



Universidad de
Oviedo



ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

INTEGRACIÓN DE SISTEMAS DE TELECOMUNICACIONES

PROYECTO "PICAR" UncertanTIC

**DAVID ARIAS-CACHERO RINCÓN
GUILLERMO LENA DÍAZ
CARMEN SÁNCHEZ GARCÍA**

DICIEMBRE 2022

Índice

1.- Protocolo MQTT	2
1.2.- Broker Mosquitto.....	2
1.2.1.- ¿Qué es Mosquitto?	2
1.2.2.- Instalación.....	2
1.2.3.- Configuración	3
1.2.4.- Publicador, subscriptor y broker.....	4
1.2.5.- Implementación en Python	6
2.- Visualización de datos en tiempo real	13
2.1.- Esquema funcional.....	13
2.2.- Formato datos	13
2.3.- Implementación	14
2.3.1.- Código Arduino	14
2.3.2.- Conexión Arduino – MR	14
2.3.3.- Conexión MR – BS.....	14
2.3.4.- Implementación BS.....	14
2.3.5.- Conexión BS – Azure	15
2.3.6.- Conexión Azure - PowerBI.....	15
Referencias.....	16

1.- Protocolo MQTT

Para realizar las comunicaciones entre el robot (MR) y la estación base (BS), se ha decidido usar MQTT, uno de los protocolos Machine to Machine (M2M) más conocidos dentro del ecosistema IoT gracias a su sencillez y ligereza ya que los dispositivos en este tipo de escenarios están limitados en potencia, consumo y ancho de banda.

Su funcionamiento se basa en el intercambio de mensajes entre los publicadores y suscriptores a través de un servidor central denominado broker. Los mensajes son filtrados en tópicos de modo que si un cliente publica en un determinado tópico, sólo recibirán aquellos que estén suscritos al mismo.

Transportando todo lo anterior a nuestro proyecto, se distinguen entonces tres segmentos bien diferenciados:

- Suscriptor: según el tipo de escenario de comunicaciones ejercerá de suscriptor la estación base o el robot
- Publicador: según el tipo de escenario, el rol de publicador lo ocupará el robot o la estación base
- Broker: el broker es estático y estará almacenado en la estación base

1.2.- Broker Mosquitto

1.2.1.- ¿Qué es Mosquitto?

Mosquitto es el software más conocido para implementar aplicaciones basadas en el protocolo MQTT. Además de ser compatible con la gran mayoría de dispositivos (ordenadores de sobremesa, portátiles o microcontroladores) dispone de una comunidad de desarrollo activa que simplificará su comprensión o resolución de errores.

1.2.2.- Instalación

El primer paso para comenzar a desarrollar las comunicaciones de nuestro proyecto es instalar el broker para que el robot pueda enviar mensajes a la estación base y viceversa. Esto lo haremos en la Raspberry de la estación base.

Para ello es necesario actualizar sus repositorios con el siguiente comando:

sudo apt update && sudo apt upgrade

Cuando la interfaz de línea de comandos nos indique que ya se ha completado la acción anterior, se procede a instalar el broker con el comando:

sudo apt install -y mosquitto mosquitto-clients

Como nos interesa que el broker se ejecute cada vez que se enciende la estación base de forma automática, cuando el paquete anterior esté instalado escribiremos el siguiente comando:

```
sudo systemctl enable mosquitto.service
```

Para comprobar si Mosquitto se ejecuta correctamente, ejecutamos el siguiente comando cuyo output se ilustra en la figura 1.1

```
mosquitto -v
```

```
root@raspberrypi:~# mosquitto -v
1670678268: mosquitto version 2.0.11 starting
1670678268: Using default config.
1670678268: Starting in local only mode. Connections will only be possible from
clients running on this machine.
1670678268: Create a configuration file which defines a listener to allow remote
access.
1670678268: For more details see https://mosquitto.org/documentation/authenticat
ion-methods/
1670678268: Opening ipv4 listen socket on port 1883.
1670678268: Opening ipv6 listen socket on port 1883.
1670678268: mosquitto version 2.0.11 running
```

Figura 1. 1.- Output del comando “mosquitto -v”

1.2.3.- Configuración

En la figura 1.1 se puede comprobar la versión del broker. Si nos fijamos más a fondo, nos indica que actualmente sólo es posible realizar conexiones dentro de la propia máquina, es decir, que está ejecutándose en modo local.

Ese no es el caso que nos interesa ya que ahora mismo no sería posible conectarse desde el robot u otro dispositivo. Para permitir el acceso remoto es necesario editar el archivo de configuración *mosquitto.conf* presente en el directorio */etc/mosquitto*

```
root@raspberrypi:~# ls /etc/mosquitto/
aclfile.example certs mosquitto.conf pskfile.example
ca_certificates conf.d passwd pwfile.example
root@raspberrypi:~#
```

Figura 1. 2.- Ubicación del archivo de configuración de Mosquitto

Guardamos el fichero anterior como *mosquitto_backup.txt* y creamos una nueva versión de *mosquitto.conf*. Lo abrimos y añadimos las siguientes líneas:

```
allow_anonymous false  
listener 1883  
password_file /etc/mosquitto/passwd  
log_dest file /etc/mosquitto/mosquitto.log
```

Con las líneas anteriores le estamos indicando al broker que queremos realizar las comunicaciones en el puerto 1883 TCP y que no permita la conexión de clientes anónimos, es decir, aquellos cuyo usuario y contraseña no se encuentren en el fichero *passwd*

Aplicamos los cambios y salimos del fichero. Ahora es momento de establecer cuál es ese usuario y contraseña. Nosotros elegimos el usuario *utic* y la contraseña *123456*. Para ello se crea el fichero *passwd* y se introducen los datos como se muestra en la figura 1.3

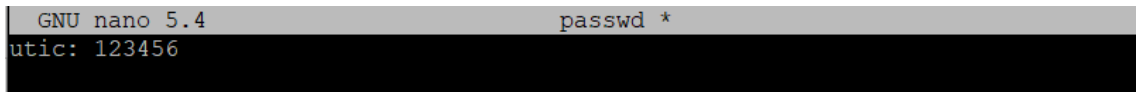
A screenshot of the GNU nano 5.4 text editor. The top status bar shows 'GNU nano 5.4' on the left and 'passwd *' on the right. The main editing area contains the text 'utic: 123456' on a single line.

Figura 1. 3.- Contenido del fichero *passwd*

Ejecutamos el siguiente comando para que Mosquitto encripte la contraseña:

```
sudo mosquitto_passwd -U passwd
```

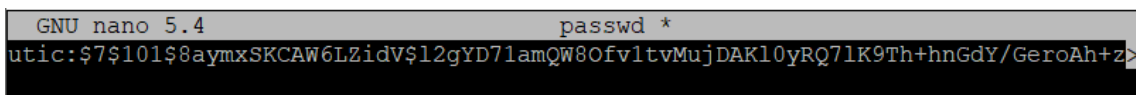
A screenshot of the GNU nano 5.4 text editor. The top status bar shows 'GNU nano 5.4' on the left and 'passwd *' on the right. The main editing area contains the text 'utic:\$7\$101\$8aymxSKCAW6LZidV\$12gYD71amQW8OfvltvMujDAKl0yRQ7lK9Th+hnGdY/GeroAh+z>' on a single line.

Figura 1. 4.- El fichero *passwd* con la contraseña encriptada

Para terminar reiniciamos el broker con el comando:

```
sudo systemctl restart mosquito
```

1.2.4.- Publicador, subscriber y broker

Antes de implementar los clientes MQTT en Python para el desarrollo del proyecto, es importante entender cómo funcionan el publicador, subscriber y broker.

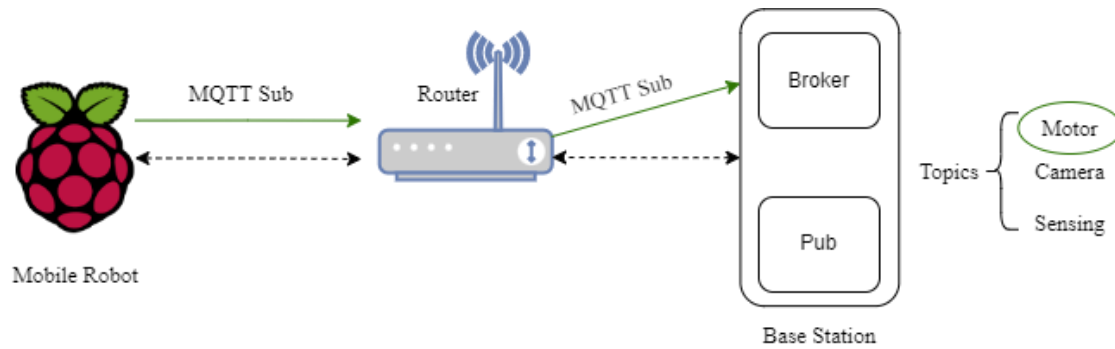


Figura 1. 5.- Proceso de subscripción en MQTT

En la figura 1.5 se ilustra cómo se suscribe el robot al tópico del motor en el broker de la estación base. La forma de hacerlo con mosquitto es a partir del siguiente comando, donde el parámetro *-h* indica el host, que en nuestro caso es la IP de la estación base y el parámetro *-t* indica en tópico, que será “Motor”:

mosquitto_sub -h 192.168.1.41 -t “Motor”

```
root@raspberrypi:~# mosquitto_sub -h 192.168.1.41 -t Motor
Error: Connection refused
```

Figura 1. 6.- Subscripción al broker sin credenciales

Si lo ejecutamos recibimos el *output* presentado en la figura 1.6, donde nos indica que la conexión ha sido rechazada. Esto ocurre porque a la hora de configurarlo indicamos que no queríamos conexiones anónimas. Con esto estamos demostrando que a priori, sólo podrían participar en nuestro ecosistema aquellos clientes que conozcan nuestras credenciales.

Para poder crear una instancia de suscriptor en un entorno MQTT con autenticación el comando varía un poco:

mosquitto_sub -h 192.168.1.41 -u utic -P 123456 -t “Motor”

```
root@raspberrypi:~# mosquitto_sub -h 192.168.1.41 -u utic -P 123456 -t "Motor"
```

Figura 1. 7.- Output de la instancia de suscriptor con credenciales

Se aprecia en la figura 1.7 que tras incluir las credenciales en el comando del suscriptor, mosquitto ya no nos rechaza la conexión. El siguiente paso es crear la instancia de publicador que actuará como indica la figura 1.8.

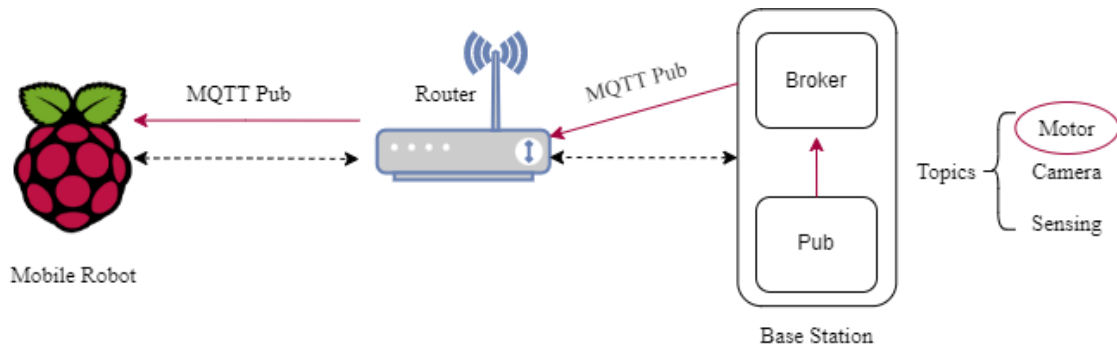


Figura 1. 8.- Proceso de publicación en MQTT

Para instanciar un publicador, el comando es muy similar. El parámetro *-m* indica el mensaje que se desea publicar en el tópico *-t*:

mosquitto_pub -h 192.168.1.41 -u utic -P 123456 -t "Motor" -m "forward"

```

root@raspberrypi:~# mosquitto_pub -h 192.168.1.41 -u utic -P 123456 -t "Motor" -m "forward"
root@raspberrypi:~# mosquitto_pub -h 192.168.1.41 -u utic -P 123456 -t "Motor" -m "backward"
root@raspberrypi:~# mosquitto_pub -h 192.168.1.41 -u utic -P 123456 -t "Motor" -m "left"
root@raspberrypi:~# mosquitto_pub -h 192.168.1.41 -u utic -P 123456 -t "Motor" -m "right"
  
```

Figura 1. 9.- Publicación de los mensajes "forward", "backward", "left" y "right" en el tópico "Motor"

```

root@raspberrypi:~# mosquitto_sub -h 192.168.1.41 -u utic -P 123456 -t "Motor"
forward
backward
left
right
  
```

Figura 1. 10.- Suscriptor MQTT en el tópico "Motor" tras la publicación de varios mensajes

Si nos fijamos en la interfaz del subscriptor (figura 1.10) ha recibido correctamente los comandos indicados por el publicador. De este modo hemos comprobado que la comunicación entre subscriptores y publicadores se realiza de forma exitosa.

1.2.5.- Implementación en Python

El software encargado de realizar las comunicaciones entre los diversos elementos del escenario utilizando el protocolo MQTT se ha escrito en Python utilizando la librería paho-mqtt

El robot ejercerá de publicador en los siguientes tópicos:

- *housekeeping*: tópico usado para enviar a la estación base los datos obtenidos por los sensores de distancia, el acelerómetro, el giroscopio y el sensor de temperatura.

- *MR_status*: tópico usado para notificar a la estación base qué función se encuentra realizando (por ejemplo si está capturando video o calculando la distancia a la misma)
- *emergency*: tópico usado para enviar un mensaje de emergencia.

Para lograr la publicación en cualquiera de los tópicos anteriores se ha escrito una función llamada *publish()*. El código se puede encontrar en el fichero *main.py* ubicado en el directorio *MobileRobot* dentro del repositorio Github del proyecto

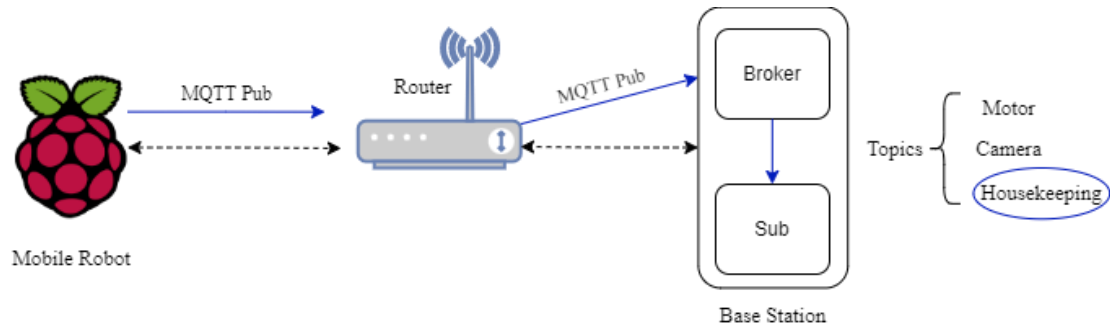


Figura 1. 11.- Funcionamiento general de la publicación de mensajes en el tópico “housekeeping”

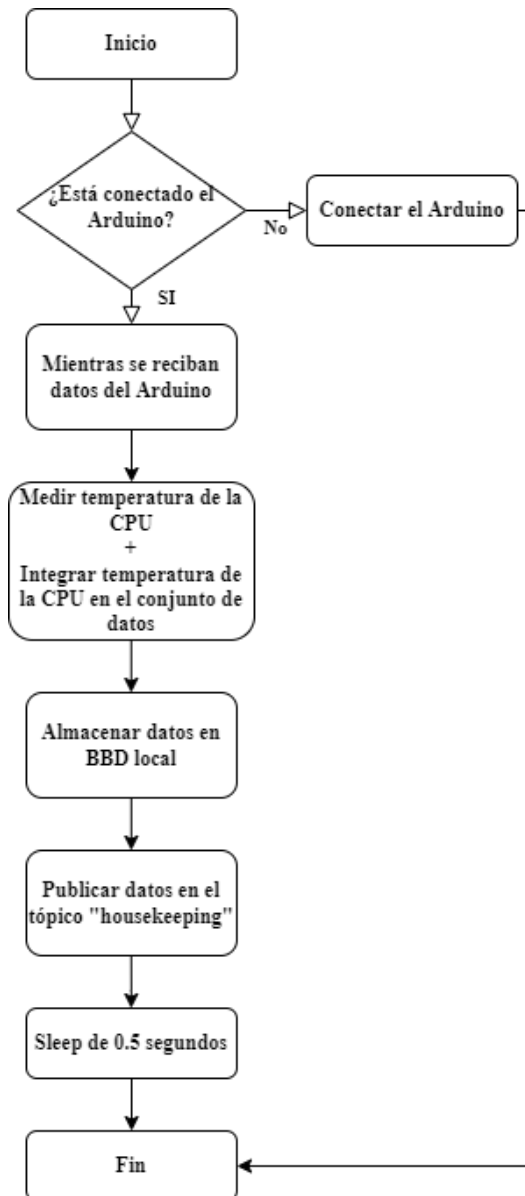


Figura 1. .- Diagrama de flujo de la función `publish()` del robot.

Por el contrario, el robot ejercerá de suscriptor cuando reciba órdenes para mover sus motores, capturar imagen, video y para estimar la distancia a la estación base. Un ejemplo de cómo es el proceso de la comunicación en este caso se puede apreciar en la figura 1.5 Los tópicos a los que se suscribe el robot son los siguientes:

- *motor*: aquí llegan los mensajes enviados desde la estación base y dispositivos móviles para mover el robot según indique el usuario
- *camera*: a este tópico le llegan los mensajes enviados desde la estación base y dispositivos móviles que accionarán la cámara para realizar una fotografía, grabar un video o iniciar una emisión en directo.
- *BS2gnuradio*: a este tópico llegan los mensajes que accionarán el bloque de SDR para posicionar el robot hasta que tenga visión directa con la estación base y posteriormente calcular la distancia que lo separa de ella.

La función encargada de realizar la suscripción a los tópicos anteriores se llama *subscribe()* y puede encontrarse en el mismo fichero.

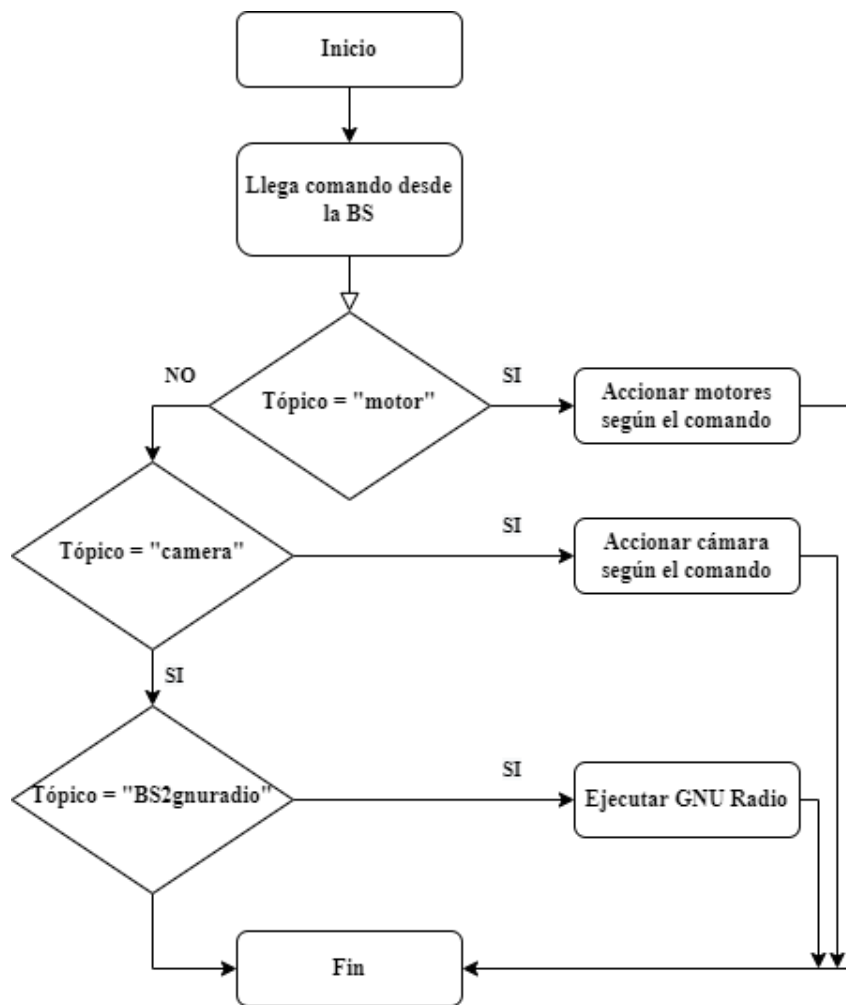


Figura 1. 13.- Diagrama de flujo de la función *subscribe()* del robot

La estación base tiene un comportamiento similar, ejerce de publicador en ciertos escenarios y de subscriptor en otros. Los tópicos en los que publica son aquellos en los que se suscribe el robot mientras que los tópicos a en los que se suscribe son aquellos en los que éste publica. Como el software diseñado para la estación base goza de una interfaz de usuario con botones (figura 1.14), la función *publish()* se llama cuando uno de éstos es pulsado mientras que la función *subscribe()* está continuamente escuchando para mostrar por pantalla los datos recogidos por los sensores del robot.

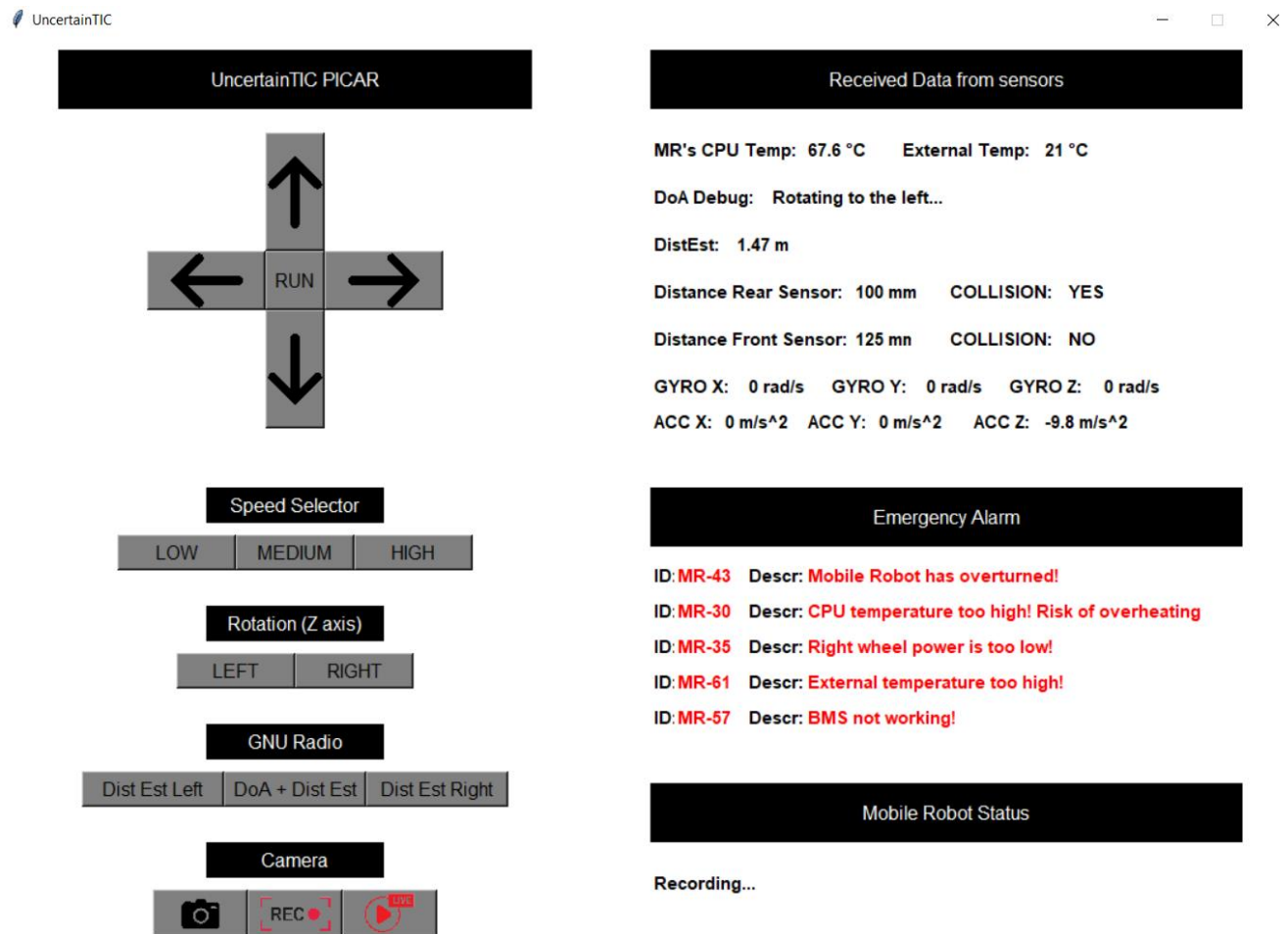


Figura 1. 14.- Interfaz de usuario de la estación base

En la figura 1.14 se pueden apreciar varias secciones:

- Parte izquierda: aquí se encuentra el mando de control para pilotar el robot (sentido de movimiento, velocidad, rotación, ejecución de SDR y uso de la cámara). Cada uno de estos botones tiene asociado un comando distinto que se publica en el tópico adecuado.
- Parte superior derecha: aquí se encuentra el *dashboard* que muestra los datos recogidos por el robot en tiempo real.
- Parte central derecha: aquí se recogen las alarmas de emergencia enviadas por los robots. Se muestra su ID y la descripción de la alerta.
- Parte inferior derecha: aquí se muestra el estado actual del robot que se está controlando. Indica si ha hecho fotos, vídeos, streaming...

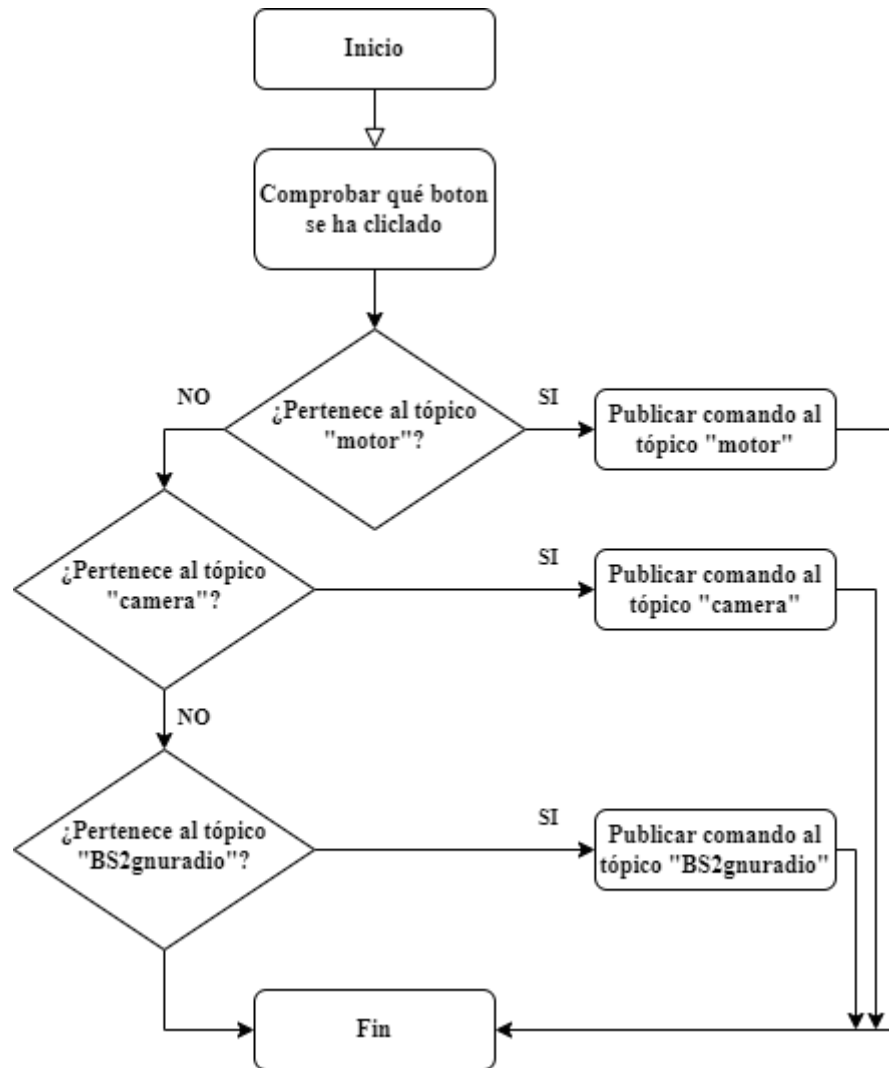


Figura 1. 15.- Diagrama de flujo de la función publish() de la estación base

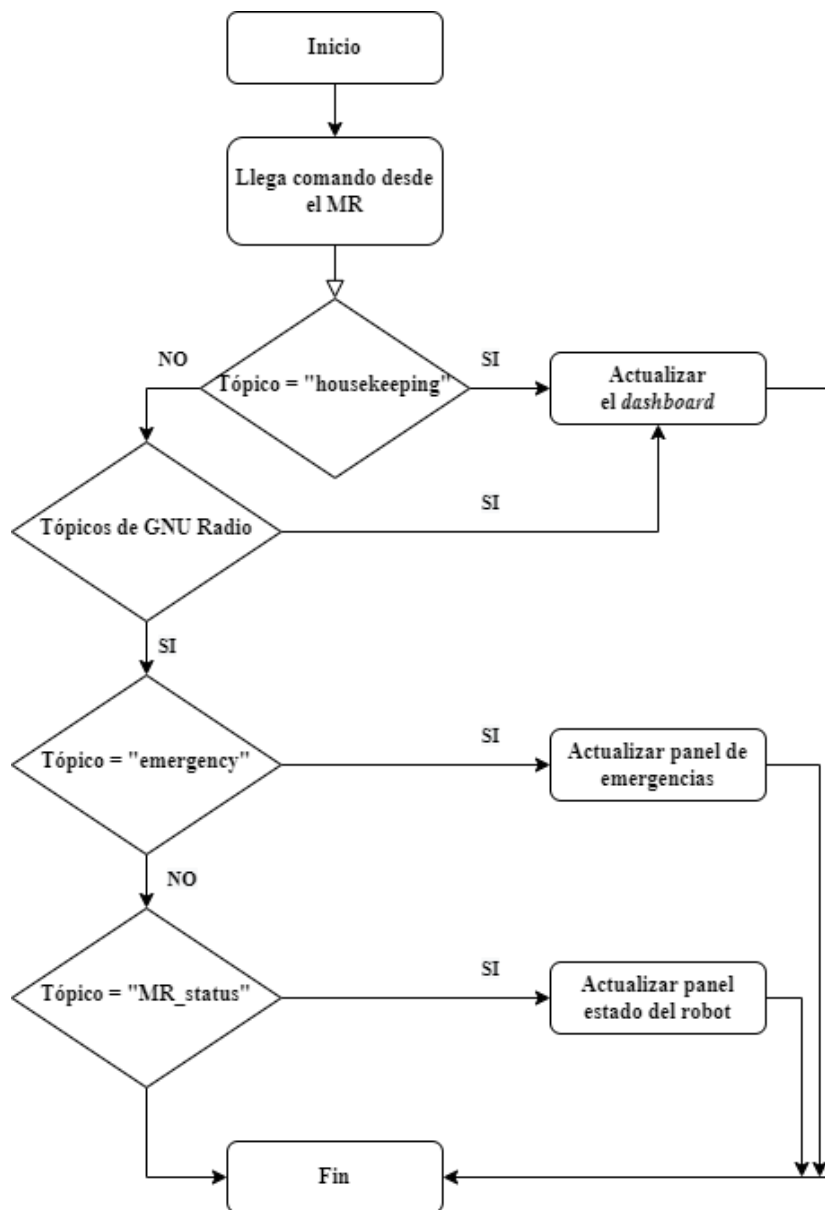


Figura 1. 16.- Diagrama de flujo de la función subscribe() de la estación base

2.- Visualización de datos en tiempo real

En este apartado se explican los pasos necesarios para realizar la solución propuesta de visualización de datos mediante una plataforma de IoT. La solución empleada se basa en parte del contenido del laboratorio la signatura de integración de servicios telemáticos: Azure y PowerBI.

2.1.- Esquema funcional

En este apartado se realizará un esquema del servicio propuesto explicando tanto el hardware (HR) como el software (SW) empleado en cada paso.

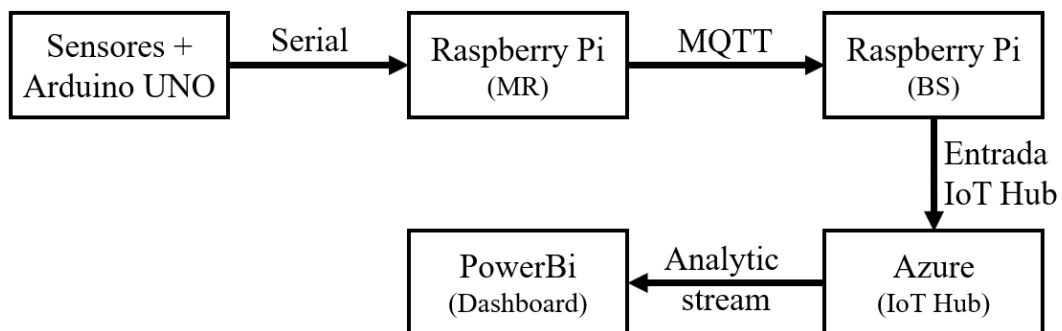


Figura 2.1.- Esquema del sistema IoT de representación

Además de los elementos que se pueden ver en la figura 2.1 en ambas Raspberry Pi se ha creado un sistema de almacenamiento basado en bases de datos.

Los sensores empleados son dos sensores de distancia, un sensor de temperatura y una IMU. Además, en el Robot Móvil (MR) se añade la temperatura de la CPU.

2.2.- Formato datos

Para transmitir los datos en los dos primeros saltos se ha empleado un formato propio que será explicado a continuación, mientras que para enviarlos al centro de IoT se deben convertir en JSON.

El formato propio diseñado consta de seis campos separados mediante puntos y comas. Cada campo está formado por dos o tres subcampos en función del número de elementos medidos, dando lugar al siguiente formato:

Timestamp;Acelerómetro;Giroscopio;Distancia;Colisión detectada;Temperatura

Los distintos subcampos son:

- Timestamp → horas minutos segundos
- Acelerómetro [m/s²] → Valor_Eje_X Valor_Eje_Y Valor_Eje_Z
- Giroscopio [rad/s] → Valor_Eje_X Valor_Eje_Y Valor_Eje_Z
- Distancia [mm] → Distancia_Trasera Distancia_Delanterá
- Colisión detectada [bool] → Colisión_Trasera Colisión_Delanterá
- Temperatura [°C] → Ambiental_MR CPU_MR

2.3.- Implementación

2.3.1.- Código Arduino

El código empleado es el mismo que se emplearía para obtener por puerto serial los resultados de las medidas a través de un ordenador. Una vez que se tiene un código funcional y depurado se cargará desde un ordenador de manera que cuando se quiera conectar al MR únicamente haya que conectarlo a la Raspberry.

En caso de que no se inicie el programa de manera adecuada al conectar el Arduino a la Raspberry se debe reiniciar el Arduino Uno en el botón rojo de la esquina.

Código para el Arduino: ToF_Distance_Ranging_Sensor_imu_temperature.ino

2.3.2.- Conexión Arduino – MR

Esta conexión se realiza mediante una interfaz serial. La parte del Arduino, como ya se ha comentado, es igual a la que sería necesaria para conectarlo a un ordenador.

En el MR se debe seleccionar el puerto al cual se conecta el Arduino:

```
ser = serial.Serial(PUERTO, 9600, timeout=1)
```

Una vez recibidas y procesadas las líneas se guardan en un sistema de archivos y se envían a la Estación Base (BS). Es importante comentar que con cada nueva misión -ejecución del código-, se borrarán los históricos de la misión anterior.

Código: main_MR.py líneas 49-50 y 469-505

2.3.3.- Conexión MR – BS

Esta comunicación se realiza mediante MQTT siendo el MR el publicador en el tópico “housekeeping” y el BS el suscriptor.

Para realizarla hay que haber creado previamente un *broker* MQTT, que se ha alojado en la Raspberry Pi empleada como BS tal y como se ha explicado en el apartado 1

Código: main_MR.py

2.3.4.- Implementación BS

La estación base cumple tres funciones principales con respecto al tratamiento de los datos. En primer lugar, los recibe desde el MR y realiza el procesado necesario, una vez procesados los guarda en un sistema de archivos y enviarlos al Hub IoT.

Para esta parte el suscriptor MQTT se encuentra implementado en el archivo: BS_IoT.py en las líneas 34-145, dentro de un hilo. Es importante comentar que, aunque se ha usado el mismo sistema de hilos que para la GUI, se ha decidido realizar en un archivo diferente duplicando el suscriptor pues la librería “tkinter” es thread sensitive.

La parte del código encargada de enviar datos al Hub IoT se implementa dentro de la función “`iothub_client_telemetry_sample_run`” al igual que el almacenamiento de los datos en el sistema de archivos

Código: BS_IoT.py líneas 34-145

2.3.5.- Conexión BS – Azure

Esta conexión se especifica en el documento indicado en el apartado 1.3.4 en las líneas 148-232. Además, se debe implementar el extremo de Azure a través de su página web.

En la Raspberry Py que contiene el MR se deben instalar los siguientes paquetes a fin de poder conectar con Azure.

`sudo pip3 install azure-iot-device`

`sudo pip3 install azure-iot-hub`

Una vez instalados los paquetes necesarios se debe crear un Hub dentro de Azure, para ello se recomienda seguir las instrucciones del tutorial [1].

Este paso se puede empleando el cloud shell que proporciona Azure. Para ello se debe ejecutar el comando (este paso solo es necesario realizarlo una vez):

`az extension add --name azure-cli-iot-ext`

Una vez que se haya instalado la extensión necesaria se puede monitorizar el tráfico entrante a un Hub IoT determinado a través de un dispositivo concreto mediante el comando:

`az iot hub monitor-events --hub-name HUB_NAME --device-id DEVICE_NAME`

Código: BS_IoT.py líneas 148-232

2.3.6.- Conexión Azure - PowerBI

Una vez que se ha comprobado que los mensajes enviados llegan correctamente a Azure se debe crear un siguiendo las instrucciones del tutorial [2],

Cuando se reciben los datos en PowerB ya se puede crear un *dashboard* para representar los datos recibidos. En este trabajo se han realizado dos diferentes uno que representa los datos en tiempo real y un segundo que representa el histórico de la última misión.

Referencias

[1] Conectar el simulador en línea de Raspberry Pi a Azure IoT Hub (Node.js)
<https://learn.microsoft.com/es-es/azure/iot-hub/iot-hub-raspberry-pi-web-simulator-get-started>

[2] Tutorial: Visualización de datos de sensor en tiempo real desde Azure IoT Hub mediante Power BI <https://learn.microsoft.com/es-es/azure/iot-hub/iot-hub-live-data-visualization-in-power-bi>