SISTEMAS EMBEBIDOS
Manual Tecnico

Docentes: Bruno Bellini
Felipe Estevez

Integrantes del equipo: Juan Valiño
Martin Fontes
Diego Silveira

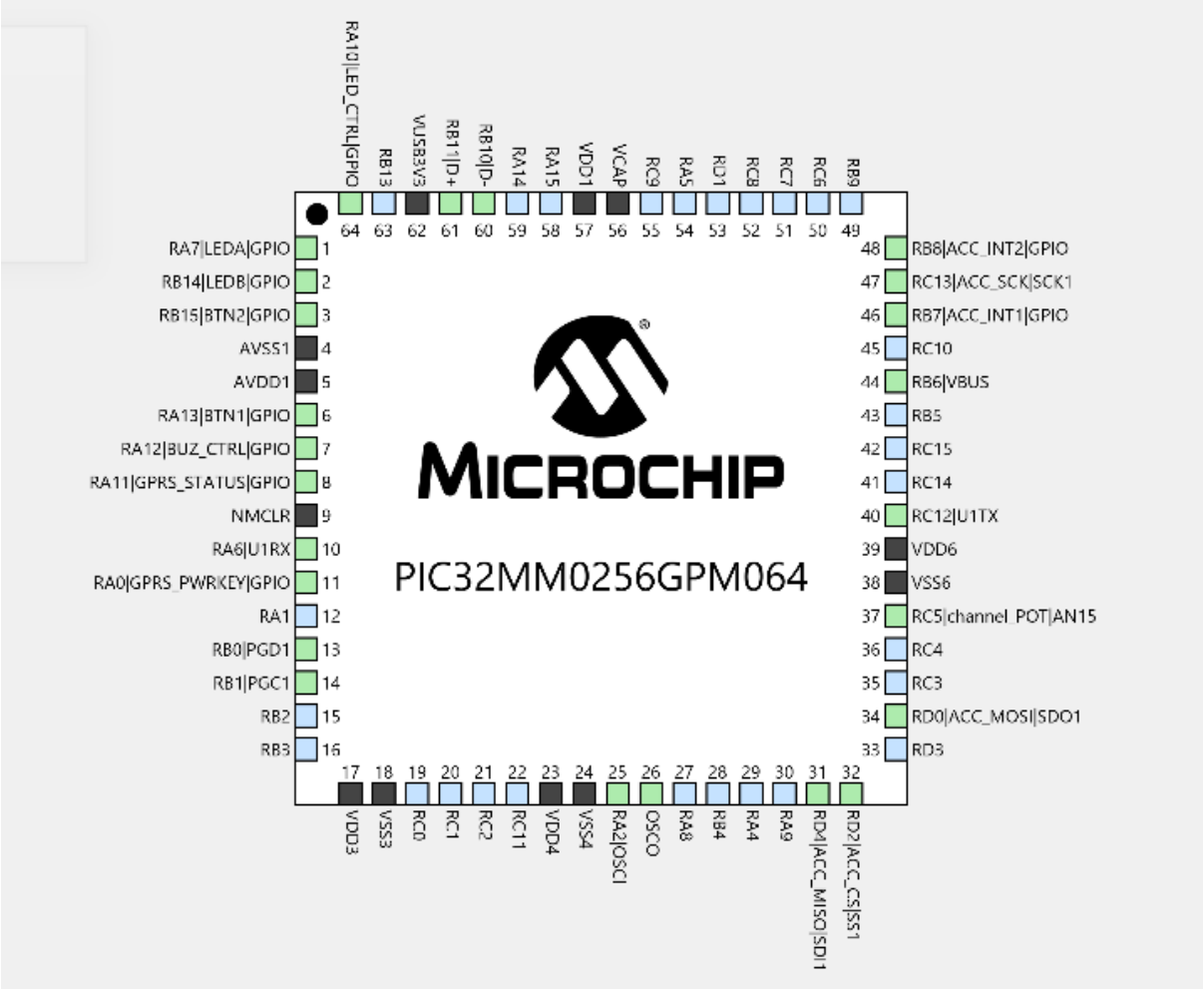
- AÑO 2021 -

Requerimientos de hardware y software para el desarrollo

Hardware

Como requerimiento de hardware, es necesario contar con la placa de desarrollo la cual tiene implementada un microprocesador PIC32MM0256GPM064.

Será necesario que se utilicen las mismas placas de desarrollo que fueron distribuidas por parte de la Universidad Católica del Uruguay debido a que el desarrollo se basa particularmente en dicho modelo. Esto incluye trabajo directo con pines particulares que pretenden habilitar distintos módulos de la placa, y por ello es esencial que se mantenga el mismo tipo de placa y no uno distinto, configurado de la siguiente forma:



Software

Para el software será necesario contar con las siguientes aplicaciones dentro de nuestro ambiente de desarrollo:

- MPLAB IDE (se recomienda versión 5.45 o mayor).
- Un compilador de C correspondiente al microprocesador (se recomienda MPLAB XC32/32++ v3.01 o superior).
- Hercules u otra aplicación que permita comunicaciones a través del puerto serial (como puede ser putty).

Por otro lado, cabe destacar que dentro de la placa se estará haciendo uso del sistema operativo para sistemas embebidos *FreeRTOS*.

Arquitectura general del sistema

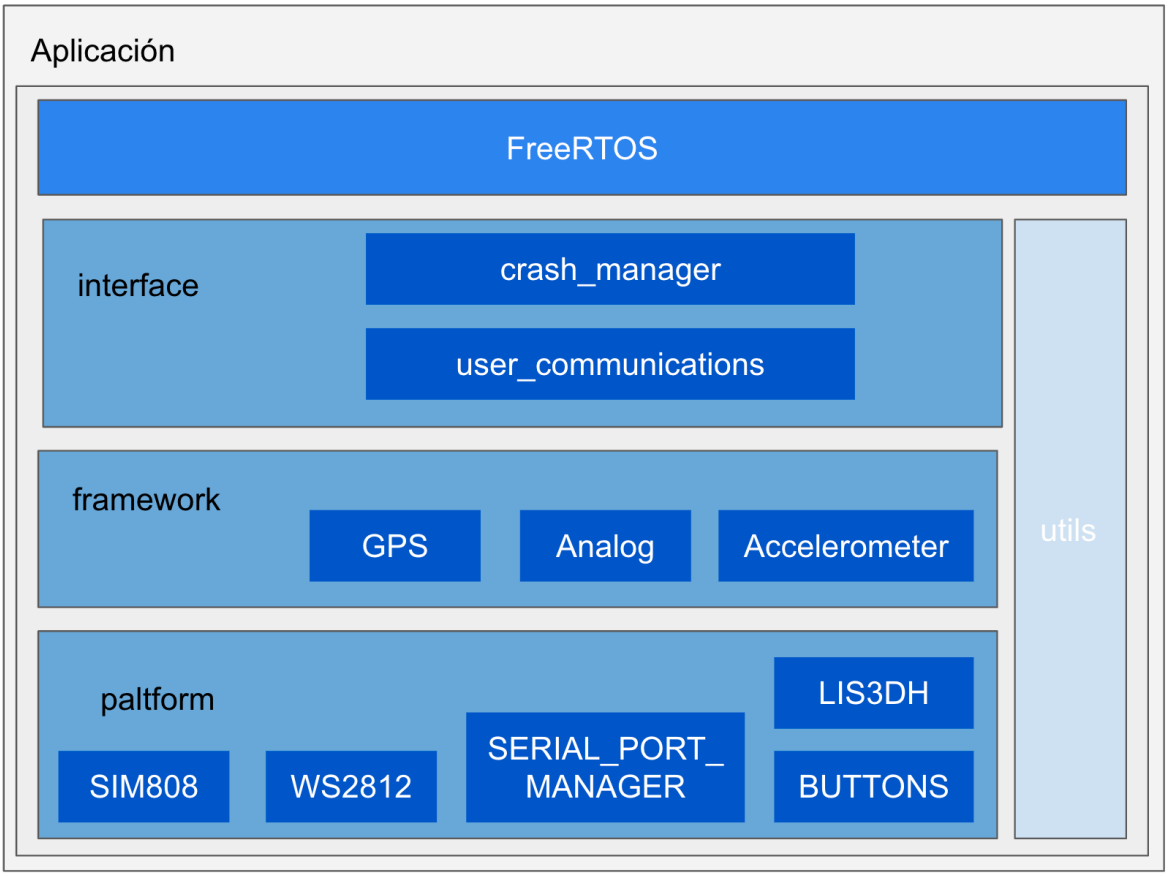


Imagen 1: diagrama de aplicación por capas.

La aplicación es desarrollada en capas. La capa de más bajo nivel contendrá todas las funciones que sean utilizadas mayoritariamente para interacción directa con el hardware. Esta capa se trata de platform y alli deberian agregar cualquier otro modulo que tenga como propósito la interacción directa con los distintos elementos de la placa.

Por otra parte se encuentra la capa que resulta como intermediaria para la interacción con el hardware, pero que no hace un uso directo del mismo. Esta capa es framework. Allí se deberán agregar todo modulo que haga utilización por ejemplo de la capa platform y a partir de ello defina funciones de utilidad para el resto de la aplicación.

Por último está la capa de más alto nivel, que es la que utiliza el resto para poder comunicarse con el usuario. Deberán agregarse a la misma los distintos módulos que se encarguen de trabajar con el usuario.

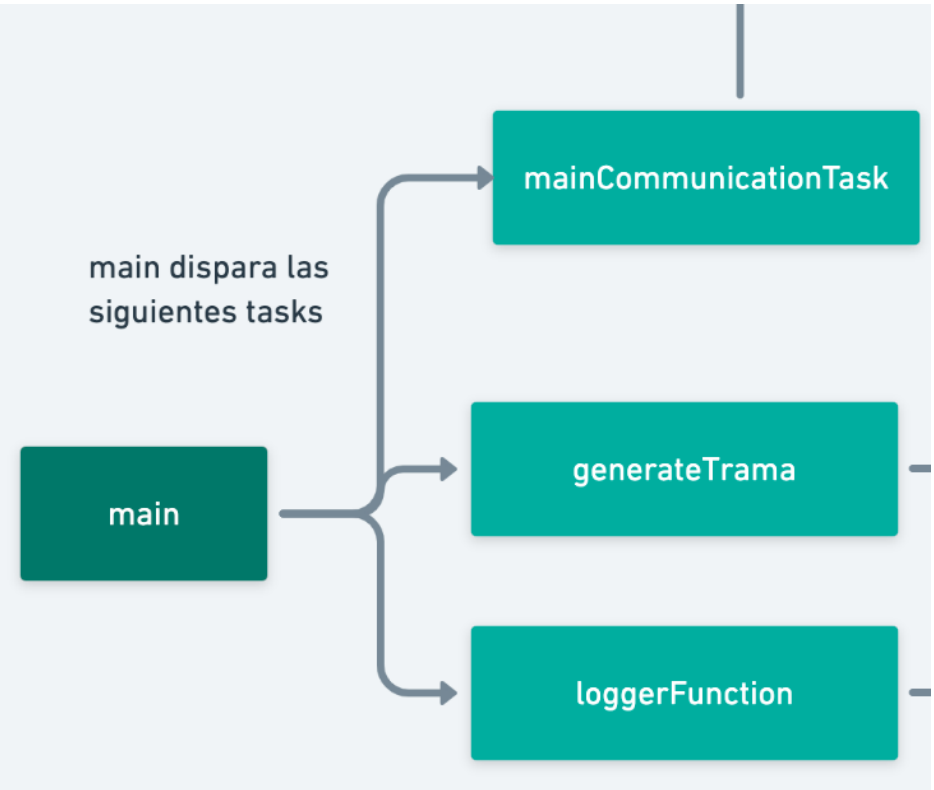
De forma paralela, se trabaja con utils, este módulo se encarga de tener todas aquellas funciones que sean de utilidad para el proyecto y sus distintas capas, y el mismo es independiente al nivel donde se encuentre, ya que cuenta con funciones que no merecen

tener un módulo propio, pero si son de alta utilidad para mantener un programa legible y sin funcionalidades repetidas.

Por último se encuentra FreeRTOS, es quien se encarga de supervisar todas aquellas tasks que sean iniciadas por los distintos módulos, por ello sería una capa que abarca a todo el resto, como si fuera una “sombrilla” hacia la aplicación.

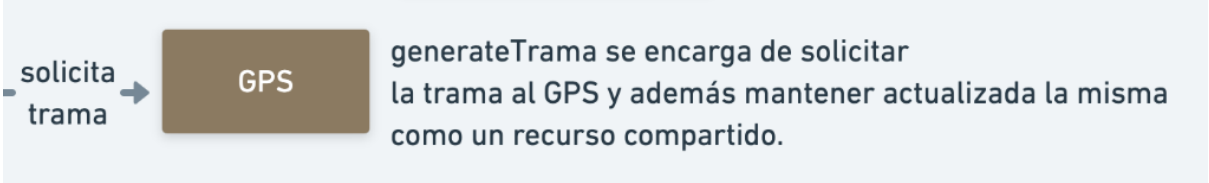
Diagrama de funcionamiento

En el siguiente diagrama se busca de forma **básica**, explicar el funcionamiento del sistema. Esto no estará 100% detallado, pero se busca orientar a futuros desarrolladores a poder tener un conocimiento básico de como se generan las cadenas de invocaciones ante los distintos eventos.
Si se desea visualizar el esquema por completo, se puede hacer [click aqui.]



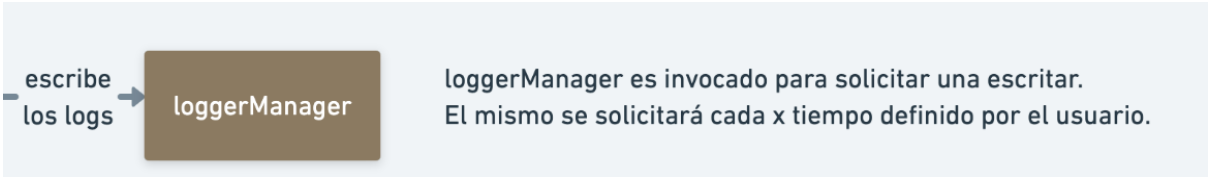
La primer imagen corresponde al inicio del programa. El mismo comienza desde la función main que se encarga de llamar a las tasks principales del mismo, cada una de estas tasks tienen responsabilidades distintas.

Comencemos por la más básica, generateTrama.



generateTrama se encarga de solicitar al GPS constantemente por nuevas tramas y así poder actualizar el recurso compartido para que otras tareas puedan solicitar donde se encuentra ubicado el sistema. Es **importante** destacar que la primera invocación a generateTrama, la tarea se va a encargar de definir el horario inicial del RTCC para que podamos contar con la fecha y hora real y no la ficticia con la que se inicia el sistema (01/01/2021).

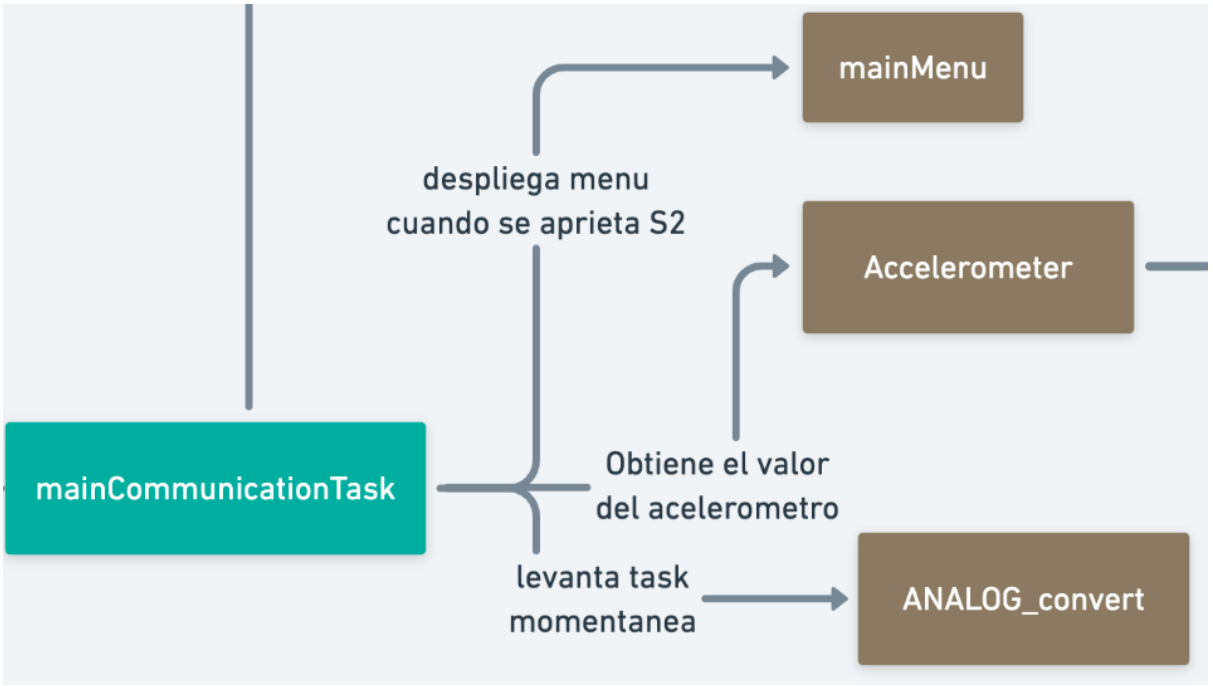
Por otra parte, seguimos con loggerFunction:



Esta tarea va a invocar a loggerManager como modo escritura cada un periodo que se define por el usuario. Por defecto viene cada 10 segundos. Se genera un log en una arreglo de 250 lugares donde se guarda un struct predefinido para tener todos los datos necesarios. Dicho arreglo cuando se llega a 250, comienza a escribir desde la posición 0 sin borrar el resto de los registros ya almacenados.

Ahora veremos mainCommunicationTask parte a parte:

En caso de que sea solicitada la impresión de los logs que mencionamos anteriormente, la task se encarga de invocar al loggerManager, pero esta vez en modo impresión para poder leer todo lo que se encuentra en el arreglo e imprimirlo. Se va a leer hasta la posición donde haya datos escritos. Si solo se tienen 100 registros, se leerá esos 100 registros. En modo impresión **solo** lee, por lo que no se hace ninguna modificación al arreglo.



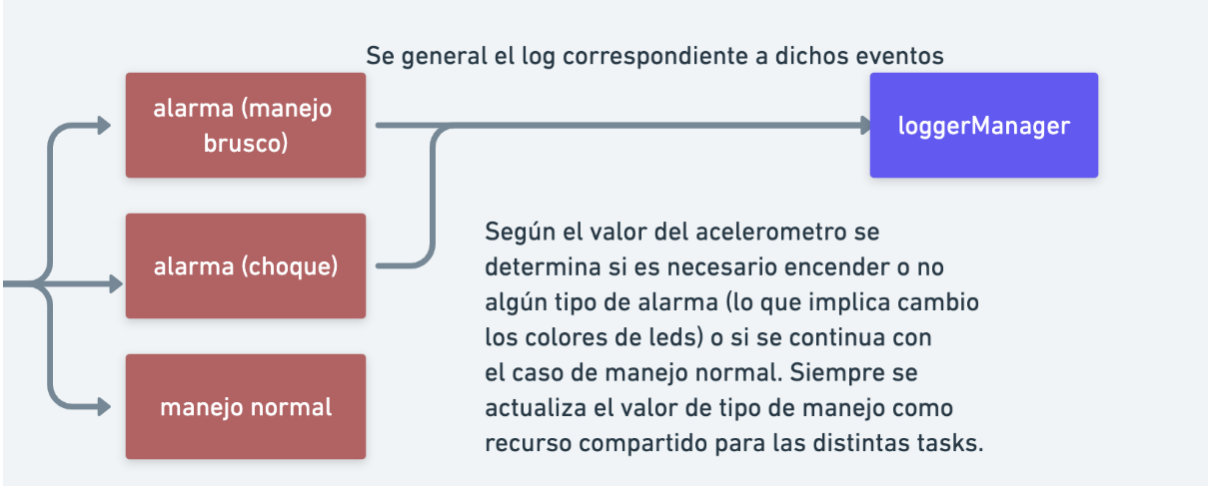
Además de la función de imprimir los logs, la task también se encarga de desplegar el menú principal cuando se apreta el botón S2.

La task es responsable de mientras no se le solicite ninguna otra funcionalidad, estar trabajando constantemente en leer el acelerómetro para detectar desvíos de manejo (que entren dentro de los umbrales), y en base a ello disparar los eventos correspondientes.

Por último, si se solicita el cambio de umbrales, levantará una task momentánea que se encarga de leer el ADC para poder generar el seteo de los umbrales según como el usuario vaya moviendo la “ruedita”.

Hay que tener en cuenta que dentro de mainCommunicationTask se realiza el chequeo de si es necesario seguir cualquiera de estos flujos en una arquitectura **round robin.** Esto nos asegura darle la misma prioridad a chequear que todos estos eventos sean necesarios de atender. Se parte bajo la premisa que el chequeo del S2 y del acelerómetro son funciones que no tienen gran costo de consulta (es decir, no bloquean significativamente a la task) y por ello pueden convivir entre sí en un round robin.

Ahora seguiremos con el flujo que se da en el acelerometro:



Cuando es consultado dicho acelerómetro, se puede detectar que el mismo está con un nivel de manejo:

- que supere el umbral de choque
- que se encuentre superior a un manejo normal y menor a un choque (manejo brusco)
- que se encuentre por debajo del manejo brusco y por ende sea un manejo normal

En los casos que no se trate de un manejo normal, el sistema lo que hará será llamar nuevamente a loggerManager como escritura para poder generar el loggeo correspondiente a estos eventos. Es importante destacar que como son dos tasks distintas que están escribiendo el mismo arreglo, estas tienen un semáforo que se encarga de asegurarse que no pueda hacerse la escritura en simultáneo, y por ende una de las dos deberá aguardar la escritura del otro para finalizar. Dado que la escritura se hace de forma casi instantánea, no habrá un desfase significativo entre los tiempos de ellas.

Contenido de los módulos

Dentro de los módulos utilizados, se encuentran los siguientes módulos particulares que se debe tener especial consideración a la hora de desarrollar ya que son los módulos principales para poder generar las funcionalidades requeridas. Estas deberían ser los primeros módulos a modificar en caso que se quiera agregar o cambiar algo. El resto de los módulos como vimos en el diagrama de las capas, son módulos de más bajo nivel y no debería ser lo primero en modificarse ya que tienen funcionalidades directas o semidirectas con el hardware.

Por otra parte, se pasa a explicar los módulos que son modificables:

utils:

Para el caso de este módulo, la misma actualmente cuenta con las siguientes funciones:

getUnixTime:

La misma se encarga de obtener el tiempo actual del sistema en formato unix.

unixTo:

Se encarga de pasar una fecha en modo unix a un formato en caracteres legible tipo timestamp UTC.

users_communications:

La funciones actuales de este módulo son las siguientes:

imprimirMenu:

Imprime el menú básico a través del puerto serial.

mainMenu:

Se encarga de manejar las distintas opciones ingresadas por el usuario a través del puerto serial para el menú dado. Según la funcionalidad que se active, esta función llama a la correspondiente para continuar con las acciones correspondientes.

Se trata básicamente de un switchcase el cual según la opción ingresada continúa con la siguiente funcionalidad.

mainComunicationTask: (task que corre constantemente)

Esta tarea es quien se encarga en forma de round robin de chequear si se pulso el botón S2 para entrar al menú o si el acelerómetro tiene un manejo brusco para generar los logs correspondientes y activar las alarmas de los mismos.

generateTrama: (task que corre constantemente)

generateTrama se encarga de dos acciones particulares, una es tener constantemente actualizada la ubicación del dispositivo, y la otra es que la primera vez que el sistema comienza su encendido, asegurarse que FreeRTOS tenga la fecha actual correcta enviada desde un satélite, a modo de asegurarnos que cuente con los datos reales para poder ser registrados posteriormente por las funciones de loggeo de datos.

loggerFunction: (task que corre constantemente)

Toma el semáforo de escritura y llama a loggerManager en modo de escritura para escribir en el array de datos.

loggerManager:

Tiene dos formas de trabajar, escritura e impresión. Escritura se encarga de llenar el arreglo en la posición correcta con los logs correspondientes. Impresión se encarga de leer el arreglo con los datos e imprimirlos a través del USB.

crash_manager:

setUmbral

Cuando el usuario comienza a generar el setteo de los umbrales, se invoca esta función. La misma se encarga de verificar que el usuario no pueda settear umbrales de forma incosistente, es decir, el umbral de manejo brusco siempre va a ser menor al umbral de choque. El umbral de manejo brusco como mínimo puede llegar a quedar en nivel 2, dejando el 1 para manejo normal, y como máximo en 7, dejando el 8 para choque.
Por otra parte el de choque, como mínimo podrá ser 1 más que el de brusco, y como máximo el 8vo nivel. Cuando se setean los umbrales son mapeados del 1 al 8 ya que el máximo nivel que obtuvimos del acelerómetro fue 10.5 y a partir del 8 ya es muy difícil alcanzarlo.

setLeds

Enciende los LEDs con un color dado o los apaga.

Los posibles colores vienen dados por macros de la siguiente forma:

- WHITE: 0
- RED: 1
- BLUE: 2
- GREEN: 3
- OFF: 4
- YELLOW: 5

prendeLedsSetearUmbrales

Dado un valor del ADC, invoca a la función setUmbral para registrar los nuevos niveles de manejo. Además prende los LEDs según la posición en la que se encuentre la ruedita y según como el usuario mueva la misma.

Deuda técnica

A lo largo del desarrollo distintos aspectos del proyecto son dejados en un estado primitivo (optimizable), con la idea de poder continuar trabajando y mejorar luego.

Es normal que parte de estos aspectos mejorables no terminen siendo mejorados en su totalidad en las primeras versiones del sistema, y esto es lo que se conoce como deuda técnica.

A continuación se detallan los siguientes puntos que deberían ser mejorados en posteriores versiones:

- Nombres de funciones y variables:

Los nombres actuales intentan ser lo más descriptivos posibles, sin embargo, esto está poco estandarizado a lo largo del código, especialmente con el uso de lenguaje inglés/español. En versiones posteriores esto debería ser revisado para mantener una buena consistencia.

- Ineficiencia en uso de heap/memoria de datos:

El uso de la memoria de datos es realizada de una forma poco eficiente, particularmente el uso del heap, ya que se cuentan con demasiadas variables que son del tipo static, las cuales son alojadas en el heap. Esto ocasiona que parte de la memoria sea reservada de antemano y la misma no pueda ser liberada ya que justamente, es estática y es trascendental a toda la ejecución del programa.

Esta ineficiencia se vio ocasionada, ya que es necesario que varias funciones sean declarada de dicha forma para la correcta funcionalidad entre los distintos tasks del programa y el uso de FreeRTOS, sin embargo se llegó a un punto donde fueron agregadas extra variables como static, de forma preemptiva a generar fallos difíciles de encontrar, y por tanto se debería buscar dichas variables que pueda evitar ser static y evitar alojarlas en el heap.

Notas para el desarrollador

Se comparten las siguientes notas de utilidad para futuros desarrolladores que implementan posteriores versiones de la aplicación, las cuales no necesariamente están directamente relacionadas con la aplicación en sí, pero deberían ser tenidas en cuenta:

- El gps en los distintos testeos que fueron realizados, el mismo puede llegar a demorar hasta unos 10 minutos en poder ser procesado, por lo cual, es posible que para obtener la primera trama se demore bastante.

- Los niveles de manejo brusco y choque vienen predeterminados en 3 y 6 respectivamente.

- El sistema comienza con RTCC con hora definida como 01/01/2021 a las 00:00.

- Se modificó configTOTAL_HEAP_SIZE dentro de FreeRTOSConfig.h de 20000 a 15000 a modo de poder contar con mayor cantidad de memoria de datos para utilizar.