Control del Motor vía MQTT

Para el control del motor por medio de mqtt es importante tenerlo instalado y funcionando tanto en tu maquina virtual como en tu raspberry (ver la lección: Instalación de Mosquito en Ubuntu 20.04 del curso Internet de las Cosas de Samsung).

Para los códigos desarrollados se usó un broker público, en este caso hivemq.com, para obtener la IP pública usamos:

>>nslookup bróker hivemq.com

Server: 127.0.0.53

Address: 127.0.0.53#53

Non-authoritative answer:

Name: broker.hivemq.com

Address: 35.157.99.2

Name: broker.hivemq.com

Address: 3.123.49.252

Tomamos la IP 3.123.49.252 para nuestro proyecto, es importante recordar que esta IP estará disponible por un corto tiempo, así que no olvides cambiarla o asegurarte de que esté vigente la que uses.

Tenemos que definir un Topic para poder direccionar nuestros dispositivos, en este caso usamos dos, capstone/salon/virtual/RPM para la velocidad y capstone/salon/virtual/voltaje para el voltaje.

También es importante mencionar que se necesita instalar en tu raspberry la librería mqtt, lo cual hacemos metiéndonos al programa **Thonny** (en la raspberry), luego damos click en **Tools** y finalmente en **Manage packages** y en la ventana que nos sale escribimos mqtt y lo instalamos.

Cuando tenemos lo anterior ya podemos comenzar a escribir nuestro código en Python y lograr el control de nuestro motor con mqtt, dejaremos un momento de lado las conexiones del motor y la raspberry pues en las lecciones anteriores ya tratamos este tema, para comenzar nuestro código primero importamos las funciones de mqtt que necesitamos:

import paho.mqtt.client as mqtt

Después necesitamos definir una clase que utilizaremos para enviar y recibir los mensajes del bróker, pero al mismo tiempo decimos que vamos a hacer con la información recibida del bróker, en este caso pensamos en que el mensaje recibido desde el bróker serán números en 0 y 100 representando los porcentajes deseados para el funcionamiento del motor, es importante mencionar que los “números ” recibidos están en formato **str, es decir,** se encuentran en formato cadena de caracteres, por este motivo debemos de convertir este valor a entero con la función **int()**, también necesitamos decodificar el mensaje enviado por el bróker, para eso se utiliza message.payload.decoded(“utf-8”) y el utf-8 utilizado es el código del teclado que normalmente usa Linux y Windows (es importante mencionar que mac puede no usar este código de caracteres dependiendo de que versión de sistema operativo tenga), por supuesto se toma en cuenta que los porcentajes deben estar entre 0 y 100 y se previene esto con un if, así como el mensaje para salir del código, es decir, si el mensaje es “-1” se requiere que se cierre el script en Python y se cierren los puertos, entonces el script en Python queda:

def messageFunction (client, userdata, message):

global x

topic = str(message.topic)

message = int(message.payload.decode("utf-8")) # transformamos el mensaje de texto a entero

if message>=0 and message <=100:

print(message)

p.ChangeDutyCycle(message) # Aquí ocurre la magia para el cambio de voltaje

elif message==-1:

x=-1

else:

print('El porcentaje debe estar entre 0 y 100')

Ahora necesitamos configurar el bróker dentro de nuestro script de Python, esto lo hacemos con:

broker\_address="18.157.172.72" # IP de hivemq,

ourClient = mqtt.Client("Alex\_Alaffita") # Crea un objeto para el cliente de mqtt

ourClient.connect(broker\_address, 1883) # Este nos conecta al broker, también funciona con la

# url es decir "hivemq.com"

ourClient.subscribe("capstone/salon/virtual") # El topic para suscribirnos

ourClient.on\_message = messageFunction # Para pegar el mensaje dentro de la variable

ourClient.loop\_start() # Para comenzar el cliente de mqtt

Finalmente la función para publicar y subscribirnos son:

ourClient.subscribe("capstone/salon/virtual/voltaje") # Subscribe message to MQTT broker ourClient.publish("capstone/salon/virtual/RPM",str(rpm)) # Publish message to MQTT broker

en estas funciones especificamos el topic correspondiente, como podemos darnos cuenta nos subscribimos a "capstone/salon/virtual/voltaje" y publicamos en "capstone/salon/virtual/RPM" el valor en formato cadena de caracteres de la velocidad con str(rpm).

El código en Python que envía la velocidad y recibe los porcentajes con mosquito queda como:

import paho.mqtt.client as mqtt # Importa la libreria MQTT

import RPi.GPIO as GPIO

from time import sleep

import pigpio

from read\_RPM import reader # Archivo read\_RPM contiene a la clase reader

# Definición de la clase para la lectura de los mensages de mqtt

def messageFunction (client, userdata, message):

global x

topic = str(message.topic)

message = int(message.payload.decode("utf-8")) # transformamos el mensaje de texto a entero

if message>=0 and message <=100:

print(message)

p.ChangeDutyCycle(message) # Aquí ocurre la magia para el cambio de voltaje

elif message==-1:

x=-1

else:

print('El porcentaje debe estar entre 0 y 100')

broker\_address="18.157.172.72" # IP de hivemq, se puede poner 'hivemq.com' pero tarda un poco

ourClient = mqtt.Client("Alex\_Alaffita") # Crea un objeto para el cliente de mqtt

ourClient.connect(broker\_address, 1883) # Este nos conecta al broker, también funciona con la

# url es decir "hivemq.com"

ourClient.subscribe("capstone/salon/virtual") # El topic para suscribirnos

ourClient.on\_message = messageFunction # Para pegar el mensaje dentro de la variable

ourClient.loop\_start() # Para comenzar el cliente de mqtt

# Pines de conexión con el shield L298N

in1 = 24

in2 = 23

en = 25

GPIO.setmode(GPIO.BCM) # Para fijar el modo para los pines

GPIO.setup(in1,GPIO.OUT) # Definimos a in1 como de salida en raspberry

GPIO.setup(in2,GPIO.OUT) # Definimos a in2 como de salida en raspberry

GPIO.setup(en,GPIO.OUT) # Definimos a en como de salida en raspberry

GPIO.output(in1,GPIO.LOW)# Para estar seguros que el motor se detiene antes de comenzar

GPIO.output(in2,GPIO.LOW)# Para estar seguros que el motor se detiene antes de comenzar

p=GPIO.PWM(en,1000) # La frecuencia para el pulso PWM en el pin en, de la variable p

p.start(25) # Para comenzar con el 25% con el movimiento del motor

print("\n")

print("Recuerda que debes de suministrar un porcentaje del voltaje (entre 0 y 100)")

print("Esperando...")

print("\n")

cont=0 #Contador de pulsos

# Set up RPM reader

RPM\_GPIO = 6 # PIN GPIO de entrada en Raspberry

SAMPLE\_TIME = 0.1 # Tiempo de muestreo para medir la velocidad

pi = pigpio.pi() # Creación de la variable pi

pi.set\_servo\_pulsewidth(RPM\_GPIO, 2500) # Maximum throttle.

sleep(2)

pi.set\_servo\_pulsewidth(RPM\_GPIO, 500) # Minimum throttle.

sleep(2)

ppc=334 # Pulsos por ciclo, el encoder tiene 334 marcas en la rueda

tach = reader(pi, RPM\_GPIO,ppc) # Función de lectura del encoder, se especifica la variable

# El pin en donde se encuentra y los pulsos por ciclo

#Iniciamos el motor de dc con el 0% del voltaje total (12 v)

GPIO.output(in1,GPIO.HIGH)

GPIO.output(in2,GPIO.LOW)

p.ChangeDutyCycle(0)

x=1

while(x==1):

rpm = tach.RPM() # Función para leer las rpm

print(rpm)

ourClient.subscribe("capstone/salon/virtual/voltaje") # Subscribe message to MQTT broker

ourClient.publish("capstone/salon/virtual/RPM",str(rpm)) # Publish message to MQTT broker

sleep(SAMPLE\_TIME) # Tiempo de espera entre cada lectura

print("Fin del programa, ¡¡¡Saludos!!!")

GPIO.output(in1,GPIO.LOW)

GPIO.output(in2,GPIO.LOW)

GPIO.cleanup() # Limpiamos todos los pines

**Modo de uso.**

1. En la raspberry abrimos una terminal y escribimos >>sudo pigpiod
2. Abrimos Thonny y ejecutamos el archivo del script anterior (recuerda que debes de tener en la misma carpeta el archivo read\_RPM.py)
3. El motor no va a arrancar, para que arranque debes de indicar un porcentaje del voltaje que requieres por medio de mqtt, supongamos que lo quieres arrancar con el 25% de los 12 volts disponibles, es decir, con 3 volts, entonces desde otra pc o maquina virtual que tenga instalado mqtt escribimos:

>>mosquitto\_pub -h 18.157.172.72 -p 1883 -i Alex-Alaffitam -q 2 -t capstone/salon/virtual/voltaje -m "25"

Con lo anterior el rotor del motor comenzará a girar.

1. Si queremos ver la velocidad a la que está girando el motor lo podemos hacer subscribiendonos de la siguiente manera:

>>mosquitto\_sub -h 18.157.172.72 -p 1883 -i Alex-Alaffita3 -q 2 -t capstone/salon/virtual/RPM

1. Si queremos ver el porcentaje del voltaje que se le va suministrando al motor nos podemos subscribir al topic “voltaje”

>>mosquitto\_sub -h 18.157.172.72 -p 1883 -i Alex-Alaffita3 -q 2 -t capstone/salon/virtual/voltaje

1. Si queremos que el motor no gire es suficiente con enviar un mensaje con el 0%

>>mosquitto\_pub -h 18.157.172.72 -p 1883 -i Alex-Alaffitam -q 2 -t capstone/salon/virtual/voltaje -m "0"

1. Si queremos cerrar el script es suficiente con enviar un mensaje con el “-1”

>>mosquitto\_pub -h 18.157.172.72 -p 1883 -i Alex-Alaffitam -q 2 -t capstone/salon/virtual/voltaje -m "-1"