Memoria Práctica Final

🤖 Realizado por Juan José Navarrete Gálvez y Daniel Bazo Correa, alumnos de Ingeniería de Sistemas Electrónicos.

Índice

1. Resumen

En este documento se detallará el proceso del proyecto realizado durante la asignatura de Microbótica para la implementación de un microbot capaz de realizar competiciones de mini-sumo.

El punto de partida ha consistido en combinar todas las prácticas anteriores realizadas durante el trascurso de la asignatura de Microbótica junto con la elaboración de una máquina de estados que veremos durante el desarrollo de este documento. Además, se ha implementado una serie de controles software para la gestión de interrupciones con la finalidad de dar una cierta “inteligencia” al microbot, con el objetivo de detectar estímulos y reaccionar acorde a ellos.

Con la finalidad de realizar un control de versiones de las diferentes prácticas, gestionar recursos, utilidades y tener los ficheros sincronizados entre los integrantes del grupo, hemos decidido utilizar GitHub. El repositorio en cuestión se encuentra en el siguiente enlace.

2. Especificaciones

2.1 Componentes

Los componentes utilizados han sido los siguientes:

- Microcontrolador: TIVA C Series EK-TM4C123GXL.
- Placa expansión Skybot-UMA y protoboard de 25x40 mm.

- Dos servos FUTABA S3003.
- Dos sensores de contacto de tipo botón ‘whisker’ o de final de carrera. Uno colocado en la parte frontal del microbot y otro en la parte trasera.
- Un sensor de distancia analógico (Sharp) GP2Y0A41SK0F con mediciones de distancia desde los 4 hasta los 30 cm.
- Cuatro sensores de línea TCRT1000.
- Dos sensores de línea CNY70.
- 4 LEDs (verde, azul, amarillo y rojo) con 4 resistencias de 220 Ohm.
- Batería de litio portátil con salida USB (5 V / 2.1 A).
- Carcasa, soporte CNY y ruedas mediante impresión 3D (Skybot).
- Cables, tornillos, pines y una canica.

2.2 Instalación de sensores

A continuación, mostramos unas imágenes de cómo han quedado instalados todos los sensores en el microbot.

Para empezar, en las imágenes 1 y 2 podemos observar una imagen general del microbot. Los sensores de línea se encuentran en las esquinas del chasis del microbot en su parte inferior. Con ello, conseguimos evitar pisar la franja circular blanca que se encuentra alrededor del tatami donde se producirá la lucha del mini-sumo.

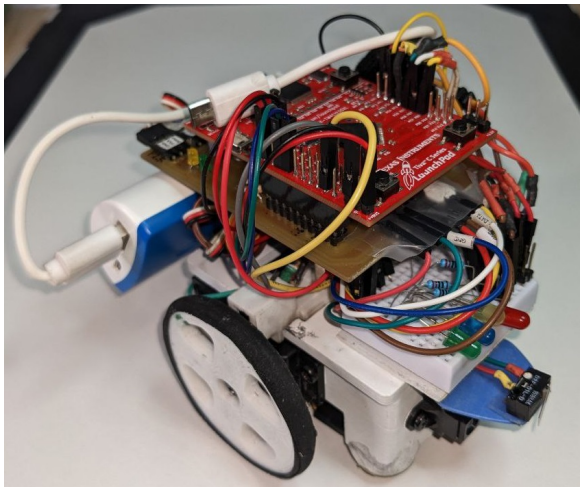

Imagen 1: Fotografía general del microbot, delantera.

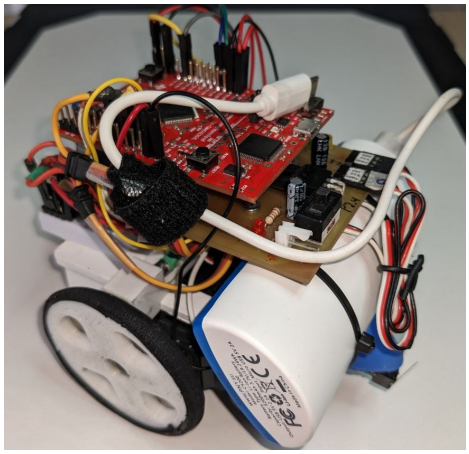

Imagen 2: Fotografía general del microbot, trasera.

En el caso del **sensor Sharp**, ha sido colocado en la parte frontal del chasis para la detección de objetos con una distancia media-lejana.

Para contar la distancia recorrida a la par que la cantidad de grados a girar, hemos colocados un sensor CNY70 en cada una de las ruedas (se puede observar la carcasa de dichos sensores en la imagen, muy cerca de las ruedas).

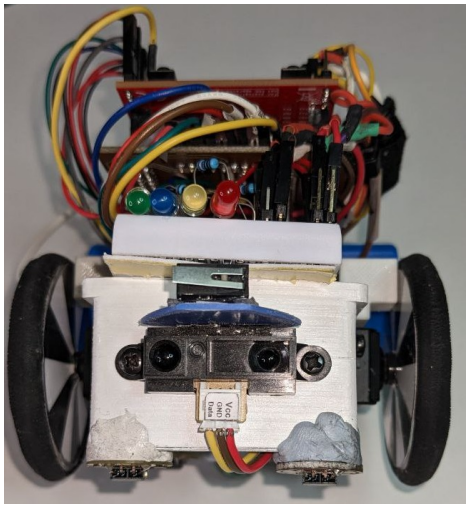

Imagen 3: Fotografía frontal del microbot.

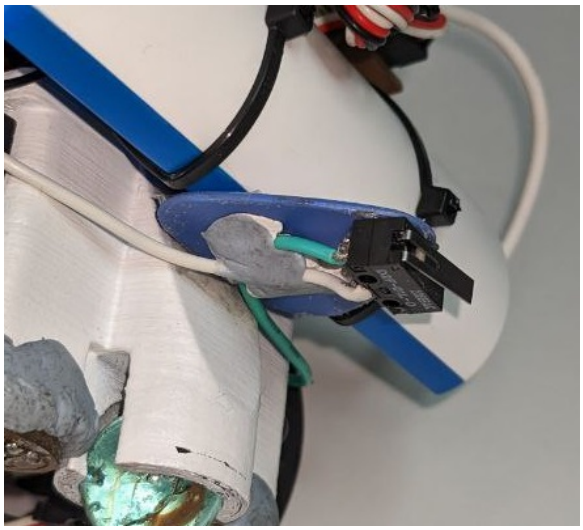

Imagen 4: Fotografía del whisker trasero del microbot.

Por último, los sensores de contacto han sido pegados en unas púas de reparación de equipos o púas de guitarra para conseguir que se activen antes de sufrir un impacto en el chasis del microbot, consiguiéndose así un margen extra de reacción frente a estímulos.

En la ‘Imagen 3’ podemos observar el sensor delantero, en la ‘Imagen 4’ el trasero.

A fin de colocar los sensores de línea cerca del suelo en una posición adecuada, se ha utilizado blu-tack. Además, se ha optado por taladrar la parte frontal del chasis con la intención de colocar los cables del sensor de contacto frontal lo más cercano del microcontrolador evitando posibles daños en los conectores y otros cables.

Debido a la gran cantidad de pines necesarios nos encontramos una limitación en relación al número de conexiones, ya que la placa de expansión seguía sin cumplir con nuestros requisitos en el número de conexiones necesario. Por ello, teniendo en cuenta la propia naturaleza de los sensores, no se deben conectar todos los pines a la misma línea de alimentación, sino que habrá que balancear las cargas utilizando diferentes pines de 5 voltios. En esta ocasión, se optó por soldar pines macho en los conectores situados en los laterales de los botones de la Tiva además de colocar una pequeña placa de prototipo pegada a la carcasa del microbot con el objetivo de utilizar múltiples pines de tensión de 5 V y GND del microcontrolador de manera cómoda. Por último, con el fin de evitar, entre otros, problemas de bucles de corriente, hemos optado por colocar todos los pines a tierra en una única toma. Podemos observar la protoboard en la imagen 5:

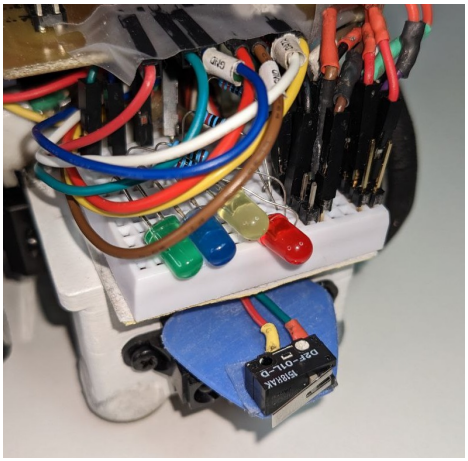

Imagen 5: Fotografía de la placa de prototipo además del whisker delantero.

Para acabar, mencionar que la placa de extensión se ha atornillado con tornillos de 20 cm de alto a la carcasa y la batería portátil se ha acoplado mediante bridas en la parte trasera del microbot apoyándonos en dichos tornillos. Los servos están atornillados por debajo.

2.2.1 Pines utilizados

A continuación mostraremos todos los pines utilizados por el microbot, además de los timers u otros elementos que hemos considerado importantes a conocer:

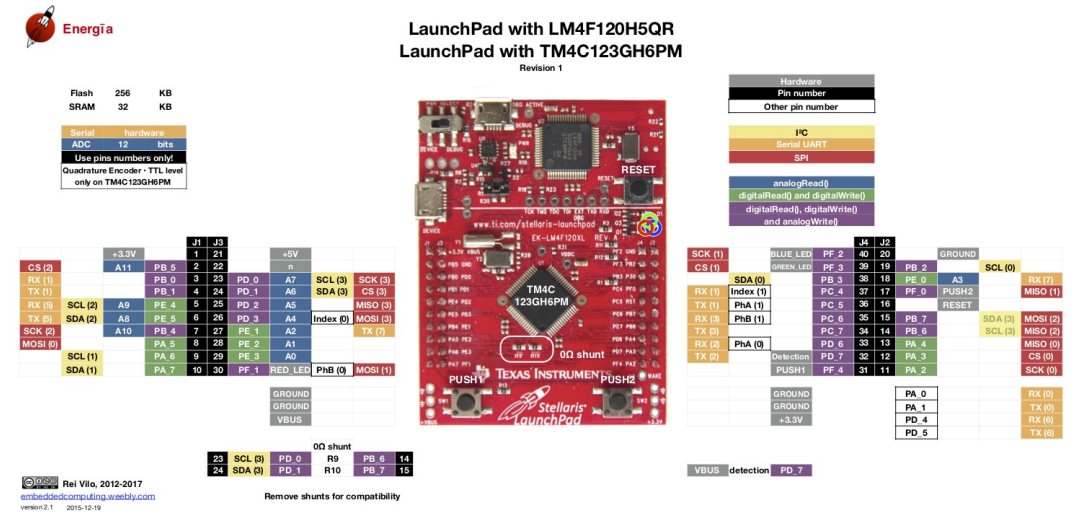

Imagen 6: Esquema de pines de la placa utilizada.

Pines	Uso	Pines	Pines
PA0	U0RX	**PB0**	PIN_LED_G (led verde) → Modo Búsqueda
PA1	U0TX	**PB1**	PIN_LED_B (led azul) → Modo Ataque

Pines	Uso	Pines	Pines
PA2	PIN_CNY_IZQ	**PB2**	PIN_LED_R (led rojo) → Zona Roja
PA3	PIN_CNY_DER	**PB3**	PIN_LED_Y (led amarillo) → Modo Huida
PA4		**PB4**	
PA5	PIN_WHISK_FRONT	**PB5**	
PA6	PIN_WHISK_BACK	**PB6**	
PA7		**PB7**	

Pines	Uso	Pines	Uso
PC0		**PD0**	
PC1		**PD1**	
PC2		**PD2**	
PC3		**PD3**	
PC4	PIN_LIN_DD (delantero derecho)	**PD4**	
PC5	PIN_LIN_DI (delantero izquierdo)	**PD5**	
PC6	PIN_LIN_DI (delantero izquierdo)	**PD6**	
PC7	PIN_LIN_DI (delantero izquierdo)	**PD7**	

Pines	Uso	Pines	Uso
PE0		**PF0**	RIGHT_BUTTON
PE1		**PF1**	
PE2		**PF2**	PWM
PE3	ADC0	**PF3**	PWM
PE4		**PF4**	LEFT_BUTTON
PE5		**PF5**	
PE6		**PF6**	
PE7		**PF7**	

- Timers → TIMER2 (usado con ADC0), TIMER5.

2.3 Requisitos

Los requisitos mínimos (nivel 1) del proyecto son los siguientes:

1. El microbot deberá tener un color claro (blanco) para que pueda ser detectado por los sensores SHARP.
2. El microbot deberá ser capaz de navegar por el tatami de forma reactiva. Es decir, cuando el microbot detecte un estímulo debe reaccionar de manera directa sin realizar un plan previo, es

acción-reacción.

3. El microbot no deberá salirse del borde blanco del tatami, detectando y reaccionando ante los posibles obstáculos.

Los requisitos de nivel 2 del proyecto son los siguientes:

1. El microbot además de contar con los requisitos anteriores deberá también incorporar estrategias de alto nivel, es decir, una estrategia deliberativa donde el microbot al presenciar un estimulo, evalúa el entorno en el que se encuentra creando un plan acorde a ello y tomando la acción que estime oportuna.

Los requisitos de nivel 3 del proyecto son los siguientes:

1. El microbot deberá combinar un control reactivo y deliberativo para conformar una arquitectura híbrida.

Como requisito extra se buscaba la implementación de un sistema de navegación dentro del microbot.

En nuestro caso, hemos implementado una arquitectura híbrida que contaremos en detalle en la parte de la programación.

3. Programación

En cuando a la programación, con el propósito de evaluar la colocación de los sensores a la par que comprobar su correcto funcionamiento para su desempeño en el mini-sumo, se ha optado por utilizar FreeRTOS junto al protocolo serie UART.

FreeRTOS es un sistema operativo en tiempo real para microcontroladores. Gracias a su uso, se consigue una abstracción mayor en el código simplificando la programación. De igual modo, permite añadir características esenciales para la elaboración del proyecto tales como flags de eventos, colas, una gran gestión de las interrupciones, capacidad de configurar el microbot en bajo consumo, etc.

La combinación de FreeRTOS con el protocolo serie UART nos ha permitido explorar, a través de un terminal serie del programa Code Composer Studio, los pasos que seguía el microbot durante la fase de pruebas, consiguiendo ver en todo momento los estados en los que se encontraba el microbot, acciones que realizaba, lectura de los sensores, etc. Con ello, hemos logrado depurar y optimizar el programa para un mejor y correcto desempeño.

Con la intención de poder visualizar los estados en los que se encontrará el microbot cuando no está conectado al ordenador (modo búsqueda, modo ataque y modo huida, los veremos a continuación) hemos colocado 3 leds que se iluminarán dependiendo del estado en el que se encuentre actualmente.

3.1 Organización del código y carpeta ‘drivers’

Para gestionar el código de una forma más clara y eficiente hemos optado por crear una serie de controladores que se relacionarán con una parte clara del código. Dichos controladores han derivado de las diferentes entregas que se han ido desarrollando a lo largo de la asignatura. En la imagen de la derecha se puede observar la distribución actual de los mismos.

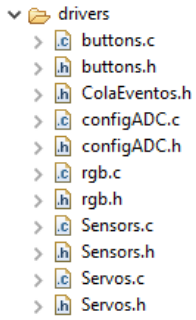

Imagen 7: Detalle de organización de la carpeta ‘drivers’.

- **buttons**: controlador propiedad de Texas Instruments encarga de la gestión de los botones de la TIVA. En nuestro caso es utilizada para inicializar el dispositivo.
- **ColaEventos**: documento de cabecera en el que se incluyen una serie de elementos clave para el funcionamiento del dispositivo, principalmente grupos de eventos con sus diferentes flags, pero también una cola y la estructura que utiliza (ambos relacionados con el movimiento).
- **configADC**: donde se configura y controla el sistema que gobierna nuestro sensor SHARP. Usamos para ello el pin PE3, utilizando el ADC0, y se configura en modo disparo por temporizador usando el TIMER2. Posteriormente, los datos obtenidos serán transformados para poder ser leídos correctamente.
- **rgb**: controlador propiedad de Texas Instruments encargado de la gestión del led RGB.
- **sensors**: archivo utilizado principalmente para la designación y configuración de los pines y puertos que se utilizarán para gestionar los diferentes sensores del microbot. Los pines se configurarán de entrada o salida y se le activarán las interrupciones según vaya siendo necesario. Adicionalmente, tanto a los whiskers como a los sensores de línea se les ha añadido una resistencia de pull-up extra. Dicho documento incluye también la tabla ‘look-up’ utilizada en la segunda entrega de la asignatura, aunque en la versión actual no se utiliza.
- **servos**: controlador de los servos, derivado de la primera entrega de la asignatura. La función que se va a utilizar principalmente es `mov_rueda_universal`, la cual posibilita la gestión independiente de cada rueda permitiendo, además, ajustar la velocidad y el sentido de las mismas. El resto de funciones no son utilizadas en la versión actual.

A continuación vamos a comentar cómo se ha programado el archivo ‘main’ para hacer que el microbot cumpla con los requisitos establecidos.

3.2 ‘Main’ y funcionamiento del programa.

Con el código elegido de la práctica anterior, decidimos elaborar una máquina de estados en el que se reflejara los modos de funcionamiento del microbot, así como de las acciones que provocarían esos cambios de estados.

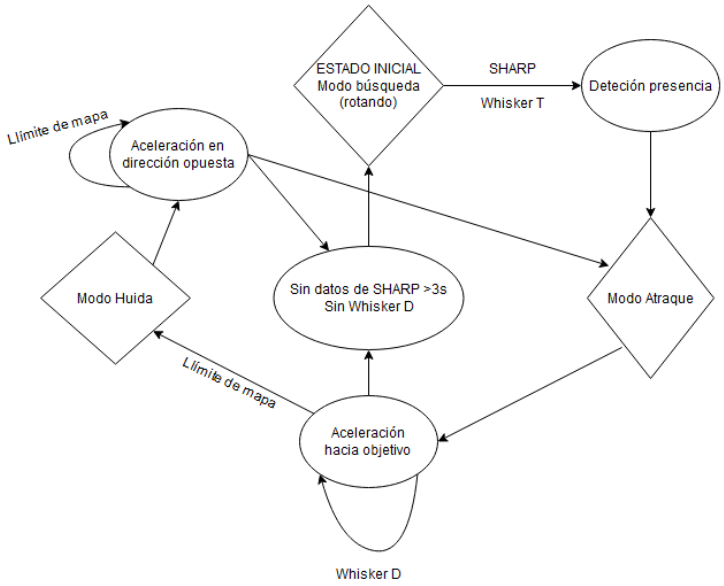

Imagen 8: Máquina de estados.

El componente principal de nuestro código es dicha máquina, donde contamos con 3 estados fundamentales: **BUSQUEDA, ATAQUE** y **HUIDA**. A continuación, mostramos las diferentes tareas utilizadas en nuestro programa:

```
static portTASK_FUNCTION (TareaMaestra, pvParameters)
{
    ...

    while(1)
    {
        // Puesto que el microbot debe dejar paso siempre a las tareas de la máquina de
        // estados, la toma de flags de eventos no debe ser bloqueante
        // por ello, como último parámetro de xEventGroupWaitBits hemos colocado un 0.
        // Cuando se active un flag de eventos provocado por una RTI, se activará el flag
        // correspondiente saltando de un estado a otro.
        eventos = xEventGroupWaitBits(FlagsEventos, TATAMI_DD | TATAMI_DI | TATAMI_TD |
                    TATAMI_TI | VAL_WHISK_FRONT | VAL_WHISK_BACK, pdTRUE, pdFALSE, 0);

        switch(modo)
        {
            case BUSQUEDA:
            {
                // El robot rota hasta encontrar un objeto. Cuando detecte datos en
                // la cola del Sharp o se activen los flags de ciertos eventos
                // (interrupciones), pasará al estado correspondiente.
                // Si durante 4 segundos (mediante un timer software, "CASIO_BUSQUEDA")
                // no encuentra ningún objeto, avanzará unos centímetros para
                // posteriormente seguir buscando.

                ...
            }

            break;


            case ATAQUE:
            {
                // Debe acelerar hacia el objetivo. Si detecta pulsaciones del whisker
                // delantero seguirá en el modo ataque.
```

```
                // Si durante el modo ataque los datos del sharp son de distancias altas
                // o medias o no detecta el whisker delantero volveremos al modo
                // de busqueda. Para gestionar el Sharp se ha utilizado un umbral.
                // Si durante el ataque al enemigo, los sensores de linea detectan
                // el limite del mapa pasamos al modo de huida

                ...
            }

            break;


            case HUIDA:
            {
                // Cuando entremos en este modo, los motores aceleran en dirección
                // contraria al sensor detectado y posteriormente realizan un giro.
                // Si durante ese giro, encuentra a alguien, vuelve al modo ataque.
                // Si los datos que recibe del sharp son de distancia media o larga
                // o no recibe datos del whisker delantero pasara al modo de busqueda

                ...
            }

            break;

            default:
            {

            }
        }
    }
}
```

Debido a que las interrupciones originadas por los sensores de línea son las más importantes dentro de los sensores utilizados, estos serán los primeros en comprobarse en casa estado. Posteriormente, recibirá mayor importancia el sensor Sharp y por último, los sensores Whiskers.

El traspaso de información del sensor Sharp con la tarea maestra se realiza a partir de valores mandados por una cola, `colaSharp` . Debido a unos problemas que tuvimos con el sensor Sharp, tuvimos que pasar de usar la función `binary_lookup` a establecer rangos de valores de manera manual dependiendo de los valores obtenidos por el sensor, aunque el uso de esta función la realizamos en la práctica 2.

Como ya se comentó anteriormente, la funcionalidad del sensor Sharp viene de la mano del driver ***configADC.c*** y ***configADC.h,*** además de una tarea `TareaADC0Sharp` que se encuentra en el ***main.c*** del proyecto.

```
static portTASK_FUNCTION (TareaADC0Sharp, pvParameters)
{
    MuestrasADC lectura;
    uint32_t valor;
    uint8_t anterior = 0;

    // TareaADC0Sharp ira comprobando el sensor Sharp y activara un evento u otro
    // dependiendo de si ve algo o no. Para evitar posibles picos espúreos, hemos
    // configurado dicha tarea para que active los eventos solo si se ha detectado
    // algo 3 veces seguidas. Ademas, anadimos un pequeno delay de 15 ms para tener
    // un control mas fino de dicha comprobacion.
    // Dicho delay se utiliza para fijar la variable 'umbralSharp': al entrar en ataque
    // el microbot avanzara, pero si al poco rato no detectamos nada el microbot
    // volvera a busqueda. Como el delay es de 15 ms y tenemos que leer 3 muestras
    // validas para enviar un evento de deteccion, necesitamos unos 45 ms para detectar
    // un obstaculo.
    // Debido a que no sabemos exactamente que el microbot haga una lectua cada 15 ms,
    // suponemos que va se va a realizar una lectura cada 50 ms.
    // Si queremos que el robot avance solo durante 600 ms: 600 / 50 = 12
    // Relacion de compromiso entre blindaje ante falsos positivos y velocidad de reaccion.
```

```
    while (1)
    {
        configADC0_LeeADC0(&lectura);

        if (lectura.chan >= 1480 && lectura.chan <= 3760)
        {
            if (anterior < 3)
            {
                anterior++;
            }
            else
            {
                valor = VAL_SHARP_CERCANO;
                xQueueSend(colaSharp, &valor, 0);

                anterior = 0;
            }
        }
        else if (lectura.chan >= 730 && lectura.chan < 1480)
        {
            if (anterior < 3)
            {
                anterior++;
            }
            else
            {
                valor = VAL_SHARP_MEDIO;
                xQueueSend(colaSharp, &valor, 0);

                anterior = 0;
            }

        }
        else if (lectura.chan >= 600 && lectura.chan < 730)
        {
            if (anterior < 3)
            {
                anterior++;
            }
            else
            {
                valor = VAL_SHARP_LEJOS;
                xQueueSend(colaSharp, &valor, 0);

                anterior = 0;
            }
        }
        else
        {
            valor = VAL_SHARP_NO_VISIBLE;
            xQueueSend(colaSharp, &valor, 0);

            anterior = 0;
        }

        vTaskDelay(15/portTICK_PERIOD_MS);
    }
}
```

Para el caso de los sensores de línea, se ha creado una RTI en la que se manda el flag de eventos `TATAMI_XY` siendo `X` :

- `D` : si se activa el sensor delantero.
- `T` : si se activa el sensor trasero.

En el caso de `Y` :

- `D` : si se activa el sensor derecho.
- `I` : si se activa el sensor izquierdo.

En dicha RTI, cambiamos directamente el modo al que pasará la máquina de estados siempre que inicialmente se halla pulsado el botón derecho, ya que este activa la variable `ON_INI` iniciando el funcionamiento de la máquina de estados y, por tanto, de la recepción de los eventos de los sensores:

```
void GPIOInt_C_Handler(void)
{
    BaseType_t higherPriorityTaskWoken = pdFALSE;
    int32_t i32PinStatus = GPIOIntStatus(GPIO_PORTC_BASE, true);

    if (ON_INI)
    {
        if ((i32PinStatus & PIN_LIN_DD))
        {
            xEventGroupSetBitsFromISR(FlagsEventos, TATAMI_DD, &higherPriorityTaskWoken);
            modo = HUIDA;
        }

        if ((i32PinStatus & PIN_LIN_DI))
        {
            xEventGroupSetBitsFromISR(FlagsEventos, TATAMI_DI, &higherPriorityTaskWoken);
            modo = HUIDA;
        }

        if ((i32PinStatus & PIN_LIN_TD))
        {
            xEventGroupSetBitsFromISR(FlagsEventos, TATAMI_TD, &higherPriorityTaskWoken);
            modo = HUIDA;
        }

        if ((i32PinStatus & PIN_LIN_TI))
        {
            xEventGroupSetBitsFromISR(FlagsEventos, TATAMI_TI, &higherPriorityTaskWoken);
            modo = HUIDA;
        }
    }

    MAP_GPIOIntClear(GPIO_PORTC_BASE, PIN_LIN_DD | PIN_LIN_DI | PIN_LIN_TD | PIN_LIN_TI);
    portEND_SWITCHING_ISR(higherPriorityTaskWoken);
}
```

Al igual que ocurre con los sensores de línea, se ha creado una RTI para los sensores Whiskers y CNY.

```
void GPIOInt_A_Handler(void)
{
    BaseType_t higherPriorityTaskWoken = pdFALSE;
    int32_t i32PinStatus = GPIOIntStatus(GPIO_PORTA_BASE, true);

    if (ON_INI)
    {
        if ((i32PinStatus & PIN_WHISK_FRONT) && (detectar == 1))
        {
            xEventGroupSetBitsFromISR(FlagsEventos, VAL_WHISK_FRONT,
                                      &higherPriorityTaskWoken);
            modo = ATAQUE;
            detectar = 0;

            if(xTimerStart(CASIO_REBOTON, 0) != pdPASS)
            {
                while(1);
            }
```

```
        }

        if ((i32PinStatus & PIN_WHISK_BACK) && (detectar == 1))
        {
            xEventGroupSetBitsFromISR(FlagsEventos, VAL_WHISK_BACK,
                                      &higherPriorityTaskWoken);
            xQueueReset(colaMovimiento);
            modo = HUIDA;
            detectar = 0;

            if(xTimerStart(CASIO_REBOTON, 0) != pdPASS)
            {
                while(1);
            }
        }

        if (i32PinStatus & PIN_CNY_IZQ)
        {
            xEventGroupSetBitsFromISR(FlagsEventos, PASOS_RUEDA_IZQ,
                                      &higherPriorityTaskWoken);
        }

        if (i32PinStatus & PIN_CNY_DER)
        {
            xEventGroupSetBitsFromISR(FlagsEventos, PASOS_RUEDA_DER,
                                      &higherPriorityTaskWoken);
        }
    }

    MAP_GPIOIntClear(GPIO_PORTA_BASE, PIN_WHISK_FRONT | PIN_WHISK_BACK |
                                            PIN_CNY_IZQ | PIN_CNY_DER);
    portEND_SWITCHING_ISR(higherPriorityTaskWoken);
}
```

De la RTI anterior, vemos que los sensores Whiskers permiten realizar saltos entre modos. Mención especial al timer software `CASIO_REBOTON`, el cual tiene una duración de 200 ms y se utiliza para crear un sistema antirrebotes en los whiskers.

Por otro lado, los sensores CNY activan sus flags correspondientes de `PASOS_RUEDA_X` (donde X es la lateralidad del sensor). Gracias a dicho flag, conseguimos conocer la cantidad de pasos que realiza el microbot, lo que nos ayuda a controlar el número de grados o centímetros que se mueve.

Para el movimiento del microbot, hemos creado un sistema de colas en el que mandamos una estructura, `pareja` , que cuenta con 2 objetos. Un primer objeto de tipo `char` permitiendo identificar el tipo de movimiento (`c` = centímetros o `g` = grados) y un segundo objeto de tipo `int32_t` determinando la cantidad a desplazarse o girar y el signo. Este sistema de colas, se ha implementado de manera que no bloquee la funcionalidad del microbot, por lo que si se recibe algún evento de los sensores de línea, se eliminarán todos los eventos anteriores dando la mayor prioridad al modo de **HUIDA**, en este caso. Lo mismo ocurriría para el resto de sensores pero con diferentes grados de prioridad. Además, la propia tarea se autogestiona, comprobando si existen más mensajes en la cola antes de parar al microbot, y se comunica con la tarea maestra utilizando la variable entera `en_movimiento`.

Debido a la longitud del código, únicamente mostraremos las partes esenciales acompañadas de una explicación:

```
static portTASK_FUNCTION (TareaMovimiento, pvParameters)
{
    // Definición de variables para su uso dentro de la tarea
    struct pareja mov_recibido;
    int objetivo_mov = 0;
    int pasos[] = {0, 0};
```

```
    EventBits_t eventos;

    while (1)
    {
        // Esperamos a recivir algún evento de movimiento o evento de los CNY
        eventos = xEventGroupWaitBits(FlagsEventos, MOV | PASOS_RUEDA_IZQ |
                            PASOS_RUEDA_DER, pdTRUE, pdFALSE, portMAX_DELAY);

        // Si el flag de eventos de Movimiento se activa
        if ((eventos & MOV) != 0)
        {
            // Y el objetivo a alcanzar (la cantidad a desplazarse o girar) es 0,
            // es decir, que no existe ningún objetivo actual.
            if (objetivo_mov == 0))
            {
                // Espero a recivir elementos de la cola de movimientos
                if (xQueueReceive(colaMovimiento, &mov_recibido, 0) == pdTRUE)
                {
                    // Paro las 2 ruedas del microbot
                    mov_rueda_universal(AMBAS_RUEDAS, 0);

                    // Si me llega una petición de desplazarme en centímetros
                    // tendré que diferenciar si la distancia a recorrer es positiva
                    // (hacia adelante) o negativa (hacia atrás)
                    if (mov_recibido.t == 'c')
                    {
                        if (mov_recibido.v >= 0)
                        {
                           ...
                        }
                        else if (mov_recibido.v < 0)
                        {
                            ...
                        }
                    }

                    // Mismo caso para los grados a girar
                    else if (mov_recibido.t == 'g')
                    {
                        if (mov_recibido.v >= 0)
                        {
                           ...
                        }
                        else if (mov_recibido.v < 0)
                        {
                            ...
                        }
                    }
                }
            }
        }

        // Si se activa el flag de ventos del CNY izquierdo
        else if ((eventos & PASOS_RUEDA_IZQ)  != 0)
        {
            // Comprueba si tengo un objetivo a cumplir
            if (objetivo_mov != 0)
            {
                ...

                // Si he cumplico con el objetivo del dezplazamiento o del giro
                // procedo a liberar los recursos
                if (pasos[0] == objetivo_mov)
                {
                    ...
                }
            }
        }

        // Al igual que ocurría con el CNY de la rueda izquierda
        // realizamos el mismo procedimiento para la rueda derecha.
        // Esto es así ya que con el tiempo se van acumulando fallos, y con este
```

```
        // mecanismo conseguimos reducirlos al conocer qué rueda ha cumplido
        // antes con su objetivo, ya que cada rueda se desplaza de manera simultanea.
        else if ((eventos & PASOS_RUEDA_DER)  != 0)
        {
            ...
        }
    }
}
```

Para acabar, comentar la última ISR utilizada, la del botón, la cual se encarga de inicializar el sistema:

```
void GPIOInt_F_Handler(void)
{
    BaseType_t higherPriorityTaskWoken = pdFALSE;

    int32_t i32PinStatus = MAP_GPIOIntStatus(GPIO_PORTF_BASE, ALL_BUTTONS);

    if ((i32PinStatus & RIGHT_BUTTON))
    {
        ON_INI = true;
        en_movimiento = 0;
        modo = BUSQUEDA;
    }

    MAP_GPIOIntClear(GPIO_PORTF_BASE, ALL_BUTTONS);
    portEND_SWITCHING_ISR(higherPriorityTaskWoken);
}
```

4. Resultados

Algunos de los resultados obtenidos se pueden ver en los siguientes vídeos:

- Practica2AEj1-Grupo2 - YouTube
- Practica2BEj3-Grupo2 - YouTube

5. Mejoras, líneas futuras y conclusiones

En cuando a las mejoras, nos hubiera gustado tener mayor tiempo para la planificación y desarrollo del proyecto, ya que conseguiríamos obtener un plan mejor detallado, con posibilidad de estudiar diferentes escenarios en los que poder utilizar el microbot (con la implementación realizada en este documento su uso tiene una gran restricción, aunque es suficiente para cumplir con los requisitos). Hubiéramos probado otros sensores o servomotores que se adaptasen mejor a nuestra funcionalidad.

Como línea futura, sería interesante estudiar el uso de otro tipo de microcontroladores (por ejemplo, Arduino) con el objetivo de simplificar la programación, aunque ello conlleve una disminución en eficiencia. En nuestro caso, hubiera sido interesante probar una Arduino Mega ya que cuenta con una mayor cantidad de pines, resolviendo muchos de los problemas con los que nos hemos enfrentado.

En caso de querer elaborar un robot con mejores prestaciones, optar por equipos empotrados más potentes como los que ofrece Nvidia con su serie Jetson sería una gran oportunidad, permitiendo conectar otros sensores que requieren de mayor computación como cámaras. Utilizar equipos como la Jetson, permitiría obtener procesados muchos más rápidos e incluso aplicar sistemas con inteligencia artificial que aprendieran del entorno y mejorasen sus respuestas ante los estímulos presentados en

dicho entorno durante su funcionamiento/aprendizaje (este tipo de problemas reciben el nombre de aprendizaje por refuerzo o *reinforcement learning*).

Consideramos que con este proyecto hemos ampliado nuestro conocimiento en los sistemas empotrados a la par que en la ingeniería debido a las ideas de diseño, toma de decisiones de sensores, resolución de problemas, etc. Nos hemos enfrentado a numerosos problemas que han requerido una solución ingeniosa por nuestra parte, y hemos ahondado más en campos más desconocidos para un ingeniero electrónico como puede ser el de la mecánica.

6. Referencias y bibliografía

- J.M. Cano, E. González, I. Herrero, Apuntes de la asignatura de Sistemas Empotrados.
- J. P. Bandera Rubio, I. A. Herrero Reder, A. C. Urdiales Garcia, Apuntes de la asignatura de Microbótica.
- Texas Instrument, TIVAWare Peripheral Driver Library Users Guide.
- Texas Instrument, Tiva TM4C123GH6PM Microcontroller Data Sheet.
- FreeRTOS.