Departamento de Ingeniería Electrónica
E.T.S.I. de Telecomunicación
Universidad Politécnica de Madrid

Enunciado del proyecto estándar de Sistemas Digitales II (SDG2)

PiTankGo: un juguete robótico
para la Raspberry Pi

Fernando Fernández Martínez (coordinador)
fernando.fernandezm@upm.es
Rubén San Segundo Hernández, Juan Manuel Montero Martínez,
Ricardo de Córdoba Herralde, José Manuel Pardo Muñoz,
Alberto Boscá Mojena, Juan José Gómez Valverde,
Luis Fernando D'Haro Enríquez, Manuel Gil Martín,
Francisco Romero

Curso 2018-2019

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Última revisión: 26, Febrero, 2019

Contenido

Índice de ilustraciones

Índice de tablas

1. Aviso inicial

Cuando aborde la lectura de este documento, hágalo con tranquilidad y detenimiento. Frases o comentarios que no se entiendan en una primera lectura pueden encerrar avisos y recomendaciones que le serán útiles a lo largo del desarrollo.

No se preocupe si no alcanza a comprender todos los términos, conceptos y detalles que se discuten. Todos ellos se irán aclarando a medida que avance en la lectura de éste y otros documentos (i.e. tutoriales y demás recursos de apoyo, que permitirán profundizar en determinados aspectos del diseño). **Por supuesto, asuma que necesitará varias lecturas y una reflexión a fondo sobre todo esto**.

Finalmente, preste especial atención a todas las referencias explícitas a aspectos que se indican como obligatorios para incluir en el sistema y en la memoria correspondiente: no quiere decir que algo no referenciado explícitamente no tenga que ser tratado, sino que nuestra experiencia demuestra que algunos de esos aspectos no son considerados por un cierto número de alumnos, lo que da lugar a desagradables sorpresas en los exámenes.

2. Introducción

SDGII tiene por objetivos docentes los siguientes: en primer lugar, la aplicación y consolidación de los conocimientos sobre sistemas basados en microprocesadores o microcontroladores adquiridos en SDGI; y en segundo lugar, y como ampliación del primero, la adquisición y aplicación de conocimientos sobre programación de sistemas autónomos o empotrados (embedded) basados en un microprocesador (incluyendo hardware y software).

Con esos propósitos, la asignatura plantea el **desarrollo de un sistema electrónico complejo basado en un microcontrolador** partiendo de una descripción y unas especificaciones básicas **conforme a un caso real de diseño** que el alumno podrá consultar en este mismo documento.

El **curso está organizado en sesiones de laboratorio** orientadas a la implementación de un nuevo módulo o una nueva versión del sistema final a implementar. En particular, cada sesión planteará al alumno la consecución de un **hito** que corresponderá a un cierto nivel de desarrollo o madurez (funcionalidad) alcanzado por el prototipo.

Las primeras tres sesiones, en las que se presentarán los conceptos y las herramientas básicas necesarios para el desarrollo del proyecto propuesto, permitirán al alumno conseguir una **primera versión simplificada pero completamente funcional del sistema** (versión 1.0). Posteriormente, la consecución de cada nuevo hito, aplicando las herramientas y fundamentos adquiridos, significará la implementación de una nueva versión del prototipo (versión 2.0, 3.0, ...) al que se le irán añadiendo **nuevos elementos hardware** (pulsadores, displays, ...) **y software** (nuevos eventos y estados, temporización, ...) **que completarán y mejorarán su funcionalidad**.

Para ello, deberá seguir las instrucciones aquí incluidas, que implicarán diversas fases de diseño, análisis, implementación y medida de los circuitos y programas propuestos. Igualmente, se contará con todos los medios disponibles en el laboratorio B-043 así como también con la ayuda de los profesores y colaboradores docentes.

2.1. Sobre la evaluación del proyecto

El alumno deberá entregar electrónicamente, y haciendo uso de los medios habilitados a tal efecto en la plataforma Moodle de la asignatura, **una copia del código desarrollado correspondiente a cada versión haciendo notar la consecución o no del hito en cuestión.**

Salvo que se indique lo contrario, el proyecto propuesto contiene las **especificaciones mínimas obligatorias** que deben cumplir los sistemas y serán **valoradas con un máximo de 7 puntos**. **Esta nota máxima sólo se conseguirá si se cumplen todas las especificaciones y los resultados de las diferentes pruebas de evaluación continua son perfectos.** Adicionalmente a las especificaciones mínimas, se presentarán sugerencias de mejora opcionales, dejando a los alumnos la libertad para que añadan nuevas mejoras o esquemas alternativos (ver sección 6.7). Con estas mejoras o montajes alternativos añadidos al prototipo básico, y dependiendo de su dificultad y realización, se podrán sumar puntos hasta alcanzar la máxima nota, 10 puntos. **Nótese que las mejoras añadidas sólo serán consideradas en caso de haber superado[1] la evaluación final sobre la versión definitiva del sistema (revisión de requisitos final y prueba individual en laboratorio, ambos aprobados).**

El trabajo realizado para la implementación de las mejoras consideradas debe quedar debidamente reflejado en una **memoria final** que contenga todos los detalles del proceso, incidiendo particularmente en los aspectos de **diseño HW y SW**, así como también en los **resultados** obtenidos y en todas aquellas cuestiones específicas que se indiquen en el presente enunciado o que sean notificadas a través de la plataforma Moodle de la asignatura. Tanto las instrucciones de entrega como la **plantilla** para la elaboración de esta memoria serán convenientemente publicadas y anunciadas a través de dicha plataforma.

2.2. Proyectos innovadores

Aquellos alumnos que deseen realizar un **proyecto innovador** basado en un **problema o un diseño propios** (o propuesto por un profesor), deberán hablar con alguno de los profesores de la asignatura y presentarle un listado de notas y una propuesta de proyecto donde describan, en 2 o 3 páginas:

- Objetivos del sistema propuesto.
- Recursos necesarios para llevarlo a cabo.
- Arquitecturas HW y SW propuestas para la resolución del problema.

Para poder abordar el proyecto será necesario contar con la aprobación de dicho profesor. No será admitido ningún proyecto (por muy complejo o perfecto que sea) que no se ajuste a estas normas. En el canal de YouTube[2] de la asignatura pueden visualizarse vídeos de demostración de los diferentes proyectos emprendidos por los alumnos de las últimas ediciones de SDG2.

[1] Para aspectos propios de la evaluación de la asignatura se remite al alumno a la guía docente: http://www.etsit.upm.es/fileadmin/documentos/estudios/grado_teleco/Guias_de_Aprendizaje/Curso_2018-19/GA_09TT_95000033_2S_2018-19.pdf

[2] https://www.youtube.com/channel/UCYIw_gl745WMJ1n0MamDzQw/

2.3.Sobre la organización del documento

Los apartados siguientes del presente documento están organizados de la siguiente manera: en primer lugar, se facilitará una descripción detallada de la **especificación de requisitos** que debe cumplir el prototipo final a implementar. A continuación, se detallarán las **arquitecturas HW y SW** del mismo, haciendo especial énfasis en la descomposición modular del sistema, en la interacción HW-SW y en los requisitos de tiempo real, todos ellos aspectos clave para abordar con éxito el proyecto. Finalmente, se describirá la **planificación recomendada** para su desarrollo, **organizada por sesiones** de laboratorio para las cuales se indicarán tanto los **objetivos a conseguir** durante las mismas como los **recursos disponibles** para ello.

3. Descripción general

3.1. Antecedentes

Hoy en día los **microcontroladores** (µC) suponen más del 50% de los Circuitos Integrados existentes. Basta analizar un hogar cualquiera para encontrar al menos una o dos docenas de microcontroladores distribuidos entre los diferentes dispositivos presentes en el mismo: lavadoras, frigoríficos, hornos, microondas, teléfonos, mandos inalámbricos, teclados, automóviles, etc. Casi cualquier dispositivo electrónico cuenta con un µC como elemento esencial para la toma de decisiones o para la supervisión del sistema.

Un buen ejemplo de su creciente importancia e interés son los **juguetes electrónicos interactivos**. Estos juguetes, cada vez más comunes en nuestra vida cotidiana, contienen, en su gran mayoría, microcontroladores. La integración del µC en la industria del juguete ha significado, para fabricantes y consumidores, la posibilidad de disfrutar de juguetes cada vez más divertidos e incluso ha logrado transformar el juguete en una plataforma educativa de enorme potencial para los niños. Mini-robots, coches, helicópteros, aviones o incluso **tanques a control remoto** (control por radio o simplemente radiocontrol, RC) son sólo algunos de los ejemplos de productos que hacen uso de microcontroladores.

Precisamente en éstos últimos, los tanques RC, está inspirado el presente proyecto. En el mercado existe una gran variedad[3] de modelos de tanques teledirigidos de diferentes calidades. Todos ellos constituyen réplicas en miniatura más o menos exactas en cuanto a la reproducción del avance, retroceso, giros y detención de un tanque real. Algunos de ellos permiten además hacer **guerras de tanques** (apoyándose para ello en las nuevas frecuencias de radio en 2.4GHz[4] que ofrecen una mejor respuesta, un mejor alcance y evitan interferencias entre los tanques teledirigidos). Éste será el aspecto en particular en el que nos concentraremos en nuestro proyecto: **las batallas**.

En particular, y dado que nuestro propósito es eminentemente docente (enfocado a la interacción entre HW y SW y a los requisitos de tiempo real), el propósito del proyecto no será la implementación de un tanque RC al completo sino que **simplificaremos su alcance** prescindiendo tanto del tren mecánico (motor, chasis, ruedas, etc.), por medio del cual se desplazan normalmente este tipo de juguetes, como de la parte de radiocontrol.

[3]Por ejemplo: https://www.juguetronica.com/kit-robot-tanque-rc

[4] http://rc.lapipadelindio.com/general/pros-contras-frecuencia-emisoras-24ghz

Por tanto, podemos resumir el objetivo del presente proyecto en la implementación de un **cañón infrarrojo de bajo coste,** similar a aquellos que montan algunos de los modelos de tanque RC anteriormente mencionados (ver Ilustración 1). En ese sentido, supondremos que un cliente (el Departamento de Ingeniería Electrónica) nos ha contratado para desarrollar un prototipo funcional completo.

3.2. Objetivo general

El objetivo del proyecto es mostrar la viabilidad de la idea de diseño básica desarrollando un prototipo de **cañón de juguete** plenamente funcional. El programa que ejecutará el micro se realizará **en lenguaje C.** El sistema digital basado en un microcontrolador será la plataforma ENT2004CF disponible en el laboratorio B-043 (construida en torno a una Raspberry Pi). Para más detalles relacionados con el sistema de desarrollo (utilización, pines de conexión, elementos disponibles), consulte la publicación [1].

Ilustración 1.Ejemplo de prototipo similar al propuesto en el presente proyecto.

El funcionamiento de nuestro cañón de juguete es sencillo. Aunque su ubicación será fija (no implementamos un tanque como tal), el **cañón** estará construido a modo de **torreta móvil operada por un par de micro servomotores** que permitirán, por un lado su oportuno **ascenso o descenso**, y por el otro, su necesario **giro a derecha e izquierda**. Para permitir la maniobrabilidad del mismo, el jugador o usuario dispondrá de un **elemento de control** (un teclado matricial o un joystick) que, al igual que los propios microservos, estarán directamente conectados **mediante cables** a nuestro microcontrolador (prescindimos del radiocontrol). Para no llevar al límite a los servos, el movimiento de la torreta deberá ser **discontinuo** produciéndose éste **a partir de pequeños incrementos** en cualquiera de los ejes, tanto para el vertical como para el horizontal. Además, la **velocidad** a la que podremos desplazar el cañón estará necesariamente **limitada**.

Obviamente, nuestro cañón deberá ser capaz de realizar **disparos** con los que poder derribar un determinado objetivo o **blanco** para lo cual nos apoyaremos en un simple **enlace infrarrojo** (i.e. contaremos con un emisor infrarrojo a modo de cañón y con su correspondiente receptor infrarrojo a modo de blanco). Además del requerido diseño e implementación de los correspondientes subsistemas HW necesarios para el establecimiento de dicho enlace será igualmente necesario **establecer las correspondientes conexiones físicas con la Raspberry Pi** empleando para ello las entradas y salidas digitales disponibles en los conectores de la placa TL04 y conforme a la correspondiente Tabla de distribución de pines (disponible, por ejemplo, en el Tutorial sobre "Iniciación al Manejo de las Entradas/Salidas del BCM 2835").

En nuestro caso, y a modo de simplificación, ya que de ese modo será posible jugar solos (i.e. controlamos tanto el cañón como el blanco), **ambos extremos del enlace, el emisor y el receptor, estarán directamente conectados a nuestro microcontrolador**. En este sentido será necesario contar con unos cables para el blanco/receptor infrarrojo cuya longitud permita disponerlo a cierta distancia de nuestro propio cañón/emisor infrarrojo para permitir así el juego en modo individual, modalidad básica de juego. Igualmente, nótese que esta consideración de diseño no afecta en modo significativo alguno a la posibilidad de jugar con otros compañeros (juego multijugador, modalidad propuesta como mejora): si jugamos solos, nuestro propio blanco se convierte en el objetivo del juego; por el contrario, si jugamos contra otros, nuestro blanco se constituye en el objetivo del resto de cañones/jugadores y, a su vez, los blancos de los demás en objetivos para el nuestro.

Para un mayor realismo, a la hora de realizar un **disparo** nuestro cañón deberá imitar la ejecución del mismo acompañándolo de la generación del correspondiente **efecto de sonido**. De igual modo, el eventual **alcance de un blanco** también irá acompañado del correspondiente efecto de sonido. Sin embargo, no implementaremos un sintetizador de gran calidad sino una versión bastante simplificada basada en la mera **generación por parte de la propia Raspberry Pi de una señal consistente en la sucesión de diferentes ondas rectangulares de un cierta duración y frecuencia**. Para percibir los efectos de sonido (con fuerte distorsión) bastará con **aplicar dicha señal directamente sobre unos auriculares**.

Además de controlar la torreta mediante el joystick, la **acción de un pulsador** será oportunamente detectada por el sistema procediendo éste a la ejecución del **disparo** (i.e. encendido del emisor infrarrojo durante un cierto periodo de tiempo) y dando comienzo a la reproducción del **efecto de sonido asociado al disparo** en cuestión. Igualmente, la eventual posterior activación del receptor infrarrojo al establecerse el enlace con el emisor (i.e. disparo con éxito, que alcanza el blanco) será también detectada por el propio sistema para proceder así a la reproducción del **efecto de sonido asociado al alcance del blanco**. La **duración** de cada disparo deberá estar necesariamente **limitada** en tiempo (i.e. el sistema final no mantendrá activo el emisor IR todo el tiempo). De esta manera, será necesario emplear un **temporizador** para encender el emisor y apagarlo una vez transcurrido el tiempo de activación del enlace definido para cada disparo (i.e. máximo 2 segundos).

El sistema deberá contar además obligatoriamente con un modo de funcionamiento, seleccionable de forma programática, que en caso de encontrarse activo permita **mantener informado al usuario** en todo momento. Para ello, se deberán mostrar diferentes **mensajes** en la pantalla del ordenador, a través de la **ventana de terminal** del entorno de desarrollo Eclipse (que actuará como display de visualización), acerca del **estado** en el que el sistema se encuentra (e.g. reproduciendo melodía) y de la ocurrencia de los diferentes **eventos de relevancia** para el funcionamiento del mismo (e.g. blanco alcanzado).

En la versión final, los **retardos** debidos a, por ejemplo, escribir por la terminal, controlar la posición de la torreta o accionar el cañón, **no deben afectar a la correcta reproducción de los efectos de sonido** (requisitos de concurrencia y tiempo real). Será, igualmente, competencia del sistema SW, generar y procesar las señales necesarias para gobernar el HW externo.

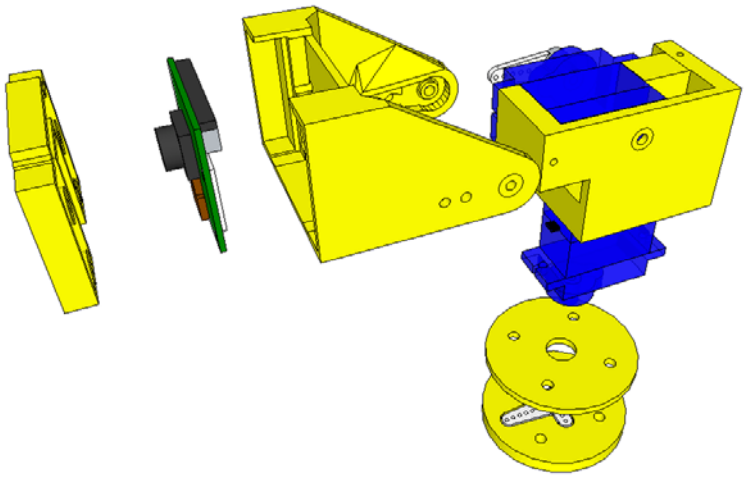

Ilustración 2. Diseño de las torretas disponibles en el laboratorio.

Los posibles **rebotes** originados tanto por el HW de control de la torreta como por el HW de detección de impactos **se suprimirán por SW**. En particular, la acción del joystick en una determinada dirección o la detección de un determinado impacto deben significar **una única actualización de la posición de la torreta** o **una única reproducción del efecto de sonido correspondiente** respectivamente eliminando cualquier posible efecto de los rebotes presentes en las señales recibidas por el HW de control o detección (la eliminación defectuosa de tales rebotes se pondrá típicamente de manifiesto en un comportamiento anómalo por parte del sistema que reaccionará como si hubiésemos accionado repetidamente el joystick o como si hubiésemos realizado múltiples disparos).

Este enunciado constituye una **guía** para el diseño e implementación de nuestro prototipo funcional completo conforme al funcionamiento anteriormente detallado. Para ello, contaremos además con la ventaja de disponer de una Raspberry Pi, un microcontrolador de referencia y con altas prestaciones, así como también de otros elementos HW de relevancia para el diseño como es el caso de las propias torretas, equipadas con sus correspondientes microservos y disponibles en cada uno de los puestos del laboratorio (ver Ilustración 2).

Adicionalmente, el alumno dispone de un **vídeo demostrativo** sobre este proyecto en YouTube: https://www.youtube.com/watch?v=K0PquX_WW6M.

En los apartados siguientes se detallarán las arquitecturas HW y SW, haciendo énfasis en la descomposición modular del sistema, tarea clave para abordar con éxito el diseño de cualquier sistema HW o SW medianamente complejo.

3.3. Especificaciones mínimas OBLIGATORIAS

Todas las especificaciones de funcionamiento recogidas en el apartado anterior tendrán el carácter de **obligatorias** con la única **excepción del enlace infrarrojo**, cuya implementación tendrá carácter **OPCIONAL**.

El cumplimiento de las **especificaciones obligatorias** de funcionamiento permitirá obtener **hasta un máximo de 7 puntos sobre 10 en la nota final de la asignatura**. Adicionalmente, y con **carácter opcional**, los alumnos podrán **mejorar** su diseño, y en consecuencia su calificación, con cualesquiera otras modificaciones complementarias a las presentes especificaciones hasta alcanzar la máxima nota. En ese sentido, la implementación del **enlace infrarrojo** permitirá obtener **hasta 2 puntos adicionales.**

Ilustración 3. Diagrama de bloques del sistema.

4. Subsistema Hardware

En la Ilustración 3 se muestra el diagrama de los bloques o módulos principales que componen el sistema. En dicho diagrama se han destacado las relaciones existentes entre los diferentes módulos tanto desde el punto de vista de diseño SW, destacadas mediante flechas en color azul, como desde el punto de vista de diseño HW, destacadas mediante flechas en color naranja. En los siguientes apartados se irán describiendo las partes más importantes que forman el subsistema HW.

4.1. Subsistema de disparo

El subsistema HW de disparo corresponde a la parte del emisor de nuestro enlace infrarrojo. El elemento esencial de dicho emisor será, por supuesto, un diodo IRLED[6] (del inglés Infrared Light Emitting Diode), un emisor de rayos infrarrojos mediante radiación situada en el espectro electromagnético en el intervalo que va desde la luz visible a las microondas (invisibles para el ojo humano[7]). Estos diodos, empleados habitualmente en el diseño de mandos de control remoto en el ámbito doméstico, se diferencian de los LED por el color de la cápsula que los envuelve que es de color azul o gris. El diámetro de ésta es generalmente de 5 mm.

Para su control se propone el montaje descrito en la Ilustración 4. La idea fundamental consiste en **controlar el apagado o encendido del led por medio de la señal PI_OUT_TX**, señal correspondiente a uno de los pines GPIO disponibles empleado como salida (entrada de este subsistema). En particular, y dependiendo del valor fijado en dicha salida, procederemos al encendido o al apagado del led conmutando el funcionamiento del transistor[8] entre la saturación (encendido) y el corte (apagado).

[6] Por ejemplo, el IR333-A, disponibles en el Servicio de Publicaciones de la Escuela, con unas características de directividad y alcance más que suficientes para nuestro proyecto.
[7] Aunque invisible al ojo humano, muchos dispositivos ópticos son capaces de captar la luz infrarroja; básicamente cualquier equipo con cámara digital integrada, por ejemplo, el móvil, permite su visualización, no obstante, el grado de sensibilidad puede variar entre un equipo y otro.
[8] Por ejemplo, un 2N2222A, disponibles en el Servicio de Publicaciones de la Escuela.

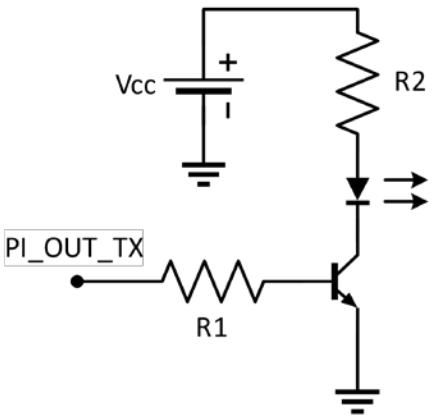

Ilustración 4. Detalle del subsistema HW de disparo propuesto.

El voltaje que alimenta el led IR fuerza la circulación de corriente a través de éste y provoca, a su vez, la emisión de un **haz de luz infrarroja** cuya intensidad depende de dicha corriente.

Nótese que la corriente de polarización del led coincide con la corriente de colector del transistor, dependiente a su vez de la corriente de base para la que será necesario establecer un valor fijo eligiendo convenientemente el valor de la resistencia R1. Dicha corriente de base será la correspondiente al encendido del emisor, situación que acontecerá cuando se produzca la escritura de un nivel alto (i.e. 5V) a la salida del pin GPIO (i.e. PI_OUT_TX) empleado al efecto.

En este sentido, a menor valor de R1, mayor intensidad de emisión y mayor alcance, no obstante, recuerde que debe evitar demandar una corriente de salida excesivamente alta a cualquiera de los pines GPIO de la Raspberry (i.e.<5mA). Igualmente, tenga en cuenta que el transistor amplificará notablemente dicha corriente haciéndola pasar por el led por lo que debe evitar niveles que puedan dañarlo.

Dado que nuestro emisor es un diodo emisor de luz infrarroja, su **polarización será similar a la de un LED** por lo que deberá consultar la caída de tensión en directa, así como el consumo de corriente recomendados por el fabricante, a fin de calcular correctamente la resistencia de polarización R2.

Nótese que, en el laboratorio B-043, las conexiones entre el HW externo y la Raspberry Pi **NO pueden ser establecidas de forma directa** e inmediata toda vez que esta última está integrada dentro de la plataforma ENT2004CF. Dicha plataforma permite acceder a los pines GPIO de la Raspi indirectamente, por medio de unos conectores ubicados en la placa externa llamada TL04. Las entradas y salidas de la TL04 están oportunamente conectadas a los pines GPIO de la propia Raspi a través de optoacopladores, situados en el interior de la plataforma e incorporados para la protección de dichos pines (dispone de más información al respecto en [3]). Aunque necesarias, estas protecciones funcionan unidireccionalmente, lo que se traduce en que **unos pines GPIO sólo pueden utilizarse como entradas y otros sólo como salidas**. En el anexo 8.1 se incluye una tabla con la descripción completa del conjunto de entradas y salidas finalmente disponibles en el laboratorio.

4.2.Subsistema de detección de impacto

Por su parte, el subsistema HW de detección de impacto consiste en el receptor de nuestro enlace infrarrojo. Este receptor consta fundamentalmente de un fotodiodo, dispositivo sensible a la incidencia de la luz, en nuestro caso, infrarroja.

Para su correcto funcionamiento es necesario polarizarlo inversamente. De esta manera, si la luz incidente tiene el nivel suficiente se generará una cierta corriente[9] traducida a un voltaje por medio de la resistencia R3 (a mayor luz incidente, mayor corriente inducida y mayor voltaje o diferencia de tensión observada en la resistencia). La tensión resultante en dicha resistencia (i.e. V+ del amplificador[10]) será entonces comparada con otra tensión de referencia (i.e. V- del amplificador), ajustable mediante la acción del potenciómetro, de tal modo que puedan producirse dos posibles situaciones dependiendo de la relación entre ambas tensiones.

La primera situación posible es que haya luz incidente que eleve la tensión en R3 por encima de la de referencia y que, por tanto, la tensión de salida (señal RX_IRQ) sea igual a Vcc. Por otro lado, la segunda situación posible correspondería al hecho de que no hubiese luz incidente (o de que ésta resultase insuficiente) quedando la tensión en R3 por debajo de la tensión de referencia y obteniéndose así un cero a la salida del amplificador.

Dependiendo de la sensibilidad del fotodiodo empleado[11] será necesario ajustar manualmente por medio del potenciómetro la tensión de referencia a fin de **obtener una salida adecuada para su oportuna lectura por una de las entradas digitales del GPIO** de la Raspberry. De igual modo, nótese que el sol produce una gran cantidad de radiación IR (también las lámparas de tungsteno), por lo que será conveniente tomar ciertas precauciones a la hora de implementar nuestro receptor. En particular, y para evitar que el fotodiodo esté expuesto a la luz directa del sol u otras fuentes de luz (mejorando así su directividad y sensibilidad), se recomienda cubrir el mismo introduciéndolo en un tubo o cánula construidos con algún material opaco como puede ser el cartón.

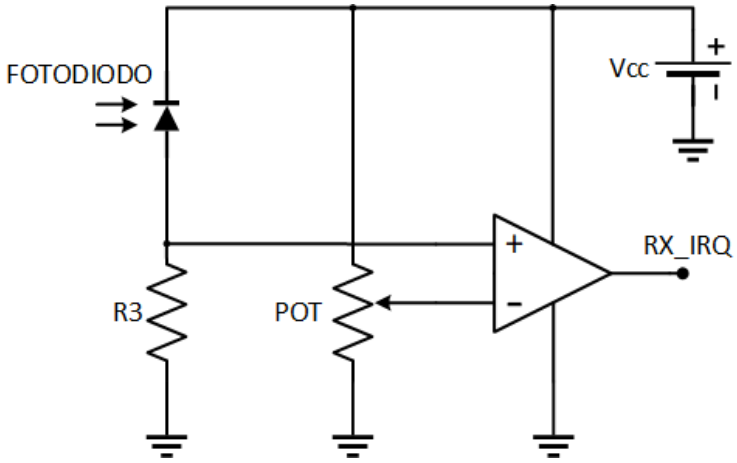

Ilustración 5. Detalle del subsistema HW de detección de impacto propuesto.

[9] En ausencia de luz la corriente presente es muy pequeña y recibe el nombre de corriente de oscuridad.
[10] Por ejemplo, un TL082, disponibles en el Servicio de Publicaciones de la Escuela.
[11] Por ejemplo, el PD333-3B/H0/L2, disponibles en el Servicio de Publicaciones de la Escuela.

Finalmente, y a modo de resumen, la **activación y caída del enlace** producirá respectivamente **flancos** de subida o bajada en la señal de salida de nuestro subsistema HW de detección de impacto. Dichos flancos podrán ser oportunamente detectados conectando para ello la salida del subsistema a alguna de las **entradas de interrupción** disponibles en la Raspberry Pi. La elección y configuración del pin en cuestión será responsabilidad del alumno.

4.3.Subsistema de movimiento y control de la torreta

4.3.1. Control de la torreta

Para la implementación del subsistema HW de control de la torreta se proponen diferentes alternativas. La más sencilla pasa por emplear el **teclado matricial de la placa TL04** disponible en cada puesto de laboratorio (ver Ilustración 6).

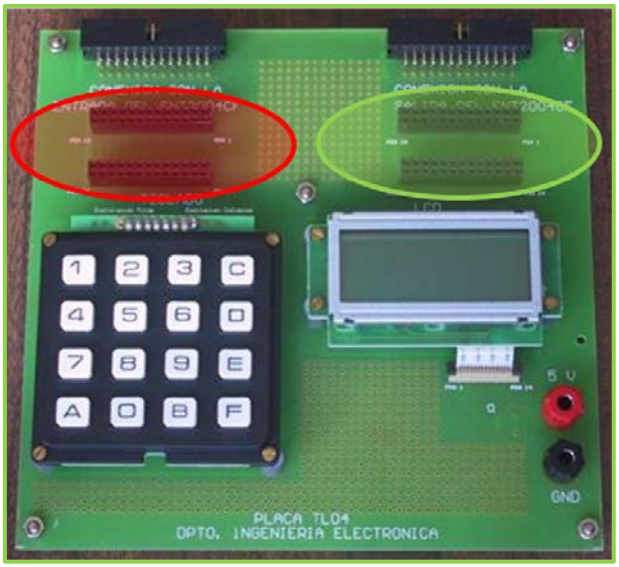

Ilustración 6. Detalle del teclado matricial disponible en la placa TL04.

Se trata de un teclado matricial de 16 teclas cuyo funcionamiento, representado en la Ilustración 7, se basa en la excitación secuencial de las columnas y la consiguiente exploración secuencial de las filas. Para ello el teclado cuenta específicamente con conexiones a los siguientes pines GPIO de la Raspberry Pi: 4 salidas para excitar las columnas y 4 entradas para explorar las filas, cuyos detalles aparecen recogidos en la siguiente tabla.

ENTRADAS				SALIDAS			
Placa TL-04 conexión entradas entrenador		Número del puerto BCM sólo utilizable como entrada	ENT2004CF LEDS ROJOS	ENT2004CF LEDS VERDES	Número del puerto BCM sólo utilizable como salida	Placa TL-04 conexión salidas entrenador	
Pin							Pin
1	Teclado Fila 1	**5**	LRE0	LVS0	**0**	Teclado Columna 1	1
2	Teclado Fila 2	**6**	LRE1	LVS1	**1**	Teclado Columna 2	2
3	Teclado Fila 3	**12**	LRE2	LVS2	**2**	Teclado Columna 3	3
4	Teclado Fila 4	**13**	LRE3	LVS3	**3**	Teclado Columna 4	4

Tabla 1. Conjunto de pines GPIO empleados por el teclado matricial.

En cada momento sólo una de las columnas permanece excitada lo que permite la detección de cualquier pulsación que pueda afectar a cualquiera de las teclas dispuestas en dicha columna (en la figura la activación o excitación de la primera columna sólo permitiría la detección de pulsaciones en las teclas 1, 4, 7 ó A).

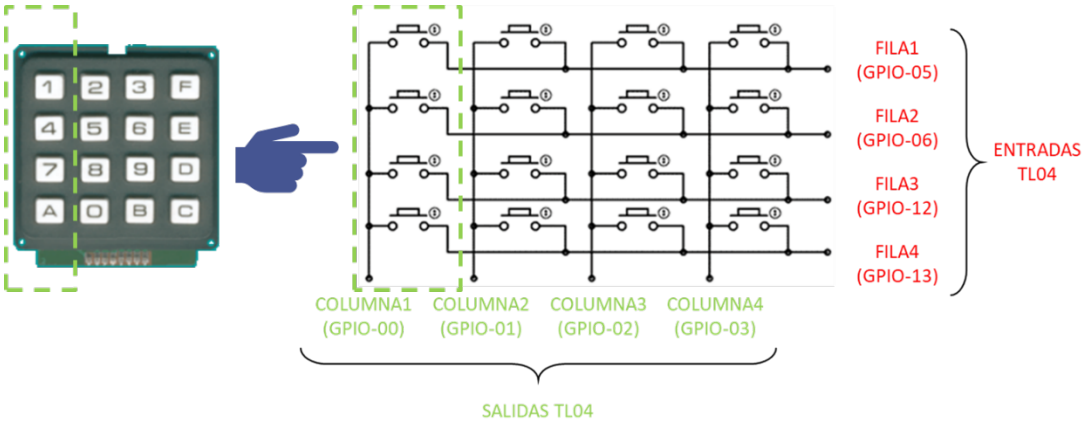

Ilustración 7. Detalle del funcionamiento del teclado matricial.

Para permitir la detección de pulsaciones en el resto de teclas, la excitación debe actualizarse periódicamente trasladándose de forma cíclica al resto de columnas. En ese sentido, el tiempo transcurrido entre diferentes excitaciones de una misma columna deberá ser lo suficientemente breve como para evitar la pérdida o no detección de lo que podríamos denominar una pulsación rápida o de corta duración.

Cada botón o tecla del teclado hace las veces de un pulsador de manera que el esquemático equivalente para cualquiera de las filas o líneas de entrada es el representado en la Ilustración 8, en la cual se pone de manifiesto la no necesidad de incorporar la habitual resistencia de pull-down (o pull-up) propia de estos montajes, al contar ya la entrenadora con dichas resistencias (i.e. de valor 10K para cada una de las entradas). Recuerde que estas resistencias permiten establecer un determinado estado lógico (i.e. un 0 en nuestro caso) en la entrada de nuestro circuito cuando el pulsador se encuentra en estado de reposo.

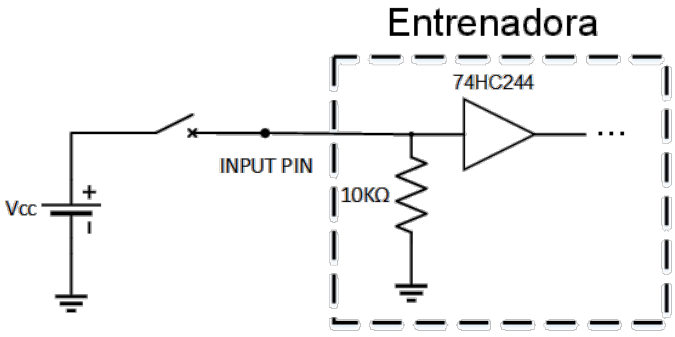

Ilustración 8. Esquemático equivalente para cada tecla.

Como alternativa al teclado de la TL04 se propone el empleo de un *interruptor de joystick*[12] similar al mostrado en la Ilustración 9, donde se incluye además su correspondiente diagrama circuital. Este joystick es un control direccional plano de cuatro direcciones (se dice que es "digital" o que, en otras palabras, sólo se pueden usar las direcciones provistas en los botones de la cruceta, sin valores intermedios), usualmente controlado por el pulgar con un botón en cada punto. Este tipo de dispositivos, similares a las crucetas o pads direccionales están presentes en casi todos los mandos de las consolas de videojuegos modernas o en los controles remotos de algunas TV.

[12] https://es.rs-online.com/web/p/products/0516316/

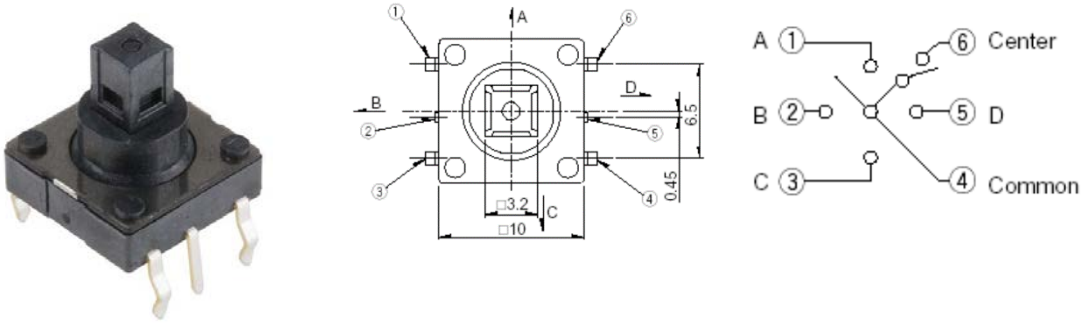

Ilustración 9. Joystick sugerido y diagrama circuital correspondiente.

Su principal ventaja es que se pueden manipular fácilmente (requieren poco movimiento del pulgar) con una precisión muy alta. Su implementación resulta incluso más sencilla que la basada en el teclado TL04 aunque su coste es más elevado, no sólo en términos económicos (es necesario adquirir el joystick), sino también en cuanto a la cantidad de entradas de interrupción necesarias (1 más que en el caso del teclado de la TL04).

4.3.2. Movimiento de la torreta: servomotores

Un servomotor (o servo) es un tipo especial de motor con características especiales de control de posición: es posible controlar la posición de su eje en todo momento. Ésto hace que su aplicación sea especialmente útil en el ámbito de la robótica o en la industria, donde suelen ser necesarios movimientos precisos y la posibilidad de mantener posiciones fijas.

Un servo está diseñado para moverse una determinada cantidad de grados y luego **mantenerse fijo en una posición**. El servo (ver Ilustración 10) es un sistema compuesto por componentes electromecánicos y electrónicos. En su interior encontramos un motor DC común y un circuito electrónico encargado de manejar el movimiento y la posición del motor. El eje del motor está acoplado a una caja de engranajes similar a la transmisión de un automóvil (a mayor velocidad, menor torque). Dicho sistema de engranajes, similar al mostrado en la figura, hace que al mover el eje motor se sienta una inercia muy superior a la de un motor común y corriente, lo que potencia el torque del motor y permite mantenerlo en una posición fija cuando se requiera.

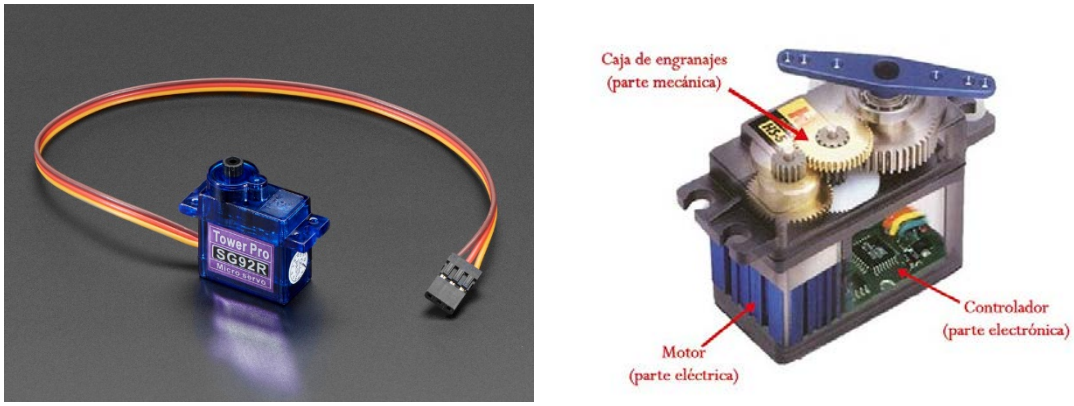

Ilustración 10. Servomotor similar a los disponibles en el laboratorio.

En particular, los servos que emplearemos en nuestro proyecto son un tipo especial de servo habitualmente utilizado en modelismo. Estos servos operan a voltajes bajos en corriente directa, específicamente entre 3 y 6 voltios.

Ilustración 11. Colores comunes de los cables de un servomotor.

Desde el punto de vista de sus características de rotación, estos servos son "de rango de giro limitado[13]" lo que implica que son incapaces de completar una vuelta completa. Este tipo de servo, el más común, solamente permite una rotación de aproximadamente 180 grados, movilidad limitada pero suficiente para nuestra torreta.

Un servo posee tres cables cuyos colores, aunque pueden variar dependiendo del fabricante, son casi siempre los mismos, lo que los hace fácilmente reconocibles. Los colores más habituales son los detallados en la Ilustración 11 y corresponden respectivamente a la alimentación, masa y control del motor. A través del **cable de control** es preciso el envío, al circuito de control interno, de una **señal de control modulada** que resulta esencial para el correcto funcionamiento del servo. Para ello se utiliza una modulación por ancho de pulsos, es decir, una **señal PWM**.

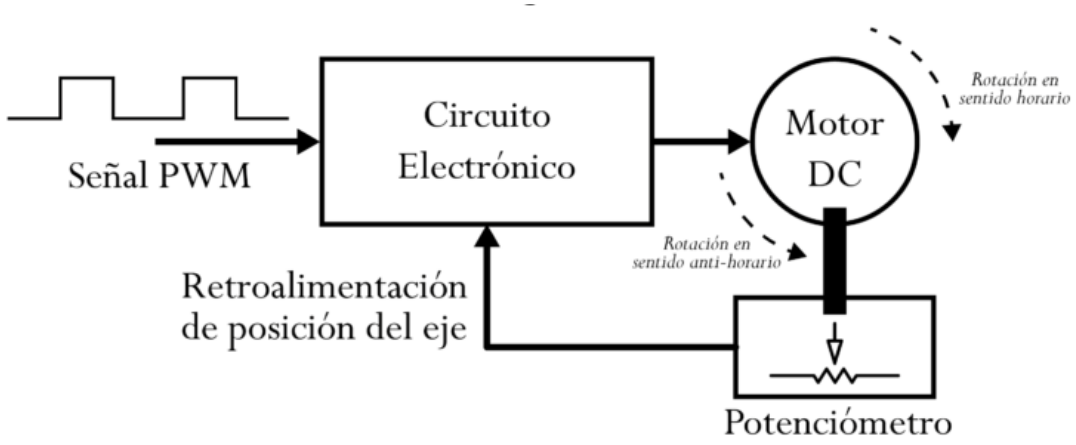

Ilustración 12. Diagrama de bloques de un servo.

El servo funciona como un sistema de control de lazo cerrado (ver Ilustración 12). Primero, el circuito de control recibe dicha señal PWM y la traduce en movimiento del motor DC. La forma de la señal PWM es la de una onda rectangular cuya **anchura de pulso nos permite especificar la posición fija que queremos que adopte el motor** (ver Ilustración 13). A continuación, el movimiento del eje del motor hace variar la posición del cursor de un potenciómetro al que el eje está acoplado. Este potenciómetro entrega entonces un voltaje proporcional a la posición actual del motor, información que es realimentada a la entrada. Finalmente, se calcula el valor del error de posición, que es la diferencia entre la posición objetiva deseada y la posición en que se encuentra el motor (calculada como la resta de los dos valores de tensión correspondientes). Un error de posición mayor significa que hay una diferencia mayor entre el valor deseado y el existente, de modo que el motor deberá rotar más rápido para alcanzarlo; uno menor, significa que la posición del motor está cerca de la deseada por el usuario, así que el motor tendrá que rotar más lentamente. Si el servo se encuentra en la posición deseada, el error será cero, y no habrá movimiento.

[13] También existen servos de rotación continua caracterizados por ser capaces de girar 360 grados, es decir, una rotación completa.

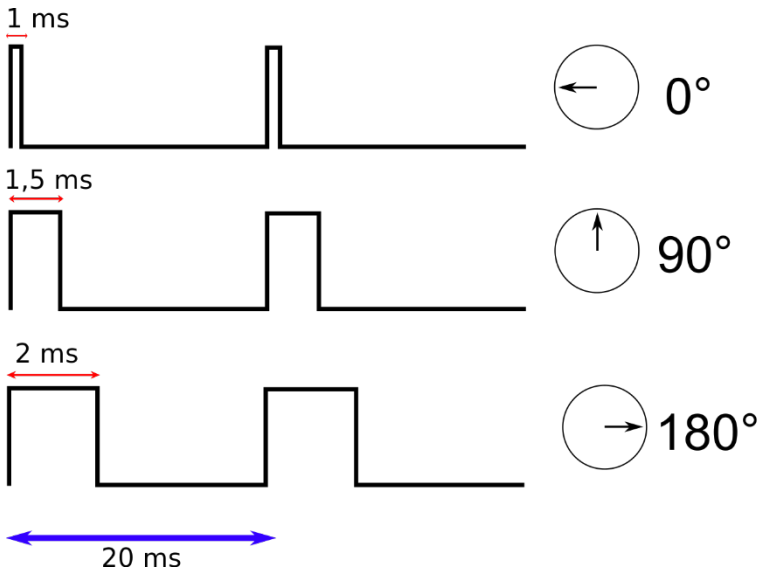

Ilustración 13. Ejemplos de trenes de pulsos para las posiciones 0º, 90º y 180º en el eje de un servo. La posición del servo tiene una proporción lineal con el ancho del pulso utilizado.

Para bloquear el servo en una posición, es necesario enviarle continuamente la señal con la posición deseada. De esta forma, el sistema de control seguirá operando, y el servo conservará su posición y se resistirá a fuerzas externas que intenten cambiarlo de posición. Si los pulsos no se envían, el servomotor quedará liberado, y cualquier fuerza externa podrá cambiarlo de posición fácilmente.

La duración del ciclo de trabajo de la señal de control PWM varía entre 15 y 25 milisegundos. Las señales mostradas en la Ilustración 13 representan ejemplos de trenes de pulsos con los que se pueden fijar diferentes posiciones para un servo, utilizando un **ciclo de trabajo de 20 milisegundos** (i.e. frecuencia de 50 Hz).

Aunque las señales PWM podrían ser generadas vía HW, por ejemplo mediante un circuito oscilador basado en un 555, en nuestro caso ambas señales de control (recuerde que debemos gobernar dos servos) serán generadas vía SW por medio de la Raspberry Pi a través de dos de sus salidas digitales y apoyándonos para ello en la librería "Software PWM[14]" de WiringPi (encontrará más información acerca de la librería en [3]).

4.4.Subsistema de reproducción de efectos de sonido

En nuestro proyecto, la plataforma ENT2004CF será la responsable de la síntesis de los diferentes efectos de sonido. Para ello, consideraremos cada efecto como un conjunto de notas musicales a reproducir durante un tiempo determinado. Para cada nota la Raspberry Pi deberá generar una señal rectangular con la frecuencia apropiada a través de uno de los pines de salida del GPIO disponibles en el conector de salidas digitales de la TL04 (ver anexo 8.1). La generación de dichas señales cuadradas se realizará vía software por medio de la librería "Software Tone Library[15]" de WiringPi (encontrará más información acerca de la librería en [3]).

[14] http://wiringpi.com/reference/software-pwm-library/
[15] http://wiringpi.com/reference/software-tone-library/

Si se aplica esta señal directamente sobre un altavoz podremos reconocer perfectamente la nota reproducida (aunque la percibiremos con una fuerte distorsión). Opcionalmente, podrá realizarse un acondicionamiento de la señal antes de enviarla al altavoz. Este acondicionamiento consistiría en un filtrado y una etapa de potencia para excitar el altavoz.

En el anexo 8.4 se muestran las frecuencias (en Hz) y las duraciones de las notas (en milisegundos) para los efectos de sonido que emplearemos en este proyecto.

4.4.1. Filtrado de la señal (opcional)

La misión del filtro tiene como objetivo principal reducir el número de armónicos, reduciendo también la sensación de distorsión de la nota reproducida. Este filtro será un filtro paso bajo con frecuencia de corte superior igual a 12kHz (para permitir al menos 2 armónicos de la nota con mayor frecuencia: última nota de la octava 7).

En un principio podríamos pensar en generar una señal sinusoidal perfecta pero este tipo de señal no simula correctamente el sonido producido por un instrumento, el cual tiene más riqueza espectral. Por esta razón, colocaremos un filtro que reduzca los armónicos pero que mantenga los 3 primeros. En el apartado de mejoras se propondrá la generación de diferentes tipos de onda para simular diferentes instrumentos.

Para la implementación de este filtro paso-bajo se propone un filtro de 2º orden Sallen-Key con ganancia unidad, similar a los desarrollados en Circuitos Electrónicos (consultar enunciado de la práctica de CELT del presente curso, apartado 4.1.6 Filtro paso bajo). Un aspecto que debemos tener en cuenta en este caso es que la señal a filtrar es una señal digital entre 0 y +5 voltios, mientras el filtro lo alimentaremos entre –5 y +5V. Por esta razón, es necesario eliminar la componente continua utilizando una red RC (en configuración filtro paso alto) con valores de C y de R elevados de forma que la frecuencia de corte sea muy baja (de unos pocos Hz). A la salida del filtro debemos tener una señal de igual o menor rango dinámico que a la entrada.

4.4.2. Amplificador de potencia (opcional)

La salida de los circuitos digitales y de muchos AO (amplificadores operacionales) con los que se construyen los filtros tiene una limitación importante en relación con la corriente que pueden ofrecer. En muchos casos, esta corriente no es suficiente para excitar unos altavoces. Por esta razón, es necesario incluir un módulo de amplificación de potencia previo a los altavoces.

Para la implementación de este módulo se recomienda utilizar el amplificador de potencia LM386 [10] utilizando alguno de los montajes que propone el fabricante. En la Ilustración 14 se muestra el montaje más sencillo.

Un aspecto importante que el alumno debe tener en cuenta es que este montaje presenta una ganancia de 20. Por esta razón, se debe incluir una etapa de atenuación (marcada con un círculo en la figura) que reduzca el margen dinámico de la señal. Al realizar el montaje se recomienda al alumno que consulte las hojas de especificaciones del LM386 con el fin de ajustar correctamente los niveles de las señales y de la alimentación. La impedancia del altavoz (auriculares) se puede aproximar por una resistencia de alrededor de 10 ohmios.

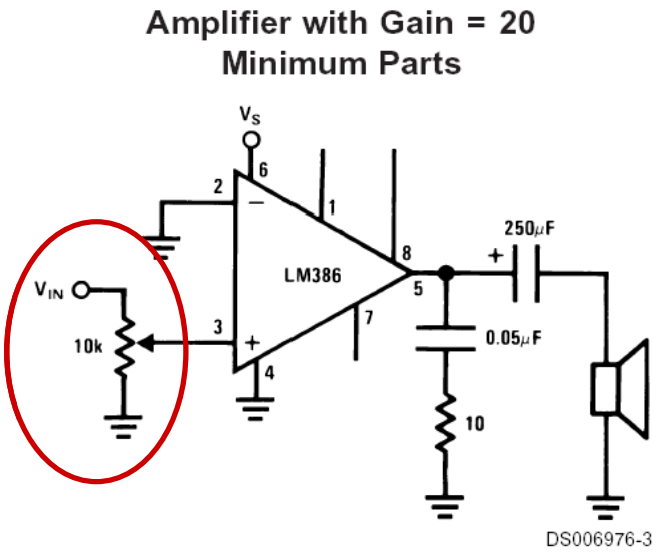

Ilustración 14. Propuesta de montaje para la implementación del amplificador de potencia.

NOTA IMPORTANTE: aunque los efectos se oigan razonablemente bien es importante comprobar con el osciloscopio el correcto funcionamiento tanto del filtro paso-bajo como del amplificador de potencia. Cuando se compruebe el funcionamiento del amplificador de potencia de forma aislada es importante probarlo utilizando señales sinusoidales (obtenidas del generador de funciones) con el fin de detectar posibles problemas de saturación.

4.4.3. Cascos o auriculares

Finalmente, la reproducción de las melodías se realizará utilizando auriculares y **NUNCA con altavoces**. El motivo es no incrementar innecesariamente el ruido del laboratorio, evitando molestar al resto de compañeros.

4.5. Observaciones adicionales sobre el HW

Es necesario recordar que los puertos de entrada disponibles en la plataforma ENT2004CF están formados por buffers digitales **con resistencias de pull-down de 10 kΩ**[16]. Si no se tiene en cuenta este hecho, se podrían producir efectos de carga no deseados al conectar el HW a la plataforma.

Igualmente importante, desde el punto de vista de la alimentación de los diferentes subsistemas, es el hecho de desacoplar las alimentaciones y alimentar en estrella.

Conviene también recordar que **el diseño y la compra** de componentes, **así como la implementación de los distintos montajes** circuitales, **se debe realizar con anterioridad a las sesiones de laboratorio**. Durante dichas sesiones, el objetivo fundamental del alumno debería ser la verificación y ajuste de los montajes requeridos. En ese sentido, resulta esencial la visualización de las diferentes señales implicadas en el osciloscopio.

[16] Los puertos de salida están disponibles directamente a través de buffers digitales.

5. Subsistema Software

El software estará basado en máquinas de estados que gestionarán la entrada/salida, las interrupciones y la temporización.

5.1. Esquema de procesos

Frecuentemente, los sistemas electrónicos digitales precisan realizar diversas acciones de una manera paralela (concurrente en varios procesadores) o aparentemente paralela (todas las acciones se ejecutan entrelazadas, produciéndose la apariencia de paralelismo si la frecuencia de conmutación entre ambas es elevada). Son los denominados **procesos**. La creación, ejecución, sincronización y destrucción entre procesos suele ser facilitada por el sistema operativo a través de diversas primitivas. Pero incluso en ese caso, el diseñador debe definir qué tareas o rutinas serán concurrentes, elegir los mecanismos de comunicación entre ellas, los tiempos de vida, etc.

En la Ilustración 15 se muestra a modo de ejemplo uno de los posibles esquemas de procesos que podrían emplearse para resolver el presente proyecto. Dicha ilustración pone de manifiesto la existencia de 2 procesos fundamentales:

- **el programa principal (PPAL)**: siempre existe y debe inicializar los objetos del sistema, inicializar el HW, inicializar y habilitar tanto los procesos de interrupción periódica como los vinculados a las interrupciones externas. Será igualmente el **responsable de inicializar y ejecutar las máquinas de estados** con las que controlaremos el sistema. Tal y como puede observarse en la Ilustración 15, dichas máquinas de estados se ejecutarán de forma permanente en un **bucle infinito** mediante la **invocación periódica** (i.e. 1 vez cada CLK_FSM milisegundos) de la función *fsm_fire* que, a su vez, será la encargada de ejecutar las funciones de comprobación (si se ha producido algún evento especial para el sistema, e.g. que haya ocurrido una interrupción) y actualización del estado del sistema. Nótese que el tiempo necesario para hacerlo (fragmentos en color azul en la figura) puede variar dependiendo del estado en el que se encuentre el sistema.

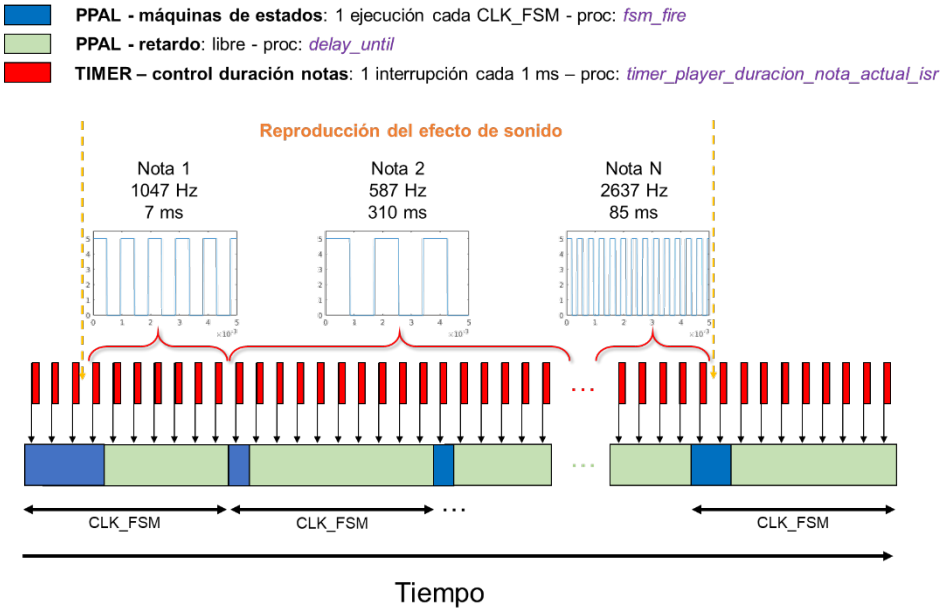

Ilustración 15. Detalle del esquema de procesos propuesto para el proyecto.

- **una interrupción periódica**[17] (**TIMER**): responsable de controlar la parte de **temporización del sistema relacionada con la reproducción de los efectos de sonido.** Tal y como sugiere la figura, el temporizador asociado actuaría a modo de contador de tiempo que permitiría controlar la duración de las notas. En particular, este temporizador se podría configurar para que interrumpiese cada 1 ms. De esta forma podríamos contar tiempos en milisegundos sin más que contar el número de interrupciones[18]. Su ejecución sería cíclica, discontinua y sucedería a intervalos regulares de tiempo (1 vez cada milisegundo); **sería concurrente con el proceso principal y estaría activo de forma permanente**, desde el mismo momento en que finalizase la inicialización del sistema e **independientemente del estado** en el que éste se encontrase. En nuestro caso adoptaremos una **solución alternativa**[19] **más simple**, también basada en un temporizador, aunque ligeramente distinta. En particular, controlaremos las duraciones de las notas a reproducir configurando dicho temporizador específicamente para cada nota de modo que éste **interrumpa una sola vez** al expirar el intervalo de tiempo correspondiente a la duración de la nota en cuestión. Como consecuencia inmediata, nuestro temporizador **sólo estará activo durante la reproducción** de las distintas notas de cada melodía.

Al margen del programa principal y de cualquier proceso que requiera de algún control del tiempo como el mencionado, **serán igualmente necesarias 5 interrupciones externas** (6 si se opta por la versión basada en el joystick de control como alternativa al teclado de la TL04) ligadas, respectivamente, a las **acciones de control de la torreta** (pulsadores o teclas para el disparo de la torreta y para el movimiento hacia arriba, abajo, izquierda y derecha de la misma) y a la **detección de impactos.**

Este esquema de procesos es obligatorio, en particular en cuanto concierne al uso de máquinas de estados.

Para la implementación de sistemas basados en máquinas de estados será de obligada lectura [5]. En ese documento encontrará la información necesaria para la definición y uso de máquinas de estados identificando para ello: los distintos estados del sistema, las posibles transiciones entre ellos, las funciones de entrada (o de comprobación de eventos), y las funciones de salida (o de actualización del sistema) que gobiernan dichas transiciones.

Para el desarrollo de software de tiempo real sobre la plataforma disponible en el laboratorio es indispensable la lectura de [4]. En ese documento encontrará la información necesaria para la definición de diferentes procesos por medio de interrupciones periódicas, interrupciones externas y el programa principal, así como para la provisión de los requeridos mecanismos de sincronización entre todos ellos.

5.2. Proceso principal y máquinas de estados

El sistema completo propuesto estará basado en **tres máquinas de estados:** una genérica para el **control de la excitación del teclado**, representada en la Ilustración 16 y cuyo funcionamiento está detallado en [3], otra para **controlar el subsistema de**

[17] Dos si se implementa el enlace IR: 2º temporizador necesario para controlar la duración del disparo.

[18] De ahí que los tiempos de las notas se expresen en milisegundos. Además, el milisegundo es una unidad suficientemente pequeña como para garantizar una resolución aceptable en su reproducción.

[19] Otra posible solución consistiría en prescindir del temporizador y medir tiempos apoyándose para ello en la temporización intrínseca que articula el comportamiento periódico de las propias máquinas de estados (i.e. 1 interrupción o invocación de la función fsm_fire cada CLK_FSM milisegundos).

reproducción de efectos de sonido, representada en la Ilustración 17, **y otra para el control de la torreta**, representada en la Ilustración 18**.** Estas dos últimas máquinas de estados serán objeto de análisis a continuación.

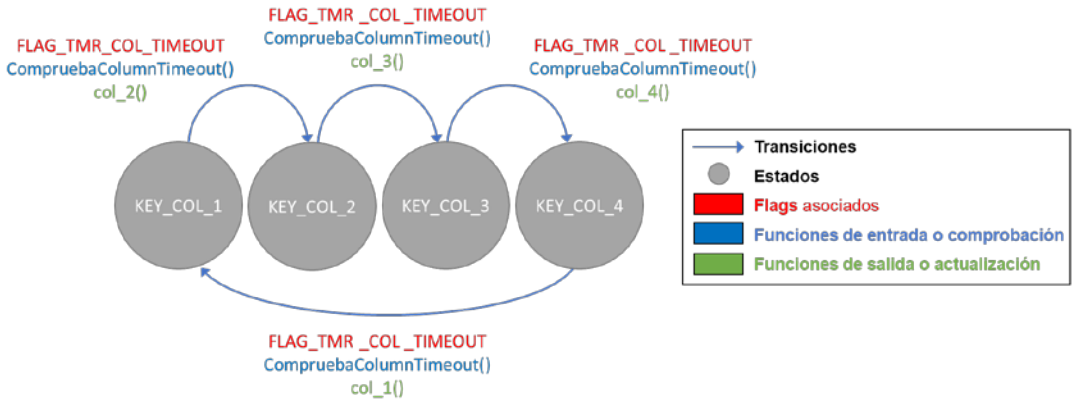

Ilustración 16. Máquina de estados propuesta para el control de la excitación del teclado.

Nótese que en todos los casos se han destacado oportunamente los siguientes elementos: los posibles **estados** de cada subsistema, las **transiciones** contempladas entre todos ellos, las **funciones de comprobación** (en azul) de cuyo resultado depende el que se hagan efectivas dichas transiciones, los **flags** específicos (en rojo) que son objeto de comprobación por parte de dichas funciones y, finalmente, las **funciones de actualización** (en verde) responsables de llevar a cabo las acciones necesarias para instaurar el nuevo estado cuando proceda efectuar la transición en cuestión.

5.2.1. Control de la reproducción

El subsistema de reproducción debe permanecer inicialmente a la espera (i.e. estado S1 o *WAIT_START*) de que se produzca un evento que dé lugar al comienzo de la reproducción, en nuestro caso el ligado al flag *FLAG_START_DISPARO* o al flag *FLAG_START_IMPACTO*, cuya activación tendrá lugar cuando el usuario accione el botón o tecla de disparo o cuando se produzca la detección de un impacto, respectivamente. Tanto el disparo como el impacto darán paso a la inicialización del procedimiento de reproducción (procedimiento *InicializaPlayDisparo* o *InicializaPlayImpacto*) y al siguiente estado (i.e. S2 o *WAIT_NEXT*) en el que se reproducirá la primera nota del efecto de sonido correspondiente. La nota reproducida podrá cambiar con cualquier transición posible desde el resto de estados al estado *WAIT_NEXT*.

Tras el comienzo de la reproducción (i.e. estado *WAIT_NEXT*) podrán producirse los siguientes eventos, cuya detección será obligatoria para conseguir que el efecto de sonido suene correctamente:

- *FLAG_NOTA_TIMEOUT*: evento ligado a uno de los flags de estado del subsistema que será indicativo de la finalización del intervalo de tiempo correspondiente a la duración prevista para la nota que se estuviese reproduciendo en ese momento. Significará la necesidad de actualizar el estado del reproductor (accediéndose al estado S3 o *WAIT_END*) para finalizar la reproducción de la nota en curso (i.e. *ActualizaPlayer*) y, acto seguido, resolver la necesidad de, eventualmente, dar por finalizada la reproducción del efecto (i.e. retorno a S1 vía *FinalEfecto*), o de dar comienzo a la reproducción de la siguiente

nota (i.e. retorno a S2 vía *ComienzaNuevaNota*), dependiendo, respectivamente, de la activación o no del flag *FLAG_PLAYER_END*[20].

- *FLAG_START_IMPACTO* (i.e. auto-transición en estado *WAIT_NEXT*): evento ligado a la detección de un impacto antes de que finalice la reproducción del efecto ligado al disparo. En este caso, damos por finalizado el efecto de disparo en curso simplemente reiniciando la reproducción de forma específica para el efecto asociado al impacto detectado (vía *InicializaPlayImpacto*).

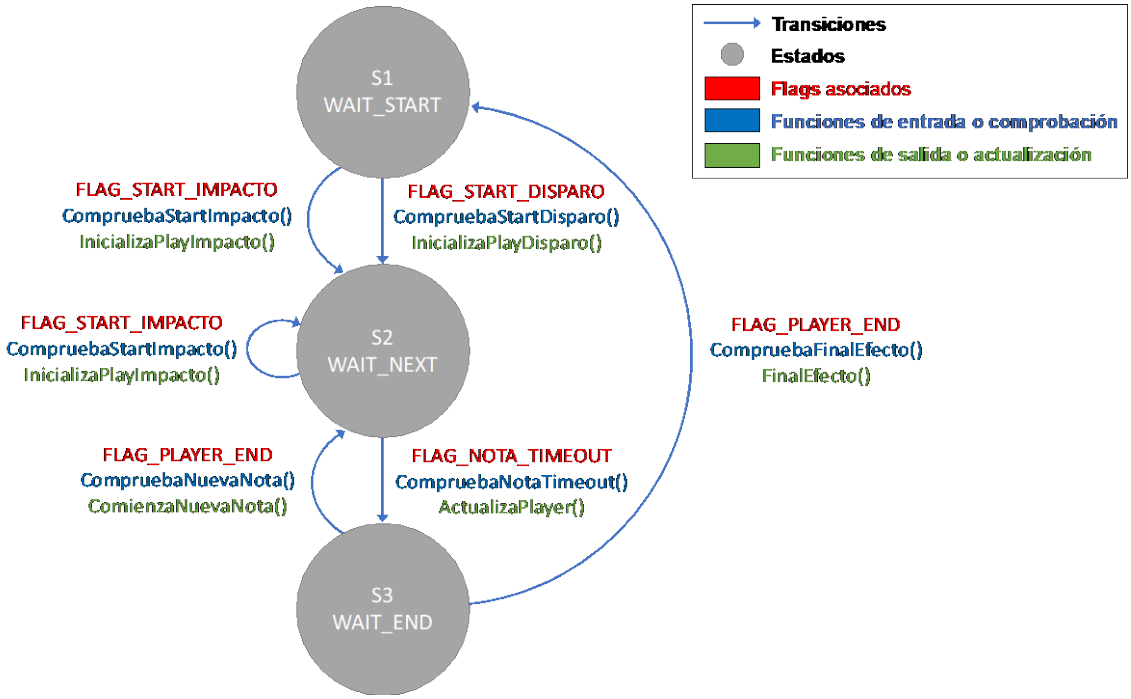

Ilustración 17. Máquina de estados propuesta para el control de la reproducción de efectos de sonido.

5.2.2. Control del juego y del cañón

Se trata, sin duda, del subsistema más importante desde el punto de vista del comportamiento del sistema final toda vez que, al margen de orquestar los procedimientos necesarios para gestionar el control de la torreta y el enlace infrarrojo, será además quién gobierne, mediante la activación y desactivación de los flags adecuados, el estado del otro subsistema: el de reproducción.

Como se ha comentado previamente, la interfaz de control para el usuario consiste fundamentalmente en el teclado matricial disponible en la TL04 (o el joystick) con el que podemos apuntar y disparar nuestro cañón infrarrojo. El comportamiento descrito a continuación se repetirá indefinidamente hasta que se apague o reinicie el sistema.

El sistema parte de un estado inicial (i.e. S1 o WAIT_START) en el que se debe llevar a cabo la oportuna inicialización del sistema y presentar el correspondiente mensaje informativo indicando al usuario que el sistema está listo para ser utilizado.

Una vez completada dicha inicialización, el sistema permanecerá a la espera de que el usuario accione el dispositivo de control (i.e. estado *WAIT_MOVE*). La ocurrencia de tal evento vendría ligada a la activación del cualquiera de los flags *FLAG_JOYSTICK_XXX*.

[20] Nótese que en el caso de la función de comprobación *CompruebaNuevaNota* es la NO activación del flag *FLAG_PLAYER_END* lo que provoca la transición correspondiente al estado *WAIT_NEXT*.

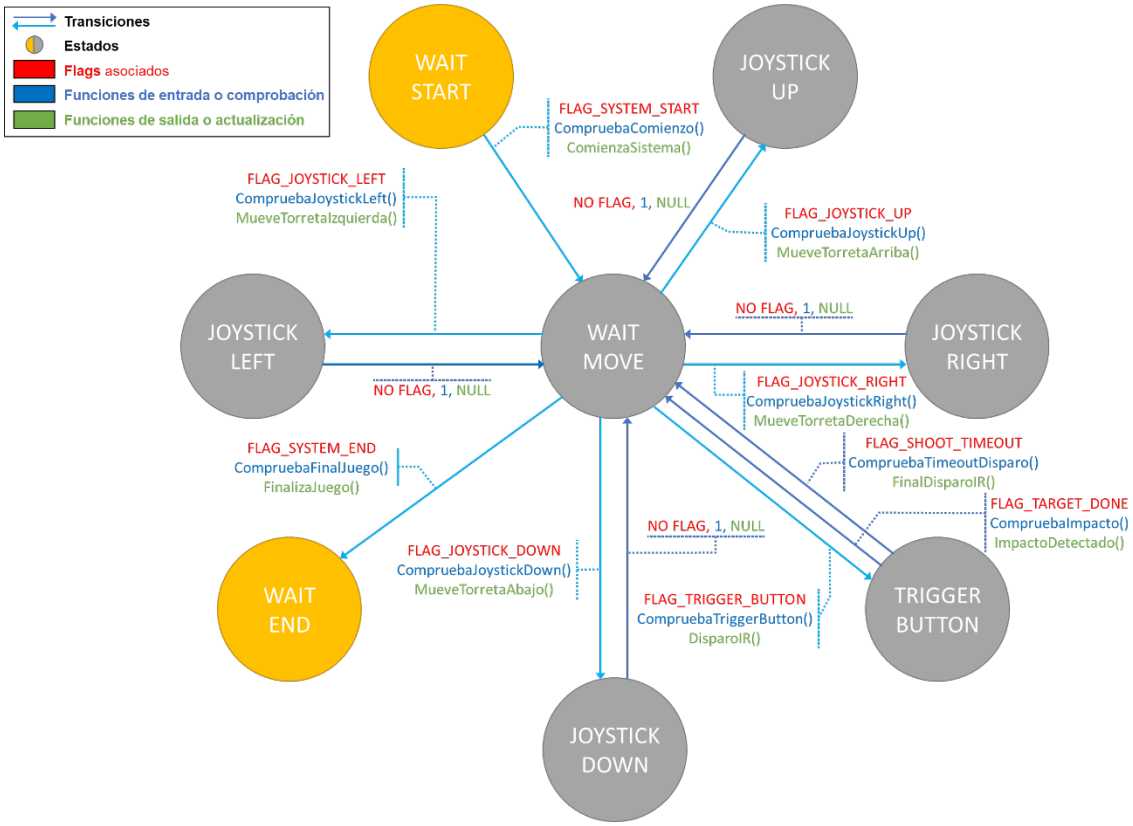

Ilustración 18. Máquina de estados propuesta para el control del juego.

En este sentido, es preciso distinguir dos situaciones claramente diferenciadas: acciones sobre las teclas de dirección que provocan la actualización de la posición de la torreta y acciones sobre la tecla de disparo que provocan la activación del enlace infrarrojo y consiguiente comprobación del posible impacto.

Las primeras son gestionadas todas ellas de la misma manera: el movimiento de la torreta es **discontinuo** por lo que debe producirse **a partir de pequeños incrementos** en cualquiera de los ejes, tanto para el vertical como para el horizontal. Además, para no forzar demasiado los servos, la **velocidad** a la que podremos desplazar la torreta estará necesariamente **limitada**. Para ello, cada pulsación de tecla de dirección (e.g. tecla “derecha”), consignada mediante la activación de su flag correspondiente (i.e. flag *FLAG_JOYSTICK_RIGHT*), producirá la transición al estado específicamente definido para el movimiento en cuestión (i.e. estado *JOYSTICK_RIGHT*), ejecutándose así el procedimiento necesario para la ejecución del movimiento (i.e. procedimiento *MueveTorretaDerecha*()) y permaneciendo en dicho estado por tiempo equivalente a un periodo de la señal de reloj CLK_FSM, que define la frecuencia de actualización de nuestras máquinas de estado.

Nótese que la única transición posible desde cualquiera de dichos estados de movimiento es la que devuelve el sistema al estado *WAIT_MOVE*, algo que sucederá siempre en la siguiente actualización (i.e. llamada a la función *fsm_fire*()) de nuestra máquina de estados (en la figura se indica un simple “1” como función de comprobación ligada a dichas transiciones lo que es equivalente a decir que no hay nada que comprobar o que, por tanto, la transición debe producirse por definición siempre que actualicemos la máquina partiendo desde cualquiera de los estados implicados).

Las acciones sobre la tecla de disparo, ligadas a la activación del flag *FLAG_TRIGGER_BUTTON*, provocarán la transición al estado *TRIGGER_BUTTON* y deberán dar lugar a la activación inmediata del enlace infrarrojo (habilitando el correspondiente temporizador cuyo oportuno timeout pueda ser aprovechado para el apagado del emisor IR) y al comienzo de la reproducción del efecto de sonido asociada al disparo.

Para ello, es fundamental que el presente subsistema de control del juego, por medio del procedimiento de actualización correspondiente (i.e. *DisparoIR*), comunique al subsistema de reproducción tal necesidad, haciéndole notar igualmente el efecto que debe ser reproducido (i.e. activación del flag *FLAG_PLAYER_START*).

Una vez que hayamos alcanzado el estado *TRIGGER_BUTTON*, nuestro sistema permanecerá en ese mismo estado hasta que se produzca alguno de los siguientes eventos:

- uno, que sin haber alcanzado el blanco expire el tiempo previsto para la duración máxima del disparo (i.e. función de comprobación o entrada *CompruebaTimeoutDisparo,* ligada al flag *FLAG_SHOOT_TIMEOUT*, activado por la rutina o procedimiento de atención a la interrupción asociada al temporizador),
- dos, que hayamos alcanzado el blanco (i.e. función de comprobación *CompruebaImpacto,* ligada al flag *FLAG_TARGET_DONE*, activado por la rutina de atención a la interrupción asociada a la entrada digital GPIO conectada al subsistema HW de detección de impacto).

En el primer caso, simplemente procederemos a apagar el emisor IR, mientras que en el segundo será además necesario actualizar el contador de blancos alcanzados, detener la reproducción del efecto de sonido ligado al disparo (i.e. activación del flag *FLAG_PLAYER_STOP* en el procedimiento de actualización correspondiente: *ImpactoDetectado* para que el subsistema de reproducción pueda detener de forma efectiva la reproducción del efecto en curso) y dar comienzo a la reproducción del otro efecto asociado al impacto, comunicándoselo al subsistema de reproducción de un modo similar al anterior.

Finalmente, y una vez alcanzado el número de blancos objetivo (i.e. *FLAG_SYSTEM_END*), pasaremos al estado de finalización del juego (i.e. *WAIT_END*), estado en el que podremos aprovechar para informar al jugador del resultado del juego y en el que podremos permanecer hasta que cualquier nueva acción por parte del jugador signifique el reinicio de una nueva partida retornándose al estado inicial.

5.3.Interrupciones

Las máquinas de estados pueden hacer uso de las interrupciones, independientemente de que estas sean periódicas (i.e. vinculada a un temporizador) o externas (i.e. vinculada a una entrada digital), por medio de la misma aproximación presentada en [5]: utilizaremos una **variable global para indicar si se ha producido o no** la interrupción en cuestión (a modo de flag). De este modo, la ocurrencia de cada interrupción provocará la llamada y ejecución de una **función de atención a la interrupción** (procedimientos cuyo nombre termina por _isr conforme a la nomenclatura empleada en [5], correspondiente a Interrupt Service Routine), **que pondrá a 1 la variable activando el correspondiente flag**.

Cuando el sistema, a través de la máquina de estados, haya tenido en cuenta el evento reflejado por dicha activación, volverá a poner la variable, y con ello el flag, a 0. Con objeto de simplificar la gestión de las diferentes interrupciones y sus correspondientes flags asociados, en vez de tener una variable para cada interrupción, como solo hace falta un bit, **utilizaremos una única variable global *flags* para todas** (tenemos para 32 interrupciones).

El uso de un conjunto de máscaras con las funciones binarias OR y AND nos permitirán referirnos a cada uno de los flags por separado para su oportuna lectura o escritura (el alumno debe leer [5] para comprender en detalle estos mecanismos; también puede consultar el anexo 8.2 incluido al final del presente documento).

Como consecuencia inmediata de simplificar, según el modo sugerido, la atención a las interrupciones, el tiempo necesario para completar dicha atención será despreciable en comparación con el requerido por *fsm_fire* para actualizar (si procede) el estado del sistema. El alumno **deberá diseñar un valor adecuado para la constante CLK_FSM** que permita una frecuencia de actualización del estado del sistema lo suficientemente alta como para que los tiempos de respuesta sean bajos (algo necesario para una buena experiencia de juego y una correcta reproducción de los efectos de sonido), y sin que por ello se vean comprometidas ninguna de las tareas a realizar en el peor escenario posible (i.e. co-ocurrencia múltiple de eventos entre sucesivas llamadas a *fsm_fire*).

Nótese que las interrupciones, ya sean de naturaleza externa o ligadas a un temporizador, pueden interrumpir la ejecución del programa principal en instantes de tiempo que resultan imprevisibles en el momento de escribir el código. Por tanto, será **fundamental contar con los mecanismos de sincronización oportunos que nos permitan actualizar correctamente todos los objetos compartidos** (conjunto de variables globales) entre los distintos procesos con el fin de intercambiar información. Preste para ello especial atención al capítulo 6 de [4] y en particular a la sección 6.5. Sincronización de threads.

6. Desarrollo recomendado

Ésta es una **planificación orientativa** en relación al posible desarrollo del proyecto. Tal y como el alumno podrá observar, el diseño propuesto ha sido dimensionado en coherencia con las 12 sesiones de laboratorio disponibles[21]. En este sentido, es importante destacar que **la parte estrictamente obligatoria** (y suficiente para aprobar la asignatura[22]), **desde el punto de vista de los requisitos funcionales que debe cumplir el sistema final, ocupa aproximadamente la mitad de las sesiones,** quedando la otra mitad a libre disposición del alumno para la realización de cuantas mejoras del sistema adicionales estime oportunas.

Por tanto, y con objeto de, no sólo poder disponer del mayor número de sesiones posible para la puesta a punto y mejora del sistema, sino también para garantizar una distribución del esfuerzo requerido a lo largo del semestre lo más racional y equilibrada posible, **se recomienda al alumno un seguimiento lo más estricto posible.**

Con idéntico propósito, y en caso de que los alumnos superen los objetivos planteados para una determinada sesión antes de la finalización de la misma, se recomienda

[21] 13, si contamos la semana adicional de turnos libres que contempla el calendario de la asignatura.
[22] Como es obvio, siempre y cuando se superen las diferentes pruebas de evaluación previstas conforme a lo especificado en la guía docente de la asignatura.

igualmente al alumno tratar de abordar los objetivos y tareas ya planteados para la siguiente sesión.

Adicionalmente, tras la consecución de cada hito el alumno deberá hacer **entrega vía Moodle de una copia del código desarrollado** para la correspondiente versión del sistema.

6.1. Sesión 0 (antes de ir al laboratorio B-043)

Aunque al principio le resulte tedioso, recomendamos al alumno que se tome su tiempo para formarse con la documentación de apoyo a la asignatura publicada.

Igualmente, con carácter general y con el fin de aprovechar al máximo las sesiones en el laboratorio, que son un recurso muy limitado, **el alumno debe prestar especial atención al trabajo previo requerido** para cada sesión. En particular, el alumno deberá haber leído las fuentes recomendadas, completado los tutoriales sugeridos, realizado los montajes HW indicados o traerse escrita de casa una primera versión de los programas que se puedan solicitar. En este sentido, y en la medida de lo posible, es fundamental aprovechar las sesiones de laboratorio para la depuración de errores de ejecución, que no de sintaxis, de los sistemas a desarrollar.

6.2. Sobre el lenguaje C

Tal y como pone de manifiesto el reciente estudio "Embedded Markets Study" [13] sobre el mercado de los sistemas empotrados, publicado en 2017 y apoyado en una encuesta realizada a ingenieros de todo el mundo sobre su trabajo y las empresas para las que trabajan, **el lenguaje C es un requisito indispensable para trabajar en el desarrollo de sistemas empotrados** hoy en día y en el corto plazo.

Una de las principales conclusiones de dicho estudio es que, a pesar del avance incontestable de otros lenguajes de programación como C++, el lenguaje C es el lenguaje por excelencia utilizado por las empresas (e instituciones académicas) que trabajan en el ámbito de los sistemas empotrados/embebidos[23]. Adicionalmente, la **alta tasa de reutilización de código**[24] de proyectos previos que existe en este mercado indica a todas luces que el lenguaje C seguirá ocupando dicha posición dominante por mucho tiempo, situación que hace del lenguaje C una **herramienta imprescindible para cualquier ingeniero** del sector de los sistemas empotrados también en el medio/largo plazo.

Esta asignatura no es ajena a dicha realidad, por lo que resulta especialmente recomendable, antes de empezar las sesiones del laboratorio, familiarizarse con dicho lenguaje de programación. Para ello proponemos la realización del **tutorial online** [11] disponible en: http://www.learn-C.org.

6.3. Sesiones 1, 2 y 3: Versión 1.0 del sistema

Durante las tres primeras sesiones se presentarán los conceptos y las herramientas básicas necesarias (familiarización con el laboratorio, su instrumental y el entorno de desarrollo) para el desarrollo del proyecto propuesto. Igualmente, se espera que dichas

[23] Más de la mitad de los proyectos (56%) actualmente en desarrollo a nivel mundial están siendo programados en C. Muy por detrás (22%) le sigue en segunda posición, el lenguaje C++.

[24] El 87% de los proyectos actualmente en desarrollo a nivel mundial están reutilizando código de proyectos previos.

sesiones sean suficientes como para permitir al alumno conseguir una **primera versión simplificada** pero funcional del sistema (versión 1.0).

En particular, esta primera versión estará orientada específicamente a la **reproducción de los efectos de sonido** y servirá igualmente para familiarizarnos con uno de los paradigmas más populares para el desarrollo de programas: las **máquinas de estados**. En este sentido, será necesario incorporar una primera máquina asociada al **subsistema de reproducción** que será la responsable de la correcta reproducción de los efectos. Para ello, el alumno hará uso de la implementación y los ejemplos presentados en [5], prestando especial atención al uso de flags de estado, y seguirá igualmente las directrices indicadas en este mismo documento, en particular, cuanto concierne a los apartados 3.1, 5.1 y 5.2.

No obstante, adoptaremos un par de **simplificaciones** importantes en relación a la reproducción posible a partir de esta primera versión. En primer lugar, nuestro sistema no contará aún con el subsistema HW de control basado en el teclado matricial, por lo que para poder incorporar a la misma los diferentes eventos ligados a dicho subsistema (e.g. el ligado al comienzo de la reproducción del efecto, *FLAG_START_DISPARO*), y recogidos por el diseño propuesto para la máquina de estados (ver Ilustración 17), será necesario **simular dichos eventos** asociándolos a la pulsación de diferentes teclas del teclado del PC.

Por otro lado, al carecer el sistema de los recursos de temporización necesarios para la reproducción automática de los efectos (su implementación está prevista en posteriores sesiones), será necesaria una segunda simplificación: la reproducción del efecto **se producirá de forma manual a partir de la pulsación de la tecla de disparo**. De este modo, cada vez que el usuario accione la tecla de disparo se comenzará la generación de una nueva nota, siendo así necesarias repetidas pulsaciones de dicha tecla para completar la reproducción del efecto elegido (obviamente, cada nota durará el tiempo que tardemos en realizar una nueva pulsación que dé comienzo a la siguiente).

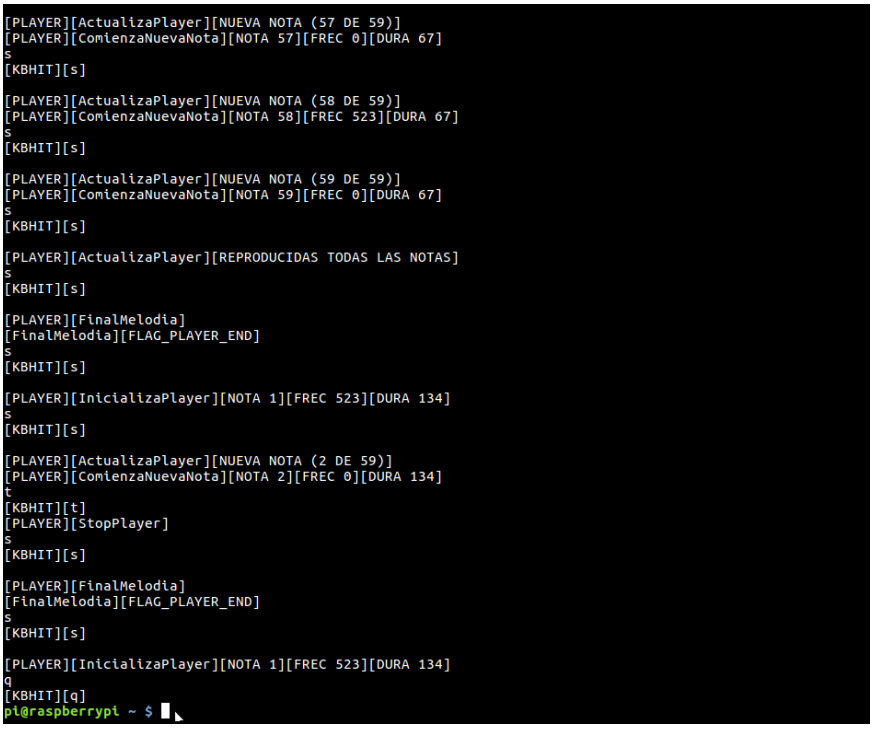

Ilustración 19. Captura de ejemplo de información a escribir por salida estándar vía "printf".

Esta versión deberá cumplir además con las siguientes especificaciones:

- **La visualización tanto de las pulsaciones detectadas como de las notas que componen el efecto se realizará a través de la ventana de terminal** de la plataforma de desarrollo. En particular, se empleará **la función printf** para reflejar en cada momento la información correspondiente a la nota cuya reproducción esté en curso incluyendo su posición, su frecuencia y su duración prevista (aunque esta última sólo tendrá utilidad a efectos informativos en esta versión). En la Ilustración 19 puede observarse una captura de la ventana de terminal del entorno de desarrollo en la que se presenta cierta información a modo de ejemplo.

- Para poder "jugar", inicialmente, el usuario empleará **el teclado convencional** del PC. Para ello, el alumno contará, como punto de partida, con un **programa** (ver anexo 8.5) **que facilitará la captura, y posterior interpretación en términos del juego, de las diferentes pulsaciones de sus teclas**. En particular, el alumno deberá gestionar la ocurrencia de los eventos básicos relacionados específicamente con la pulsación de las teclas de movimiento (i.e. teclas de dirección) o disparo (i.e. tecla 's').

- Para la reproducción de cada nota se empleará uno de los pines GPIO[25] debidamente configurado para **generar una señal rectangular con la frecuencia correspondiente**. La Tabla 3 recoge una posible distribución de los diferentes pines GPIO de salida necesarios para la implementación del proyecto. De igual modo, se facilita también la Tabla 2 para los pines GPIO de entrada. En cualquier caso, será de obligada lectura [3].

Puerto de entrada	13	12	11	10	9	8	7	6	5	4	3	2	1
Conexión	-	-	-	-	-	GPIO-19 LIBRE	GPIO-16 RX_IRQ	GPIO-15 LIBRE	GPIO-09 LIBRE	GPIO-13 FILA4	GPIO-12 FILA3	GPIO-06 FILA2	GPIO-05 FILA1
Puerto de entrada	26	2	24	23	22	21	20	19	18	17	16	15	14
Conexión	-	GND	GND	-	-	-	-	-	-	GPIO-27 LIBRE	GPIO-26 LIBRE	GPIO-21 LIBRE	GPIO-20 LIBRE

Tabla 2. Propuesta de uso de pines GPIO de entrada.

Puerto de salida	13	12	11	10	9	8	7	6	5	4	3	2	1
Conexión	-	-	-	-	-	GPIO-10 LIBRE	GPIO-08 LIBRE	GPIO-07 LIBRE	GPIO-04 LIBRE	GPIO-03 COL4	GPIO-02 COL3	GPIO-01 COL2	GPIO-00 COL1
Puerto de salida	26	25	24	23	22	21	20	19	18	17	16	15	14
Conexión	-	GND	GND	-	-	GPIO-25 LIBRE	GPIO-24 PI_OUT_TX	GPIO-23 TONE	GPIO-22 PWM2	GPIO-18 PWM1	GPIO-17 LIBRE	GPIO-14 LIBRE	GPIO-11 LIBRE

Tabla 3. Propuesta de uso de pines GPIO de salida.

Para poder comprobar la correcta generación de cada una de las notas a través del pin del puerto de salida reservado al efecto, emplearemos el osciloscopio, prestando en este caso especial atención a la frecuencia de las señales visualizadas en el mismo y a su posible desviación con respecto al valor esperado.

El alumno dispone de más información acerca del programa de ejemplo con el que comenzar, así como de una descripción más detallada del modelo de implementación a adoptar, en el apartado correspondiente a ANEXOS incluido en este mismo documento.

[25] Por ejemplo, el pin GPIO-18 correspondiente al pin 17 del conector de salidas digitales de la TL04.

Sesiones 1, 2 y 3) Versión 1.0 del sistema	
Trabajo previo	• Serán de obligada lectura [1], [3] y [5]. • Tutorial online programación en C
Objetivos docentes / de aprendizaje	Adquirir los conocimientos suficientes para poder: • Comprender el funcionamiento del entorno: - Perspectivas 'Debug' y 'Remote System Explorer' • Saber crear y configurar un proyecto desde cero - Saber crear y configurar un nuevo proyecto a partir de otro • Saber compilar el proyecto - Y entender el proceso… • Saber depurar: uso de breakpoints, ejecución paso a paso, visualización de variables, … • Adquirir conocimientos de C necesarios y suficientes para abordar el proyecto, en particular: - Uso de arrays y estructuras de datos • Config. básica pines GPIO para generación señales cuadradas • Ser capaces de reformular el sistema a partir de la primera versión como una máquina de estados identificando para ello: - Los distintos estados del sistema: wait_start, wait_next, wait_end - Las posibles transiciones entre ellos - Las funciones de entrada (o de comprobación de eventos, e.g. FLAG_PLAYER_START), y las funciones de salida (o de actualización del sistema) que gobiernan dichas transiciones
Recursos adicionales	• Programa de ejemplo con rutina de detección de pulsaciones, declaración de todos los objetos básicos y prototipos de las principales funciones a implementar • Programa ejemplo con fsm y proceso que corre rutina de detección de pulsaciones
Resultados para el proyecto	• Familiarización con el entorno y herramientas básicas • 1ª versión del sistema jugable desde terminal y teclado del PC • 1ª FSM de reproducción de efectos de sonido: sucesivas pulsaciones de tecla de disparo avanzan la reproducción

Tabla 4. Resumen del trabajo previo, objetivos, recursos y resultados de las sesiones 1,2 y 3.

6.4. Sesión 4: Versión 2.0 del sistema

Con esta sesión se pretende completar una nueva **versión del sistema que será completamente automática** desde el punto de vista de la reproducción de los efectos de sonido. Para ello, el alumno deberá incorporar un **temporizador** que mida el tiempo de reproducción de las distintas notas (o cualquiera de las soluciones alternativas para la gestión y control del tiempo descritas en 5.1) y que, agotada la duración prevista para las mismas, dispare los eventos necesarios (i.e. *FLAG_NOTA_TIMEOUT* o *FLAG_PLAYER_END*, para pasar a la siguiente nota o para dar por finalizada la reproducción, respectivamente) para que la máquina de estados, asociada al subsistema de reproducción e implementada en el hito anterior, funcione correctamente de modo que el efecto se reproduzca al completo sin intervención manual alguna por

parte del usuario (debería bastar una única pulsación de la tecla de disparo para comenzar y completar la reproducción íntegra del efecto).

Sesión 4) Versión 2.0 del sistema	
Trabajo previo	• Será de obligada lectura [4] • Versión 1.0 del sistema
Objetivos docentes / de aprendizaje	• Manejo de temporizadores • Capacidad para gestionar correctamente el acceso por parte de las rutinas de atención a la interrupción a los recursos compartidos por éstas con el resto de procesos del sistema (exclusión mutua)
Recursos adicionales	-
Resultados para el proyecto	• Segunda versión del sistema que incorpora temporizador para la reproducción automática de efectos de sonido

Tabla 5. Resumen del trabajo previo, objetivos, recursos y resultados de la cuarta sesión.

6.5.Sesión 5: Versión 3.0 del sistema

El objetivo de esta sesión consiste en incorporar al sistema el **teclado matricial** disponible en la placa TL04 (o el joystick alternativo) necesario para controlar el movimiento y disparo de nuestro cañón.

Para ello nos apoyaremos en una **FSM orientada específicamente a la excitación periódica de las columnas** del teclado que hace posible la detección de las pulsaciones de las teclas. En particular, nos apoyaremos en la solución propuesta en [3]. Dicha solución, avanzada en el apartado 4.3.1 del presente enunciado, se apoya en que la acción de una tecla produce un flanco de subida en la fila en la que ésta se encuentra. Esta situación provoca una interrupción en la entrada correspondiente de modo que, al estar cada fila ligada a una interrupción diferente y al haber una única columna activa en cada momento, la atención específica de dicha interrupción hace posible la identificación de la tecla pulsada (i.e. tecla ubicada en la columna excitada y en la fila ligada a la interrupción).

Gracias a la TL04 el **teclado ya está convenientemente conectado a la Raspberry Pi** conforme a la propuesta de pines recogida en la Tabla 1 (columnas conectadas a los 4 primeros pines de salida y filas conectadas a los 4 primeros pines de entrada de la TL04), por lo que, salvo que queramos emplear nuestro propio teclado o un joystick como alternativa de control que justifique la elección de un conjunto de pines distinto para este fin, no será necesario establecer conexión física adicional alguna.

La cuestión que sí será necesaria es la correcta **configuración** de todos los pines GPIO implicados, unos como salidas y otros como entradas. Para estos últimos en particular será además necesaria la definición de las correspondientes **rutinas de atención** a la interrupción ligadas respectivamente a cada interrupción externa.

Una vez incorporados y debidamente configurados dichos recursos, el alumno procederá a verificar su correcto funcionamiento ayudándose para ello, específicamente, de **puntos de parada ubicados al comienzo de ambas subrutinas de atención** a la interrupción. A este respecto deberá prestar especial atención a los **posibles rebotes** (i.e. interrupciones espurias que producen transiciones de un estado

a otro no deseadas) que puedan producirse en las entradas digitales (puede hacerse uso del osciloscopio para su visualización). Dichos rebotes deberán ser **eliminados por SW** de modo que **cada acción sobre cualquiera de las teclas de dirección o disparo vayan acompañados de una única actualización de la posición de la torreta** o de **un único disparo** respectivamente.

Finalmente, la inclusión plena del teclado en nuestro sistema se completará de forma efectiva **asignando a las respectivas rutinas de atención a la interrupción la responsabilidad de activar los flags previstos** para las diferentes acciones.

Sesión 5) Versión 3.0 del sistema	
Trabajo previo	• Serán de obligada lectura [3] y [4] • Versión 2.0 del sistema
Objetivos docentes / de aprendizaje	• Dimensionar e identificar los recursos (puertos GPIO) necesarios • Capacidad para configurar convenientemente las entradas de interrupción dedicadas (R interna de pull-up/pull-down, limitaciones entrenadora, …) • Capacidad para habilitar soluciones SW / HW frente al problema de rebotes presente en las entradas de interrupción • Verificar el correcto funcionamiento del montaje por medio de los recursos disponibles (osciloscopio, entorno de desarrollo,…)
Recursos adicionales	• Programa de ejemplo de uso del teclado matricial basado en FSM
Resultados para el proyecto	• Tercera versión del sistema que incluye el teclado matricial y que, por tanto, permite prescindir del teclado del PC

Tabla 6. Resumen del trabajo previo, objetivos, recursos y resultados de la quinta sesión.

6.6. Sesiones 6 y 7: Versión 4.0 del sistema

Durante la sesión 7 (correspondiente a la semana 7 del calendario académico) tendrá lugar una **revisión intermedia obligatoria por parte de los profesores del estado del sistema en desarrollo**. En este punto, **el alumno debería haber completado con éxito la versión 3.0 del sistema**.

Estas dos sesiones están orientadas a la implementación de la segunda FSM, la de control del juego, y a la incorporación de los servos al sistema para un control real de la torreta. Con ellas queda cubierto íntegramente el conjunto de especificaciones de funcionamiento de carácter obligatorio propuesto para el proyecto.

El alumno deberá **incorporar al sistema la segunda máquina de estados** prevista, responsable en este caso la **gestión directa del juego, incluyendo el control total del cañón** y la capacidad de iniciar **la reproducción de los efectos de sonido**[26]. Para esto último, será necesario establecer los mecanismos de comunicación oportunos (i.e. por medio de flags de estado) que permitan a esta máquina gobernar el estado de la otra.

Para ello, una vez más, el alumno podrá hacer uso de la implementación y ejemplos presentados en [5], prestando especial atención al uso de flags, y seguirá igualmente

[26] Nótese que en caso de optar por la NO implementación del enlace IR la reproducción de efectos de sonido se limita a la correcta reproducción del efecto de disparo.

las directrices indicadas en este mismo documento, en particular, cuanto concierne a los apartados 3.1, 5.1 y 5.2.

En relación a la inclusión de los servos será necesario establecer las correspondientes conexiones físicas entre los servos y la Raspberry Pi empleando para ello las salidas digitales disponibles en los conectores de la placa TL04. En la Tabla 3, presentada anteriormente, puede encontrar una propuesta al respecto.

Igualmente, se hace notar al alumno que los servos necesitan una **tensión de alimentación, en nuestro caso de 5V**, por lo que será necesario emplear también la fuente de alimentación facilitada en el propio laboratorio.

El alumno contará con un **programa de prueba** que permitirá un control básico de los servos y que será de utilidad para verificar y validar el correcto funcionamiento de los mismos a partir de las conexiones HW establecidas.

Desde el punto de vista SW, para el control de los servos nos apoyaremos fundamentalmente en el uso de la librería "**Software PWM Library**" de WiringPi, por lo que se remite al alumno a la oportuna consulta de la documentación disponible en relación a la misma (http://wiringpi.com/reference/software-pwm-library/).

Sesiones 6 y 7) Versión 4.0 del sistema	
Trabajo previo	• Será de obligada lectura [4] • Versión 2.0 del sistema
Objetivos docentes / de aprendizaje	• Consolidar los conceptos básicos relacionados con FSMs
Recursos adicionales	• Torreta con microservos • Programa de prueba que permite control básico de ambos servos • Documentación "Software PWM Library" de WiringPi
Resultados para el proyecto	• Cuarta versión del sistema, completa desde el punto de vista de especificaciones básicas obligatorias, que incorpora los servos para el movimiento de la torreta y 2ª FSM para control del juego

Tabla 7. Resumen del trabajo previo, objetivos, recursos y resultados de las sesiones 6 y 7.

6.7. Resto de sesiones: Implementación de mejoras (OPCIONAL) y puesta a punto del sistema final

Durante estas últimas sesiones se espera que el alumno ponga a punto la versión final de su sistema incorporando para ello diversas mejoras, de su elección, que le permitan alcanzar la máxima nota (recuerde que la implementación del proyecto propuesto cubriendo sólo las **especificaciones mínimas obligatorias** será valorada **con un máximo de 7 puntos; la realización de mejoras permitirá incrementar la puntuación** hasta alcanzar la máxima nota, 10 puntos)**.**

6.7.1. Versión 5.0: sistema completo con enlace IR

Mejora consistente en la implementación de la versión 5.0 del sistema. Está valorada con **hasta 2 puntos**. Esta versión incorporará finalmente **el enlace infrarrojo**. Con éste, el alumno conseguirá un prototipo 100% funcional que cumplirá íntegramente el conjunto de especificaciones descrito en la sección 3.1 del presente documento.

Para ello, el alumno **completará el diseño** propuesto para los **subsistemas HW de disparo y detección de impacto** (ver secciones 4.1 y 4.2), **los implementará y verificará** su correcto funcionamiento. Asimismo, se deberá completar la configuración e integración de los recursos SW necesarios para cumplir con las especificaciones de funcionamiento definidas para cada extremo del enlace: un **temporizador** vinculado al emisor para medir la duración de cada disparo (i.e. tiempo máximo durante el cual el enlace infrarrojo puede permanecer activo) y una **entrada de interrupción** vinculada al receptor para poder detectar los posibles impactos. Para este último, además, será necesario **incorporar la reproducción del efecto** de sonido específico para tal situación.

Como planificación tentativa se propone dedicar una primera sesión a la implementación del emisor y una segunda al receptor y puesta a punto del sistema al completo.

En cuanto concierne al emisor deberemos realizar las conexiones HW oportunas. En particular, deberemos identificar un pin GPIO disponible en la TL04 que podamos emplear como salida para "atacar" así el subsistema HW de disparo (ver propuesta recogida en Tabla 3). Una vez establecida dicha conexión, deberemos comprobar la adecuada polarización del led infrarrojo al accionar la tecla de disparo (acción que debería modificar el nivel lógico presente en la mencionada salida) y cómo éste efectivamente se activa visualizando su encendido **a través de la cámara del móvil**[27]. Igualmente, deberemos comprobar también su oportuno apagado al transcurrir el tiempo o duración límite prevista para cada disparo.

Ilustración 20. Detalle del encendido de un led infrarrojo visto a través de la cámara de un teléfono móvil.

Desde el punto de vista del receptor, será igualmente importante establecer las conexiones HW necesarias. En este caso será necesario identificar un pin GPIO disponible en la TL04 que podamos emplear como entrada para "leer" la salida del subsistema HW de detección de disparo (ver propuesta recogida en Tabla 2).

En cuanto a su correcto funcionamiento deberemos comprobar, **haciendo uso del osciloscopio** disponible**,** como el nivel lógico presente a la salida del HW de detección varía en consonancia, nivel alto o bajo, con el establecimiento y posterior caída del enlace. Nótese que es probable que necesite realizar un ajuste adecuado del potenciómetro incluido en la solución propuesta (ver Ilustración 5).

[27] Éste es un truco doméstico habitual para la comprobación de las baterías de un control remoto.

A continuación, volverá a hacerse uso del osciloscopio prestando en esta ocasión especial atención a los **posibles rebotes** (i.e. interrupciones espurias que producen transiciones de un estado a otro no deseadas) que puedan producirse en la entrada digital correspondiente. Estos rebotes deberán ser **eliminados por SW** de modo que, **cada activación del enlace vaya acompañada de una única interrupción**.

Finalmente, y para un mejor y más sencillo análisis de los eventuales problemas que puedan surgir durante la implementación de la presente mejora, nótese que es posible **reemplazar, eventualmente, todo el HW del enlace IR por un simple cable que conecte directamente el pin GPIO de salida de nuestro emisor con el pin GPIO de entrada de nuestro receptor**. De este modo podremos determinar si los posibles problemas que afectan a nuestro sistema son de naturaleza HW o SW (o sencillamente nos facilitará progresar en la implementación de la mejora aún cuando, por cualquier motivo, no se disponga de los componentes HW requeridos para ello).

Sesiones 8 y 9) Versión 5.0 del sistema (OPCIONAL)	
Trabajo previo	• Serán de obligada lectura [3] y [4] • Elementos de montaje HW necesarios para las conexiones a los pines GPIO (cables tipo Arduino, tiras de pines, …) • Versión 4.0 del sistema
Objetivos docentes / de aprendizaje	• Adquirir los conocimientos y capacidades básicas para controlar cualquier dispositivo externo conectado a la Raspberry Pi a través de las entradas y salidas digitales disponibles en el entrenador
Recursos adicionales	• Propuesta de diseño para ambos subsistemas • Led infrarrojo ya instalado en la parte superior de la torreta
Resultados para el proyecto	• Quinta y definitiva versión del sistema que incluye el enlace IR y que (a falta de otras mejoras) cubre al 100% las especificaciones propuestas: tanto las obligatorias como las opcionales

Tabla 8. Resumen del trabajo previo, objetivos, recursos y resultados de las sesiones 8-9.

6.7.2. Algunas mejoras con puntuación predefinida

Uso de Raspberry Pi propia	
Descripción	Implementación del sistema conforme a las especificaciones descritas en el presente documento pero haciendo uso de una Raspi convencional desprovista de las protecciones propias de la plataforma ENTCF2004
Requisitos de implementación	Para aspirar a la máxima puntuación posible, esta mejora implica OBLIGATORIAMENTE demostrar, a la coordinación de la asignatura o a alguno de los profesores responsables del turno, el correcto funcionamiento en dicha Raspi de alguna de las distintas versiones definidas para el sistema ANTES de que finalice la semana 8 del curso. IMPORTANTE: No se aceptará como mejora cualquier uso de Raspis distintas de las del laboratorio del que no se tenga constancia, en los términos anteriormente expresados, antes de dicha fecha.

Puntuación	+1 punto

Acondicionamiento de señal	
Descripción	Aunque para reproducir los efectos de sonido es suficiente con enviar la señal cuadrada generada directamente a los auriculares, en este caso se propone implementar las etapas de filtrado y amplificación descritas en la sección 4.4 a modo de acondicionamiento de la señal.
Requisitos de implementación	Los recogidos en los apartados 4.4.1 y 4.4.2
Puntuación	+1 punto

Tabla 9. Mejoras con puntuación predefinida.

6.7.3. Otras mejoras

Las sugerencias recogidas en este apartado tienen un carácter meramente orientativo. La valoración de la dificultad (de diseño y de implementación) que se hace de cada una de ellas es igualmente orientativa. No obstante, y con carácter general, se valorarán especialmente aquellas mejoras que conlleven algún tipo de esfuerzo por parte del alumno a un mayor número de niveles. En este sentido, se tendrán en cuenta los 3 niveles de la arquitectura típica de cualquier sistema embebido: nivel software o de aplicación (nivel superior en verde de la pila presentada en la Ilustración 21), nivel hardware (nivel inferior en rojo), y nivel de software de sistema o nivel dedicado a la integración hardware-software (nivel intermedio en naranja), concretada a través de la figura de "driver" (i.e. controlador) SW para el control de un dispositivo HW.

De este modo, el modelo de valoración adoptado, resumido en la Ilustración 21, reflejará respectivamente el impacto de la mejora sobre:

- sólo el primero de los niveles: BAJA valoración, por tratarse de una mejora que sólo requiere modificar código;
- sólo los dos primeros niveles: valoración MEDIA, por tratarse de una mejora que, además de modificar el código, requiera del uso adicional de algún nuevo recurso de la Raspberry Pi (como por ejemplo otro temporizador) o de una configuración alternativa de los mismos (ya sean éstos HW o SW);
- todos los niveles anteriormente mencionados: valoración ALTA, por afectar a todos los niveles de diseño del sistema, aspecto para el que será esencial la incorporación de nuevo HW al sistema.

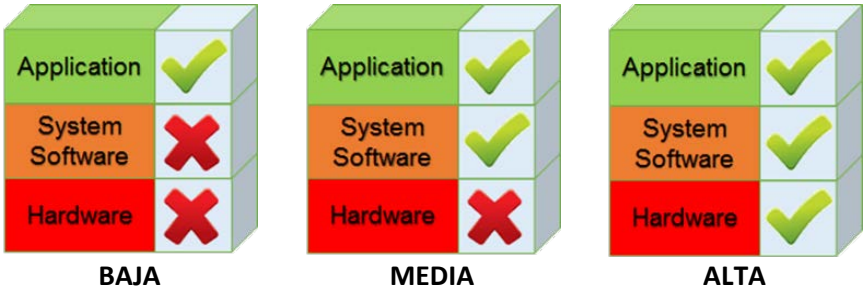

Ilustración 21. Niveles de valoración de las posibles mejoras a implementar.

Respectivamente, y con carácter igualmente orientativo, la calificación de otras mejoras con una valoración BAJA, MEDIA o ALTA, razonablemente, podría traducirse a efectos numéricos de la siguiente manera:

- BAJA: entre 0,3 y 0,5,
- MEDIA: entre 0,5 y 1,
- ALTA: entre 1 y 1,5.

En cualquier caso, todas las mejoras propuestas podrán llevarse a cabo de diversas maneras (i.e. **complejidad**) y con distintos grados de perfección (i.e. **calidad**) por lo que la calificación final de las mismas dependerá necesariamente de ambos aspectos.

Del mismo modo, la **originalidad** de la mejora, en particular cuando respondan a alguna iniciativa novedosa por parte del alumno, será igualmente otro aspecto o factor importante a tener en cuenta a efectos de su calificación.

En resumen, hacer una mejora no implica necesariamente conseguir toda la puntuación prevista. No obstante, esto aplica en ambos sentidos, es decir, también podría suceder que el alumno obtuviese incluso más puntos de los previstos si el montaje o la solución propuesta así lo merecieran.

La valoración de toda mejora, independientemente de que esté o no recogida en el Enunciado, se regirá por estos criterios generales de valoración.

En caso de duda, se recomienda contactar con el coordinador docente (Fernando Fernández-Martínez, fernando.fernandezm@upm.es) o con cualquiera de los profesores de la asignatura.

Nuevas características añadidas al sistema	
Descripción	• Sistema de menús que permita al usuario seleccionar o configurar, entre otros: - la duración del disparo, - la velocidad de movimiento de los servos, - Otros… • Perfil de usuario que gestione las preferencias o estadísticas de uso de distintos usuarios • Otros…
Requisitos de implementación	-
Puntuación	BAJA

Cambios sustanciales del sistema	
• Descripción	• Reproducción de melodías almacenadas en la tarjeta SD • Empleo de librerías de control GPIO alternativas a WiringPi (e.g. PiGPIO, http://abyz.me.uk/rpi/pigpio/) • Incorporación de pulsadores, interruptores o leds con alguna funcionalidad específica (e.g. visualización del

	disparo/señalización del impacto/impactos restantes y demás información de juego, …) • Incorporación de motores adicionales (e.g. vibración para impactos) • Construcción de un prototipo del HW en PCB mediante el uso de un programa comercial de diseño • Incorporación de múltiples blancos • Reproducción de melodías de juego de fondo • Otros…
Requisitos de implementación	-
Puntuación	MEDIA

Integración con otros dispositivos o sistemas	
Descripción	• Uso de otros microcontroladores (e.g. Arduino) / FPGA con algún fin o propósito específico • Construcción de torreta propia • Implementación del tanque al completo integrando el cañón en un vehículo para su posible desplazamiento y consiguiente control del mismo. • Incorporación de cualquier dispositivo para el radiocontrol del cañón. • Implementación íntegra del sistema en Python • Modulación IR para juego multijugador • Programación (mediante comunicación serie) de un kit de reproducción MP3 • Empleo de otros displays (como alternativa a la ventana de terminal para la visualización, para la representación de caras que acerquen el juguete a la idea de un avatar con el cuál poder interaccionar, marcador electrónico para juego multijugador…): - LCD - Matriz de leds RGB - Otros… • Empleo de otros dispositivos de control: - móvil, - acelerómetro 3D, - sensores de ultrasonidos, - módulo BLE para comunicación inalámbrica, - control remoto convencional (e.g. TV) - Otros… • Esquemas circuitales alternativos • Integración con otros sistemas / dispositivos (interfaz de visualización alternativa, carga de nuevas melodías/efectos, configuración del sistema, …): - Web o app móvil, - Nube, - Otros…

Requisitos de implementación	-
Puntuación	ALTA

Tabla 10. Otras mejoras.

Para conocer detalles adicionales sobre la implementación o la valoración de éstas u otras mejoras, no dude en ponerse en contacto con el coordinador o con cualquiera de los profesores de la asignatura.

6.8. Fechas importantes

A continuación se facilita un calendario en el que se han destacado algunas **fechas relevantes** para el transcurso de la asignatura. Revíselas concienzudamente pues algunas de ellas afectan específicamente a los procedimientos de evaluación de la misma.

En la Ilustración se ha añadido a la izquierda del calendario la planificación por sesiones recomendada y su correspondencia con las diferentes semanas del curso. Nótese que, debido a la acumulación de festivos a lo largo del curso, algunos grupos pueden alcanzar ciertas sesiones 1 semana antes o después que el resto.

1	FEBRERO	S1	28	29	30	31	1	2	3
2		S2	4	5	6	7	8	9	10
3	V1.0	S3	11	12	13	14	15	16	17
4	V2.0	S4	18	19	20	21	22	23	24
5	MARZO	S5	25	26	27	28	1	2	3
6	V3.0	S6	4	5	6	7	8	9	10
7	V4.0	S7	11	12	13	14	15	16	17
8		S8*	18	19	20	21	22	23	24
9	V5.0	S9*	25	26	27	28	29	30	31
10	ABRIL	S10	1	2	3	4	5	6	7
11		S11	8	9	10	11	12	13	14
12			15	16	17	18	19	20	21
13		S12	22	23	24	25	26	27	28
14	MAYO		29	30	1	2	3	4	5
15		LIBRE	6	7	8	9	10	11	12
16			13	14	15	16	17	18	19
17			20	21	22	23	24	25	26
18	JUNIO		27	28	29	30	31	1	2
19			3	4	5	6	7	8	9

Ilustración 22. Calendario de fechas importantes para la asignatura.

Resumen de fechas importantes:

- 24-27 ENE: Selección de grupo
- 29 ENE-8h: Presentación SDGII (Salón de Actos Edificio A)
- 29 ENE-4 FEB: 1ª sesión
    - 28 ENE NO lectivo (Santo Tomás de Aquino)

  ~~11 MAR-12h: Prueba – HITO INTERMEDIO~~
    - **ACTUALIZACIÓN: cada pareja realizará la prueba en laboratorio correspondiente al HITO INTERMEDIO al comienzo de la sesión 7 en su turno asignado**
- 12-18 MAR: Revisión requisitos – HITO INTERMEDIO
    - 12 MAR: Revisión – HITO INTERMEDIO MT
    - 13 MAR: Revisión – HITO INTERMEDIO XT
    - 14 MAR: Revisión – HITO INTERMEDIO JT
    - 15 MAR: Revisión – HITO INTERMEDIO VC
    - 18 MAR: Revisión – HITO INTERMEDIO LT
- 30 ABR: TURNO LIBRE
- 7-10 MAY: TURNO LIBRES
- 13-17 MAY: REVISIÓN Y PRUEBA INDIVIDUAL FINAL
    - 10 MAY: TODOS - Fecha límite entrega código y memoria definitivos
    - 13 MAY: FINAL LT
    - 14 MAY: FINAL XT (CLASE DE MIÉRCOLES)
    - 16 MAY: FINAL JT
    - 17 MAY: FINAL VC
    - 21 MAY: FINAL MT

Para cualquier duda en relación a los procedimientos y fechas indicadas, no dude en ponerse en contacto con el coordinador docente o administrativo de la asignatura (Juan José Gómez Valverde, jjgvalverde@die.upm.es).

6.9. Entregas electrónicas del código previstas

Las diferentes versiones a implementar del sistema exigirán la entrega vía Moodle del SW desarrollado. Concretamente, los alumnos deberán realizar una entrega del mismo al final de la última sesión dedicada a cada versión (e.g. primera entrega al finalizar la sesión 3).

En el calendario reflejado en la Ilustración 22 se ha añadido una columna adicional a la izquierda (e.g. “V1.0”) que hace referencia a la versión del sistema prevista a la finalización de cada semana y que indica, por tanto, la obligatoriedad de realizar una entrega del código desarrollado hasta ese momento independientemente de que cumpla o no los requisitos previstos.

En resumen, los alumnos **deberán realizar un total de 5 entregas**: 4 correspondientes a las diferentes versiones 1.0-4.0 del sistema, conforme a las especificaciones básicas y obligatorias, y una última entrega final con las posibles mejoras que se hayan podido añadir al mismo. Nótese que la V5.0, a pesar de aparecer indicada en el calendario, es opcional por lo que no será obligatoria su entrega.

7. Material complementario

[1] Introducción al entorno de desarrollo en C para Raspberry Pi

[2] Prácticas de lenguaje C para Raspberry Pi

[3] Iniciación al Manejo de las Entradas/Salidas del BCM 2835

[4] Manejo de temporizadores, interrupciones y procesos con la Raspberry Pi

[5] Introducción a las máquinas de estados en C para Raspberry Pi

[6] Manejo de DAC/ADC vía SPI.

[7] Manejo de Display LCD con Raspberry Pi B+

[8] MFRC522 Standard performance MIFARE and NTAG frontend
https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf

[9] Circuitos Electrónicos (CELT). Descripción del proyecto. Curso 2018-2019.
https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/882276/mod_resource/content/22/Enunciado_2018_19_vMAS_v11.pdf

[10] Hojas de características del LM386
http://www.ti.com/lit/ds/symlink/lm386.pdf

[11] Tutorial de C online e interactivo
http://www.learn-C.org

[12] Tutorial de C online
https://www.cprogramming.com/tutorial/c-tutorial.html

[13] 2017 Embedded Markets Study, Integrating IoT and Advanced Technology Designs, Application Development & Processing Environments. April 2017. EE|Times.
https://m.eet.com/media/1246048/2017-embedded-market-study.pdf

8. ANEXOS

8.1. Tabla de pines de la placa auxiliar TL04

ENTRADAS				SALIDAS			
Placa TL-04 conexión entradas entrenador		Número del puerto BCM sólo utilizable como entrada	ENT2004CF LEDS ROJOS	ENT2004CF LEDS VERDES	Número del puerto BCM sólo utilizable como salida	Placa TL-04 conexión salidas entrenador	
Pin							Pin
1	Teclado Fila 1	**5**	LRE0	LVS0	**0**	Teclado Columna 1	1
2	Teclado Fila 2	**6**	LRE1	LVS1	**1**	Teclado Columna 2	2
3	Teclado Fila 3	**12**	LRE2	LVS2	**2**	Teclado Columna 3	3
4	Teclado Fila 4	**13**	LRE3	LVS3	**3**	Teclado Columna 4	4
5	Libre	**09** (SPI_MISO)	LRE4	LVS4	**04** (GPCLK0)	Libre	5
6	Libre	**15** (UART_RXD)	LRE5	LVS5	**07** (SPI_CS1)	Libre	6
7	Libre	**16**	LRE6	LVS6	**08** (SPI_CS0)	LCD_RS	7
8	Libre	**19**	LRE7	LVS7	**10** (SPI_MOSI)	LCD_Enable	8
9	NC	NC	-	-	NC	NC	9
10	ADC/DAC SPI_CLK	ADC/DAC SPI_CLK	-	-	ADC SSTRB	ADC SSTRB	10
11	ADC/DAC SPI_Din	ADC/DAC SPI_Din	-	-	ADC/DAC SPI_Dout	ADC/DAC SPI_Dout	11
12	SPI_CS_ADC	SPI_CS_ADC	-	-	NC	NC	12
13	SPI_CS_DAC	SPI_CS_DAC	-	-	NC	NC	13
14	Libre	**20**	LRE8	LVS8	**11** (SPI_CLK)	LCD_Bit 0	14
15	Libre	**21**	LRE9	LVS9	**14** (UART_TXD)	LCD_Bit1	15
16	Libre	**26**	LRE10	LVS10	**17**	LCD_Bit 2	16
17	Libre	**27**	LRE11	LVS11	**18**	LCD_Bit 3	17
18	NC	NC	LRE12	LVS12	**22**	LCD_Bit 4	18
19	NC	NC	LRE13	LVS13	**23**	LCD_Bit 5	19
20	NC	NC	LRE14	LVS14	**24**	LCD_Bit 6	20
21	NC	NC	LRE15	LVS15	**25**	LCD_Bit 7	21
22	NC	NC	-	-	NC	NC	22
23	NC	NC	-	-	NC	NC	23
24	GND_Exterior	GND_Exterior	-	-	GND-Exterior	GND_Exterior	24
25	GND_Exterior	GND_Exterior	-	-	GND-Exterior	GND_Exterior	25
26	NC		-	-		NC	26

8.2.Sobre el uso de flags y máscaras

En nuestro proyecto utilizaremos una única variable global, la variable flags, para indicar la ocurrencia de todos los eventos que afectan al funcionamiento del sistema (e.g. si se ha producido la inserción de una tarjeta o, como veremos en los próximos ejemplos, si se ha producido la pulsación de una determinada tecla o no).

Si tenemos en cuenta que basta un bit para indicar la ocurrencia de un determinado evento, dado que la variable flags es de tipo int, disponemos de 32 bits para gestionar la ocurrencia de hasta un total de 32 eventos distintos (más que suficiente para cubrir nuestras necesidades en el proyecto).

En este sentido, las máscaras juegan un papel fundamental toda vez que nos permiten referirnos a cada uno de los flags (i.e. bits) por separado por medio de las funciones binarias OR y AND (| y &).

En particular, un flag determinado puede activarse con la función binaria OR de nuestra variable flags con una constante (i.e. máscara) que tenga todos los valores a 0 salvo aquél que se desea activar. Por ejemplo:

```
#define FLAG_TECLA_OFF 0x02

flags|=FLAG_TECLA_OFF; // Pone a 1 únicamente el segundo bit de
los 32 disponibles en la variable
```

Igualmente, un flag puede limpiarse o desactivarse con la función binaria AND de nuestra variable flags con una máscara que tenga todos los valores a 1 salvo aquél que se desea desactivar. Para ello podemos hacer uso de la función binaria NOT (~) para obtener la versión negada de la máscara del flag. Por ejemplo:

```
flags&= ~FLAG_TECLA_OFF; // Pone a 0 únicamente el segundo bit de
los 32 disponibles en la variable
```

FinaImente, también podemos consultar el valor de un flag con la función binaria AND de nuestra variable flags con la misma máscara que empleamos para su activación (i.e. constante que tenga todos los valores a 0 salvo aquél que se desea consultar). Por ejemplo:

```
result= (flags& FLAG_TECLA_OFF); // Devuelve el valor, 1 ó 0,
correspondiente al flag en cuestión
```

Para que estos 3 procedimientos funcionen correctamente es fundamental que las máscaras sólo afecten a un único bit. A continuación se muestra a modo de ejemplo la correcta definición de máscaras para los 6 primeros bits de la variable flags:

```
#define MASCARA_FLAG_BIT1 0x01
#define MASCARA_FLAG_BIT2 0x02
#define MASCARA_FLAG_BIT3 0x04
#define MASCARA_FLAG_BIT4 0x08
#define MASCARA_FLAG_BIT5 0x10
#define MASCARA_FLAG_BIT6 0x20
...
```

Ejemplos de definición incorrecta de tales máscaras, fundamentalmente por no cumplir con el requisito de afectar a un único bit, podrían ser los siguientes:

```
#define MASCARA_FLAG_BIT3 0x03
#define MASCARA_FLAG_BIT4 0x04
#define MASCARA_FLAG_BIT5 0x05
```

```
#define MASCARA_FLAG_BIT6 0x06
...
```

8.3. Creación de clases y objetos en C

C, a diferencia de Java, no es un lenguaje orientado a objetos, y por tanto no existen clases ni objetos. Como alternativa aparecen unas estructuras de datos definibles por el programador denominadas “struct” que permiten aglutinar diferentes tipos de datos de modo similar a las clases de Java; no obstante, no permiten definir métodos propios de ese tipo.

Su sintaxis es la que sigue:

```
typedef struct { // declaración de los elementos que componen la
estructura.
   int altura;
   int anchura;
}Tarmario;

void main (void) {
   int obtenido;
   Tarmario ropero; // crea un "objeto" de tipo armario llamado
ropero
   ropero.altura = 2; // inicializa sus variables.
   ropero.anchura= 3;
   obtenido = ropero.altura; // pasa el valor de altura del
ropero a obtenido
}
```

También se puede utilizar un array de struct o utilizar struct dentro de otra struct:

```
#define MAX_LONG_NOMBRE 100
#define NUM_BASES_DATOS 2

typedef struct{
   char nombre[MAX_LONG_NOMBRE];
} Tbasedatos;

typedef struct {
   Tbasedatos[NUM_BASES_DATOS] tipos; // estructuras del tipo
TbaseDatos
}Tnombres;

Tnombres franceses; // Creación estructura

strcpy(franceses.tipos[0].nombre, "Jaques"); // Inicialización
de la estructura
strcpy(franceses.tipos[1].nombre, "François");

while (franceses.tipos[1].nombre != 0) {
   printf ("%s", franceses.tipos[1].nombre++); // accede a una
posición.
}
```

En este caso, Tnombres está formado por una estructura de tipo Tbasedatos que es un puntero de tipo char.

Al inicializar se han dado valores a los diferentes Tbasedatos con la sintaxis conocida:

```
NombreEstructura.ComponenteAInicializar
```

Al ser dos estructuras:

```
NombreEstructura1.NombreEstructura2.ComponenteAInicializar
```

Finalmente se ha impreso el parámetro nombre de una TBASEDATOS.

8.4. Efectos de sonido

En la Ilustración 23 se muestran las teclas de un piano correspondientes a una de las escalas. En esta ilustración podemos observar que se dispone de un total de 12 teclas. Cada una de estas teclas es capaz de generar una señal periódica de una frecuencia determinada. Generalmente podemos pensar que la forma de la señal generada es sinusoidal, pero en la realidad esta forma depende enormemente del instrumento y en muchos casos se parece más a una señal triangular o en diente de sierra que a una sinusoidal.

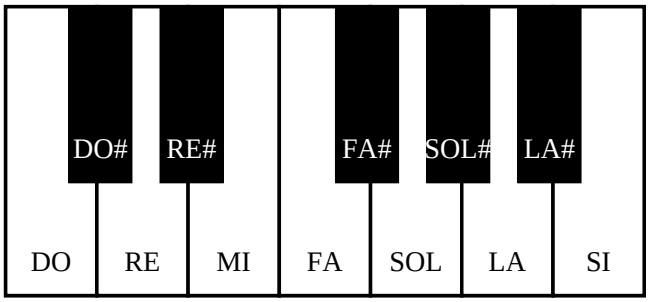

Ilustración 23. Teclas de una octava de un piano.

En cualquier instrumento se dispone de varias escalas. A medida que subimos de escala las frecuencias de las notas se multiplican por 2: la nota DO de una escala genera una señal de frecuencia el doble que la misma nota DO de una escala anterior. Las escalas se identifican según la frecuencia de la nota LA. La escala de referencia es aquella para la que la nota LA genera una frecuencia de 440Hz. En escalas superiores la nota LA generará frecuencias de 880Hz, 1760Hz y 3520Hz respectivamente. En la figura siguiente se muestran las frecuencias (redondeadas a números enteros) de las notas correspondientes a las 4 escalas con las que vamos a trabajar en este proyecto.

OCTAVA	DO	DO#	RE	RE#	MI	FA	FA#	SOL	SOL#	LA	LA#	SI
4	262	277	294	311	330	349	370	392	415	**440**	466	494
5	523	554	587	622	659	699	740	784	831	**880**	932	988
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	**1760**	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	**3520**	3729	3951

Tabla 11. Frecuencias de las notas musicales para las escalas 4,5,6 y 7.

```
int frecuenciasDisparo[16] = {2500, 2400, 2300, 2200, 2100, 2000,
1900, 1800, 1700, 1600, 1500, 1400, 1300, 1200, 1100, 1000};

int tiemposDisparo[16] = {75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75,
75, 75, 75, 75, 75};

int frecuenciasImpacto[32] = {97, 109, 79, 121, 80, 127, 123, 75, 119,
96, 71, 101, 98, 113, 92, 70, 114, 75, 86, 103, 126, 118, 128, 77, 114,
119, 72};

int tiemposImpacto[32] = {10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
10, 10, 10, 10};
```

8.5. Proyecto inicial: piTankGo_1

Con objeto de facilitar el desarrollo del sistema se facilitará al alumno un proyecto y sus correspondientes fuentes que servirán de punto de partida para la implementación de la primera versión del mismo. Este proyecto proporcionará, entre otras, **la definición de los objetos SW básicos** (i.e. `typedef struct{…} tipo_objeto;`) necesarios para su implementación (ver Ilustración 24), incluyendo:

- Objetos de tipo **TipoEfecto** y **TipoPlayer**, objetos que contendrán la información característica de dichos elementos (un objeto de tipo **TipoPlayer** será capaz de procesar otro de tipo **TipoEfecto** y reproducir así el correspondiente efecto),
- Objeto de tipo **TipoTorreta,** básicamente una combinación de parámetros que definen el movimiento de la torreta y su correspondiente posición.
- y finalmente el propio objeto sistema de tipo **TipoSistema,** objeto que define en todo momento las características y los elementos de nuestro juguete y que garantiza la integridad del mismo conforme al conjunto de reglas o restricciones básicas definidas para éste. Este tipo de objeto se define fundamentalmente como la combinación de un reproductor (objeto **TipoPlayer**), una torreta (objeto de tipo **TipoTorreta**) y el estado general del mismo.

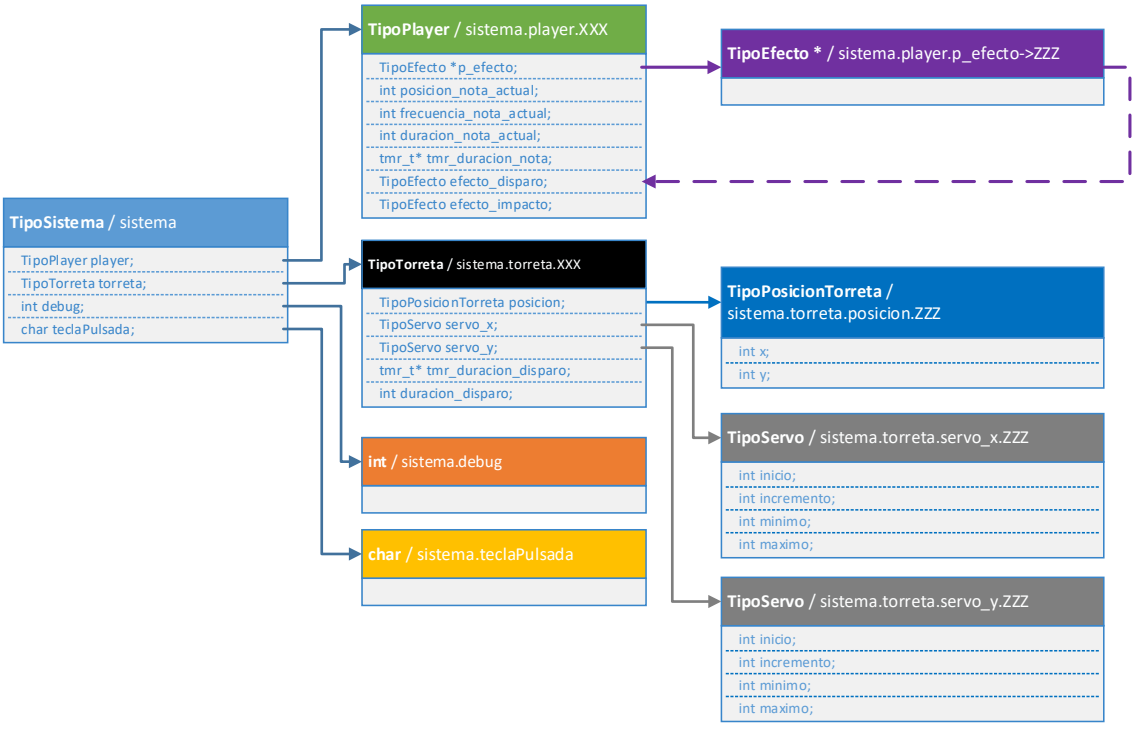

Ilustración 24. Modelo de clases y objetos propuesto.

Igualmente, se proporcionan los **prototipos** (i.e. declaración de una función, disponible en el correspondiente fichero de cabecera) **para la mayoría de las funciones básicas** necesarias para la implementación del sistema, incluyendo:

- **InicializaEfecto**: función encargada de la inicialización de toda variable o estructura de datos específicamente ligada a la reproducción de efectos.

- **ConfiguraSistema**: método encargado de llevar a cabo la oportuna configuración del sistema incluyendo, en particular, la de aquellos elementos y recursos HW que resulten necesarios para el correcto funcionamiento del sistema.
- **InicializaSistema**: método encargado de llevar a cabo la oportuna inicialización del sistema incluyendo, en particular, la de aquellos objetos SW que resultan necesarios para el correcto funcionamiento del sistema.

Será tarea del alumno completar la definición de dichas funciones. Nótese que será igualmente responsabilidad del alumno completar la definición de aquellas otras funciones destinadas a la reproducción de los efectos y a la visualización del estado del sistema a través de la ventana de terminal o consola de la plataforma.

Finalmente, se facilita también tanto **una definición parcial de la máquina de estados responsable de la reproducción de efectos de sonido** como la provisión de un **thread específico para la detección de pulsaciones ligadas al teclado del PC**. Dicho thread llama periódicamente la función ***kbhit*** para detectar si se ha producido la pulsación de alguna de las teclas del teclado, en cuyo caso se llama a la función ***kbread*** para recuperar la tecla pulsada (ambas funciones se facilitan íntegramente al alumno). Posteriormente se lleva a cabo la interpretación de la pulsación detectada.

El alumno deberá completar el código requerido para la adecuada interpretación de todas las teclas necesarias y para el correcto funcionamiento de la citada máquina de estados, en especial las funciones de entrada y salida ligadas a la misma.