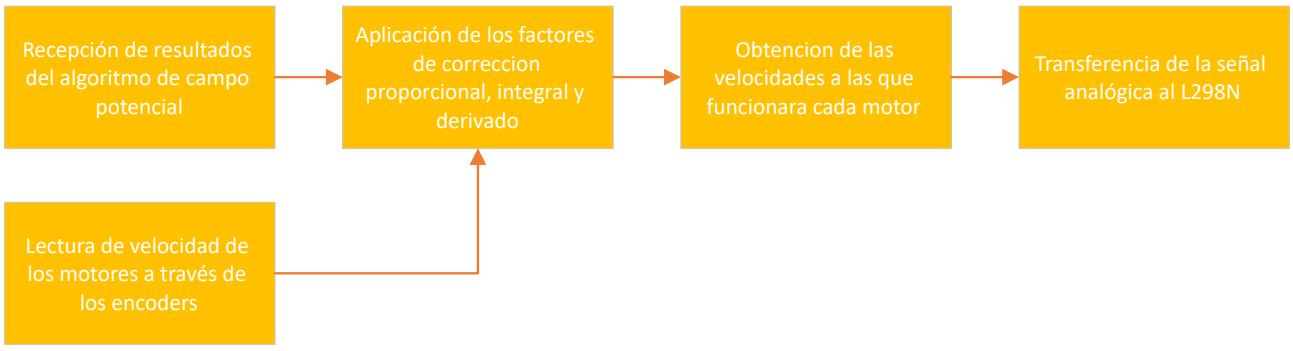


Figura 8: Diagrama de aplicación del PID

5. Implementación

5.1. Conexionados

Para nuestra fortuna, el Raspberry Pi 3 Modelo B cuenta con las entradas y salidas necesarias para esta implementación. En la **figura 9** se muestra los pines empleados y los usos para cada pin. En las siguientes secciones, se detallara el conexionado para cada elemento del robot individualmente.

Pin	Funcion	Se conecta a...
1	3.3VDC	MPU(VCC)
2	5VDC	Ultrasonicos (Vcc)
3	SDA1(I1C)	MPU(SDA)
4		
5	SCL1(I2C)	MPU(SCL)
6	GND	MPU(GND)
7		
8		
9		
10		
11	GPIO 0	Ultrasonico 1 (Trig)
12		LM298(ENA)
13	GPIO 2	ultrasonico 1 echo
14		
15	GPIO 3	Ultrasonico 3 (Trig)
16		LM298(IN1)
17		
18		LM298(IN2)
19		
20		

(a) Conexión de los primeros 20 pines

Pin	Funcion	Se conecta a...
21		Encoder 2
22	GPIO 6	Conversor (a definir)
23		Encoder 2
24		
25		
26		
27		
28		
29	GPIO 21	motor
30		
31	GPIO 22	motor
32		
33	GPIO 23	motor
34		
35		
36	GPIO 27	Ultrasonico 2 (Trig)
37	GPIO 25	conversor (a definir)
38		Encoder 1
39	GND	Ultrasonicos (Gnd)
40		Encoder 1

(b) Conexión de los ultimos 20 pines

Figura 9: Pines empleados en el Raspberry Pi

5.1.1. Sensores Ultrasónicos

Para la conexión del sensor ultrasónico al Raspberry se empleo el conexionado de la **figura 10** para cada sensor ultrasónico. Se emplea un convertidor lógico de tensión entre el Raspberry y el sensor ultrasónico para realizar la conversión de 5 a 3.3 [V].

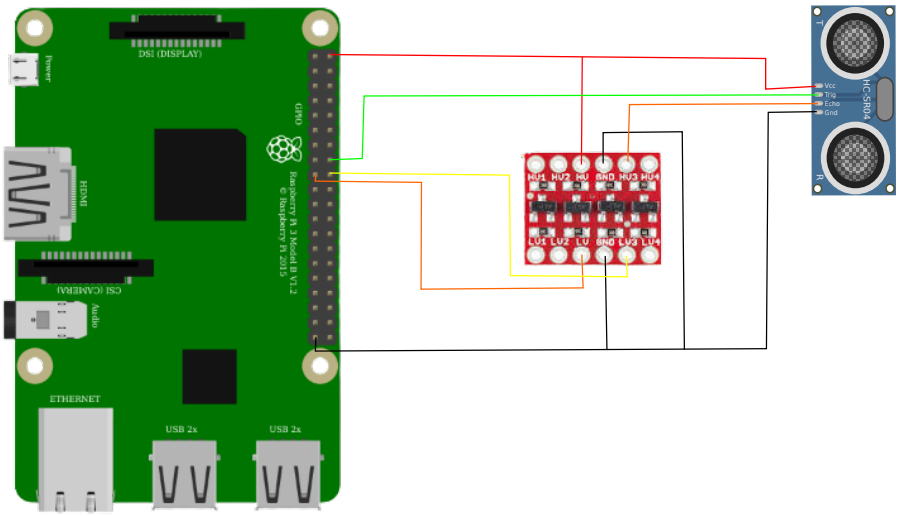


Figura 10: Conexionado del sensor ultrasónico

5.1.2. Motor con L298N

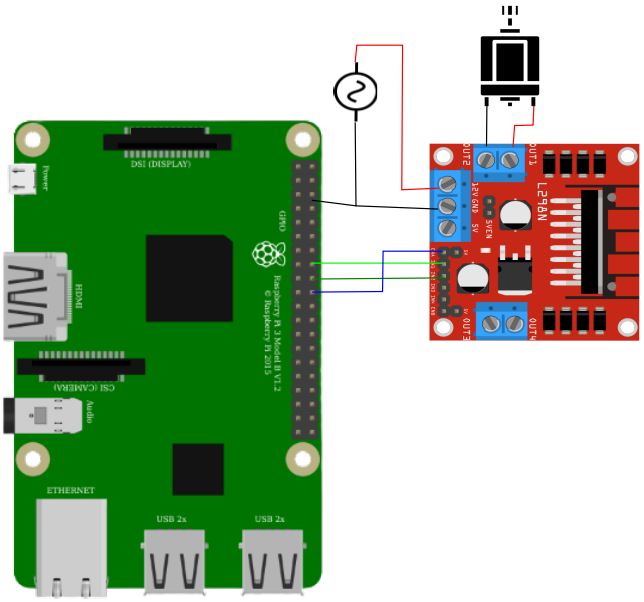


Figura 11: Conexionado del motor con el puente H

5.2. Código de Implementación

5.2.1. Sensor Ultrasonico

```
# Importamos la paquteria necesaria
import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
TRIG1 = 11 #Variable que contiene el GPIO al cual conectamos la señal TRIG del sensor
ECHO1 = 13 #Variable que contiene el GPIO al cual conectamos la señal ECHO del sensor

TRIG2 = 36 #Variable que contiene el GPIO al cual conectamos la señal TRIG del sensor
ECHO2 = 37 #Variable que contiene el GPIO al cual conectamos la señal ECHO del sensor

TRIG3 = 15 #Variable que contiene el GPIO al cual conectamos la señal TRIG del sensor
ECHO3 = 22 #Variable que contiene el GPIO al cual conectamos la señal ECHO del sensor

GPIO.setmode(GPIO.BOARD)     #Establecemos el modo según el cual nos refiriremos a los GPIO de nuestra RPi
GPIO.setup(TRIG1, GPIO.OUT) #Configuramos el pin TRIG como una salida
GPIO.setup(ECHO1, GPIO.IN)  #Configuramos el pin ECHO como una salida
GPIO.setup(TRIG2, GPIO.OUT) #Configuramos el pin TRIG como una salida
GPIO.setup(ECHO2, GPIO.IN)  #Configuramos el pin ECHO como una salida
GPIO.setup(TRIG3, GPIO.OUT) #Configuramos el pin TRIG como una salida
GPIO.setup(ECHO3, GPIO.IN)  #Configuramos el pin ECHO como una salida

#Contenemos el código principal en un aestructura try para limpiar los GPIO al terminar o presentarse un error
try:
    #Implementamos un loop infinito
    while True:
#ULTRASONICO 1-------------------------------------------------
        # Ponemos en bajo el pin TRIG y después esperamos 0.5 seg para que el transductor se estabilice
        GPIO.output(TRIG1, GPIO.LOW)
        time.sleep(0.5)

        #Ponemos en alto el pin TRIG esperamos 10 uS antes de ponerlo en bajo
        GPIO.output(TRIG1, GPIO.HIGH)
        time.sleep(0.00001)
        GPIO.output(TRIG1, GPIO.LOW)

        # En este momento el sensor envía 8 pulsos ultrasónicos de 40kHz y coloca su pin ECHO en alto
        # Debemos detectar dicho evento para iniciar la medición del tiempo

        while True:
            pulso_inicio = time.time()
            if GPIO.input(ECHO1) == GPIO.HIGH:
                break

        # El pin ECHO se mantendrá en HIGH hasta recibir el eco rebotado por el obstáculo.
        # En ese momento el sensor pondrá el pin ECHO en bajo.
        # Prodedemos a detectar dicho evento para terminar la medición del tiempo

        while True:
            pulso_fin = time.time()
            if GPIO.input(ECHO1) == GPIO.LOW:
                break

        # Tiempo medido en segundos
        duracion1 = pulso_fin - pulso_inicio

        #Obtenemos la distancia considerando que la señal recorre dos veces la distancia a medir y que la velocidad
        ↪  del sonido es 343m/s
        distancia1 = (34300 * duracion1) / 2
#ULTRASONICO 2-------------------------------------------------------------------
        time.sleep(0.5)
        # Ponemos en bajo el pin TRIG y después esperamos 0.5 seg para que el transductor se estabilice
        GPIO.output(TRIG2, GPIO.LOW)
        time.sleep(0.5)

        #Ponemos en alto el pin TRIG esperamos 10 uS antes de ponerlo en bajo
        GPIO.output(TRIG2, GPIO.HIGH)
        time.sleep(0.00001)
        GPIO.output(TRIG2, GPIO.LOW)

        # En este momento el sensor envía 8 pulsos ultrasónicos de 40kHz y coloca su pin ECHO en alto
```

```
        # Debemos detectar dicho evento para iniciar la medición del tiempo

        while True:
            pulso_inicio = time.time()
            if GPIO.input(ECHO2) == GPIO.HIGH:
                break

        # El pin ECHO se mantendrá en HIGH hasta recibir el eco rebotado por el obstáculo.
        # En ese momento el sensor pondrá el pin ECHO en bajo.
        # Prodedemos a detectar dicho evento para terminar la medición del tiempo

        while True:
            pulso_fin = time.time()
            if GPIO.input(ECHO2) == GPIO.LOW:
                break

        # Tiempo medido en segundos
        duracion2 = pulso_fin - pulso_inicio

        #Obtenemos la distancia considerando que la señal recorre dos veces la distancia a medir y que la velocidad
        ↪  del sonido es 343m/s
        distancia2 = (34300 * duracion2) / 2

#ULTRASONICO 3------------------------------------------------------------
            # Ponemos en bajo el pin TRIG y después esperamos 0.5 seg para que el transductor se estabilice
        time.sleep(0.5)
        GPIO.output(TRIG3, GPIO.LOW)
        time.sleep(0.5)

        #Ponemos en alto el pin TRIG esperamos 10 uS antes de ponerlo en bajo
        GPIO.output(TRIG3, GPIO.HIGH)
        time.sleep(0.00001)
        GPIO.output(TRIG3, GPIO.LOW)

        # En este momento el sensor envía 8 pulsos ultrasónicos de 40kHz y coloca su pin ECHO en alto
        # Debemos detectar dicho evento para iniciar la medición del tiempo

        while True:
            pulso_inicio = time.time()
            if GPIO.input(ECHO3) == GPIO.HIGH:
                break

        # El pin ECHO se mantendrá en HIGH hasta recibir el eco rebotado por el obstáculo.
        # En ese momento el sensor pondrá el pin ECHO en bajo.
        # Prodedemos a detectar dicho evento para terminar la medición del tiempo

        while True:
            pulso_fin = time.time()
            if GPIO.input(ECHO3) == GPIO.LOW:
                break

        # Tiempo medido en segundos
        duracion3 = pulso_fin - pulso_inicio

        #Obtenemos la distancia considerando que la señal recorre dos veces la distancia a medir y que la velocidad
        ↪  del sonido es 343m/s
        distancia3 = (34300 * duracion3) / 2
#RESULTADOS ---------------------------------------------------------

        # Imprimimos resultado
        print( "Distancia 1: %.2f cm" % distancia1)
        # Imprimimos resultado
        print( "Distancia 2: %.2f cm" % distancia2)
        # Imprimimos resultado
        print( "Distancia 3: %.2f cm" % distancia3)
finally:
    # Reiniciamos todos los canales de GPIO.
    GPIO.cleanup()
```

5.3. Motor con Encoder

```
def PID_function():

    global previous_time
    global previous_error
    global Integral
    global D_cycal
    global Kp
    global Ki
    global Kd

    error = int(Set_RPM) -feedback

    if (previous_time== 0):
         previous_time =time.time()

    current_time = time.time()
    delta_time = current_time - previous_time
    delta_error = error - previous_error

    Pout = (Kp/10 * error)

    Integral += (error * delta_time)


    if Integral>10:
        Integral=10

    if Integral<-10:
        Integral=-10

    Iout=((Ki/10) * Integral)


    Derivative = (delta_error/delta_time)
    previous_time = current_time
    previous_error = error

    Dout=((Kd/1000 )* Derivative)

    output = Pout + Iout + Dout                    # PID

    if ((output>D_cycal)&(D_cycal<90)):
        D_cycal+=1

    if ((output<D_cycal)&(D_cycal>10)):
        D_cycal-=1

    return ()
"""  Calculos de RPM    """

def RPM_function():
    global feedback
    tc=time.time()


    while (GPIO.input(22)==False):
        v=0
        ts=time.time()
        time_count=ts-tc

        if (time_count>7):
            print("Sin lectura de feedback ")
            feedback=0
            return ()


    while (GPIO.input(22)==True):
        i=0
        ts=time.time()
        time_count=ts-tc
        if (time_count>7):
            print("sin lectura de feedback ")
            feedback=0
            return ()
```

```
v = time.time()                                 # primer punto en el tiempo
while (GPIO.input(22)==False):
    s=0
while (GPIO.input(22)==True):
    h=0
h=time.time()                                   # segundo punto en el tiempo

w=(60/(h-v))                                    # Velocidad del motor en RPM
feedback = w
```

6. Anexos

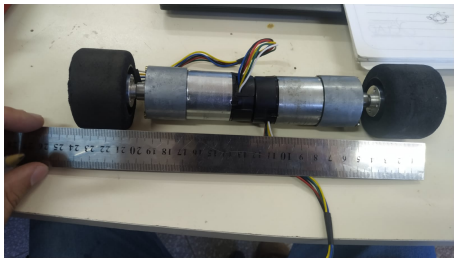
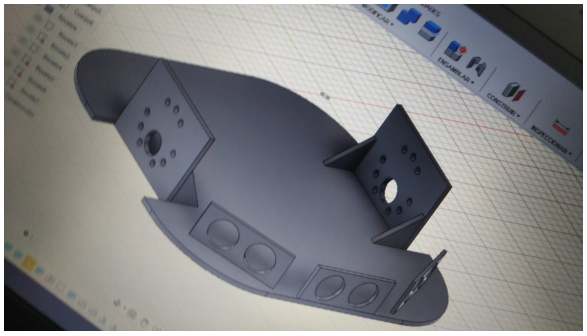
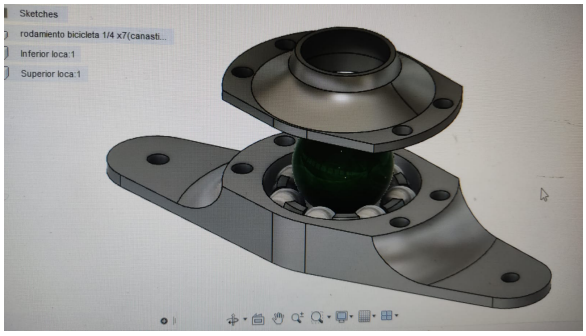


Figura 12: Medición de los motores



(a) Diseño de la base del chasis del robot



(b) Diseño de la rueda loca

Figura 13: Disenhos realizados en Fusion 360

Figura 14: Motores montados en el chasis diseñado

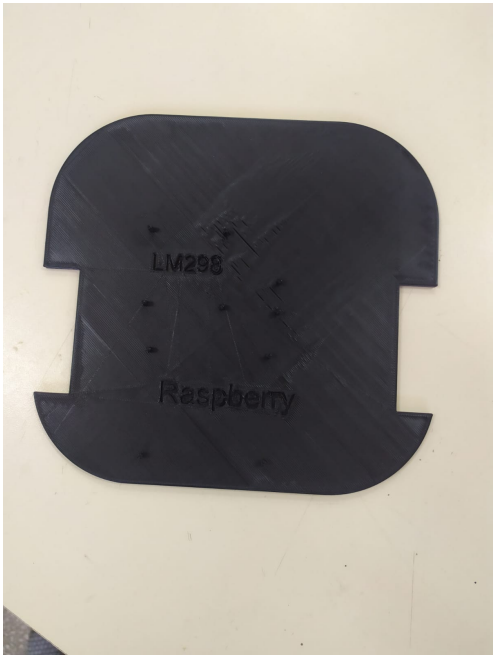


Figura 15: Tapa superior del robot

Figura 16: ubicación de los elementos de control dentro de la tapa superior

Figura 17: Ensamblado de la tapa del robot sobre el robot

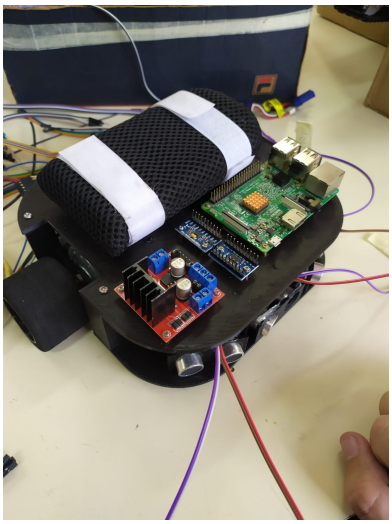


Figura 18: Fijado de la bateria del Raspberry en el robot

Figura 19: Conexionados preliminares de prueba

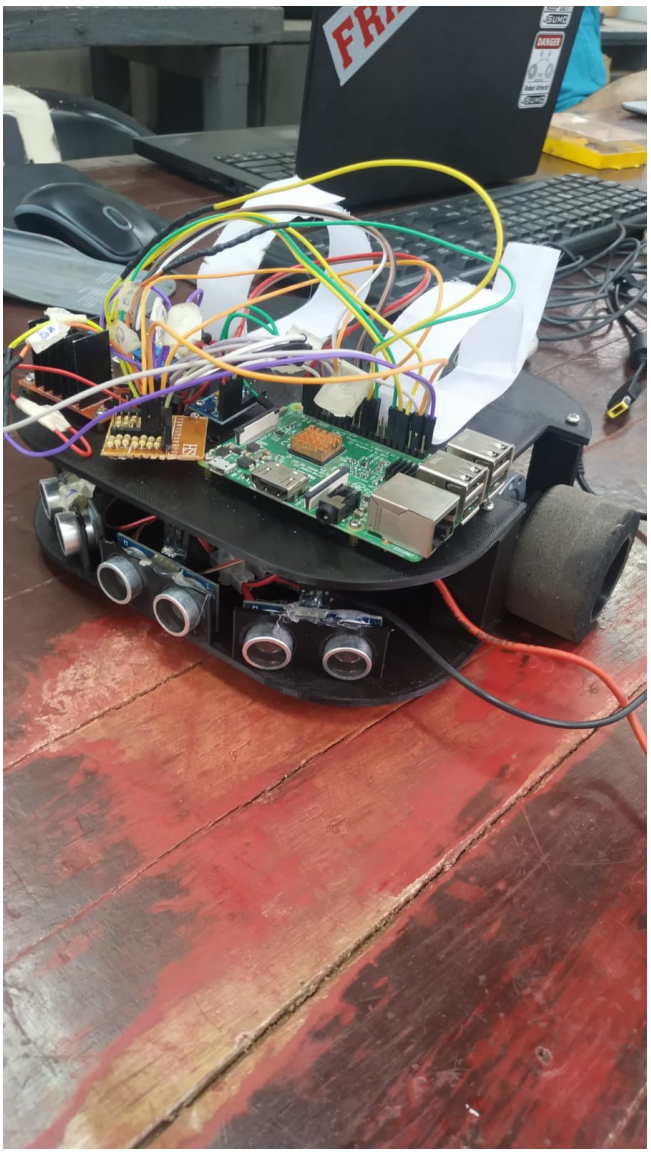


Figura 20: Caption

7. Conclusión

Para esta presentación hemos adquirido los conocimientos necesarios para controlar un robot movil empleando el Raspberry Pi. Nos interiorizamos mas en Python y sus librerías para el Raspberry Pi, adquirimos los conocimientos necesarios para controlar un robot empleando ciertos algoritmos, como el PID, y pulimos nuestras capacidades de diseño con impresión 3D.

Bibliografía

[1] Yong Koo Hwang, Narendra Ahuja et al. «A potential field approach to path planning.» En: *IEEE transactions on robotics and automation* 8.1 (1992), págs. 23-32.

[2] Carl Knospe. «PID control». En: *IEEE Control Systems Magazine* 26.1 (2006), págs. 30-31.

[3] *Local Path Planning Using virtual Potential Field in Python — by ASLAN — Nerd For Tech — Medium.* URL: https://medium.com/nerd-for-tech/local-path-planning-using-virtual-potential-field-in-python-ec0998f490af.

[4] *Módulo controlador de motores L298N — Tienda y Tutoriales Arduino.* URL: https://www.prometec.net/l298n/.

[5] *Sensor Ultrasonido HC-SR04.* URL: https://naylampmechatronics.com/sensores-proximidad/10-sensor-ultrasonido-hc-sr04.html.

[6] *Teoría PID explicada - NI.* URL: https://www.ni.com/es-cr/innovations/white-papers/06/pid-theory-explained.html.