Dispositivo de medida de distancias con sensor de ultrasonidos.

David serrano moreno.
IoT. Creación de prototipos con Raspberry.
15 Julio 2021.

Presentación.

En esta practica se ha pretendido hacer un dispositivo para la detección de distancias y luego trazar un perfil del perímetro detectado.
Consiste en un detector de ultrasonidos HC-SR04 que es el típico que viene en los kits para Arduino / Raspberry.
Un motor de paso a paso 28BYJ-48.que también es típico en los kits para Arduino / Raspberry.

Se ha tenido que fabricar una pieza de soporte para el sensor de ultrasonidos y una pieza de soporte para el motor, con impresión 3D.

Se ha utilizado una Raspberry pi B+ como sistema de gestión del dispositivo sensor.

El receptor es un programa que se ejecuta en el PC.

Se ha realizado tanto el programa emisor que se ejecuta en la Rapsberry con el programa receptor que se ejecuta en el PC, en python3.

El protocolo de comunicación entre emisor y receptor es MQTT.

Emisor.

Consiste en el cabezal que es el sensor de ultrasonidos HC-SR04 el soporte del sensor fabricado en impresión 3D, el motor de paso a paso 28BYJ-48, el soporte del motor fabricado en impresión 3D, un pulsador que sirve de detector de fin de carrera, colocado en el soporte del motor.

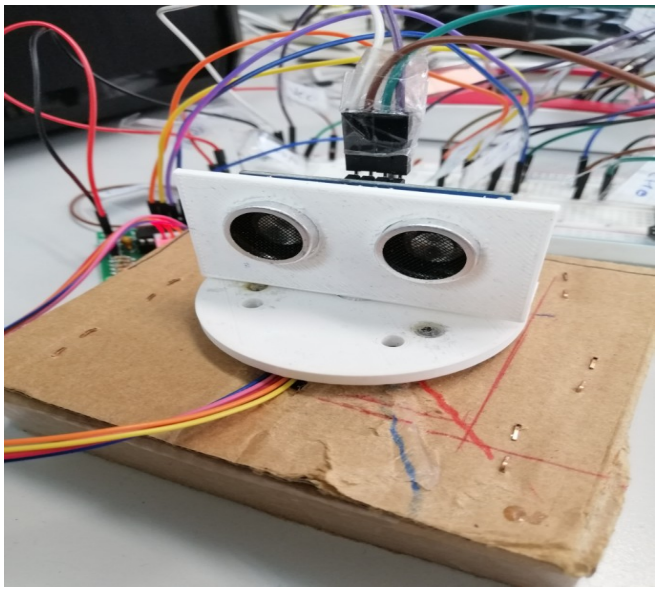
La Raspberry pi 3B+ sirve como controlador del emisor. Tiene instalado un linux debian lite. Y ejecuta el programa del emisor.

Una placa Protoboard donde se hacen las conexiones entre el motor , sensor y pulsador con la Raspberry.

Una fuente de alimentación de 5V para el sensor y el pulsador que va conectada directamente a la placa protoboard.

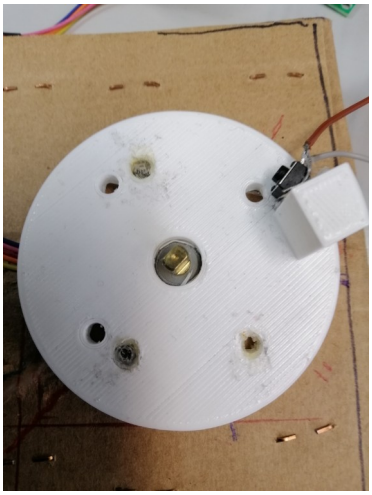
Una fuente de alimentación de 12v que alimenta el motor paso a paso 28BYJ-48. El motor paso a paso viene con una pequeña placa de circuito impreso con el chip ULN2001A. Y la conexión de 12v se hace directamente a esta placa.

Cabezal.

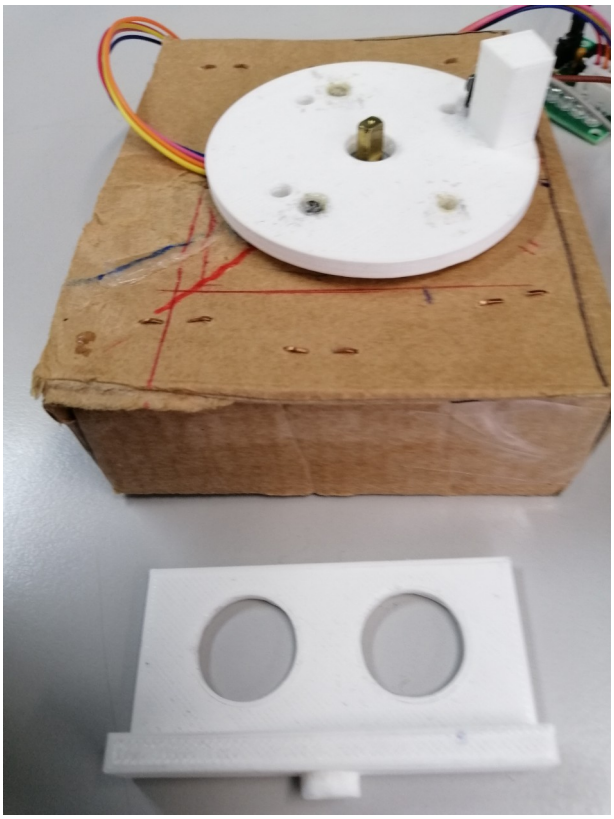


Esquema de la pieza del cabezal.

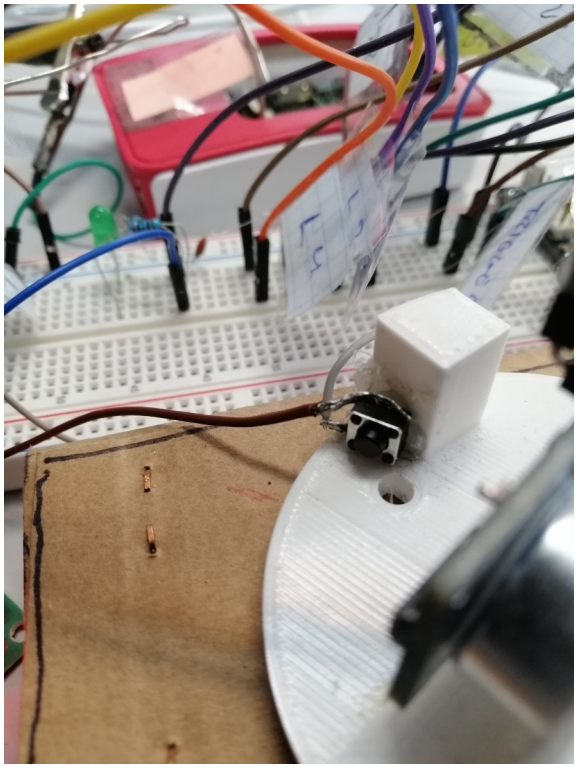
Base del motor.



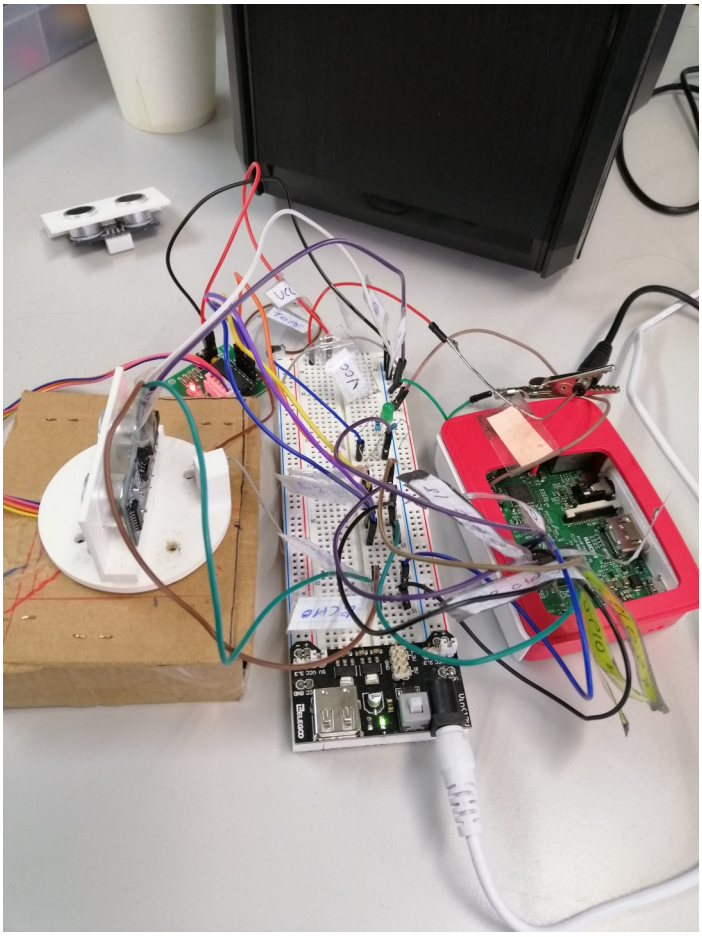
Esquema de la pieza de la base del motor.



Pulsador de fin de carrera



Proyecto.



Programa emisor en la Raspberry.

El programa emisor consiste en tres hilos.
El primero ejecuta las funciones que mueven el motor paso a paso.
Primero inicializa los puertos de la raspberry y ejecuta una vuelta en sentido horario para buscar el pulsador de tope, necesita ese tope para encontrar un punto de referencia. Cuando este pulsador se activa se aborta la funcion que ejecuta la primera vuelta y se inicia otra vuelta en sentido antihorario hasta un número de pasos máximo. A partir de entonces entra en un bucle en el que se ejecutan alternativamente una vuelta en sentido horario y otra en sentido antihorario.

El segundo hilo, es para el sensor de ultrasonidos. Esta función envía un pulso a la entrada trigger del sensor (pulso de disparo) y se queda escuchando el eco del sensor. Cuando se recibe un eco se mide la diferencia de tiempo entre el disparo y el eco recibido, a partir del cual se puede medir la distancia hasta el objeto.

El tercero es para el emisor MQTT. Este hilo envía continuamente las distancias detectadas y los pasos que ha dado el motor paso a paso, para que en el emisor se pueda reconstruir un perfil del área barrida por el sensor. Tambien envía el sentido de la vuelta en la que esta, si es horario o antihorario.

Programa emisor en la Raspberry.

```
import time
#raspberry
import RPi.GPIO as GPIO
# módulo para multihilo
import threading

#protocolo mqtt
import paho.mqtt.publish as publish
#credenciales MQTT
USUARIO = 'pi'
CONTRASENA = 'gaticos16'
IP_MQTT =  "10.199.160.173" #"192.168.1.5"
PUERTO = 1883

#establece pines de GPIO para el sensor de ultrasonidos
GPIO_TRIGGER = 23
GPIO_ECHO = 24
#establece pines de GPIO para el motor paso a paso
BOBINA1 = 4   #AZUL
BOBINA2 = 17  #ROSA
BOBINA3 = 18  #AMARILLO
BOBINA4 = 27  #NARANJA
#detectar el límite de barrido
GPIO_TOPE_DERECHO = 22
#secuencia de activacion de las bobinas para el giro
ciclo_de_pasos_horario = ((1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1))
ciclo_de_pasos_antihorario = ((0, 0, 0, 1), (0, 0, 1, 0), (0, 1, 0, 0), (1, 0, 0, 0))
#pasos por barrido
PASOS_INICIO = 700  #un giro de tope a tope son 675 pasos
PASOS_INICIO_DOS = 510
PASOS = 500
#velocidad de barrido
DEMORA = 0.01
#variables globales
DISTANCIA = 0
INDICE_PASOS = 0
SENTIDO_GIRO = 0



#----------------------------------------hilo3----------------------------------------------------
#-------------------------------------------------------------------------------------------------

#-------------------------funciones para el emisor de datos-----------------------------------------

# MQTT Publish Demo
# publica dos mensajes, con dos "topics" (paso y distancia) diferentes

#ip de la raspberry
#usuario y contraseña del archivo /etc/mosquitto/pwfile del PC.
def envia_datos():
    global DISTANCIA
    global INDICE_PASOS
    global SENTIDO_GIRO
    indice_pasos = 0
    while True:
        if INDICE_PASOS != indice_pasos :   # si se avanza un paso
            indice_pasos = INDICE_PASOS    # se envían los datos
            publish.single("paso", INDICE_PASOS, hostname=IP_MQTT, port=PUERTO, auth = {'username': USUARIO, 'password': CONTRASENA})
            publish.single("distancia", DISTANCIA, hostname=IP_MQTT, port=PUERTO, auth = {'username': USUARIO, 'password': CONTRASENA})
            publish.single("giro", SENTIDO_GIRO, hostname=IP_MQTT, port=PUERTO, auth = {'username': USUARIO, 'password': CONTRASENA})



#----------------------------------------hilo2----------------------------------------------------
#-------------------------------------------------------------------------------------------------

#-------------------------funciones para el sensor de ultrasonidos-----------------------------------
def sensor():
    global DISTANCIA
    while True:
        # pone Trigger a alto(HIGH)
        GPIO.output(GPIO_TRIGGER, True)

        # pone Trigger tras 0.01ms a bajo(LOW)
        time.sleep(0.00001)
        GPIO.output(GPIO_TRIGGER, False)

        StartTime = time.time()
        StopTime = time.time()

        # guarda el momento de envio "StartTime"
        while GPIO.input(GPIO_ECHO) == 0:
            StartTime = time.time()

        # guarda el momento de llegada del eco
        while GPIO.input(GPIO_ECHO) == 1:
            StopTime = time.time()

        # diferencia entre tiempo de envio y llegada
        TimeElapsed = StopTime - StartTime
        # se multiplica por la velocidad del sonido (34300 cm/s)
        # y se divide entre 2, porque es ida y vuelta
        DISTANCIA = (TimeElapsed * 34300) / 2
```

```
#---------------------------------------hilo1-----------------------------------------------------
#-------------------------------------------------------------------------------------------------

#----------------------------funciones para el motor paso a paso-----------------------------------
def activar_bobina(paso,demora):
    GPIO.output(BOBINA1,paso[0])
    GPIO.output(BOBINA2,paso[1])
    GPIO.output(BOBINA3,paso[2])
    GPIO.output(BOBINA4,paso[3])
    time.sleep(demora)

def primera_vuelta(vuelta):
    global Vuelta
    if vuelta == 2:
        return Vuelta
    else:
        Vuelta = vuelta
        return Vuelta

def giro(pasos,ciclo_de_pasos,demora):
    global DISTANCIA
    global INDICE_PASOS
    global SENTIDO_GIRO
    # sentido de giro 0 = horario, 1 = antihorario.
    SENTIDO_GIRO = ciclo_de_pasos[0][0]

    indice_pasos_ciclo = 0
    for indice_pasos in range (1,pasos+1,1):
        activar_bobina(ciclo_de_pasos[indice_pasos_ciclo],demora)
        indice_pasos_ciclo +=1
        if indice_pasos_ciclo > 3:
            indice_pasos_ciclo = 0
        if primera_vuelta(2) == 0:      # ¿es la primera vuelta?
            if GPIO.input(GPIO_TOPE_DERECHO) == True:
                break

        # consulta al sensor
        distancia = DISTANCIA
        #visualiza_distancia(distancia, indice_pasos)
        # envia pasos al MQTT, la distancia ya esta en la variable DISTANCIA
        INDICE_PASOS = indice_pasos

def posicion_inicial():
    primera_vuelta(0)
    giro(PASOS_INICIO,ciclo_de_pasos_horario,DEMORA)
    primera_vuelta(1)
    giro(PASOS_INICIO_DOS,ciclo_de_pasos_antihorario,DEMORA)

def programa():
    global SENTIDO_GIRO
    #busca posicion inicial
    posicion_inicial()
    while True:
        giro(PASOS,ciclo_de_pasos_horario,DEMORA)
        giro(PASOS,ciclo_de_pasos_antihorario,DEMORA)



#-------------------------------------------------------------------------------------------------
#-------------------------------------------------------------------------------------------------

#-----------------------------inicio--------------------------------------------------------------

def apaga_puertos():
    GPIO.output(GPIO_TRIGGER,GPIO.LOW)
    GPIO.output(BOBINA1,GPIO.LOW)
    GPIO.output(BOBINA2,GPIO.LOW)
    GPIO.output(BOBINA3,GPIO.LOW)
    GPIO.output(BOBINA4,GPIO.LOW)

def configurar_puertos():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(GPIO_ECHO,GPIO.IN)
    GPIO.setup(GPIO_TRIGGER,GPIO.OUT)
    GPIO.setup(BOBINA1,GPIO.OUT)
    GPIO.setup(BOBINA2,GPIO.OUT)
    GPIO.setup(BOBINA3,GPIO.OUT)
    GPIO.setup(BOBINA4,GPIO.OUT)
    GPIO.setup(GPIO_TOPE_DERECHO,GPIO.IN)
    # pone los puertos de salida a cero
    apaga_puertos()

def main():
    configurar_puertos()

    hilo1 = threading.Thread(target=programa)
    hilo2 = threading.Thread(target=sensor)
    hilo3 = threading.Thread(target=envia_datos)
    hilo1.start()
    hilo2.start()
    hilo3.start()

    try:
        while True:
            pass
    finally:
        # pone los puertos de salida a cero
        GPIO.cleanup()

if __name__ == '__main__' :
    main()
```

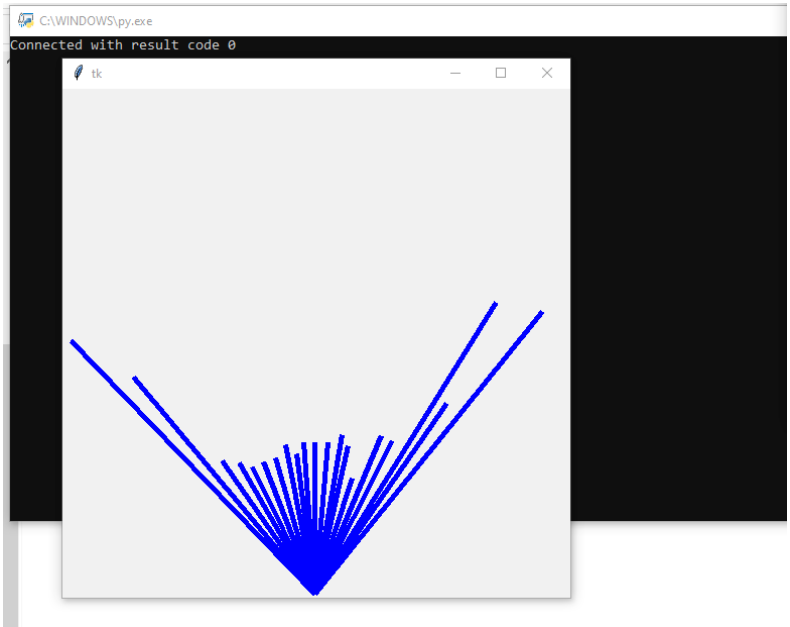
Receptor

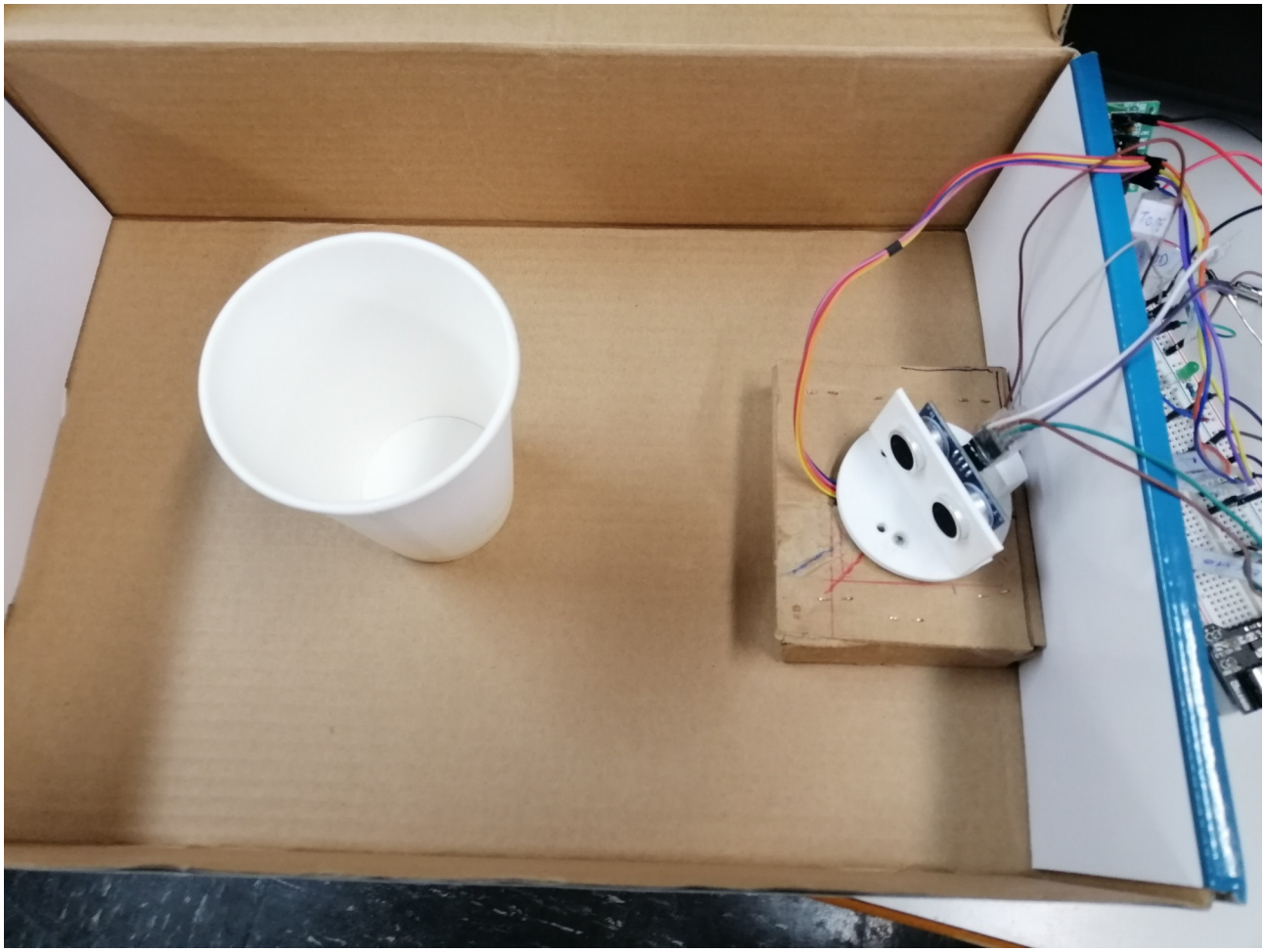
El programa receptor consiste en dos hilos.
El primero se encarga de establecer la conexión con el emisor MQTT, y de escuchar continuamente los datos recibidos.
El segundo se encarga de mostrar una pantalla gráfica en la que se muestra el área barrida por el sensor.

programa receptor en el PC.

```
# módulo para multihilo
import threading
# libreria matemática
import math
# libreria gráfica
from tkinter import *
# cliente MQTT
# continuamente monitorea los "topics" (etiquetas) de datos
# sudo pip install paho-mqtt
import paho.mqtt.client as mqtt
# credenciales para el protocolo MQTT
USUARIO = 'pi'
CONTRASENA = 'gaticos16'
IP_MQTT =  "10.199.160.173" #"192.168.1.5"
PUERTO = 1883
#variables globales
PASO = 0
DISTANCIA = 0
SENTIDO_GIRO = 0

#-----------------------------------hilo2----------------------------------------
#--------------------------------------------------------------------------------

#-----------------------------------funciones de visualizacion---------------------
def calcula_coordenadas(paso, distancia, sentido_giro):
    global alcada, amplada
    coordenadas = [0,0]

    if distancia > 50:
        distancia = 50

    grado = int(paso/5.6)  #5.6 pasos por grado

    # ¿es giro antihorario?
    if sentido_giro == 0:
        grado = grado + 45

    # ¿es giro horario?
    if sentido_giro == 1:
        grado = 135 - grado

    # se aplica un factor de escalabilidad
    distancia = distancia*10
    polarx = abs(int(distancia*math.cos(math.pi*grado/180)+250))
    polary = abs(int(distancia*math.sin(math.pi*grado/180)-500))

    coordenadas[0] = polarx

    coordenadas[1] = polary
    return coordenadas

def visualizacion():

    global PASO
    global DISTANCIA
    global SENTIDO_GIR
    global alcada
    global amplada
    sentido_giro = SENTIDO_GIRO
    paso = 0
    coordenadas = [0,0]
    ventana = Tk()
    amplada = ventana.winfo_screenwidth()
    alcada = ventana.winfo_screenheight()
    # canvas = widget que se usa para dibujar graficos simples en una ventana
    grafico = Canvas(ventana,height=500,width=500)  # crea un area de 500x500 donde se dibujaran las funciones

    while True:
        # si se avanza un paso se dibuja una linea
        if PASO != paso :
            paso = PASO

            #¿ha llegado al final? -> cambia de sentido
            if SENTIDO_GIRO != sentido_giro:
                sentido_giro = SENTIDO_GIRO

                #limpia la ventana gráfica
                grafico.delete(ALL)
            else:
                coordenadas = calcula_coordenadas(paso, DISTANCIA, sentido_giro)
                # dibuja una linea desde el punto X,Y al punto 250,500
                grafico.create_line(coordenadas[0],coordenadas[1],250,500,fill="blue",width=5)
                grafico.pack()
                #se actualiza la ventana gráfica
                ventana.update()

    ventana.mainloop()
```

```
#-------------------------------------hilo1----------------------------------------
#----------------------------------------------------------------------------------

#-------------------------------------funciones del receptor-----------------------
# conexion. el cliente recibe una respuesta CONNACK desde el servidor.
def on_connect(cliente, userdata, flags, rc):
    # para saber si se ha podido conectar.
    print("Connected with result code " + str(rc))

    # subscribirse en on_connect() - si se pierde la conexion y
    # reconecta las subscripciones se renovarán.
    cliente.subscribe("paso")
    cliente.subscribe("distancia")
    cliente.subscribe("giro")

# recepción de mensaje. el mensaje se recibe desde el servidor.
def on_message(cliente, userdata, msg):
    global PASO
    global DISTANCIA
    global SENTIDO_GIRO
    paso = 0
    distancia = 0
    sentido_giro = 0
    # comprueba si los datos recibidos coinciden con los "topics" (etiquetas) predefinidos
    # recoge los datos enviados
    if msg.topic == "paso":
        #convierte el dato enviado de bit a entero
        paso = float(msg.payload)
        paso = int(paso)
        PASO = paso

    if msg.topic == "distancia":
        #convierte el dato enviado de bit a entero
        distancia = float(msg.payload)
        distancia = int(distancia)
        DISTANCIA = distancia

    if msg.topic == "giro":
        #convierte el dato enviado de bit a entero
        sentido_giro = float(msg.payload)
        sentido_giro = int(sentido_giro)
        SENTIDO_GIRO = sentido_giro


def receptor():
    # crear un cliente MQTT y adjuntar las funciones de conexión y recepción de mensajes
    cliente = mqtt.Client()
    cliente.on_connect = on_connect
    cliente.on_message = on_message

    cliente.username_pw_set(USUARIO, CONTRASENA)
    cliente.connect(IP_MQTT, PUERTO, 60)

    # procesa el trafico de red y reparte las respuestas. tambien gestiona las reconexiones
    # revisa la documentacion en
    # https://github.com/eclipse/paho.mqtt.python
    # por información de como usar otras funciones loop*()
    cliente.loop_forever()

#-------------------------------------inicio---------------------------------------

def main():
    hilo1 = threading.Thread(target = receptor, args=())
    hilo2 = threading.Thread(target = visualizacion, args=())
    hilo1.start()
    hilo2.start()

if __name__ == '__main__' :
    main()
```





conclusión.

El desarrollo e implementación de este dispositivo ha exigido la ayuda de varias disciplinas como son la electrónica, el diseño 3D, la fabricación con impresora 3D, y la programación en Python. En este proyecto se ha hecho patente que llevar a término un producto es una tarea multidisciplinar en la que pueden intervenir varios profesionales o expertos. Por sencillo que sea el objetivo del proyecto.