Sistemas de Tiempo Real
2018

Proyecto Final
Sistema de estacionamiento asistido

Abba, Pedro Nicolás
Ares, Charo
Galán, Martín Andrés

UNIVERSIDAD
NACIONAL
DE LA PLATA

Introducción:

El proyecto que se nos ha asignado como trabajo final de la materia ha sido la implementación de un **sistema de estacionamiento asistido**. El principal objetivo es modelar este sistema tan difundido en la industria automotríz utilizando componentes electrónicos básicos proporcionados por la cátedra y programar un microcontrolador para que dicho sistema reaccione a las variables reales que se toman del medio, en un tiempo acotado y acorde.

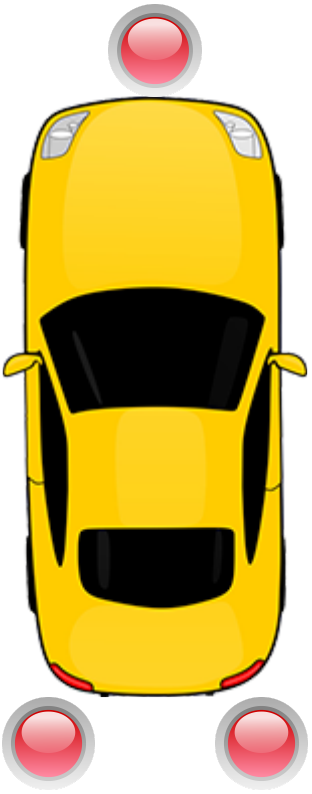
La **consigna** original fue la siguiente:
Se debe desarrollar un dispositivo que simule el funcionamiento del sistema de alarma de estacionamiento de los autos.
Debe tener un botón que inicie el censado y uno que lo corte.
Al iniciar el censado se debe comenzar un timer para contar el tiempo que se tarda en estacionar.
Se deben encender las luces correspondientes al iniciar el dispositivo.

Teniendo en cuenta esto, nos dispusimos a diseñar un sistema cuyo funcionamiento se centra en un microcontrolador conectado a sensores ultrasónicos que recogen datos sobre distancias y se informan a través de leds, y adicionalmente, una aplicación Android conectada mediante bluetooth. Como especifica la consigna, utilizamos un botón para prendido y apagado. Cuando el sistema se enciende, se inicia un timer y se muestra dicho tiempo en la app.

En el siguiente informe se detallan los objetivos, el hardware utilizado y el software del proyecto así como los resultados finales de nuestro sistema de estacionamiento.

Diseño de la solución:

Para resolver el problema provisto por la cátedra decidimos modelar el sistema como un “vehículo” con tres sensores ultrasónicos ubicados uno por delante al centro y dos por detrás y a los costados ya que nos pareció la disposición ideal para optimizar la cantidad la cantidad de sensores y a la vez obtener toda la información necesaria para un estacionamiento exitoso. En el siguiente gráfico cada punto rojo identifica la posición de los sensores.



A su vez cada sensor tendrá asociado un led que parpadeará dependiendo de la distancia que tome el sensor.

Para mostrar los datos decidimos añadir valor agregado al proyecto y crear una aplicación Android que reciba del sistema los datos a través de un módulo Bluetooth y los muestre en una interfaz intuitiva y agradable.

Se nos preguntó por preferencias para a la hora de escoger el microcontrolador que lleve adelante el sistema y respondimos que podría ser cualquier Arduino ya que para este proyecto en particular no requerimos de tanta capacidad de procesamiento ni de tantos pines. Dado esto nos ofrecieron un Arduino UNO R3 y lo aceptamos.

Por último decidimos que para el encendido y apagado del sistema sería suficiente con un solo botón para ahorrarnos pines y conexionado. Solo debemos tener cuidado con el rebote que puede producir el pulsador.

Requisitos del proyecto:

Los requisitos que nos planteamos para el proyecto son:

-Poder encender el sistema a través de un botón y que inmediatamente se inicie un timer para contar el tiempo que se tarda en estacionar.

-Poder apagar el sistema a través del mismo botón y que inmediatamente se detenga el timer para poder saber cuánto tiempo tardó el usuario en estacionar.
.
-Sensar cada cierto tiempo uno a uno la distancia de los sensores de ultrasonido a su objeto más cercano, para que el sistema tenga información en tiempo real y pueda tomar acción si un objeto se encuentra demasiado cerca.

-Encender leds para indicar que existe una distancia corta entre uno o más de los sensores ultrasónicos y sus objetos más cercanos y de esta forma evitar que el usuario “choque” contra dichos objetos.

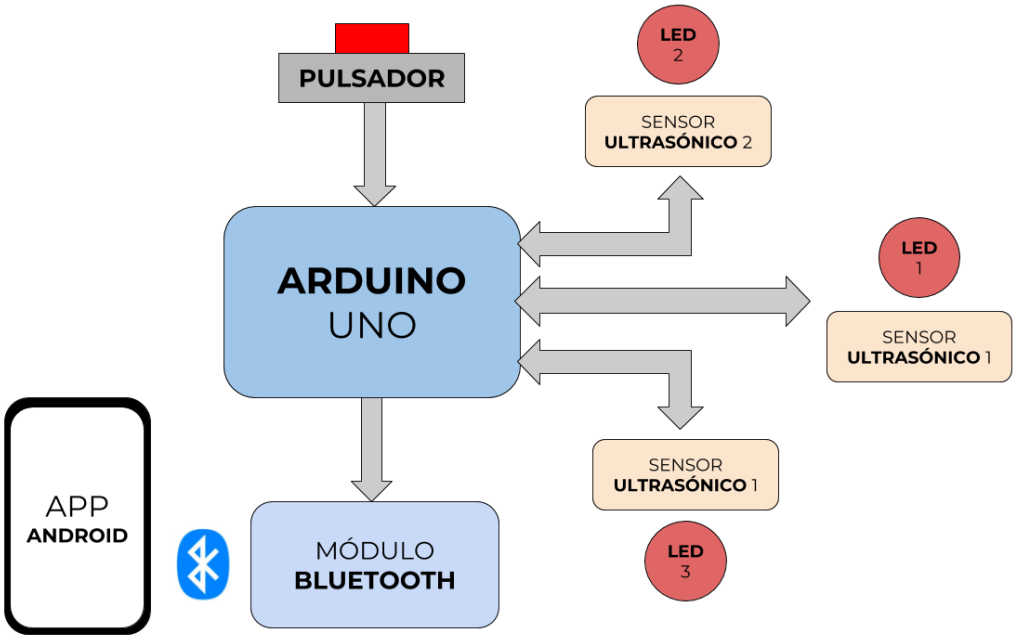
-Enviar datos a través de un dispositivo Bluetooth a una aplicación Android creada por nosotros: las distancias tomadas por cada sensor, el tiempo de estacionamiento.

-Informar las distancias de cada sensor en la App en formato de texto y complementarlo con un gráfico del estado de los sensores.

-Llevar un registro histórico de los estacionamientos recientes dentro de la aplicación.

Diseño del hardware

Diagrama en bloques del sistema



Para nuestro proyecto vamos a requerir los siguientes componentes:

- Arduino UNO R3
- 3 sensores ultrasónicos HC-SR04
- 3 LED rojos
- Módulo bluetooth HC-05 ZC-040
- Pulsador 2 patas
- Cables macho-macho
- Cables hembra-hembra
- 1 resistencia de 10k ohm para el pulsador
- 3 resistencias de 120 ohm para los leds
- Protoboard para armado del prototipo

Se presentarán uno a uno los componentes.

Arduino UNO R3

Características:
Microcontrolador ATmega328.
Voltaje de entrada 7-12V.
14 pines digitales de I/O (6 salidas PWM). 2 pueden producir interrupciones.
6 entradas análogas.
32k de memoria Flash.
Reloj de 16MHz de velocidad.

Como se mencionó anteriormente este modelo de Arduino nos alcanza y sobra para el proyecto que estamos encarando: la velocidad del reloj es más que suficiente para temporizar las tareas, la cantidad de pines está bien ya que utilizaremos los analógicos como si fuesen digitales y sólo requerimos de un pin que produzca interrupción (la del pulsador).

Sensores ultrasónicos HC-SR04

Este dispositivo posee 4 patas que se conectarán al microcontrolador: VCC que conectaremos a un pin que entregue 5V constantemente; GND que conectaremos a un pin de tierra; TRIG cuya función es que, al mandarle una señal en alto, se desencadena el proceso de envío de la señal ultrasónica; y ECHO que nos “devuelve” el tiempo que tarda la señal en volver al sensor.

En nuestro diagrama conectamos:

VCC y GND: Pines 5V y GND. **ECHO:** Pines 9 y 11 (para el prototipo solo utilizamos 2 sensores). **TRIG:** Pines 8 y 10

Multiplicado el dato que obtenemos como entrada del pin ECHO por la velocidad del sonido (340 m/s) y dividiendo por 2 conseguimos la distancia del sensor al objeto más cercano, lo que será la base de todo el sistema.

3 LED rojos

Los LEDs de este sistema son incorporados para informar visualmente al usuario sobre las distancias que captan los sensores ultrasónicos. Son genéricos, de color rojo, por lo que teniendo en cuenta que el Arduino entrega aproximadamente 5V por cada uno de sus puertos de salida debemos incorporar **3 resistencias de 120** ohm al conexionado para proteger los componentes.

En el conexionado una pata del LED se conecta a tierra y la otra a un pin que controlará su encendido o apagado pasando por la resistencia ya mencionada. Estos se conectan a los pines 5, 4 y 3 del Arduino.

Módulo bluetooth HC-05 ZC-040

Utiliza el protocolo UART RS 232 serial. Para el envío de datos deberán usarse los comandos AT que son ciertos comandos que el dispositivo espera para poder cumplir con el envío. Compatible con el protocolo Bluetooth V2.0. Voltaje de alimentación: 3.3 V – 6 V. Voltaje de operación: 3.3 V.

Este dispositivo posee 6 patas, 4 de las cuales se conectarán al microcontrolador: RX de Arduino se conectará a TX del módulo, y será por donde se enviarán mediante comunicación serie asincrónica los datos; análogamente TX de Arduino se conectará a RX del módulo; VCC a 5V del micro y GND a GND del micro.

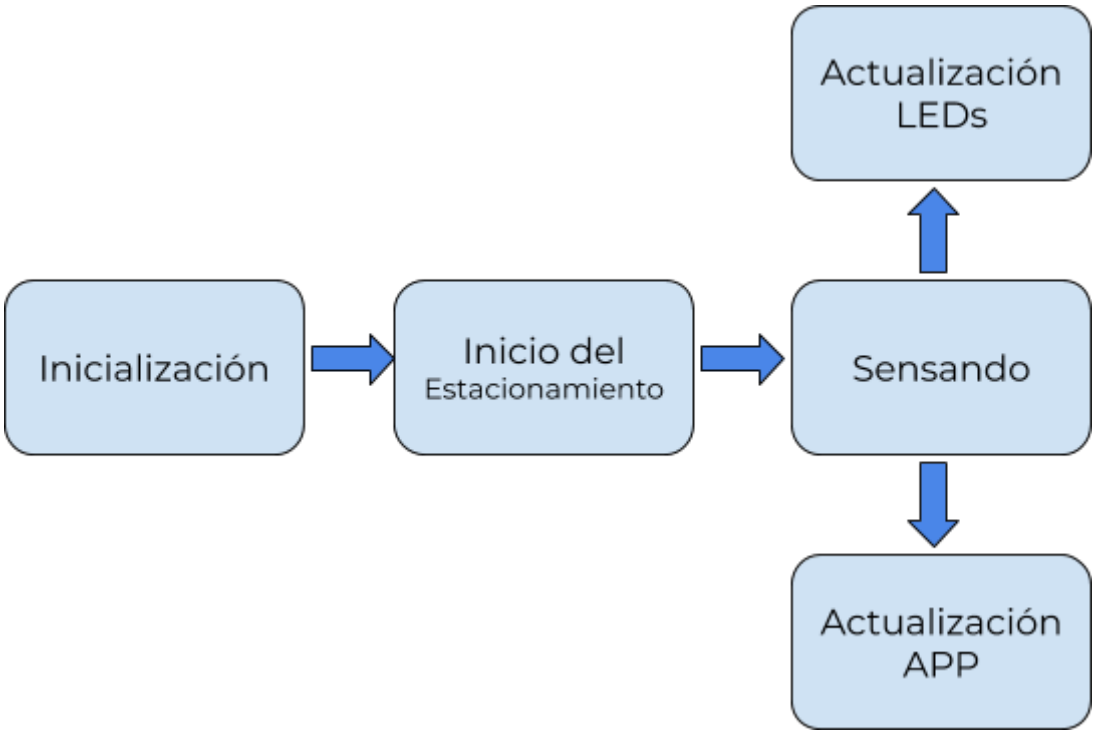
STATE y EN no requieren ser usados en esta ocasión ya que el dispositivo siempre tomará el rol de esclavo quedando a la escucha de peticiones de conexión. En el modo maestro es él quien se conecta otros módulos bluetooth.

Pulsador 2 patas

Para encender y apagar el timer que pide la consigna, que calcula el tiempo que lleva el estacionamiento actual, la cátedra nos proporcionó un pulsador genérico que posee dos pines: uno estará conectado directamente a GND. El otro se conectará a través de una **resistencia** de pull up **de 10k ohm** a una tensión VCC. A la salida de este circuito se conectará un pin del micro capaz de generar interrupciones para que el sistema reaccione lo más rápido posible al pulsador.

Diseño del Software

El Software del sistema se basó en la siguiente idea:



Inicialización. (*void Init(void)*). Se inicializan los componentes: ***Init_Led().*** los LEDs comienzan apagados y sus puertos como salidas. ***Init_BLE().*** Establezco la comunicación. Defino y envío las distancias de los sensores como 0. Defino y envío los minutos y segundos como 0. ***Init_Pulsador().*** Se activa la resistencia de pull-up interna del puerto 2 para el pulsador. Se define una interrupción al evento de bajada en el pin 2 (uno de los dos pines capaces de generar una interrupción), y en la función ISR asociada iniciamos el timer.

Inicio del estacionamiento. Se espera a que se detecte que se ha tocado el pulsador a través de la interrupción. Cuando esto sucede se pone en alto la variable *estadoSistema* que desencadena el sensado, el envío de los datos y el encendido de un led del arduino (pin 13) para indicar visualmente que el sistema inicio . También existen los flags flagSensor, flagReloj y flagLed[3] que sirven para temporizar las tareas (Arquitectura Time-Triggered). El Timer1 del Arduino es configurado para interrumpir cada 100 ms. Cada vez que se interrumpe se incrementan los ticks del sistema. Cuando se cumplen una cierta cantidad de ticks estos flags se ponen en alto y producen envíos de actualizaciones a la aplicación o el comienzo del sensado.

Sensado, Actualización de LEDs y de APP. Se explican detalladamente en las próximas páginas.

Implementación del Sensado con sensores Ultrasónicos

Este sistema se basa enteramente en los 3 sensores de ultrasonido colocados estratégicamente para cumplir la función de medir las distancias del sensor al objeto linealmente más cercano.

Por esta razón es fundamental que el sensado sea cada un tiempo adecuado: para que el sistema sea de tiempo real y reaccione con una respuesta adecuada en el momento justo.

El tiempo que nos pareció prudente para tomar los datos de los sensores y tomar acción según su valor fue de 500 ms. Dentro de ese período cada uno de los tres envía una señal ultrasónica, se recepciona el eco de la misma, se calcula la distancia en función del tiempo de retorno, se envían los datos a la aplicación y se mantiene o cambia el estado de los leds indicadores.

La función que se encarga del sensado y el retorno de las distancias es ***void comenzarSensado(void)*** que modifica un vector de dimensión 3 con los valores de las distancias medidas por cada sensor.

Luego estas distancias se envían a la aplicación a través de la función ***EnviarSensor()*** que se describirá más adelante en el informe.

Una vez terminado el pasaje de los datos se ejecutará la función ***ActualizarLed()*** que, dependiendo de las distancias sensadas se mantendrán apagados o titilarán a distintas frecuencias:

Para este prototipo:

Distancia mayor a 40 cm ➔ LED apagado

Distancia entre 40 y 30 cm ➔ LED titila cada 400 ms

Distancia entre 30 y 20 cm ➔ LED titila cada 300 ms

Distancia entre 20 y 10 cm ➔ LED titila cada 200 ms

Distancia entre 10 y 0 cm ➔ LED titila cada 100 ms

Cuando se terminan de actualizar los LED físicos se pasa a enviar la información de los LED a la APP. Por simplicidad solo enviamos un true si el LED está encendido y un false caso contrario, mediante la función ***EnviarLed(boolean led1, boolean led2, boolean led3)***.

Comunicación Bluetooth y Aplicación Android

Para complementar la parte física del sistema quisimos añadir una parte de software para mostrar los datos en de forma clara y con una interfaz intuitiva y amigable.

La forma en la que los datos llegan a la aplicación es a través de la comunicación serie del módulo Bluetooth. Periódicamente el programa envía los datos a la salida del HC-05 ZC-040 y la aplicación, habiendo activado previamente el Bluetooth del celular y establecido la conexión, la aplicación los recibe y los muestra gráficamente.

Envío de los datos

La aplicación debe recibir varios tipos de datos:

- Señal de Start
- Estado de los LEDs
- Distancias de los sensores
- Minutos y segundos del Timer
- Señal de Stop

En primer lugar inicializamos el módulo de bluetooth como normalmente con la función void ***InitBLE()*** que pone el baud rate en 9600 y envía 0 a las distancias de los sensores y ***void ConfigurarBT()*** función con la cual comenzamos la comunicación, configuramos el nombre, la contraseña, el rol, etcétera con los comandos AT. Luego, cuando se inicia el estacionamiento presionando el pulsador comienza el envío de los datos.

Los datos llegan al módulo Bluetooth mediante una función ***void EnviarDato (String, int)*** a la que se le pasa como parámetro un String y un entero que indica cuál de los parámetros se está enviando:

INDICE_RELOJ → 1 ; INDICE_SENSOR1 → 2 ; INDICE_SENSOR2 → 3 ;INDICE_SENSOR3 → 4 ; INDICE_LED1 → 5 ; INDICE_LED2 → 6; INDICE_LED3 → 7 ; INDICE_STOP → 8

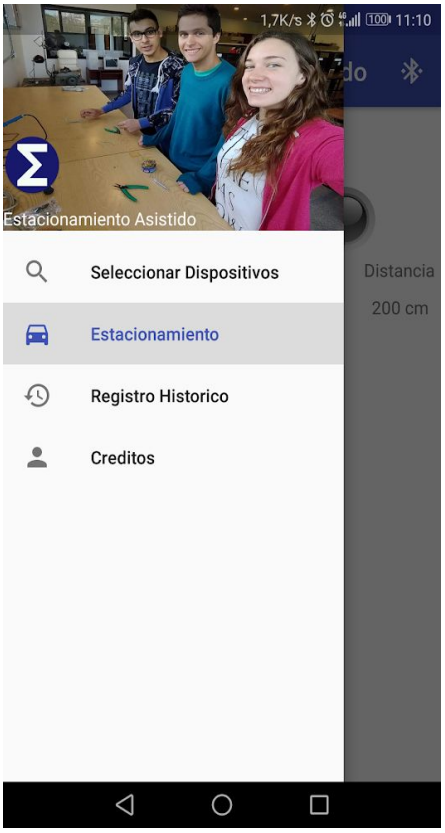
Esta es invocada desde las funciones: *void **EnviarReloj**(int minuto,int segundo); void **EnviarSensor**(int sensor1, int sensor2, int sensor3) ; void **EnviarLed**(boolean led1, boolean led2, boolean led3) ; void **EnviarStop**()* con los datos corresponientes.

En cada llamado a dicha función concatenamos el índice, la información correspondiente, le agregamos un '#' indicando el fin de la cadena y lo enviamos como un String con la función ***BTserial.print(String);***

Interfaz de la APP

-Menú desplegable para moverse entre las distintas vistas

Existen 4 vistas: una para seleccionar el dispositivo Bluetooth, la central que es la del estacionamiento en curso, la tabla de registro de estacionamientos y los créditos de la aplicación.

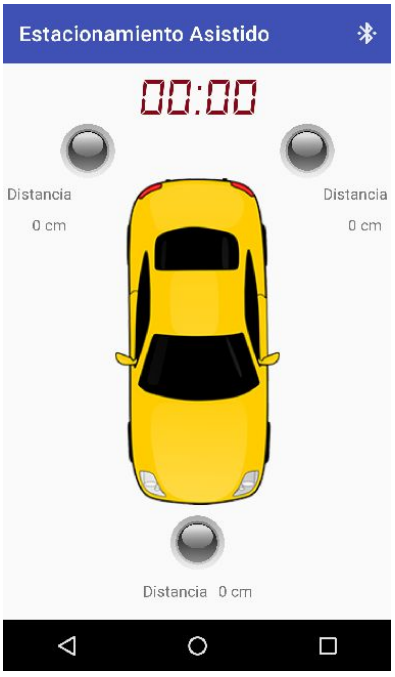


-Vista para seleccionar el dispositivo Bluetooth con el que se va a vincular.

-Vista del Estacionamiento Asistido

Indica el tiempo que lleva el estacionamiento (00:00 si todavía no comenzó), la distancia de cada sensor al objeto más cercano en centímetros y el estado de cada sensor (encendido si la distancia del sensor al objeto es menor a 40cm).

Estado Inicial:



Estacionamiento en curso:

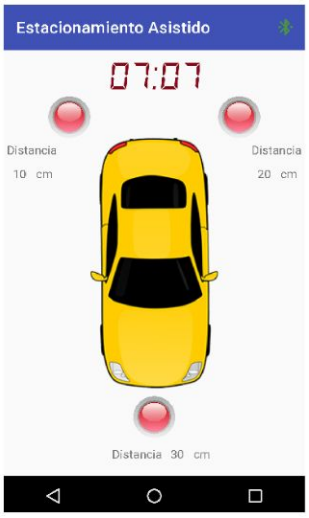


Tabla de Estacionamientos Registrados:

Cuando se finaliza el estacionamiento se registra el tiempo en una tabla.

Tiempo	Fecha	Hora
00:20	06-10-2018	17:43:42
00:40	06-10-2018	17:44:02
01:20	06-10-2018	17:53:17
01:40	06-10-2018	17:53:31
11:11	03-10-2018	22:43:51
13:21	03-10-2018	22:44:29
22:22	03-10-2018	22:44:04

En esta se guardan el tiempo que se tardó en estacionar, la fecha y la hora del estacionamiento.
La tabla se puede ordenar por fecha o por tiempo de estacionamiento.

El código de la APP está subido a GitHub con el link:
https://github.com/Galandrus/EstacionamientoAsistidoApp

Inconvenientes

El primer inconveniente con el que nos encontramos fue el hecho que teniamos 2 sensores ultrasónicos en vez de los 3 que habíamos dicho, pero lo solucionamos fácilmente sensando dos veces con el mismo sensor, es decir que los dos sensados de la parte trasera del vehículo son realizados por el mismo sensor, pero en dos oportunidades diferentes.

El segundo inconveniente fue el efecto rebote del pulsador, debido al ruido que se produce en los flancos de señal por el choque de las placas. Dicho ruido puede generar múltiples disparos de una interrupción y por lo tanto interferir el flujo normal de nuestro programa.

Logramos atenuar el problema realizando ciertas verificaciones por software cuando interrumpía el botón.

El tercer inconveniente se nos apareció cuando probamos todo el sistema funcionando a la vez. Si bien cada módulo funcionaba perfecto por separado, para nuestra sorpresa no ocurrió lo mismo cuando los aunamos. La aplicación Android no daba abasto para cumplir en tiempo y forma los envíos realizados por el Arduino, es decir que los datos se mostraban con un gran delay, ocasionando un bajo rendimiento del sistema en general. Para solucionar este problema o tratar de atenuarlo, procedimos a enviarle la mínima e indispensable cantidad de datos posibles, una forma de lograr esto fue no volver a enviar el dato si era igual al enviado anteriormente, ya que se mostraria lo mismo y no valía el esfuerzo.

RTOS

Una vez que tuvimos el sistema funcionando correctamente nos propusimos implementarlo con la librería FreeRTOS. Idealmente decidimos que haya una tarea para el reloj, una para los sensores y una para cada uno de los leds, como así también un semáforo para cada una para lograr tener sincronización entre las mismas y que no se vea afectado el flujo normal del sistema.

Desgraciadamente no pudimos hacerlo funcionar como pensamos, porque por alguna razón nunca se llegaban a ejecutar las tareas, estando creadas y con el planificador activo. Llegamos a la conclusión a base de prueba y error que no se podían crear más de 3 tareas porque el programa se “rompía”.

Teniendo en cuenta lo dicho anteriormente tampoco pudimos hacer que funcione, a lo que se nos ocurrió que la interrupción del Timer podría estar interfiriendo con la ejecución de las tareas. Para ello creamos una tarea que supla la función del Timer, tratando de simular los ticks con ayuda de la función *vTaskDelay().* Luego creamos dos tareas más, una para controlar el reloj y otra para manejar los sensores junto con sus leds. Esta solución funcionaba pero con un muy bajo rendimiento, porque la tarea Timer no proporcionaba el tiempo exacto para poder implementar el reloj eficazmente y además no se podía enviar dato alguno a la aplicación.

Bibliografía

Datasheets

Arduino UNO R3:
http://www.fecegypt.com/uploads/dataSheet/1522237550_arduino%20uno%20r3.pdf

HC-SR04:
https://components101.com/sites/default/files/component_datasheet/HCSR04%20Datasheet.pdf

HC-05:
https://electronilab.co/wp-content/uploads/2014/02/HC-0-Bluetooth-Electronilab-datasheet.pdf