

REDES INALÁMBRICAS DE SENSORES



Universidad de Sevilla

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

Reporte de prácticas

Autor:

Sergio León Doncel & Álvaro García Lora

Diciembre 2023

Índice general

1. Objetivos y alcance	1
1.1. Idea de producto	1
1.2. Objetivos	1
1.3. Planteamiento del problema	2
1.4. Aplicaciones	2
2. Desarrollo del proyecto	3
2.1. Launchpad CC2650	3
2.2. Uso del convertidor analógico digital	5
2.3. Caracterización del sensor de nivel de agua	6
2.4. Comunicación bajo estándar IEEE 802.15.4 entre cliente-servidor . . .	8
2.5. Despliegue de servicios de visualización y métricas	9
3. Conclusiones y prueba de funcionamiento	10
4. Ampliaciones para producto final	14
5. Archivos adjuntos	15

Resumen

Este documento pretende reportar el procedimiento y resultados conseguidos durante el desarrollo del proyecto de la asignatura Redes Inalámbricas de Sensores para el Máster de Ingeniería Electrónica, Robótica y Automática.

Se procederá a la recolección de medidas de un sensor analógico de nivel de agua, al establecimiento de una comunicación cliente-servidor entre 2 plataformas IoT compatibles con ContikiNG, y al uso de los servicios necesarios para la muestra de métricas en remoto.

Capítulo 1

Objetivos y alcance

1.1. Idea de producto

Hoy en día, la sociedad se encuentra cada vez más concentrada en urbes de población. Sin embargo, la productividad de las zonas agrícolas tiene un peso muy importante. La tecnología IoT puede permitir recolectar información cómodamente sin tener que acudir al lugar, lo que supone un ahorro en tiempo, dinero y contaminación. Así mismo posibilita automatizar los sistemas para aumentar su eficiencia.

Dentro de este mismo contexto, un problema cotidiano que se les presenta a los agricultores es el de conocer la precipitación de lluvia caída sobre los cultivos en sus terrenos a lo largo del año. En este aspecto, los sensores IoT se presentan como una solución efectiva y de bajo coste.

1.2. Objetivos

El objetivo final de este proyecto es el de crear un sistema prototipo de pluviómetro IoT que muestre un posible abordamiento del problema.

Para ello, se plantea hacer uso de una placa Launchpad modelo LAUNCHXL-CC2650 de Texas Instruments para la lectura y envío de los datos procesados obtenidos de un sensor analógico de nivel de agua HW-038. Esta se comunicará de forma inalámbrica usando el estándar IEEE 802.15.4 con una placa dongle nrf52840 de Nordic Semiconductors, estableciendo una conexión de cliente a servidor respectivamente.

La aplicación final incluirá MQTT como protocolo de red, y aprovechará los servicios de Prometheus y Grafana para la recolección de los datos y su visualización, de cara a una interfaz de monitorización más amigable y clara para el usuario.

1.3. Planteamiento del problema

Para lograr los objetivos mencionados en el punto anterior, hemos de descomponer el problema en los siguientes aspectos:

- Familiarización con placa LAUNCHXL-CC2650 de Texas Instruments:
 - Programación del dispositivo
 - Uso del ADC en Contiki
- Caracterización del sensor de nivel de agua HW-038
- Comunicación inalámbrica entre dongle nrf52840 y launchpad cc2650 bajo estándar IEEE 802.15.4
- Uso de MQTT, Prometheus y Grafana para recolección y visualización de datos

1.4. Aplicaciones

Los pluviómetros tienen utilidades muy diversas en función del campo de aplicación:

- **Agricultura:** Permite estimar la humedad de agua en los terrenos, de forma que se generen acciones que ayuden a mantener este nivel dentro de los límites adecuados para cultivos y cosechas, por ejemplo, en sistemas de regadío inteligentes.
- **Ciudades inteligentes:** Para prevención de desastres naturales por inundaciones, desbordamientos de caudal y sequías, consiguiendo proteger las poblaciones.
- **Investigación:** Las mediciones permiten estudiar la evolución del cambio climático a través de histogramas, para realizar comparativas temporales.

Por los motivos mencionados, entre otros, el despliegue de una red de nodos con sensores de agua cubriendo un cierto área puede ser interesante.

De igual forma, este sistema podría colocarse en cauces de canales y ríos para controlar las subidas y bajadas de caudal, o incluso en zonas donde el tránsito de agua puede causar una situación peligrosa en poblaciones y es necesario recibir avisos del riesgo con antelación para no lamentar daños.

Capítulo 2

Desarrollo del proyecto

2.1. Launchpad CC2650

Para poder cargar nuestro programa en la placa launchpad cc2650, se hace uso del ROM bootloader, con el comando: `make TARGET=cc26x0-cc13x0 BOARD=launchpad/cc2650 <program_name>.upload`

La compilación del mismo se realiza con el mismo comando sin indicar el nombre del programa ya que se usa un fichero Makefile: `make TARGET=cc26x0-cc13x0 BOARD=launchpad/cc2650`

No obstante, es requisito previo desbloquear el bootloader, ya que originalmente de fábrica estará bloqueado [1]. Para ello, debe borrarse toda la flash con la herramienta SmartRF Flash Programmer (Windows) [2] o UniFlash (Linux) [3]. Esto último sólo es necesario la primera vez que se use la placa de fábrica. Para poder cargar nuestro programa una vez desbloqueado el bootloader, debemos habilitarlo, presionando los botones 1 (de forma mantenida) y el botón RESET, de manera que la placa entre en este modo.

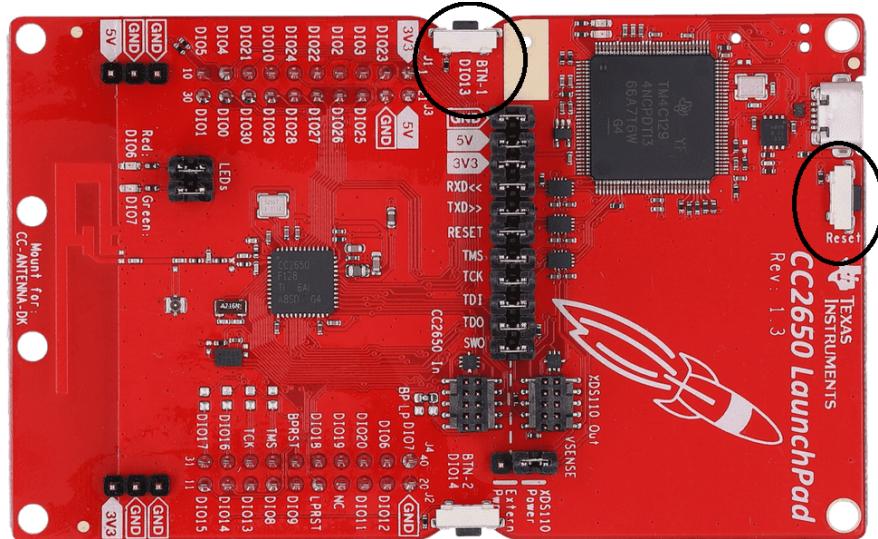


Figura 2.1: Vista cenital del launchpad

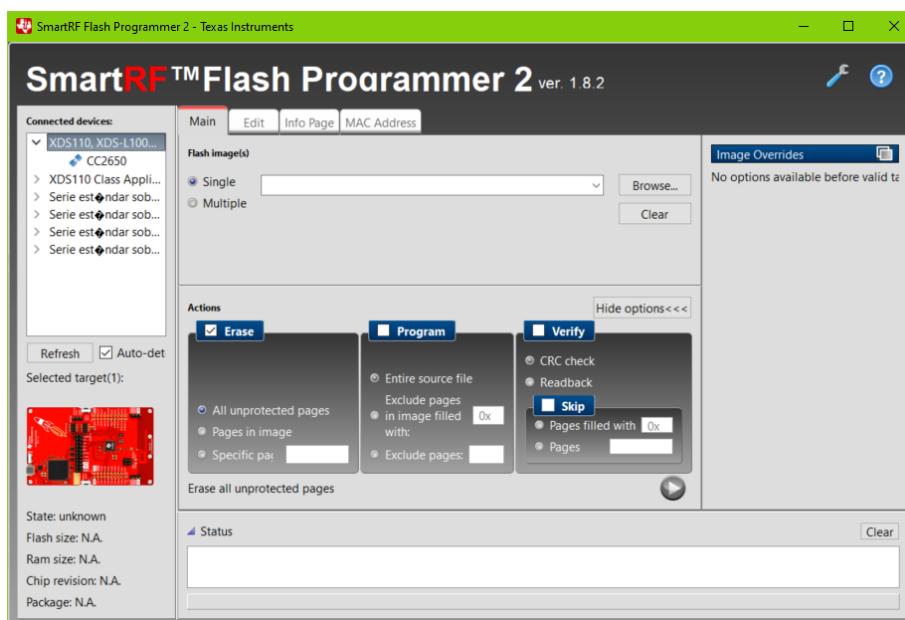


Figura 2.2: Herramienta Flash Programmer

```

user@vn-devel:~/contiki-ng/IoT-WSN-Contiki-ng/project/launchpad-nrf-water/launchpads make TARGET=cc26x0-cc13x0 BOARD=launchpad/cc2650 udp-client-launchpad.upload
CC      ./../arch/cpu/cc26x0-cc13x0/lib/cc26xxware/startup_files/ccfg.c
CC      ./../arch/cpu/cc26x0-cc13x0/.iieee-addr.c
CC      ./../arch/cpu/cc26x0-cc13x0/.fault-handlers.c
CC      ./../arch/cpu/cc26x0-cc13x0/lib/cc26xxware/startup_files/startup_gcc.c
CC      udp-client-launchpad.c
LD      build/cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.elf
OBJCOPY build/cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.elf --build=cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.bin
./../.tools/c2538-bsl/c2538-bsl.py -e -w -v -p /dev/ttyACM0 build/cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.bin
Opening port /dev/ttyACM0, baud 500000
Reading data from build/cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.bin
Cannot auto-detect firmware filetype: Assuming .bin
Connecting to target...
CC2650 PG2.3 (Tx/Rx): 128KB Flash, 20KB SRAM, CCFG_BL_CONFIG at 0x0001FF08
Primary IEEE Address: 00:12:4B:00:12:04:D5:07
    Performing mass erase
Erase done
Writing 131072 bytes starting at address 0x00000000
Write 128 bytes at 0x0001FF08
Verifying by comparing CRC32 calculations.
    Verified (match: 0xdead539e)
rm build/cc26x0-cc13x0/launchpad/cc2650/obj/fault-handlers.o build/cc26x0-cc13x0/launchpad/cc2650/obj/startup_gcc.o udp-client-launchpad.o

```

Figura 2.3: Carga del programa de forma correcta

Es posible que durante el proceso de flasheo del programa obtengamos ciertos errores. A continuación se muestran los más comunes y el motivo de los mismos.

```

user@vn-devel:~/contiki-ng$ cd IoT-WSN-Contiki-ng/project/launchpad-nrf-water/launchpad/
user@vn-devel:~/contiki-ng/IoT-WSN-Contiki-ng/project/launchpad-nrf-water/launchpads make TARGET=cc26x0-cc13x0 BOARD=launchpad/cc2650 udp-client-launchpad.upload
load
CC      ./../arch/cpu/cc26x0-cc13x0/lib/cc26xxware/startup_files/ccfg.c
CC      ./../arch/cpu/cc26x0-cc13x0/.iieee-addr.c
CC      ./../arch/cpu/cc26x0-cc13x0/.fault-handlers.c
CC      ./../arch/cpu/cc26x0-cc13x0/lib/cc26xxware/startup_files/startup_gcc.c
CC      udp-client-launchpad.c
LD      build/cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.elf
OBJCOPY build/cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.elf --build=cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.bin
./../.tools/c2538-bsl/c2538-bsl.py -e -w -v -p /dev/ttyACM0 build/cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.bin
Opening port /dev/ttyACM0, baud 500000
Reading data from build/cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.bin
Cannot auto-detect firmware filetype: Assuming .bin
Connecting to target...
ERROR: Timeout waiting for ACK/NACK after 'Synch (0x55 0x55)'
ERROR: Timeout waiting for ACK/NACK after 'Synch (0x55 0x55)'
ERROR: Timeout waiting for ACK/NACK after 'Synch (0x55 0x55)'
ERROR: Timeout waiting for ACK/NACK after 'Synch (0x55 0x55)'
ERROR: Timeout waiting for ACK/NACK after 'Synch (0x55 0x55)'
ERROR: Timeout waiting for ACK/NACK after 'Synch (0x55 0x55)'
make: *** [udp-client-launchpad.upload] Error 1
rm build/cc26x0-cc13x0/launchpad/cc2650/obj/fault-handlers.o build/cc26x0-cc13x0/launchpad/cc2650/obj/startup_gcc.o udp-client-launchpad.o
user@vn-devel:~/contiki-ng/IoT-WSN-Contiki-ng/project/launchpad-nrf-water/launchpads

```

Figura 2.4: ERROR: Timeout waiting for ACK/NACK after 'Synch (0x55 0x55)'

```

user@vn-devel:~/contiki-ng$ cd IoT-WSN-Contiki-ng/project/launchpad-nrf-water/launchpad/
user@vn-devel:~/contiki-ng/IoT-WSN-Contiki-ng/project/launchpad-nrf-water/launchpads make TARGET=cc26x0-cc13x0 BOARD=launchpad/cc2650 udp-client-launchpad.upload
load
CC      ./../arch/cpu/cc26x0-cc13x0/lib/cc26xxware/startup_files/ccfg.c
CC      ./../arch/cpu/cc26x0-cc13x0/.iieee-addr.c
CC      ./../arch/cpu/cc26x0-cc13x0/.fault-handlers.c
CC      ./../arch/cpu/cc26x0-cc13x0/lib/cc26xxware/startup_files/startup_gcc.c
CC      udp-client-launchpad.c
LD      build/cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.elf
OBJCOPY build/cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.elf --build=cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.bin
./../.tools/c2538-bsl/c2538-bsl.py -e -w -v -p /dev/ttyACM0 build/cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.bin
Opening port /dev/ttyACM0, baud 500000
Reading data from build/cc26x0-cc13x0/launchpad/cc2650/udp-client-launchpad.bin
Cannot auto-detect firmware filetype: Assuming .bin
Connecting to target...
ERROR: Timeout waiting for ACK/NACK after 'Synch (0x55 0x55)'
ERROR: Timeout waiting for ACK/NACK after 'Synch (0x55 0x55)'
make: *** [udp-client-launchpad.upload] Error 1
rm build/cc26x0-cc13x0/launchpad/cc2650/obj/fault-handlers.o build/cc26x0-cc13x0/launchpad/cc2650/obj/startup_gcc.o udp-client-launchpad.o
user@vn-devel:~/contiki-ng/IoT-WSN-Contiki-ng/project/launchpad-nrf-water/launchpads

```

Figura 2.5: ERROR: [Errno 6] could not open port /dev/ttyACM0

- Error Figura 2.4: Es debido a que no estamos en el modo del bootloader. Solución: Presionar botón 1 + Reset para habilitar el bootloader.
- Error Figura 2.5: Es debido a que contiker no identifica el dispositivo. Solución: Cerrar los procesos de docker contiker abiertos y acceder nuevamente al contenedor. Verificar que la placa está conectada correctamente y contiker la identifica.

2.2. Uso del convertidor analógico digital

Para utilizar sensores externos al launchpad, se hace uso de sus pines del GPIO disponibles. En concreto, dada la naturaleza analógica de la señal de salida del sensor, para este caso se utiliza el pin 'DIO30', dado que es de entrada analógica.

Como resultado, el conexionado final queda de la siguiente manera:



Figura 2.6: Conexionado del launchpad CC2650 y el sensor HW-038

De esta manera, el sensor se conecta al pin de entrada analógica, a la vez que a los pines de 5V y GND del launchpad para alimentarse.

En software, esto se traduce en el uso de los drivers disponibles en contiki para nuestro dispositivo, ubicados en *arch/cpu/cc26x0-cc13x0/dev*. Más concretamente, para el uso del convertidor analógico digital, el fichero *adc-sensor.c*, el cual define la estructura *adc_sensor* del tipo *sensors_sensor*.

Dentro de ella, se utiliza el puntero a función *configure* para seleccionar el canal del comparador y activarlo.

```
1 adc_sensor.configure(ADC_SENSOR_SET_CHANNEL, ADC_COMPB_IN_AUXIO0);  
2 adc_sensor.configure(SENSORS_ACTIVE, 1);
```

A su vez, también se comprueba que el sensor esté correctamente conectado gracias al puntero a función *status*, de forma que el programa mostrará un error del sensor si no fuese el caso que finalizará la ejecución.

```
1 if (adc_sensor.status(SENSORS_ACTIVE) == 0)  
2 {  
3     return SENSOR_ERROR;  
4 }
```

Por último, para obtener el valor de conversión obtenido, se utiliza el puntero a función *value*, el cuál en principio devolverá el valor en microvoltios del ADC. Dado que para nuestra aplicación este valor no es de interés, se ha adaptado su implementación para quedarse con el valor devuelto en crudo por el conversor.

2.3. Caracterización del sensor de nivel de agua

Para este proyecto, se ha utilizado el sensor HW-038, uno de los más extendidos para aplicaciones de bajo coste en detección de agua. Este sensor es muy barato y sus prestaciones son bajas. Para una primera aproximación del sistema, tal y como lo es nuestro prototipo, es útil. Sin embargo, el sensor debería mejorarse de cara a la

fiableidad de las medidas en un pluviómetro real, si se requiere tener mayor precisión.

Su principio de funcionamiento se basa en el fenómeno de resistencia variable que presenta junto al cambio de profundidad del agua al cortocircuitarse sus huellas con la tierra. Gracias al circuito amplificador que estaá formado principalmente por un transistor y unas líneas metálicas en el PCB, la señal de la profundidad del agua es convertida en señal eléctrica, y podemos conocer el cambio de profundidad del agua.

Para caracterizar la respuesta del sensor, se ha realizado un estudio experimental, donde se han obtenido las distintas salidas que el ADC presenta ante los diferentes niveles de agua. El recipiente usado como pluviómetro contiene marcas de los distintos niveles de volumen de agua ocupado en centímetros cúbicos o equivalentemente, en mL.

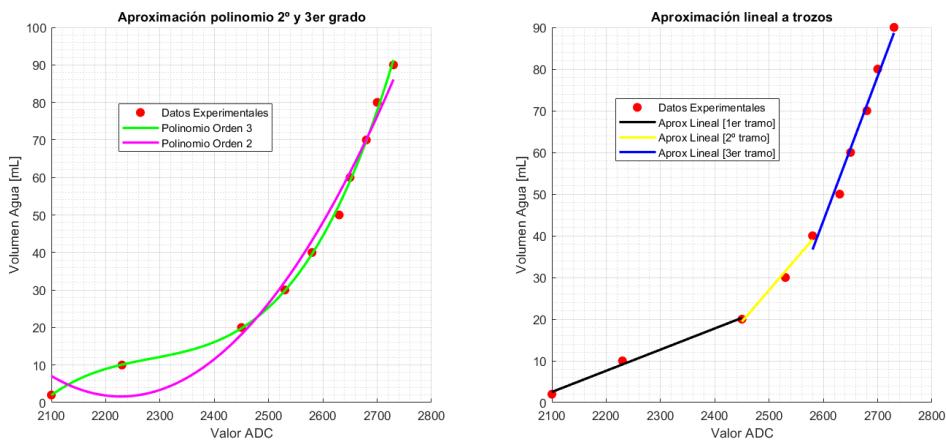


Figura 2.7: Relación experimental de medidas ADC con volumen de agua ocupado

El ajuste de los datos tomados por polinomios mostrados en la Figura 2.7 se ha realizado con la función *polyfit()* de *Matlab*.

Durante las pruebas se han identificado una serie de limitaciones del sensor usado:

- Diferencia abrupta de comportamiento en el rango requerido: En la figura 2.7 puede verse como para volúmenes bajos, pequeñas variaciones producen grandes cambios en el valor obtenido del ADC; mientras que para volúmenes de mayor magnitud, las diferencias obtenidas de lectura en el ADC son más difíciles de distinguir.
- Baja precisión del sensor y medidas poco estables con ruido. Pequeños movimientos en el recipiente provocan cambios notorios en las medidas del sensor debido a la agitación en el líquido.
- Poca robustez frente a perturbaciones: Dependiendo de la composición del agua utilizada, así como su temperatura. La conductividad variará en función de estos factores, lo cuál afecta a la lectura del sensor.
- Tiempo de respuesta del sensor es lento, demorándose varios minutos en alcanzar un permanente en la medida, con gran sobreoscilación (típicamente al

introducir agua se obtienen valores altos del ADC que bajan con el tiempo hasta un valor aproximado a la capacidad ocupada).

En un principio, a partir de los puntos experimentales obtenidos, podría plantearse en uso de un polinomio de 3er grado para obtener un mejor ajuste de la función de conversión que se usará para traducir el valor del ADC a los mililitros de agua recogidos por el pluviómetro. A pesar de esto, debemos tener en cuenta las limitaciones de las plataformas embebidas y redes de sensores de bajo consumo. Si bien es cierto que el microprocesador dispone de FPU para cálculos con mantisa, no es recomendable su uso dado el alto consumo de recursos que supone, lo cual supone una reducción en la autonomía del dispositivo. Por otro lado la aproximación por el polinomio de orden 2 mostrado no es posible debido al mínimo que presenta dicha función, el cual provocaría incoherencias en la conversión.

Debido a todo ello, se ha decidido optar por una solución de compromiso. Dado que se pueden identificar claramente 3 rangos de funcionamiento diferenciados, se recurre a una función a trozos caracterizada por 3 ecuaciones lineales con mejor ajuste a los puntos experimentales de sus respectivos tramos. Esta es una solución simple para el problema y suficiente dado que el cuello de botella en este caso está en el propio sensor, por lo que este resultado debe tomarse como una prueba de concepto. Implementar estas zonas de funcionamiento en la característica estática del sensor nos permite reducir la capacidad de cómputo frente a usar polinomios de orden superior.

2.4. Comunicación bajo estándar IEEE 802.15.4 entre cliente-servidor

Para la transmisión de las lecturas al nrf, se hace uso de las funciones de contiki que implementan la comunicación inalámbrica bajo el estándar IEEE 802.15.4 (capa física y de enlace del modelo OSI).

Tal y como ha venido haciendo en las prácticas de la asignatura, se configura para cliente y servidor el mismo PAN ID y canal, y se hace uso de las funciones *simple_udp_register* y *simple_udp_sendto*, registrando una conexión UDP y asociando un callback como respuesta a los mensajes recibidos, así como enviando como payload un código de estado de respuesta 'OK' al cliente, para confirmar la recepción del mensaje por parte del servidor.

El launchpad (cliente) enviará el valor resultante de la lectura del sensor en mililitros (tras la conversión de las medidas del ADC). La placa dongle nrf (servidor) enviará una cadena por puerto serie con el formato:

```
1 "msg\_to\_mqtt\_exporter: %d\n", *data"
```

Esto permitirá como se explicará en el siguiente punto, filtrar mensajes por puerto serie para parsear las cadenas y que mqtt_exporter sepa por tanto qué cadenas van dirigidas hacia él.

2.5. Despliegue de servicios de visualización y métricas

Para este proyecto se ha mantenido la estructura de bloques planteada en prácticas, tal y como muestra la figura 2.8, utilizando MQTT como protocolo de red. De esta manera, mqtt_exporter publica un tópico, Mosquitto hace de broker, y Prometheus se suscribe a éste para recibir la información y almacenarla. Finalmente, Grafana extrae información de Prometheus y muestra los datos de una manera visual con los widgets de *Gauge* y *Bar Chart*.

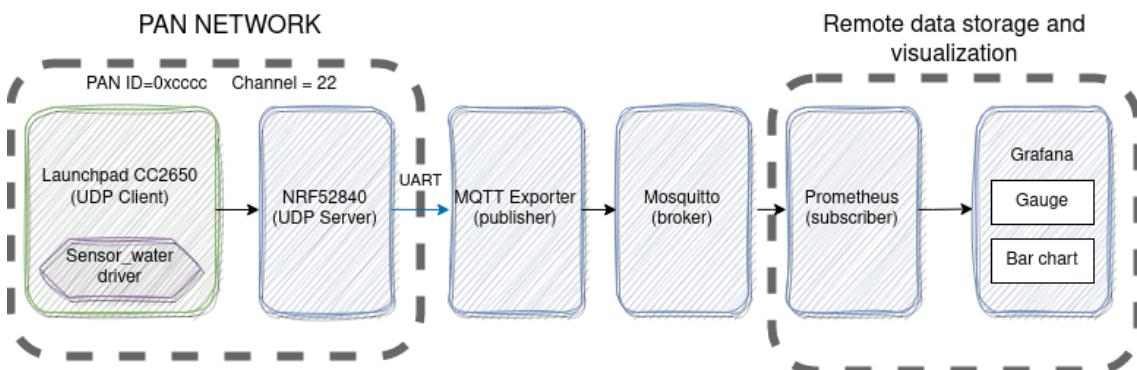


Figura 2.8: Diagrama de bloques

La diferencia fundamental con lo aplicado en anteriores prácticas está en el procesamiento de las líneas leídas por puerto serie. Para este caso, se utiliza como patrón la cadena ”msg_to_mqtt_exporter: ” para ignorar todos los mensajes con un comienzo distinto a este. Para el caso afirmativo, se extrae de la cadena el valor indicado y se publica en el tópico *water_ml*.

El resultado final que muestra Grafana para la interfaz de usuario es:



Figura 2.9: Interfaz en Grafana

Capítulo 3

Conclusiones y prueba de funcionamiento

Para el cierre de esta memoria, se muestra a continuación la imagen del montaje final en la Figura 3.1 donde se distinguen las placas usadas y prototipo de pluviómetro con el sensor de agua.



Figura 3.1: Montaje del sistema

A continuación se listan los resultados de las pruebas realizadas:

Figura 3.2: Mensajes por puerto serie en experimento con 15 mL

En la figura 3.2 se puede ver como el vaso se encontraba inicialmente vacío arrojando un valor de 0 mL y al precipitar agua, en torno a 15 mL, se produce un cambio en los valores obtenidos del sensor.

```
[INFO: Launchpad-client] ADC value: 2671  
[INFO: Launchpad-client] Water measure (ml): 68  
[INFO: Launchpad-client] Server response status code: 'OK' received from fd00::f6ce:3608:25a9:3e38  
[INFO: Launchpad-client] ADC value: 2674  
[INFO: Launchpad-client] Water measure (ml): 69  
[INFO: Launchpad-client] Server response status code: 'OK' received from fd00::f6ce:3608:25a9:3e38  
[INFO: Launchpad-client] ADC value: 2675  
[INFO: Launchpad-client] Water measure (ml): 69  
[INFO: Launchpad-client] Server response status code: 'OK' received from fd00::f6ce:3608:25a9:3e38  
[INFO: Launchpad-client] ADC values: 3  
[INFO: Launchpad-client] Water measure (ml): 0  
[INFO: Launchpad-client] Server response status code: 'OK' received from fd00::f6ce:3608:25a9:3e38  
[INFO: Launchpad-client] ADC value: 1  
[INFO: Launchpad-client] Water measure (ml): 0  
[INFO: Launchpad-client] Server response status code: 'OK' received from fd00::f6ce:3608:25a9:3e38  
[INFO: Launchpad-client] ADC value: 0  
[INFO: Launchpad-client] Water measure (ml): 0  
[INFO: Launchpad-client] Server response status code: 'OK' received from fd00::f6ce:3608:25a9:3e38  
[INFO: Launchpad-client] ADC value: 1  
[INFO: Launchpad-client] Water measure (ml): 0  
[INFO: Launchpad-client] Server response status code: 'OK' received from fd00::f6ce:3608:25a9:3e38
```

Figura 3.3: Mensajes por puerto serie en experimento con 70 mL

Por otro lado, en la figura 3.3 se muestra una medición inicial de unos 70 mL y la actualización de la medida al vaciar el vaso.

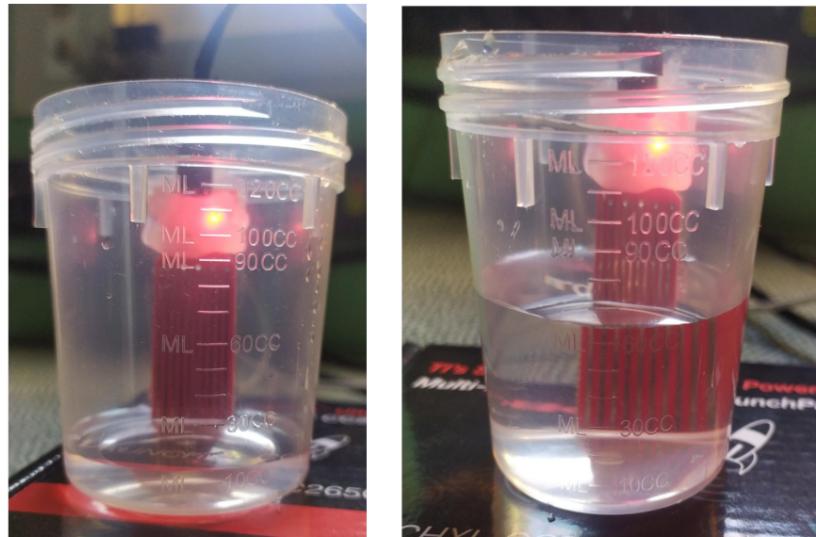


Figura 3.4: Estado final prueba 15 mL y estado inicial prueba 70 mL

```

# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 294.0
python_gc_objects_collected_total{generation="1"} 189.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable objects found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 44.0
python_gc_collections_total{generation="1"} 4.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython", major="3", minor="10", patchlevel="13", version="3.10.13"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 2.779136e+07
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 2.0021248e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.70343478384e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.28
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 15.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP number_msgs_total Number of received messages
# TYPE number_msgs_total counter
number_msgs_total 4.0
# HELP number_msgs_created Number of received messages
# TYPE number_msgs_created gauge
number_msgs_created 1.70343478487169e+09
# HELP water_ml Water Measure [ml]
# TYPE water_ml gauge
water_ml 24.0

```

Figura 3.5: Escucha por el puerto 9000 de MQTT Exporter

En el puerto 9000 de la máquina local, podemos escuchar los mensajes que MQTT Exporter está publicando hacia Prometheus, tal y como muestra la figura 3.5

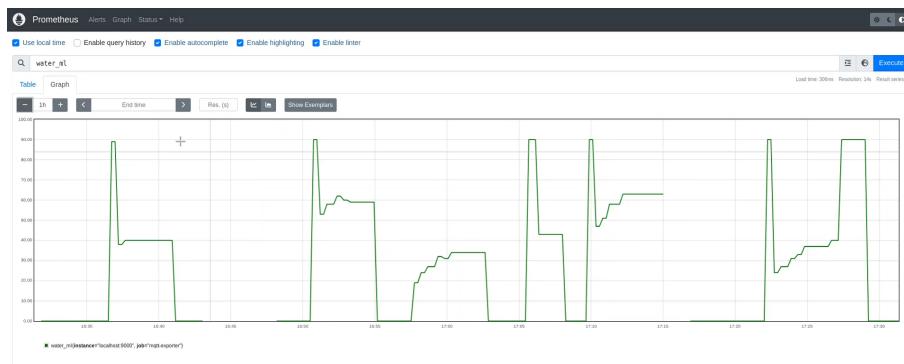


Figura 3.6: Escucha por el puerto 9090 de Prometheus

Mientras que en el puerto 9090, accedemos al servidor web de Prometheus.

Por último, comentar que en Grafana, accesible por el puerto 3000, se puede ver como la medida satura por abajo y por arriba, ajustado a los límites físicos del pluviómetro, para evitar lecturas fuera de rango:



Figura 3.7: Escucha por el puerto 3000 de Grafana con vaso vacío



Figura 3.8: Escucha por el puerto 3000 de Grafana con vaso lleno

A modo de conclusión, este proyecto asienta las bases sobre la posibilidad de desarrollar un producto final al público general. No obstante, tal y como se ha ido mencionando con anterioridad, las medidas del sensor no son lo suficientemente confiables, dado que para un mismo valor de nivel de agua, el convertidor analógico digital arroja valores distintos en función de diversos factores no controlables.

Sin embargo, el proyecto demuestra la posibilidad de desarrollar comunicaciones entre dispositivos embebidos de bajo coste, que alimentados por batería y desplegados por zonas rurales, podrían tener vidas útiles de varios años. Una vez ya programados los dispositivos, la interfaz de usuario es simple, por lo que no requiere de conocimiento previo para su uso.

Capítulo 4

Ampliaciones para producto final

Una funcionalidad interesante para este producto sería la posibilidad del vaciado del pluviómetro en caso de alcanzar el límite de la capacidad. De esta forma se podría medir mayor cantidad de precipitación en caso de que se diera. Se tendría en cuenta a nivel software el número de veces que se ha puesto a cero el dispositivo para estimar la acumulación de agua total.

Para llevar a cabo esta idea una posible alternativa consistiría en la adición de un servomotor en el recipiente que provocara el giro del mismo para derramar intencionadamente el contenido líquido en caso de alcanzar el límite del envase y tras ese movimiento volviera a la posición original para seguir recolectando agua. Esta idea puede verse en la imagen de la Figura 4.1.

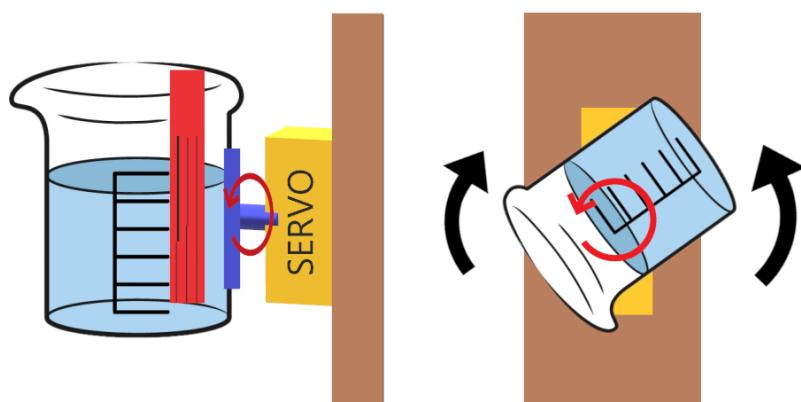


Figura 4.1: Concepto de inclusión servomotor para vaciado del pluviómetro

Incluir un servomotor en el prototipo actual es posible usando alguno de los pines del GPIO del launchpad como salida PWM para controlar la posición del servomotor en función de la medida actual de mL de agua. La inclusión de este actuador, como se ha comentado, puede ser útil de cara a un producto final, aunque diverge de los objetivos de esta asignatura, razón por la cual se ha mostrado simplemente como idea de concepto.

Capítulo 5

Archivos adjuntos

Junto a la memoria de proyecto, se hace entrega del código fuente desarrollado. Dentro de la carpeta **launchpad-nrf-water** adjuntada, se pueden ver los archivos y directorios:

- *launchpad/* : Contiene el código fuente `udp-client-launchpad.c` y `project-conf.h`, los cuales son compilados para generar la imagen binaria que será cargada en el launchpad cc2650 para funcionar como lector del sensor de agua y cliente udp que envía la información recogida en mL, tras ser procesada.
- *mosquitto/* : Original facilitado en las prácticas
- *mqtt_exporter/* : Dentro del cuál el archivo `main.py` ha sido modificado para adaptarlo a nuestra aplicación.
- *nrf/* : Contiene el código fuente `udp-server-nrf.c` y `project-conf.h`, los cuales son compilados para generar la imagen que será cargada en el dongle nrf52840 para funcionar como servidor UDP que recibe mensajes del cliente, y publica por MQTT esta información hacia Prometheus.
- *prometheus/* : Original facilitado en las prácticas
- *docker-compose.yml* : Fichero para la configuración de los servicios de nuestra aplicación, original facilitado en las prácticas.

A su vez, también se proporciona un vídeo demostrativo de la funcionalidad del proyecto, con nombre *demo_pluviometro.mp4*.

Bibliografía

- [1] ContikiNG contributors y maintainers. *cc26x0-cc13x0: TI cc26x0 and cc13x0 platforms*. 2023. URL: http://w3techs.com/technologies/overview/content_language/all.
- [2] Texas Instruments. *FLASH-PROGRAMMER Software programming tool*. 2023. URL: <https://www.ti.com/tool/FLASH-PROGRAMMER>.
- [3] Texas Instruments. *UniFlash*. 2023. URL: <https://www.ti.com/tool/UNIFLASH>.