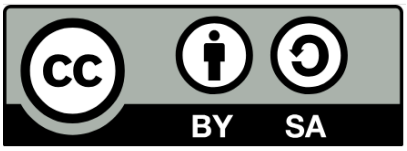
Universidad
Rey Juan Carlos

Grado de Ingeniería en Robótica Software

Sistema empotrados y de tiempo real

Práctica 1

Multihilo y aproximación a tareas esporádicas en C



En esta práctica se trabajarán conceptos de multithreading, deadlines, tiempos de coste en C utilizando pthreads.

Objetivos:

- Repaso de C y pthreads
- Ver la diferencia de ejecutar: Un proceso multihilo en un sistema monocore o multicore
- Comprobar prácticamente que Linux no es un sistema de tiempo real
- No podemos decidir cuando las tareas empiezan o terminan.
- Demostrar que no se pueden asegurar los plazos de ejecución.

Desarrolla un programa (practica1.c) que realice la siguiente funcionalidad.

- El programa debe crear 4 threads que se ejecuten a la vez y no secuencialmente.
- Cada thread es una tarea periódica con C=**0.5** segundos y P=**0.9** segundos (asumimos que el deadline y periodo son iguales).
- Los propios threads son los encargados de estimar el tiempo utilizado en su ejecución y controlar la frecuencia a la que se ejecutan.
- Cada thread debe ejecutar 5 iteraciones, teniendo en cuenta que son iteraciones periódicas (P=0.9).
- Para los cálculos de tiempo utiliza clock_gettime y timespec (mira páginas de manual).
- En cada iteración del thread debes asegurarte que está ejecutando un código random durante 0.5 segundos. Puedes utilizar el siguiente código, pero asegurate que su ejecución es aproximadamente de 0.5 segundos.

  ```
  volatile unsigned long long j;
  for (j=0; j < 400000000ULL; j++);
  ```

- El programa debe mostrar una línea por cada iteración y thread ejecutado con la siguiente información por pantalla (thread e iteraciones empiezan en 1 siempre).

  ```
  [1663147910.736590668] Thread 1 - Iteracion 1: Coste=0.50 s.
  ```

- Si la restricción temporal no se cumplido debe mostrar lo siguiente:

  ```
  [1663147910.736590668] Thread 1 - Iteracion 1: Coste=1.5 s. (fallo temporal)
  ```

- La ejecución del programa debe realizarse en 4 escenarios distintos
  - Ejecución multicore
  - Ejecución monocore
  - Ejecución multicore + stress (todas las cpus)
  - Ejecución monocore + stress (todas las cpus)

- Para ejecutar procesos utilizando una configuración multicore o monocore puedes utilizar el comando taskset
    - Multicore:
        - **`./practica1`**
    - Monocore:
        - **`taskset -c 0 ./practica1`**

- Comandos útiles para la práctica
    - htop: visualizador de procesos en linux
    - stress: genera stress computacional en las cpus
    - time: calcula el tiempo de ejecución de un proceso

PREGUNTAS:

1. Ejecutar los siguientes casos y justifica su comportamiento:
    a. ./practica1 (multicore)
    b. ./practica1 (monocore)
    c. ./practica1 (monocore) + *stress*
    d. ./practica1 (multicore) + *stress*

2. ¿En qué casos de ejecución (nombrados anteriormente) el sistema es capaz de cumplir las restricciones temporales (tanto tiempo de cómputo como periodicidad)?
3. ¿Qué número mínimo de cpus se necesitan para que tu programa ejecute correctamente sin fallos de restricciones temporales? Usa el comando taskset para comprobarlo.
4. ¿Qué solución se podría proponer para cumplir plazos estrictos temporales de periodicidad en la ejecución de los *threads* SIN cambiar la configuración actual que tienen los ordenadores del laboratorio?

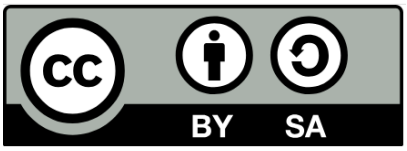
Universidad
Rey Juan Carlos

Grado de Ingeniería en Robótica Software

Sistema empotrados y de tiempo real

Práctica 2

Cácuulo de latencias en sistemas RT y no-RT



En esta segunda práctica pondremos en práctica los conceptos adquiridos en clase de teoría sobre cálculo de latencias y sistemas de tiempo real.

1. Análisis con cyclictest.

Utiliza la utilidad cyclictest para medir la latencia en los laboratorios de la universidad, tanto en ordenadores con kernel normal como en ordenadores con el kernel de RT. Utiliza el siguiente comando, pero **asegúrate** de mirar la página de manual para entender los parámetros utilizados.

```
$ sudo cyclictest -p99 -N --smp -D 60
```

La medición de la tenencia siempre se debe realizar en 3 escenarios distintos:

- S1: idle
- S2: Estresando el planificador (hackbench)
  - `sudo hackbench -l 1000000 -s 1000 -T `nproc``
- S3: Estresando las operaciones I/O (bonnie++)
  - `bonnie++ -d /tmp/ -D -r 2048 -f -b -u $USER -c `nproc``

Además, utilizaremos 3 configuraciones distintas de hardware y kernel para todas las pruebas:

- Laboratorios f-l3109: kernel no-RT (accede por ssh)
- RaspberryPi: kernel no-RT
- RaspberryPi: kernel RT

RBPi Kernel no-RT	**RBPi Kernel RT**
rbpi-01: 10.110.128.64	rbpi-10: 10.110.128.73
rbpi-02: 10.110.128.65	rbpi-11: 10.110.128.74
rbpi-03: 10.110.128.66	rbpi-12: 10.110.128.75
rbpi-04: 10.110.128.67	rbpi-13: 10.110.128.76
rbpi-05: 10.110.128.68	rbpi-14: 10.110.128.77

Para acceder a las RBPI, que tienen IPs privadas, debes hacerlo siempre desde ordenadores de la universidad. Puedes utilizar tu mismo usuario que en los laboratorios. Además, tendrás tu HOME montado igualmente.

Calcula las latencias máximas y medias en los 3 escenarios descritos arriba (S1, S2, S3) para las 3 configuraciones de hardware y kernel disponibles. Muestra los resultados obtenidos en la siguiente tabla (si lo prefieres puedes usar gráficos o plots de barras mientras la información quede correctamente mostrada).

cyclictestURJC						
	Laboratorios		RaspberryPi			
	Kernel NO RT		Kernel NO RT		Kernel RT	
	Latencia media (ns)	Latencia max (ns)	Latencia media (ns)	Latencia max (ns)	Latencia media (ns)	Latencia max (ns)
S1						
S2						
S3						

2. Desarrollo de cyclictestURJC.

Desarrolla una aplicación en C llamada *cyclictestURJC* que mida la latencia de planificación de un sistema GNU/Linux utilizando las técnicas explicadas en clase. cyclictestURJC debe ejecutar continuamente durante 1 minuto y debe realizar la siguiente funcionalidad:

- Generar un thread por cada core del ordenador. Para saber el número de cores, puedes utilizar la siguiente función:
  - `int N = (int) sysconf(_SC_NPROCESSORS_ONLN);`
- Cada thread debe ejecutarse en la cola de prioridad SCHED_FIFO, con prioridad 99.
  - `pthread_setschedparam`
- Cada thread debe ejecutarse en un core diferente, por tanto, establece la afinidad con
  - `pthread_setaffinity_np`
- El thread debe estar constantemente calculando la latencia de planificación mediante técnicas de sleep. **RECOMENDACIÓN**: realiza esperas del orden de milisegundos.
- Para configurar Linux con la mínima latencia en DMA, asegúrate que tu programa siempre comienza abriendo el fichero `/dev/cpu_dma_latency` y escribiendo un 0. **NO** cierres el descriptor hasta el final del programa (si lo cierras, restablecerá la configuración previa).

```
static int32_t latency_target_value = 0;
latency_target_fd = open("/dev/cpu_dma_latency", O_RDWR);
write(latency_target_fd, &latency_target_value, 4);
```

- Al finalizar el programa (después de 1 minuto de ejecución), debe mostrar el siguiente resumen con la latencia media y latencia máxima por thread/cpu.

```
$ ./cyclictestURJC
[0]    latencia media = 000001984 ns. | max = 000016865 ns
[1]    latencia media = 000001984 ns. | max = 000017124 ns
[2]    latencia media = 000001999 ns. | max = 000014698 ns
[4]    latencia media = 000001995 ns. | max = 000016550 ns
[3]    latencia media = 000002011 ns. | max = 000014936 ns
[5]    latencia media = 000002005 ns. | max = 000014396 ns

Total  latencia media = 000001996 ns. | max = 000017124 ns
```

- Además, tu programa debe guardar en un fichero CSV (cyclictestURJC.csv) en formato texto plano (https://en.wikipedia.org/wiki/Comma-separated_values) todas las medidas obtenidas en el siguiente formato:

CPU, NUMERO_ITERACION, LATENCIA

MUY IMPORTANTE: Asegúrate de realizar esta operación de escritura a fichero, una vez terminada toda la evaluación de la latencia, para no interferir con las interrupciones al escribir.

MUY IMPORTANTE: Asegúrate que cuando realices las mediciones en las RBPi, no haya ningún otro estudiante realizando las mediciones también. Obtendréis resultados erróneos.

Utilizando la herramienta cyclictestURJC que has desarrollado, rellena la siguiente tabla del mismo modo que hiciste en el apartado 1.

cyclictestURJC						
	Laboratorios		RaspberryPi			
	Kernel NO RT		Kernel NO RT		Kernel RT	
	Latencia media (ns)	Latencia max (ns)	Latencia media (ns)	Latencia max (ns)	Latencia media (ns)	Latencia max (ns)
S1						
S2						
S3						

Por último, crea un gráfico que muestre la distribución de la latencia para los distintos escenarios en Raspberry Pi. Deberás incluir los 2 tipos de kernel (real-time y no real-time) y los 3 escenarios definidos (idle, hackbench, etc.). Asegúrate de que todos los gráficos de latencia utilicen el mismo rango en el eje X, expresado en microsegundos, para facilitar la comparación entre ellos. Los datos deben provenir de los archivos CSV generados para cada escenario.

En cada gráfico, que deben aparecer las leyendas correctamente identificando los escenarios, incluye la desviación estándar calculada para cada escenario. Puedes ponerlo en la propia leyenda del plot.

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.hist.html

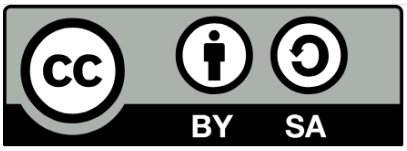
Universidad
Rey Juan Carlos

Grado de Ingeniería en Robótica Software

Sistema empotrados y de tiempo real

Práctica 3

Controlador Máquina Expendedora desarrollado en Arduino



Descripción de la práctica

Se busca diseñar e implementar un controlador para una máquina expendedora que esté basado en Arduino UNO y en los sensores/actuadores que se proporcionan en el kit Arduino. La práctica tendrá que integrar obligatoriamente los siguientes componentes hardware:

- Arduino UNO
- LCD
- Joystick
- Sensor temperatura/Humedad DHT11
- Sensor Ultrasonido
- Boton
- 2 LEDS Normales (LED1, LED2)

Funcionalidad

La práctica debe implementar la siguiente funcionalidad software.

1. Arranque:
    a. Al inicio del sistema, el LED1 debe parpadear 3 veces a intervalos de 1 segundo. Al mismo tiempo debe mostrarse el mensaje “CARGANDO …” en el LCD. Al cabo de los 3 parpadeos el LED1 debe apagarse y la pantalla debe mostrar la información de la funcionalidad “Servicio”.
2. Servicio.
    a. Si el usuario se encuentra a menos de 1 metro de la máquina se debe pasar al estado b). En caso contrario el LCD debe mostrar “ESPERANDO CLIENTE”.
    b. El LCD debe mostrar la temperatura y humedad durante 5 segundos y acto seguido deberá mostrar la lista de productos que el usuario puede seleccionar. Los productos y precios a mostrar son:
        i. Cafe Solo 1€
        ii. Cafe Cortado 1.10 €
        iii. Cafe Doble 1.25 €
        iv. Cafe Premium 1.50 €
        v. Chocolate 2.00 €

Debes permitir la navegación por esa lista utilizando el joystick (arriba / abajo). Y para su selección debes usar el switch del propio joystick. Una vez seleccionado debes mostrar el mensaje “Preparando Cafe …” durante un tiempo aleatorio entre 4 y 8 segundos. Cada ejecución puede ser un tiempo distinto (debes hacerlo aleatorio). Usa ese mismo tiempo para hacer que el LED2 se encienda de manera incremental, de tal manera que la intensidad del LED2 indica igualmente el progreso de la preparación del café. Una vez terminada la preparación del cafe, el LDC debe mostrar “RETIRE BEBIDA” durante 3 segundos y volver a la funcionalidad inicial de Servicio.

En cualquier momento del estado b), el usuario puede reiniciar el estado (no la placa) si pulsa el botón durante el rango 2-3 segundos. Por lo que deberá ejecutar de nuevo la funcionalidad de Servicio.

3. Admin
    a. Es posible acceder a la interfaz de administración de la máquina en cualquier momento. Para ello el usuario debe presionar el botón durante no menos de 5 segundos.
    b. Mientras el usuario esté en la vista de Admin, ambos LEDS deben estar encendidos.
    c. El siguiente menú debe ser mostrado en el LCD:
        i. Ver temperatura
        ii. Ver distancia sensor
        iii. Ver contador
        iv. Modificar precios

    Debes permitir la navegación por esa lista utilizando el joystick (arriba / abajo). Y para su selección debes usar el switch del propio joystick. Además debes permitir volver al menú utilizando el joystick (movimiento izquierda). A continuación se detalla lo que debe mostrar cada menú:
    - i) Temp: XX ºC Hum: YY % (debe cambiar dinámicamente)
    - ii) Distancia: XX cm (debe cambiar dinámicamente)
    - iii) Tienes que llevar un contador en segundos desde que la placa está arrancada. Ese contador en segundos es el que se muestra en este menú. Mientras estás en esa pantalla se tiene que observar como el contador va incrementando con el paso de los segundos.
    - iv) Debes mostrar el listado de productos y permitir el cambio de precio utilizando el joystick. Los incrementos o decrementos se realizan en 5 céntimos (utiliza el joystick arriba y abajo para cambiar el precio). Para confirmar el valor del precio utiliza el switch del joystick y para cancelar y volver a la lista de precio utiliza el movimiento “izquierda” del joystick. Ahora si vas a la lista ofrecida en la funcionalidad 2a) deberías ver los precios actualizados. (Los precios no persisten si la placa se reinicia).

    d. Para salir de la vista admin se debe pulsar de nuevo el pulsador durante no menos de 5 segundos, volviendo a la funcionalidad de Servicio.

Recomendaciones

- Haz uso de todas las librerías y técnicas vistas en clase.
- Mantén tu código seguro utilizando el watchdog para evitar bloqueos.

Documentación:

- https://www.arduino.cc/reference/en/

Bibliotecas utilizadas:

- LiquidCrystal: https://www.arduino.cc/en/Reference/LiquidCrystal
- ArduinoThread: https://www.arduino.cc/reference/en/libraries/arduinothread/
- Watch Dog: https://create.arduino.cc/projecthub/rafitc/what-is-watchdog-timer-fffe20
- DHT-sensor-library: https://github.com/adafruit/DHT-sensor-
- TimerOne: https://www.arduino.cc/reference/en/libraries/timerone/

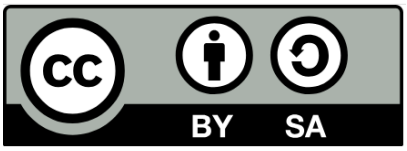
Universidad
Rey Juan Carlos

Grado de Ingeniería en Robótica Software

Sistema empotrados y de tiempo real

Práctica 4

Sigue Linea en Arduino basado en sistemas RT



Sumario

1. Descripción..... 3
2. Esquema..... 4
3. Montaje..... 6
4. Arduino..... 6
    4.1 Ultrasonido..... 6
    4.2 Sensor Infra-rojo..... 6
    4.3 Motores..... 6
    4.4 LED..... 7
5. ESP32..... 7
    5.1 Arduino-IDE y librerías..... 7
    5.2 Comprobar la conexión WiFi..... 9
6. Comunicación Serie..... 9
7 Comunicación IoT..... 10
    7.1 MQTT..... 10
    7.2 Mensajes..... 11
        Mensaje de inicio de vuelta..... 11
        Mensaje de Fin de vuelta..... 11
        Mensaje Obstáculo Detectado..... 12
        Mensaje Línea Perdida..... 12
        Mensajes PING..... 12
        Mensaje Inicio Búsqueda de Linea (OPCIONAL)..... 12
        Mensaje Fin Búsqueda de Linea (OPCIONAL)..... 13
        Mensaje Línea Encontrada (OPCIONAL)..... 13
        Mensaje Estadísticas Línea Visible (OPCIONAL)..... 13
8 Evaluación..... 14
    8.1 Día de Test y Día de Examen..... 14
    8.2 Bonificaciones y Penalizaciones..... 14
    8.3 Nota Final..... 14
9. Otros recursos..... 15

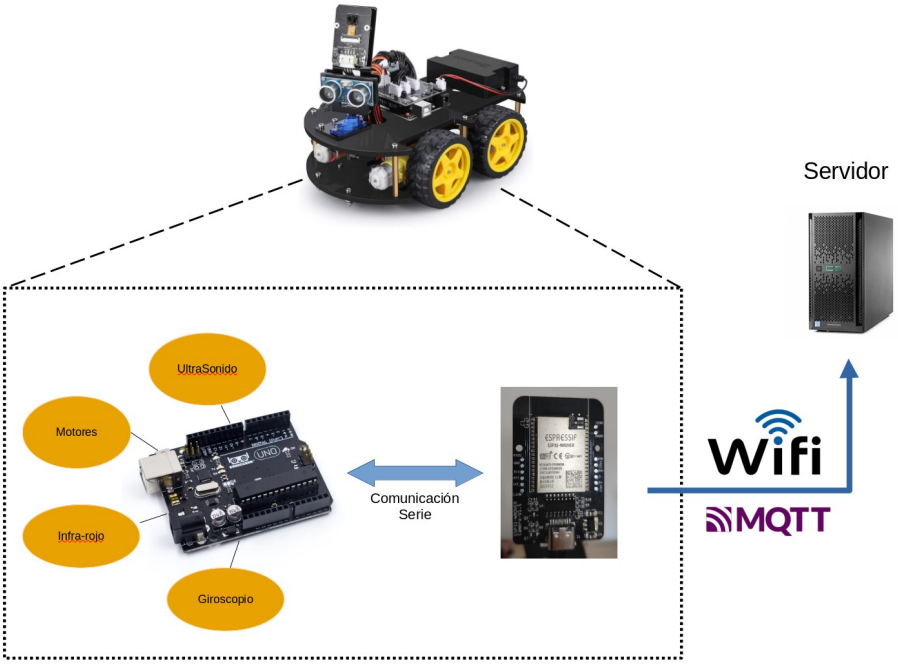
1. Descripción

La práctica se realizará por grupos de 2 estudiantes. Se utilizará el kit de Smart Robot Car Kit v4.0 compuesto por un arduino uno y un ESP32 camera

La evaluación y objetivos de esta práctica son:

- Sigue la línea lo más rápido posible sin salirse.
- Comunicación IoT a través de MQTT
- Comunicación serie entre ESP32 y Arduino UNO
- Si el robot pierde la línea se permite realizar una búsqueda de la línea de nuevo.
- Detección de obstáculos

2. Esquema



3. Montaje

Consulta el manual que viene con el kit y el siguiente vídeo para realizar el montaje correctamente. Es el primer paso de la práctica.

4. Arduino

El robot incorpora un Ardunio UNO que actúa de cerebro del sistema controlando todos los sensores y actuadores.

4.1 Ultrasonido

Los sensores de ultrasonido los puedes controlar uitlizando los siguientes pines.

```
#define TRIG_PIN 13
#define ECHO_PIN 12
```

4.2 Sensor Infra-rojo

Los sensores infra-rojos, son sensores analógicos. Usa toda su potencia y rango continuo de valores. **No los utilices como sensores digitales**

```
#define PIN_ITR20001-LEFT   A2
#define PIN_ITR20001-MIDDLE A1
#define PIN_ITR20001-RIGHT  A0
```

4.3 Motores

El Smart Robot Car incorpora 4 motores DC que se controlan a través de una pequeña placa controladora incorporada en la placa de extensión. Las velocidades las comandaremos por grupos de motores (los del lado derecho, y los del lado izquierdo). A continuación se muestran los pines de control, velocidad y dirección.

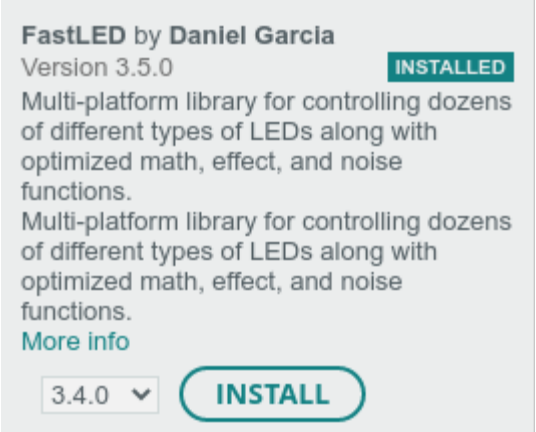
```
// Enable/Disable motor control.
//  HIGH: motor control enabled
//  LOW: motor control disabled
#define PIN_Motor_STBY 3

// Group A Motors (Right Side)
// PIN_Motor_AIN_1: Digital output. HIGH: Forward, LOW: Backward
#define PIN_Motor_AIN_1 7
// PIN_Motor_PWMA: Analog output [0-255]. It provides speed.
#define PIN_Motor_PWMA 5

// Group B Motors (Left Side)
// PIN_Motor_BIN_1: Digital output. HIGH: Forward, LOW: Backward
#define PIN_Motor_BIN_1 8
// PIN_Motor_PWMB: Analog output [0-255]. It provides speed.
#define PIN_Motor_PWMB 6
```

4.4 LED

La placa de expansión contiene un LED RGB que podemos utilizar para plasmar colores según los estados de tu comportamiento (muy útil para depuración en el circuito).

IMPORTANTE: Es obligatorio que siempre que tu robot detecte la línea (con cualquiera de los 3 sensores), debe mostrar el color verde en el LED. Si no detecta la línea, debe mostrar el LED en rojo.

Es necesario instalar la siguiente librería (FastLED):



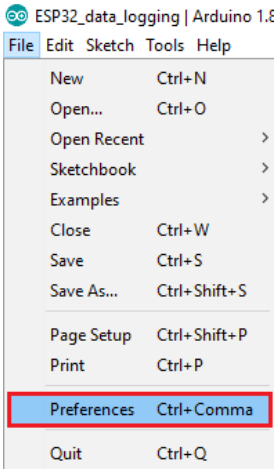
Puedes consultar el siguiente código de ejemplo para entender el funcionamiento DemoLed

5. ESP32

5.1 Arduino-IDE y librerías

El robot incluye un modelo ESP32 CAM, que nos permite comunicarnos a través de cualquier red WiFi. Utilizaremos Arduino-IDE para programarlo, pero es necesario realizar las siguientes acciones:

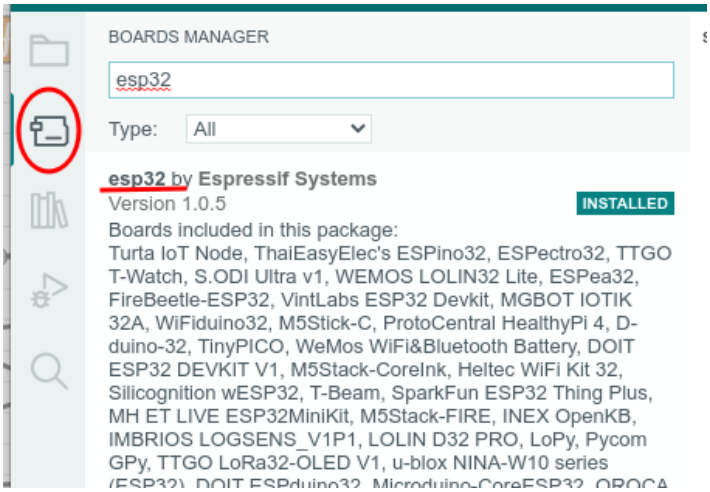
- Primero, añade el repositorio para ESP32 dentro de tu Arduino IDE



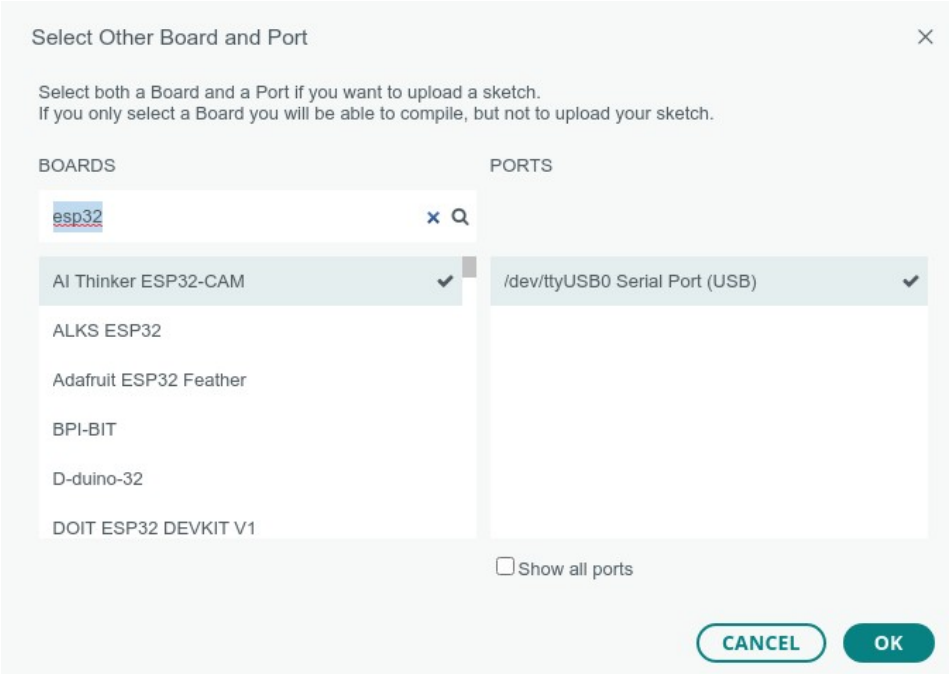
Introduce la siguiente URL en "Additional Board Manager URLs"

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

Asegurate que tienes instalado el paquete ESP32 dentro de "Boards Manager" en tu arduino IDE



Asegurate de configurar correctamente el modelo de placa "AI Thinker ESP32-CAM"



Por último asegurate que en los laboratorios, te aparecen 2 puertos detectados (ttyS4 y ttyUSB0). Utiliza éste último para realizar la programación del ESP32.

5.2 Comprobar la conexión WiFi

- Puedes comprobar la conexión WiFi (ESP) y la IP asignada. Para ello utilizaremos la red wifi eduroam. Puedes utilizar de base el código de ejemplo de **eduroam** del repositorio.

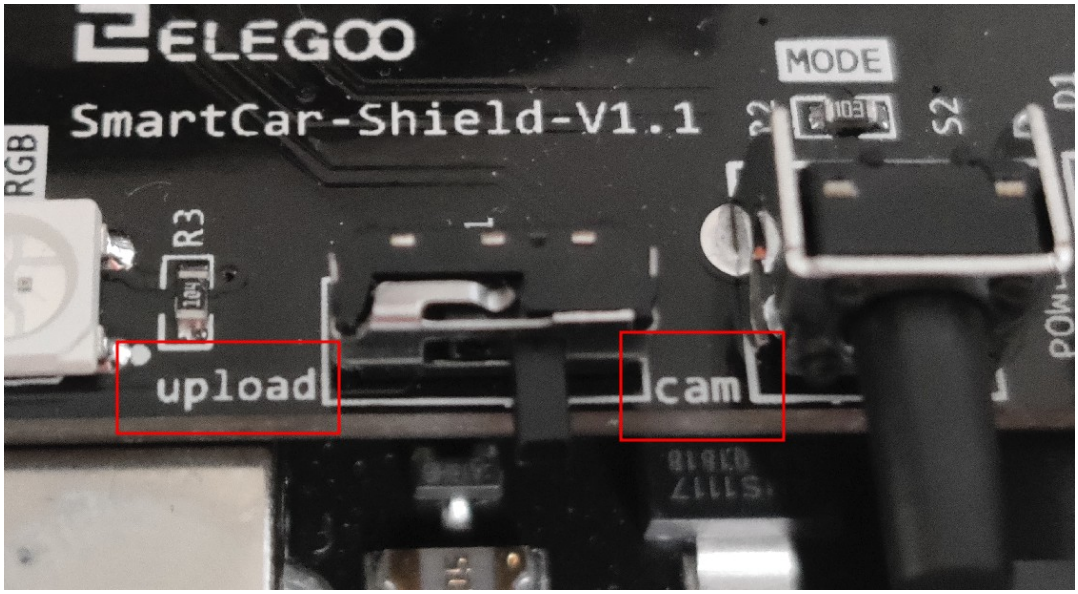
Si ves que la wifi no te da IP, puedes realizar pruebas con la wifi que levantes en tu smartphone (Hotspot) o con cualquier otra wifi (incluso la de tu casa, si haces pruebas allí).

6. Comunicación Serie

Revisa el código de ejemplo que encontrarás en SerialCommunication para entender como es posible comunicar los puertos serie del ESP y de Arduino. Este ejemplo muestra comunicación en ambos sentidos y hace uso de LedFast.

La comunicación serie puede ser bidireccional, es decir, podemos mandar mensajes del ESP al arduino y viceversa. El comportamiento sigue línea NUNCA puede comenzar hasta que el ESP confirme que tiene WiFi y está conectado el servidor MQTT.

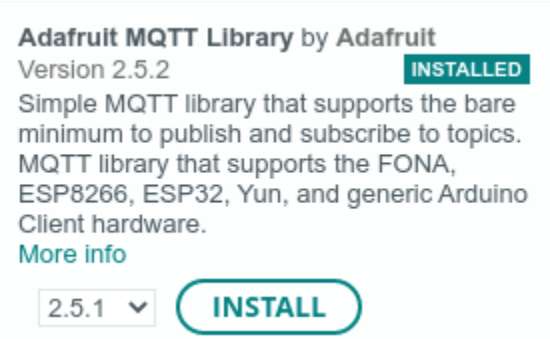
IMPORTANTE: Para que la comunicación serie entre el ESP y Arduino funcione correctamente, el switch S1 de la placa de expansión debe estar en la posición "CAM". Recuerda que ese switch debe estar en la posición "UPLOAD" para cargar el programa en la placa Arduino.



7 Comunicación IoT

7.1 MQTT

Desde el ESP32 tendrás que conectarte y mantener la conexión abierta para mandar mensajes al servidor según vayas completando el circuito. Para ello necesitarás instalar la librería **Adafruit-MQTT**



Aquí es necesario configurar un servicio MQTT publico para el acceso de los estudiantes. Se deja un pequeño manual de como realizarlo.

7.2 Mensajes

Tu robot debe mandar los siguientes mensajes siempre conectando al servidor MQTT y utilizando obligatoriamente el siguiente TOPIC

```
/SETR/2023/$ID_EQUIPO/
```

Para comprobar los mensajes que envía tu robot a través de MQTT puedes utilizar el siguiente comando que debes ejecutar en los ordenadores del laboratorio. Básicamente es un subscriptor a un topic determinado que mostrará todo lo que llega a ese topic.

```
mosquitto_sub -v -h 193.147.53.2 -p 21883 -t /SETR/2023/$ID_EQUIPO/
```

Para realizar pruebas puedes probar a publicar de la siguiente manera:

```
mosquitto_pub -h 193.147.53.2 -p 21883 -t /SETR/2023/$ID_EQUIPO/ -m "hello world!"
```

Mensaje de inicio de vuelta

- Descripción: Este mensaje debe enviarse siempre justo antes de empezar la vuelta al circuito. Por tanto debe realizarse sólo 1 vez. **IMPORTANTE: La vuelta nunca podrá comenzar si no hay conexión a la red WiFi ni a MQTT.**

- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "START_LAP"
}
```

Mensaje de Fin de vuelta

- Descripción: Este mensaje debe enviarse siempre al finalizar una vuelta, que lo sabrás porque tendrás un obstáculo delante. Por tanto debe realizarse sólo 1 vez. Presta atención al campo "time", debe contener el tiempo (en milisegundos) transcurrido desde que enviaste tu START_LAP hasta que envías tu END_LAP. La vuelta ha finalizado justo después de detectar un obstáculo.
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "END_LAP",
        "time": 000000
}
```

Mensaje Obstáculo Detectado

- Descripción: Este mensaje debe enviarse siempre que detectes un obstáculo en el camino de la línea. El coche se debe detener entre 5 y 8 cm antes del obstáculo. En el campo distancia debes enviar la distancia detectada en centímetros.
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "OBSTACLE_DETECTED",
        "distance" : 000
}
```

Mensaje Línea Perdida

- Descripción: Este mensaje debe enviarse siempre que tu robot se haya salido de la línea.
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "LINE_LOST"
}
```

Mensajes PING

- Descripción: Mensaje de estado mandado cada **4 segundos** (empezando en 0). El campo time debe reflejar el tiempo (en milisegundos) desde que comenzó la vuelta (START_LAP). Debe priorizar el comportamiento sigue linea al envío de mensajes PING.
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "PING",
        "time": 000000
}
```

Mensaje Inicio Búsqueda de Linea (OPCIONAL)

- Descripción: Opcionalmente tienes la posibilidad de implementar un comportamiento "encuentra línea" una vez que te hayas salido del camino (esto te permitirá reducir la penalización).
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "INIT_LINE_SEARCH"
}
```

Mensaje Fin Búsqueda de Linea (OPCIONAL)

- Descripción: Si has implementado la búsqueda de línea una vez que la has perdido, deberás mandar este mensaje en el momento que la hayas encontrado y por tanto finalizado el comportamiento "encuentra línea"
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "STOP_LINE_SEARCH"
}
```

Mensaje Línea Encontrada (OPCIONAL)

- Descripción: Este mensaje debe enviarse siempre que tu robot haya encontrado la línea. Además este mensaje sólo se debe enviar si anteriormente se ha enviado el mensaje **LINE_LOST**
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
```

```
        "action": "LINE_FOUND"
}
```

Mensaje Estadísticas Línea Visible (OPCIONAL)

- Descripción: Este mensaje debe enviarse al finalizar la vuelta, es decir, cuando encuentres el obstáculo. Debes enviar el % de veces que has detectado la línea utilizando los infra-rojos. Es decir, si durante la vuelta has leído 1000 veces el infrarojo, y 900 has detectado la línea con alguno de los sensores, y 100 veces no la has detectado, deberás mandar un 90.0 %
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "VISIBLE_LINE",
        "value": 0.00
}
```

8 Evaluación

8.1 Día de Test y Día de Examen

El **Martes 19 de Diciembre** en horario de clase cada equipo dispondrá de al menos 5 minutos para probar su solución en el circuito real y con la parte de comunicaciones realizada. Podrá comprobar así que los mensajes MQTT están bien formados y son correctos.

El **Jueves 21 de Diciembre** en horario de clase cada equipo dispondrá de máximo de **2 rondas/tests** para realizar el circuito. Entre ronda y ronda está permitido modificar el código del programa.

El circuito será idéntico para ambos días.

8.2 Bonificaciones y Penalizaciones

- Cuando el coche se salga del circuito, ese test se dará por finalizado y se declarará nulo.
- Para aprobar la práctica el coche debe terminar el circuito correctamente y mandar los mensajes de comunicación establecidos mediante MQTT. El no envío de mensajes MQTT supondrá tener la práctica suspensa.
- El coche debe parar en el rango [5-8] cm antes del obstáculo. Por cada cm fuera de rango se aplicará una penalización de un 1% al tiempo final.
- Si el coche pierde la línea tiene un máximo de 5 segundos para encontrarla. Si es capaz de recuperarla correctamente y mandar los mensajes MQTT correspondientes obtendrá una rebaja del tiempo final del 2%.

8.3 Nota Final

La nota final dependerá de:

- La posición final en la tabla de tiempos
- Código arduino y mecanismos utilizados.
- Mensajes MQTT enviados y periodicidad (PING)
- Post en el blog explicando la solución.

IMPORTANTE: La práctica estará suspensa si:

- El coche no es capaz de realizar 1 vuelta completa.
- El coche tarda más de 20 segundos en completar la vuelta.

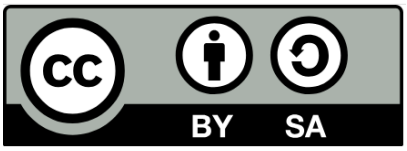
Universidad
Rey Juan Carlos

Grado de Ingeniería en Robótica Software

Sistema empotrados y de tiempo real

Práctica 1

Multihilo y aproximación a tareas esporádicas en C



En esta práctica se trabajarán conceptos de multithreading, deadlines, tiempos de coste en C utilizando pthreads.

Objetivos:

- Repaso de C y pthreads
- Ver la diferencia de ejecutar: Un proceso multihilo en un sistema monocore o multicore
- Comprobar prácticamente que Linux no es un sistema de tiempo real
- No podemos decidir cuando las tareas empiezan o terminan.
- Demostrar que no se pueden asegurar los plazos de ejecución.

Desarrolla un programa (practica1.c) que realice la siguiente funcionalidad.

- El programa debe crear 4 threads que se ejecuten a la vez y no secuencialmente.
- Cada thread es una tarea periódica con C=**0.5** segundos y P=**0.9** segundos (asumimos que el deadline y periodo son iguales).
- Los propios threads son los encargados de estimar el tiempo utilizado en su ejecución y controlar la frecuencia a la que se ejecutan.
- Cada thread debe ejecutar 5 iteraciones, teniendo en cuenta que son iteraciones periódicas (P=0.9).
- Para los cálculos de tiempo utiliza clock_gettime y timespec (mira páginas de manual).
- En cada iteración del thread debes asegurarte que está ejecutando un código random durante 0.5 segundos. Puedes utilizar el siguiente código, pero asegurate que su ejecución es aproximadamente de 0.5 segundos.

  ```
  volatile unsigned long long j;
  for (j=0; j < 400000000ULL; j++);
  ```

- El programa debe mostrar una línea por cada iteración y thread ejecutado con la siguiente información por pantalla (thread e iteraciones empiezan en 1 siempre).

  ```
  [1663147910.736590668] Thread 1 - Iteracion 1: Coste=0.50 s.
  ```

- Si la restricción temporal no se cumplido debe mostrar lo siguiente:

  ```
  [1663147910.736590668] Thread 1 - Iteracion 1: Coste=1.5 s. (fallo temporal)
  ```

- La ejecución del programa debe realizarse en 4 escenarios distintos
  - Ejecución multicore
  - Ejecución monocore
  - Ejecución multicore + stress (todas las cpus)
  - Ejecución monocore + stress (todas las cpus)

- Para ejecutar procesos utilizando una configuración multicore o monocore puedes utilizar el comando taskset
    - Multicore:
        - **`./practica1`**
    - Monocore:
        - **`taskset -c 0 ./practica1`**

- Comandos útiles para la práctica
    - htop: visualizador de procesos en linux
    - stress: genera stress computacional en las cpus
    - time: calcula el tiempo de ejecución de un proceso

PREGUNTAS:

1. Ejecutar los siguientes casos y justifica su comportamiento:
    a. ./practica1 (multicore)
    b. ./practica1 (monocore)
    c. ./practica1 (monocore) + *stress*
    d. ./practica1 (multicore) + *stress*

2. ¿En qué casos de ejecución (nombrados anteriormente) el sistema es capaz de cumplir las restricciones temporales (tanto tiempo de cómputo como periodicidad)?
3. ¿Qué número mínimo de cpus se necesitan para que tu programa ejecute correctamente sin fallos de restricciones temporales? Usa el comando taskset para comprobarlo.
4. ¿Qué solución se podría proponer para cumplir plazos estrictos temporales de periodicidad en la ejecución de los *threads* SIN cambiar la configuración actual que tienen los ordenadores del laboratorio?

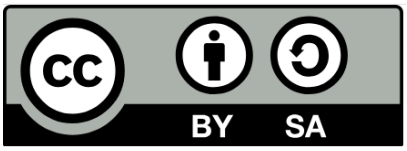
Universidad
Rey Juan Carlos

Grado de Ingeniería en Robótica Software

Sistema empotrados y de tiempo real

Práctica 2

Cácuulo de latencias en sistemas RT y no-RT



En esta segunda práctica pondremos en práctica los conceptos adquiridos en clase de teoría sobre cálculo de latencias y sistemas de tiempo real.

1. Análisis con cyclictest.

Utiliza la utilidad cyclictest para medir la latencia en los laboratorios de la universidad, tanto en ordenadores con kernel normal como en ordenadores con el kernel de RT. Utiliza el siguiente comando, pero **asegúrate** de mirar la página de manual para entender los parámetros utilizados.

```
$ sudo cyclictest -p99 -N --smp -D 60
```

La medición de la tenencia siempre se debe realizar en 3 escenarios distintos:

- S1: idle
- S2: Estresando el planificador (hackbench)
  - `sudo hackbench -l 1000000 -s 1000 -T `nproc``
- S3: Estresando las operaciones I/O (bonnie++)
  - `bonnie++ -d /tmp/ -D -r 2048 -f -b -u $USER -c `nproc``

Además, utilizaremos 3 configuraciones distintas de hardware y kernel para todas las pruebas:

- Laboratorios f-l3109: kernel no-RT (accede por ssh)
- RaspberryPi: kernel no-RT
- RaspberryPi: kernel RT

RBPi Kernel no-RT	**RBPi Kernel RT**
rbpi-01: IP1	rbpi-10: IP6
rbpi-02: IP2	rbpi-11: IP7
rbpi-03: IP3	rbpi-12: IP8
rbpi-04: IP4	rbpi-13: IP9
rbpi-05: IP5	rbpi-14: IP10

Para acceder a las RBPI, que tienen IPs privadas, debes hacerlo siempre desde ordenadores de la universidad. Puedes utilizar tu mismo usuario que en los laboratorios. Además, tendrás tu HOME montado igualmente.

Calcula las latencias máximas y medias en los 3 escenarios descritos arriba (S1, S2, S3) para las 3 configuraciones de hardware y kernel disponibles. Muestra los resultados obtenidos en la siguiente tabla (si lo prefieres puedes usar gráficos o plots de barras mientras la información quede correctamente mostrada).

cyclictestURJC						
	Laboratorios		RaspberryPi			
	Kernel NO RT		Kernel NO RT		Kernel RT	
	Latencia media (ns)	Latencia max (ns)	Latencia media (ns)	Latencia max (ns)	Latencia media (ns)	Latencia max (ns)
S1						
S2						
S3						

2. Desarrollo de cyclictestURJC.

Desarrolla una aplicación en C llamada *cyclictestURJC* que mida la latencia de planificación de un sistema GNU/Linux utilizando las técnicas explicadas en clase. cyclictestURJC debe ejecutar continuamente durante 1 minuto y debe realizar la siguiente funcionalidad:

- Generar un thread por cada core del ordenador. Para saber el número de cores, puedes utilizar la siguiente función:
  - `int N = (int) sysconf(_SC_NPROCESSORS_ONLN);`
- Cada thread debe ejecutarse en la cola de prioridad SCHED_FIFO, con prioridad 99.
  - `pthread_setschedparam`
- Cada thread debe ejecutarse en un core diferente, por tanto, establece la afinidad con
  - `pthread_setaffinity_np`
- El thread debe estar constantemente calculando la latencia de planificación mediante técnicas de sleep. **RECOMENDACIÓN**: realiza esperas del orden de milisegundos.
- Para configurar Linux con la mínima latencia en DMA, asegúrate que tu programa siempre comienza abriendo el fichero `/dev/cpu_dma_latency` y escribiendo un 0. **NO** cierres el descriptor hasta el final del programa (si lo cierras, restablecerá la configuración previa).

```
static int32_t latency_target_value = 0;
latency_target_fd = open("/dev/cpu_dma_latency", O_RDWR);
write(latency_target_fd, &latency_target_value, 4);
```

- Al finalizar el programa (después de 1 minuto de ejecución), debe mostrar el siguiente resumen con la latencia media y latencia máxima por thread/cpu.

```
$ ./cyclictestURJC
[0]    latencia media = 000001984 ns. | max = 000016865 ns
[1]    latencia media = 000001984 ns. | max = 000017124 ns
[2]    latencia media = 000001999 ns. | max = 000014698 ns
[4]    latencia media = 000001995 ns. | max = 000016550 ns
[3]    latencia media = 000002011 ns. | max = 000014936 ns
[5]    latencia media = 000002005 ns. | max = 000014396 ns

Total  latencia media = 000001996 ns. | max = 000017124 ns
```

- Además, tu programa debe guardar en un fichero CSV (cyclictestURJC.csv) en formato texto plano (https://en.wikipedia.org/wiki/Comma-separated_values) todas las medidas obtenidas en el siguiente formato:

CPU, NUMERO_ITERACION, LATENCIA

MUY IMPORTANTE: Asegúrate de realizar esta operación de escritura a fichero, una vez terminada toda la evaluación de la latencia, para no interferir con las interrupciones al escribir.

MUY IMPORTANTE: Asegúrate que cuando realices las mediciones en las RBPi, no haya ningún otro estudiante realizando las mediciones también. Obtendréis resultados erróneos.

Utilizando la herramienta cyclictestURJC que has desarrollado, rellena la siguiente tabla del mismo modo que hiciste en el apartado 1.

cyclictestURJC						
	Laboratorios		RaspberryPi			
	Kernel NO RT		Kernel NO RT		Kernel RT	
	Latencia media (ns)	Latencia max (ns)	Latencia media (ns)	Latencia max (ns)	Latencia media (ns)	Latencia max (ns)
S1						
S2						
S3						

Por último, crea un gráfico que muestre la distribución de la latencia para los distintos escenarios en Raspberry Pi. Deberás incluir los 2 tipos de kernel (real-time y no real-time) y los 3 escenarios definidos (idle, hackbench, etc.). Asegúrate de que todos los gráficos de latencia utilicen el mismo rango en el eje X, expresado en microsegundos, para facilitar la comparación entre ellos. Los datos deben provenir de los archivos CSV generados para cada escenario.

En cada gráfico, que deben aparecer las leyendas correctamente identificando los escenarios, incluye la desviación estándar calculada para cada escenario. Puedes ponerlo en la propia leyenda del plot.

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.hist.html

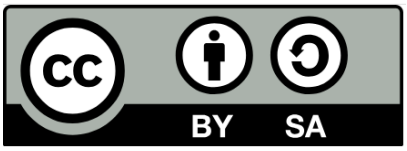
Universidad
Rey Juan Carlos

Grado de Ingeniería en Robótica Software

Sistema empotrados y de tiempo real

Práctica 3

Controlador Máquina Expendedora desarrollado en Arduino



Descripción de la práctica

Se busca diseñar e implementar un controlador para una máquina expendedora que esté basado en Arduino UNO y en los sensores/actuadores que se proporcionan en el kit Arduino. La práctica tendrá que integrar obligatoriamente los siguientes componentes hardware:

- Arduino UNO
- LCD
- Joystick
- Sensor temperatura/Humedad DHT11
- Sensor Ultrasonido
- Boton
- 2 LEDS Normales (LED1, LED2)

Funcionalidad

La práctica debe implementar la siguiente funcionalidad software.

1. Arranque:
    a. Al inicio del sistema, el LED1 debe parpadear 3 veces a intervalos de 1 segundo. Al mismo tiempo debe mostrarse el mensaje “CARGANDO …” en el LCD. Al cabo de los 3 parpadeos el LED1 debe apagarse y la pantalla debe mostrar la información de la funcionalidad “Servicio”.
2. Servicio.
    a. Si el usuario se encuentra a menos de 1 metro de la máquina se debe pasar al estado b). En caso contrario el LCD debe mostrar “ESPERANDO CLIENTE”.
    b. El LCD debe mostrar la temperatura y humedad durante 5 segundos y acto seguido deberá mostrar la lista de productos que el usuario puede seleccionar. Los productos y precios a mostrar son:
        i. Cafe Solo 1€
        ii. Cafe Cortado 1.10 €
        iii. Cafe Doble 1.25 €
        iv. Cafe Premium 1.50 €
        v. Chocolate 2.00 €

Debes permitir la navegación por esa lista utilizando el joystick (arriba / abajo). Y para su selección debes usar el switch del propio joystick. Una vez seleccionado debes mostrar el mensaje “Preparando Cafe …” durante un tiempo aleatorio entre 4 y 8 segundos. Cada ejecución puede ser un tiempo distinto (debes hacerlo aleatorio). Usa ese mismo tiempo para hacer que el LED2 se encienda de manera incremental, de tal manera que la intensidad del LED2 indica igualmente el progreso de la preparación del café. Una vez terminada la preparación del cafe, el LDC debe mostrar “RETIRE BEBIDA” durante 3 segundos y volver a la funcionalidad inicial de Servicio.

En cualquier momento del estado b), el usuario puede reiniciar el estado (no la placa) si pulsa el botón durante el rango 2-3 segundos. Por lo que deberá ejecutar de nuevo la funcionalidad de Servicio.

3. Admin
    a. Es posible acceder a la interfaz de administración de la máquina en cualquier momento. Para ello el usuario debe presionar el botón durante no menos de 5 segundos.
    b. Mientras el usuario esté en la vista de Admin, ambos LEDS deben estar encendidos.
    c. El siguiente menú debe ser mostrado en el LCD:
        i. Ver temperatura
        ii. Ver distancia sensor
        iii. Ver contador
        iv. Modificar precios

    Debes permitir la navegación por esa lista utilizando el joystick (arriba / abajo). Y para su selección debes usar el switch del propio joystick. Además debes permitir volver al menú utilizando el joystick (movimiento izquierda). A continuación se detalla lo que debe mostrar cada menú:
    - i) Temp: XX ºC Hum: YY % (debe cambiar dinámicamente)
    - ii) Distancia: XX cm (debe cambiar dinámicamente)
    - iii) Tienes que llevar un contador en segundos desde que la placa está arrancada. Ese contador en segundos es el que se muestra en este menú. Mientras estás en esa pantalla se tiene que observar como el contador va incrementando con el paso de los segundos.
    - iv) Debes mostrar el listado de productos y permitir el cambio de precio utilizando el joystick. Los incrementos o decrementos se realizan en 5 céntimos (utiliza el joystick arriba y abajo para cambiar el precio). Para confirmar el valor del precio utiliza el switch del joystick y para cancelar y volver a la lista de precio utiliza el movimiento “izquierda” del joystick. Ahora si vas a la lista ofrecida en la funcionalidad 2a) deberías ver los precios actualizados. (Los precios no persisten si la placa se reinicia).

    d. Para salir de la vista admin se debe pulsar de nuevo el pulsador durante no menos de 5 segundos, volviendo a la funcionalidad de Servicio.

Recomendaciones

- Haz uso de todas las librerías y técnicas vistas en clase.
- Mantén tu código seguro utilizando el watchdog para evitar bloqueos.

Documentación:

- https://www.arduino.cc/reference/en/

Bibliotecas utilizadas:

- LiquidCrystal: https://www.arduino.cc/en/Reference/LiquidCrystal
- ArduinoThread: https://www.arduino.cc/reference/en/libraries/arduinothread/
- Watch Dog: https://create.arduino.cc/projecthub/rafitc/what-is-watchdog-timer-fffe20
- DHT-sensor-library: https://github.com/adafruit/DHT-sensor-
- TimerOne: https://www.arduino.cc/reference/en/libraries/timerone/

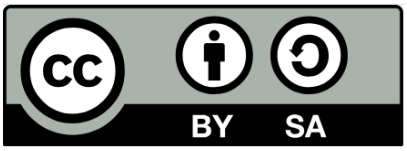
Universidad
Rey Juan Carlos

Grado de Ingeniería en Robótica Software

Sistema empotrados y de tiempo real

Práctica 4

Sigue Linea en Arduino basado en sistemas RT



Sumario

1. Descripción..... 3
2. Esquema..... 4
3. Montaje..... 6
4. Arduino..... 6
    4.1 Ultrasonido..... 6
    4.2 Sensor Infra-rojo..... 6
    4.3 Motores..... 6
    4.4 LED..... 7
5. ESP32..... 7
    5.1 Arduino-IDE y librerías..... 7
    5.2 Comprobar la conexión WiFi..... 9
6. Comunicación Serie..... 9
7 Comunicación IoT..... 10
    7.1 MQTT..... 10
    7.2 Mensajes..... 11
        Mensaje de inicio de vuelta..... 11
        Mensaje de Fin de vuelta..... 11
        Mensaje Obstáculo Detectado..... 12
        Mensaje Línea Perdida..... 12
        Mensajes PING..... 12
        Mensaje Inicio Búsqueda de Linea (OPCIONAL)..... 12
        Mensaje Fin Búsqueda de Linea (OPCIONAL)..... 13
        Mensaje Línea Encontrada (OPCIONAL)..... 13
        Mensaje Estadísticas Línea Visible (OPCIONAL)..... 13
8 Evaluación..... 14
    8.1 Día de Test y Día de Examen..... 14
    8.2 Bonificaciones y Penalizaciones..... 14
    8.3 Nota Final..... 14
9. Otros recursos..... 15

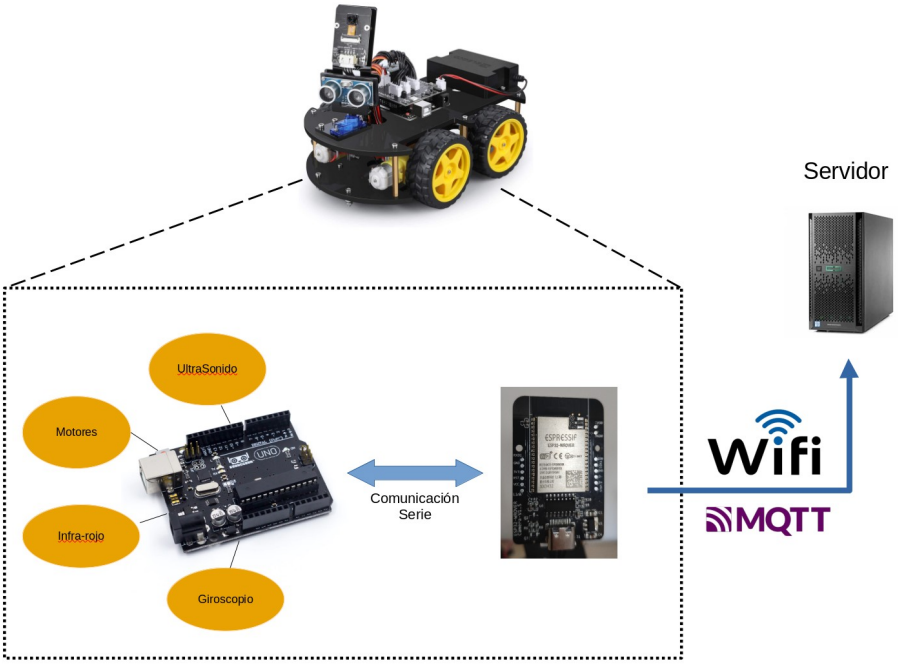
1. Descripción

La práctica se realizará por grupos de 2 estudiantes. Se utilizará el kit de Smart Robot Car Kit v4.0 compuesto por un arduino uno y un ESP32 camera

La evaluación y objetivos de esta práctica son:

- Sigue la línea lo más rápido posible sin salirse.
- Comunicación IoT a través de MQTT
- Comunicación serie entre ESP32 y Arduino UNO
- Si el robot pierde la línea se permite realizar una búsqueda de la línea de nuevo.
- Detección de obstáculos

2. Esquema



3. Montaje

Consulta el manual que viene con el kit y el siguiente vídeo para realizar el montaje correctamente. Es el primer paso de la práctica.

4. Arduino

El robot incorpora un Ardunio UNO que actúa de cerebro del sistema controlando todos los sensores y actuadores.

4.1 Ultrasonido

Los sensores de ultrasonido los puedes controlar uitlizando los siguientes pines.

```
#define TRIG_PIN 13
#define ECHO_PIN 12
```

4.2 Sensor Infra-rojo

Los sensores infra-rojos, son sensores analógicos. Usa toda su potencia y rango continuo de valores. **No los utilices como sensores digitales**

```
#define PIN_ITR20001-LEFT   A2
#define PIN_ITR20001-MIDDLE A1
#define PIN_ITR20001-RIGHT  A0
```

4.3 Motores

El Smart Robot Car incorpora 4 motores DC que se controlan a través de una pequeña placa controladora incorporada en la placa de extensión. Las velocidades las comandaremos por grupos de motores (los del lado derecho, y los del lado izquierdo). A continuación se muestran los pines de control, velocidad y dirección.

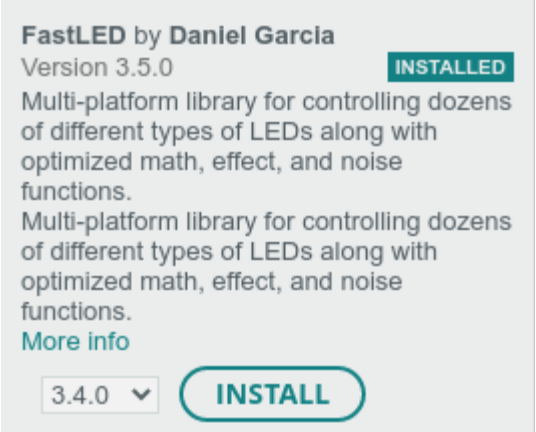
```
// Enable/Disable motor control.
//  HIGH: motor control enabled
//  LOW: motor control disabled
#define PIN_Motor_STBY 3

// Group A Motors (Right Side)
// PIN_Motor_AIN_1: Digital output. HIGH: Forward, LOW: Backward
#define PIN_Motor_AIN_1 7
// PIN_Motor_PWMA: Analog output [0-255]. It provides speed.
#define PIN_Motor_PWMA 5

// Group B Motors (Left Side)
// PIN_Motor_BIN_1: Digital output. HIGH: Forward, LOW: Backward
#define PIN_Motor_BIN_1 8
// PIN_Motor_PWMB: Analog output [0-255]. It provides speed.
#define PIN_Motor_PWMB 6
```

4.4 LED

La placa de expansión contiene un LED RGB que podemos utilizar para plasmar colores según los estados de tu comportamiento (muy útil para depuración en el circuito).

IMPORTANTE: Es obligatorio que siempre que tu robot detecte la línea (con cualquiera de los 3 sensores), debe mostrar el color verde en el LED. Si no detecta la línea, debe mostrar el LED en rojo.

Es necesario instalar la siguiente librería (FastLED):



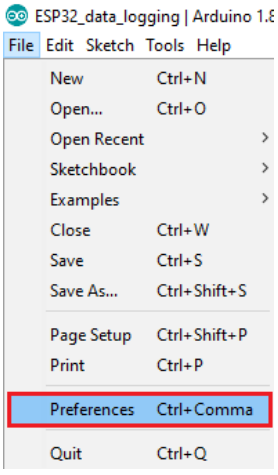
Puedes consultar el siguiente código de ejemplo para entender el funcionamiento DemoLed

5. ESP32

5.1 Arduino-IDE y librerías

El robot incluye un modelo ESP32 CAM, que nos permite comunicarnos a través de cualquier red WiFi. Utilizaremos Arduino-IDE para programarlo, pero es necesario realizar las siguientes acciones:

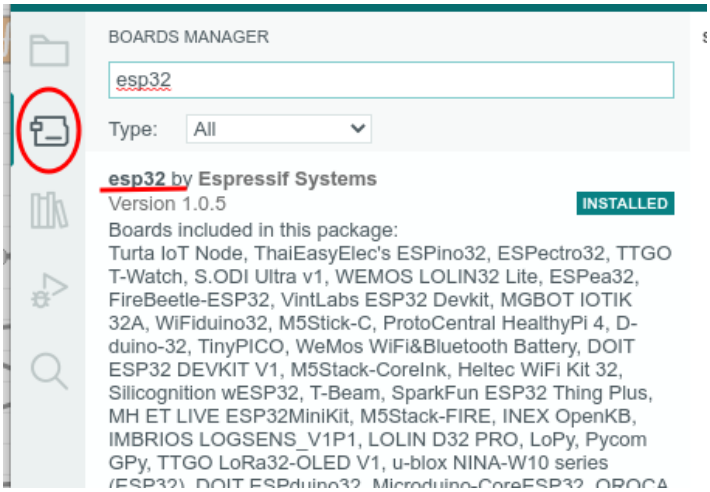
- Primero, añade el repositorio para ESP32 dentro de tu Arduino IDE



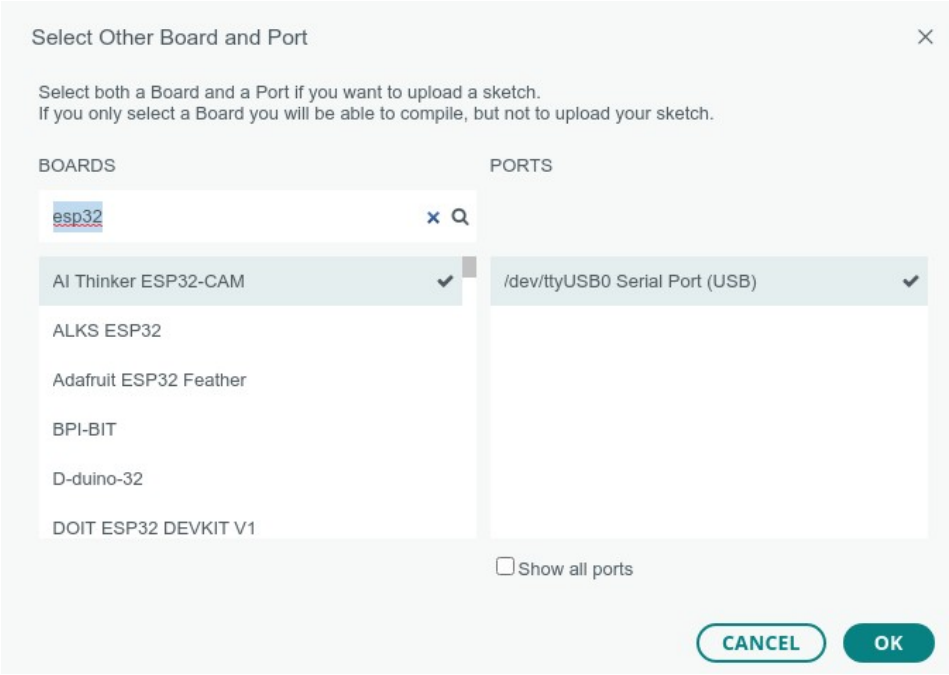
Introduce la siguiente URL en "Additional Board Manager URLs"

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

Asegurate que tienes instalado el paquete ESP32 dentro de "Boards Manager" en tu arduino IDE



Asegurate de configurar correctamente el modelo de placa "AI Thinker ESP32-CAM"



Por último asegurate que en los laboratorios, te aparecen 2 puertos detectados (ttyS4 y ttyUSB0). Utiliza éste último para realizar la programación del ESP32.

5.2 Comprobar la conexión WiFi

- Puedes comprobar la conexión WiFi (ESP) y la IP asignada. Para ello utilizaremos la red wifi eduroam. Puedes utilizar de base el código de ejemplo de **eduroam** del repositorio.

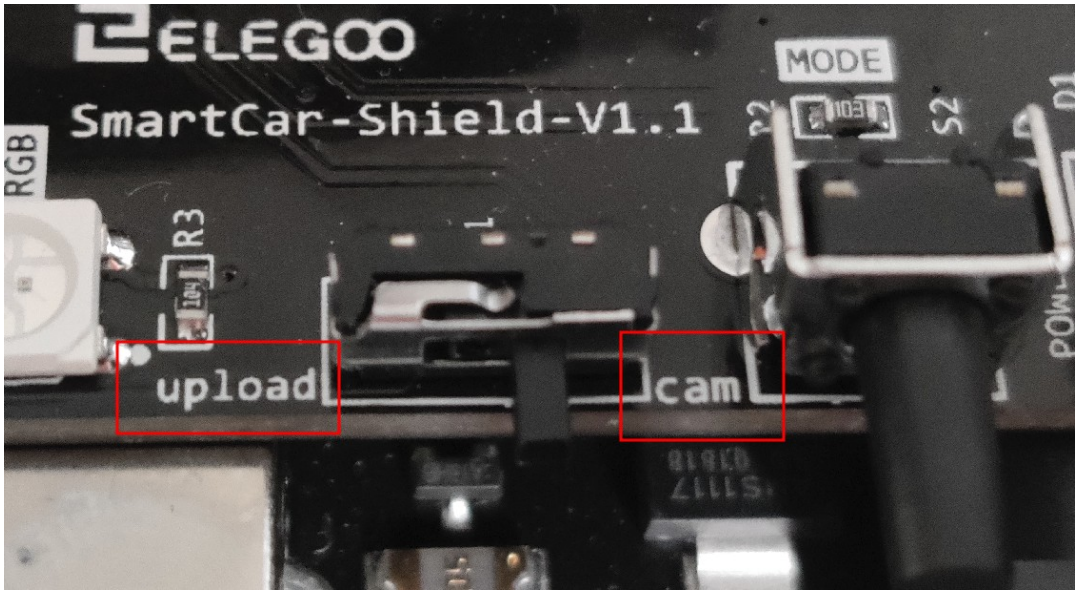
Si ves que la wifi no te da IP, puedes realizar pruebas con la wifi que levantes en tu smartphone (Hotspot) o con cualquier otra wifi (incluso la de tu casa, si haces pruebas allí).

6. Comunicación Serie

Revisa el código de ejemplo que encontrarás en SerialCommunication para entender como es posible comunicar los puertos serie del ESP y de Arduino. Este ejemplo muestra comunicación en ambos sentidos y hace uso de LedFast.

La comunicación serie puede ser bidireccional, es decir, podemos mandar mensajes del ESP al arduino y viceversa. El comportamiento sigue línea NUNCA puede comenzar hasta que el ESP confirme que tiene WiFi y está conectado el servidor MQTT.

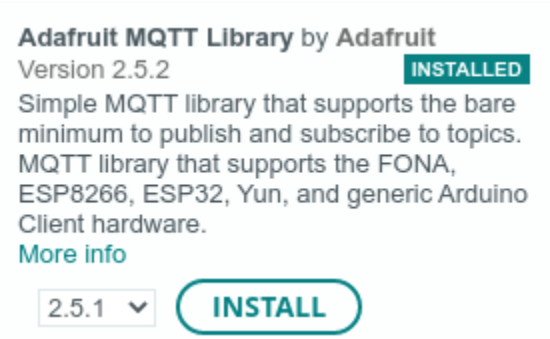
IMPORTANTE: Para que la comunicación serie entre el ESP y Arduino funcione correctamente, el switch S1 de la placa de expansión debe estar en la posición "CAM". Recuerda que ese switch debe estar en la posición "UPLOAD" para cargar el programa en la placa Arduino.



7 Comunicación IoT

7.1 MQTT

Desde el ESP32 tendrás que conectarte y mantener la conexión abierta para mandar mensajes al servidor según vayas completando el circuito. Para ello necesitarás instalar la librería **Adafruit-MQTT**



Aquí es necesario configurar un servicio MQTT publico para el acceso de los estudiantes. Se deja un pequeño manual de como realizarlo.

7.2 Mensajes

Tu robot debe mandar los siguientes mensajes siempre conectando al servidor MQTT y utilizando obligatoriamente el siguiente TOPIC

```
/SETR/2023/$ID_EQUIPO/
```

Para comprobar los mensajes que envía tu robot a través de MQTT puedes utilizar el siguiente comando que debes ejecutar en los ordenadores del laboratorio. Básicamente es un subscriptor a un topic determinado que mostrará todo lo que llega a ese topic.

```
mosquitto_sub -v -h 193.147.53.2 -p 21883 -t /SETR/2023/$ID_EQUIPO/
```

Para realizar pruebas puedes probar a publicar de la siguiente manera:

```
mosquitto_pub -h 193.147.53.2 -p 21883 -t /SETR/2023/$ID_EQUIPO/ -m "hello world!"
```

Mensaje de inicio de vuelta

- Descripción: Este mensaje debe enviarse siempre justo antes de empezar la vuelta al circuito. Por tanto debe realizarse sólo 1 vez. **IMPORTANTE: La vuelta nunca podrá comenzar si no hay conexión a la red WiFi ni a MQTT.**

- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "START_LAP"
}
```

Mensaje de Fin de vuelta

- Descripción: Este mensaje debe enviarse siempre al finalizar una vuelta, que lo sabrás porque tendrás un obstáculo delante. Por tanto debe realizarse sólo 1 vez. Presta atención al campo "time", debe contener el tiempo (en milisegundos) transcurrido desde que enviaste tu START_LAP hasta que envías tu END_LAP. La vuelta ha finalizado justo después de detectar un obstáculo.
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "END_LAP",
        "time": 000000
}
```

Mensaje Obstáculo Detectado

- Descripción: Este mensaje debe enviarse siempre que detectes un obstáculo en el camino de la línea. El coche se debe detener entre 5 y 8 cm antes del obstáculo. En el campo distancia debes enviar la distancia detectada en centímetros.
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "OBSTACLE_DETECTED",
        "distance" : 000
}
```

Mensaje Línea Perdida

- Descripción: Este mensaje debe enviarse siempre que tu robot se haya salido de la línea.
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "LINE_LOST"
}
```

Mensajes PING

- Descripción: Mensaje de estado mandado cada **4 segundos** (empezando en 0). El campo time debe reflejar el tiempo (en milisegundos) desde que comenzó la vuelta (START_LAP). Debe priorizar el comportamiento sigue linea al envío de mensajes PING.
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "PING",
        "time": 000000
}
```

Mensaje Inicio Búsqueda de Linea (OPCIONAL)

- Descripción: Opcionalmente tienes la posibilidad de implementar un comportamiento "encuentra línea" una vez que te hayas salido del camino (esto te permitirá reducir la penalización).
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "INIT_LINE_SEARCH"
}
```

Mensaje Fin Búsqueda de Linea (OPCIONAL)

- Descripción: Si has implementado la búsqueda de línea una vez que la has perdido, deberás mandar este mensaje en el momento que la hayas encontrado y por tanto finalizado el comportamiento "encuentra línea"
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "STOP_LINE_SEARCH"
}
```

Mensaje Línea Encontrada (OPCIONAL)

- Descripción: Este mensaje debe enviarse siempre que tu robot haya encontrado la línea. Además este mensaje sólo se debe enviar si anteriormente se ha enviado el mensaje **LINE_LOST**
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
```

```
        "action": "LINE_FOUND"
}
```

Mensaje Estadísticas Línea Visible (OPCIONAL)

- Descripción: Este mensaje debe enviarse al finalizar la vuelta, es decir, cuando encuentres el obstáculo. Debes enviar el % de veces que has detectado la línea utilizando los infra-rojos. Es decir, si durante la vuelta has leído 1000 veces el infrarojo, y 900 has detectado la línea con alguno de los sensores, y 100 veces no la has detectado, deberás mandar un 90.0 %
- Payload JSON:

```
{
        "team_name": "$TU_NOMBRE_DE_EQUIPO ",
        "id": "$ID_EQUIPO",
        "action": "VISIBLE_LINE",
        "value": 0.00
}
```

8 Evaluación

8.1 Día de Test y Día de Examen

El **Martes 19 de Diciembre** en horario de clase cada equipo dispondrá de al menos 5 minutos para probar su solución en el circuito real y con la parte de comunicaciones realizada. Podrá comprobar así que los mensajes MQTT están bien formados y son correctos.

El **Jueves 21 de Diciembre** en horario de clase cada equipo dispondrá de máximo de **2 rondas/tests** para realizar el circuito. Entre ronda y ronda está permitido modificar el código del programa.

El circuito será idéntico para ambos días.

8.2 Bonificaciones y Penalizaciones

- Cuando el coche se salga del circuito, ese test se dará por finalizado y se declarará nulo.
- Para aprobar la práctica el coche debe terminar el circuito correctamente y mandar los mensajes de comunicación establecidos mediante MQTT. El no envío de mensajes MQTT supondrá tener la práctica suspensa.
- El coche debe parar en el rango [5-8] cm antes del obstáculo. Por cada cm fuera de rango se aplicará una penalización de un 1% al tiempo final.
- Si el coche pierde la línea tiene un máximo de 5 segundos para encontrarla. Si es capaz de recuperarla correctamente y mandar los mensajes MQTT correspondientes obtendrá una rebaja del tiempo final del 2%.

8.3 Nota Final

La nota final dependerá de:

- La posición final en la tabla de tiempos
- Código arduino y mecanismos utilizados.
- Mensajes MQTT enviados y periodicidad (PING)
- Post en el blog explicando la solución.

IMPORTANTE: La práctica estará suspensa si:

- El coche no es capaz de realizar 1 vuelta completa.
- El coche tarda más de 20 segundos en completar la vuelta.