
**Facultad de Ciencias**

# SOPORTE Y USO DE LIBRERÍAS COMPATIBLES CON ARDUINO EN TARJETAS STM32 Y MARTE OS

## (SUPPORT AND USE OF ARDUINO COMPATIBLE LIBRARIES ON STM32 CARDS AND MARTE OS)

Trabajo de Fin de Grado

para acceder al

**Grado en Ingeniería Informática**

Autor: Alejandro Fernández Gutiérrez

Director: Mario Aldea Rivas

Co-Director: Héctor Pérez Tijero

Santander, noviembre 2021

# Resumen

El presente Trabajo de Fin de Grado realiza un estudio de las librerías compatibles con Arduino para utilizarlas en el microcontrolador STM32F769I y en MaRTE OS. Se centra en desarrollar un marco de trabajo en el que se puedan utilizar dispositivos hardware en MaRTE OS utilizando el soporte que proporciona Arduino sin tener que preocuparse de desarrollar drivers de bajo nivel.

Inicialmente el proyecto arrancó con la adaptación de la librería stm32duino a nuestro entorno, lo cual nos proporcionaría una base sobre la que utilizar dispositivos compatibles con Arduino en MaRTE OS, con sus correspondientes librerías nativas sin tener que desarrollarlas por nuestra cuenta. Una vez adaptados varios dispositivos y tras la detección de un conflicto con un elemento entre Arduino y MaRTE OS para portar ciertas librerías de dispositivos mas complejas, se opta por desarrollar unas nuevas para proporcionar una funcionalidad lo mas parecida a la que nos daba la original.

Como parte final del trabajo se han desarrollado demostradores para los dispositivos que se han adaptado anteriormente. También se han construido dos aplicaciones mas complejas que utilizan múltiples dispositivos a la vez en el microcontrolador para controlar un coche, poniendo en practica los beneficios de poder utilizar estos en un sistema de tiempo real.

**Palabras clave**: microcontrolador STM32, MaRTE OS, controladores de dispositivos, Arduino API.

# Abstract

The present Final Bachelor's Dissertation is based on a study concerning compatible Arduino libraries in order to use it in STM32F769I microcontroller and MaRTE OS. It focuses on a framework develop in which we can use hardware devices on MaRTE OS using Arduino support without having to worry about build low-level drivers.

Initially, the proyect began with the stm32duino library adaptation to our enviroment. This provide us an initial point where we can use Arduino compatible devices in MaRTE OS with their own native libraries without the need of develop it. Once several devices have been adapted a conflict between Arduino and MaRTE Os element have been discovered to port it in certain complex devices. So, it has been chosen to develop new libraries to this devices in order to give a similar funcionality like the originals.

Finally, some test application has been developed for devices that were adapted previously. Two complex applications have also been built. Each one use multiple devices at the same time in the microcontroller, controlling a toy car as a target getting some benefits using real time tools.

**Palabras clave**: STM32 microcontroller, MaRTE OS, device drivers, Arduino API.

# Índice

# Índice de figuras

# Índice de Listados

# 1. Introducción y objetivos.

## 1.1. Motivación

Durante los últimos años el uso de los microcontroladores se ha extendido enormemente, y no solo para aplicaciones industriales sino también en ámbitos personales o educativos. Esto se produce gracias a la disminución del coste de estos dispositivos junto con la aparición de tecnologías como Arduino que facilitan tanto el uso como el aprendizaje de estos entornos.

A lo largo del Grado en Ingeniería Informática se han utilizado microcontroladores para el aprendizaje en los laboratorios de algunas asignaturas, estos venían en computadores empotrados los cuales no están al alcance del alumnado y la incorporación de dispositivos y sensores a este entorno son complicadas. En este Trabajo de Fin de Grado se desarrolla un proyecto que intenta facilitar precisamente la incorporación de estos dispositivos a un entorno con un sistema operativo de tiempo real que se ha desarrollado en la Universidad de Cantabria MaRTE OS[1][2] y la posibilidad de integrar librerías y aplicaciones previamente desarrolladas para estos.

Un punto interesante a destacar es la falta de concurrencia que tiene Arduino nativamente, por lo que al integrarlo en el proyecto junto con MaRTE OS da la posibilidad de desarrollar prácticas incluyendo el soporte de librerías desarrolladas para Arduino. Esto proporciona versatilidad y facilidad para utilizar dispositivos de manera sencilla poniendo en practica los conceptos teóricos como son la concurrencia o la planificación de tareas bajo distintas prioridades.

Esta posibilidad se materializa en microcontroladores industriales como son los de la familia STM32. Los cuales ofrecen facilidad para desarrollar este tipo de proyectos junto con un buen equilibrio entre rendimiento, consumo y conectividad.

## 1.2. Objetivos

El objetivo principal de este proyecto reside en la creación de un marco de trabajo para facilitar la programación de aplicaciones para MaRTE OS sobre la tarjeta STM32F769I, permitiendo integrar en estas aplicaciones muchos drivers y librerías desarrolladas por la comunidad de Arduino y que pueden ser útiles para desarrollar practicas para los alumnos en los laboratorios de sistemas distribuidos y tiempo real.

Como objetivos secundarios relacionados con el principal se encuentran:

- Estudio y adaptación de la librería STM32DUINO[3] para adaptarlo al sistema operativo MaRTE OS y su ejecución en la placa STM32F769I
- Automatizar el proceso que conlleva la compilación del núcleo de Arduino y sus librerías utilizando su propio IDE, posteriormente integrar también los módulos resultantes con el resto del proyecto(la parte de MaRTE OS).
- Uso de dispositivos(tanto sensores como actuadores) en MaRTE OS a través de la API de Arduino. Tanto para las funciones básicas de la placa(mostrar

mensajes por pantalla, encender los leds integrados...) como para incluir sensores y actuadores básicos a estos programas, como pueden ser servos, sensores digitales, sensores analógicos...

- Desarrollar un demostrador que utilice varios sensores de forma conjunta, haciendo uso de la multitarea proporcionada por MaRTE OS que podría servir como modelo de una practica de complejidad media propuesta a los alumnos.

# 2. Herramientas y tecnologías.

En esta sección se abordarán las herramientas utilizadas tanto para la adaptación de la librería stm32duino como para desarrollo de programas para la placa STM32F769. Además también se dará a conocer los entornos en los que se han trabajado, tanto para la parte de MaRTE OS como para Arduino. Pero primero de todo es esencial explicar las características del controlador que se utiliza y la librería stm32duino para poner en un contexto adecuado antes de empezar a profundizar en la adaptación y creación de librerías.

## 2.1. Microcontrolador STM32F769I.

Esta placa forma parte de la familia STM32[1], fabricada por la empresa ST. Este microcontrolador está basado en un procesador ARM Cortex M7 de 32 bits con varias interfaces de expansión como un panel táctil, gráficos, conectividad a un puerto Ethernet y otras características extras[5]. Esto hace que sea una placa muy adecuada para una amplia variedad de proyectos multipropósito en un formato muy reducido y con un consumo pequeño de energía. Una de sus características mas relevantes y la razón principal de su elección como microcontrolador es su compatibilidad con Arduino, con el objetivo de aprovechar la gran multitud de dispositivos y librerías que ya hay desarrolladas por la comunidad.

En este proyecto se ha usado solamente un pequeño subconjunto de estas características. Se ha utilizado toda la conectividad de pines de Arduino para la conexión de dispositivos y sensores de esta plataforma y también se ha utilizado el display LCD para mostrar tanto resultados de los programas de prueba. Por ultimo se hace uso del puerto st-link para cargar los programas desarrollados en la placa y que también utiliza como fuente de alimentación.

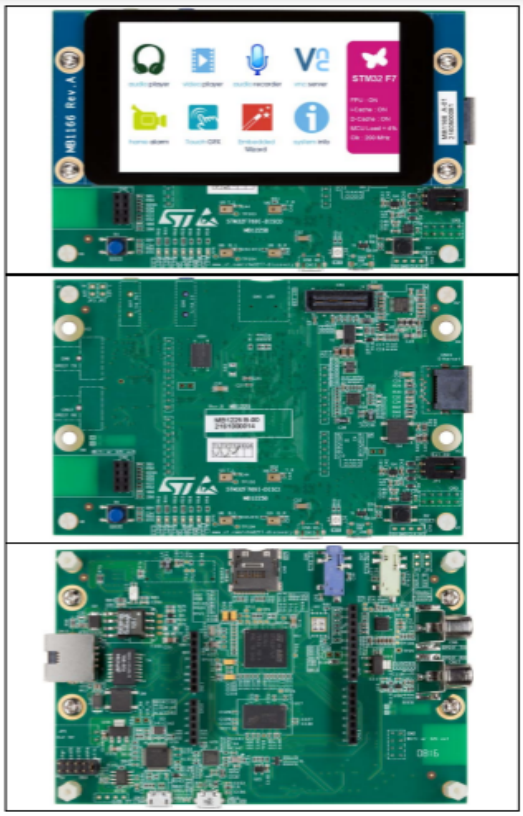
En la Figura 1 se muestra un resumen de las características que tiene este modelo de microcontrolador[2].

[1] La familia de microcontroladores STM32 están basados siempre en núcleos ARM[4], pero que pueden contener varios conjuntos de periféricos, memoria RAM y FLASH y que se utilizan desde proyectos como hobbies hasta aplicaciones industriales.

[2] `https://www.st.com/resource/en/data_brief/32f769idiscovery.pdf`

**Features**

- STM32F769NIH6 microcontroller featuring 2 Mbytes of Flash memory and 512+16+4 Kbytes of RAM, in BGA216 package
- On-board ST-LINK/V2-1 supporting USB reenumeration capability
- USB ST-LINK functions: virtual COM port, mass storage, debug port
- 4" capacitive touch LCD display with MIPI® DSI connector (on STM32F769I-DISCO only)
- SAI audio codec
- Two audio line jacks, one for input and one for output
- Stereo speaker outputs
- Four ST MEMS microphones on DFSDM inputs
- Two SPDIF RCA input and output connectors
- Two push-buttons (user and reset)
- 512-Mbit Quad-SPI Flash memory
- 128-Mbit SDRAM
- Connector for microSD card
- Wi-Fi or Ext-EEP daughterboard connector
- USB OTG HS with Micro-AB connector
- Ethernet connector compliant with IEEE-802.3-2002
- Five power supply options:
  - ST LINK/V2-1
  - USB HS connector
  - 5 V from RJ45 (Power Over Ethernet)
  - 5 V from Arduino™ or external connector
  - USB charger
- Power Over Ethernet based on IEEE 802.3af (Powered Device, 48 V to 5 V, 3 W)
- Power supply output for external applications: 3.3 V or 5 V
- Arduino™ Uno V3 connectors



1. Pictures are not contractual. From top to bottom: STM32F769I-DISCO top view, STM32F769I-DISC1 top view, STM32F769I-DISCO and STM32F769I-DISC1 bottom view.

- Comprehensive free software including a variety of examples, part of the STM32Cube package
- Supported by a wide choice of integrated development environments

Figura 1: Resumen de características del microcontrolador STM32F769I

## 2.2. STM32DUINO

STM32duino es un proyecto desarrollado por el propio fabricante de la placa, que agrupa un conjunto de librerías que tienen el propósito de explotar las características de los microcontroladores basados en STM32. Este repositorio es público [3] en github[3]. De este conjunto de librerías el proyecto se va a centrar en la que denominan 'Arduino_Core_STM32'.

Esta librería es una capa de abstracción intermedia entre Arduino y el hardware que proporciona un conjunto de herramientas[4] como; llamadas API, interfaces comunes a la familia STM32, o un compilador para la arquitectura del microcontrolador que da la posibilidad de explotar el entorno de Arduino casi al completo en este tipo de placas.

[3] `https://github.com/stm32duino`
[4] `https://github.com/stm32duino/Arduino_Core_STM32#Introduction`

Stm32cubeprogrammer [6] es un software desarrollado también por la empresa ST y enfocada a los productos de la familia del microcontrolador. Provee de un entorno completo para leer, escribir y verificar la memoria de estos dispositivos, también para depurar aplicaciones y proporcionar una interfaz para cargar estas a la placa (lo que comúnmente se llama bootloader). En este proyecto unicamente se utilizará este software para este ultimo propósito, dado que para todo lo demás utilizaremos el GNAT studio.

## 2.3. GNAT y GNAT studio

Este es un entorno de desarrollo integrado (IDE), el cual proporciona un entorno en el que se agrupan herramientas útiles para el desarrollo de proyectos, como podría ser un explorador de ficheros, llamadas al compilador, llamada al cargador de ficheros a la placa o incluso al depurador de programas.

GNAT en sus siglas inglesas significaba Traductor GNU[5] de Ada de la NYU[6]. A destacar de este producto, es un entorno multilenguaje con soporte por ejemplo para C,C++ y Python, pero originalmente fue desarrollado para el lenguaje de programación Ada y basado en la estructura de compilación GCC. Está escrito en su mayoría en Ada.

GNAT es el software que se ha elegido para encargarse del proceso de compilación[7] (o al menos una parte como se explicará mas adelante) en el proyecto. Por otra parte, GNAT studio [8] nos proporciona también una interfaz tanto para programar, compilar, ejecutar, depurar y ejecución remota del programa generado al Microcontrolador STM.

## 2.4. Arduino y Arduino IDE

Arduino es un proyecto nacido en 2003 en el instituto de diseño Interactivo de Ivrea en Italia como una necesidad para proporcionar a los estudiantes de electrónica microcontroladores de bajo coste de fácil aprendizaje y fácil uso para sus aulas. La gran razón por la que Arduino se ha convertido en una plataforma tan grande, fue a causa de la decisión de liberar y hacer publico tanto el código como los diagramas y especificaciones hardware para fabricar los dispositivos. Esto implica que cualquier persona o empresa pueda crear sus propias placas manteniendo el núcleo desarrollado por el proyecto original.

El software libre de esta manera ha proporcionado a la plataforma un empuje de la mano de toda la comunidad de desarrolladores la cual se ha encargado de ampliar y dar vida a este proyecto dándole un gran alcance, sobre todo para usuarios que no necesariamente tienen que tener unos avanzados conocimientos en electrónica e informática por su facilidad de uso. Esto ha hecho que se forme un nicho de mercado

[5]GNU General Public License, también conocida como General Public License (GPL), es una licencia de software libre ampliamente utilizada, originalmente escrita por Richard Stallman para el Proyecto GNU. El Proyecto GNU se lanzó en 1984 para desarrollar un sistema operativo completo de estilo UNIX que es software libre: el sistema GNU.

[6]Universidad privada de New York, Estados Unidos.

en el que se incluyen a personas que lo utilizan para proyectos caseros, como hobbie y también por supuesto en el ámbito educativo como era su objetivo en su nacimiento.

El entorno de desarrollo integrado (IDE) de Arduino[9] es una aplicación multiplataforma (disponible en Windows, Mac y Linux) escrita en lenguaje Java y publicado bajo la licencia GNU que admite los lenguajes de programación C y C++. El IDE es un entorno de programación que soporta tanto edición, compilación como depuración de los programas que desarrolles para Arduino. Ademas estas características las proporciona bajo una GUI que también se encarga de de cargar el programa generado en la memoria flash de nuestra placa Arduino.

En el caso de este proyecto, esta herramienta se utiliza no solo para desarrollar y probar, sino para acceder a una compilación de librerías y el core de Arduino de una manera simple.

Este IDE fue una herramienta relevante en las primeras fases del proyecto para verificar el correcto funcionamiento de las librerías que posteriormente se integraron en la plataforma de ejecución.

## 2.5. MaRTE OS.

Es un sistema operativo de tiempo real desarrollado por el grupo de Ingeniería del software y Tiempo Real en la Universidad de Cantabria[7]. La mayoría del software está escrito en Ada con algunas partes en C. Este proyecto permite el desarrollo cruzado entre aplicaciones C y Ada usando los compiladores GNU (Gnat y gcc). En el caso particular de la 'run-time-library' ha sido adaptada para correr en el kernel de MaRTE OS.

MaRTE OS soporta dos tipos de arquitectura, x86 y Raspberry(modelos 1 y zero). Para la arquitectura x86 se forma un entorno cruzado formado por un PC que utiliza Linux denominado 'host' y una placa x86 denominada 'target'. Ambos sistemas conectados a una red ethernet y una conexión serie.

MaRTE OS implementa muchas de las funcionalidades POSIX[10] de tiempo real como son el manejo de threads, planificación de estos con prioridades, relojes y timers. Es interesante ver como estas características se han utilizado en este proyecto para integrarlas en aplicaciones complejas o incluso en el desarrollo de librerías de bajo nivel en algunos dispositivos para dotarlas de concurrencia en el sistema.

[7] `https://marte.unican.es`

# 3. Adaptación de librerías compatibles con Arduino

Uno de los objetivos del proyecto es generar un entorno en el que sea posible utilizar las librerías que existen para las placas de Arduino, esto es, las librerías para utilizar dispositivos como pueden ser de entrada/salida, tanto digitales como analógicos, e incluso aparatos mas complejos como un servomotor, un mando infrarrojo o una pantalla LCD . Al integrar este código y aplicaciones generadas tanto por los fabricantes como por la amplia comunidad que tiene Arduino, se abre un abanico muy grande de posibilidades para desarrollar multitud de ejercicios, prácticas o para la práctica de la programación sin tener centrarse en el desarrollo de drivers de bajo nivel en cada dispositivo.

Por lo tanto, uno de los primeros pasos es generar un entorno en el que se tenga la capacidad de compilar programas que utilizan tanto los recursos de MaRTE OS como las cabeceras de Arduino. Este proceso de compilación cruzada[8] es necesario dado que en el GNAT studio unicamente se compilan las aplicaciones desarrolladas en C junto con las cabeceras de MaRTE OS.

Para compilar la parte del núcleo de Arduino se debe utilizar un compilador distinto que también admita el lenguaje C++. En este proyecto se ha utilizado el que incluye el propio entorno de Arduino en forma de linea de comando.

Por lo tanto para utilizar las librerías del núcleo de Arduino se ha desarrollado un script[9] que automatiza todo este proceso. El script realiza en primer lugar una compilación de las librerías propias de Arduino, utilizando la linea de comandos propia de este entorno. Los objetos resultantes generados se incluyen en el directorio que contiene los ficheros de MaRTE OS, para luego compilarlo todo junto desde el GNAT studio. Una vez en este punto, se pueden hacer los programas para ejecutar en el microcontrolador con sistema operativo MaRTE OS pero pudiendo utilizar los recursos del núcleo de Arduino.

Como punto también importante se ha configurado el script para poder compilar el núcleo de Arduino con las opciones de depuración. De esta manera se puede utilizar el 'debugger' en tiempo de ejecución para tener una mayor facilidad de detectar y corregir problemas tanto los que han surgido durante el desarrollo del proyecto como para posibles futuros proyectos.

Otro de los objetivos que tiene el proyecto es la ejecución de librerías externas al core de Arduino. Las cabeceras de dichas funciones (.h) hacen referencia a código C++, el cual el compilador no admite. Por ello, se ha creado una capa de abstracción o wrappers[11] del código de C++ de las librerías para que pueda ser invocado desde las aplicaciones que se ejecuten sobre MaRTE OS. Por tanto con la compilación cruzada que explicábamos antes, podemos generar los objetos de estas librerías, y utilizar el wrapper desde el GNAT para llamar a las funciones como si estuviesen programadas en C.

[8]Es una técnica en la que se compila el código en un sistema diferente(HOST) del que en el que se ejecuta. Esto es usual en este tipo de entornos con microcontroladores pequeños como el que se utiliza en este proyecto.

[9]Script es una secuencia de comandos los cuales los ejecuta un interprete que lee el fichero con el código fuente.

En el Código 1 se puede ver un ejemplo de un wrapper:

Source Code 1: Ejemplo de Wrapper.

```
#include <Ultrasonic_c.h>
#include <Ultrasonic.h>

EXTERNC Ultrasonic_t ultrasonic_create(int val) {

        return static_cast<Ultrasonic_t>(new Ultrasonic(val));


}

EXTERNC long medida_centimetros(Ultrasonic_t ultrasonic) {

        Ultrasonic* con = static_cast<Ultrasonic*>(ultrasonic);
        return con->MeasureInCentimeters();

}
```

Como se ve en el ejemplo, estas funciones del wrapper hacen las transformaciones necesarias para no necesitar las llamadas a funciones originales en C++, consiguiendo que puedan utilizarse en las aplicaciones, las cuales posteriormente cargaremos (con el bootloader[10] integrado en el GNAT studio) y ejecutaremos en la placa.

En la sección 5.2 se explica y muestra el funcionamiento de varios de estos dispositivos que se han hecho funcionar en el microcontrolador utilizando wrappers y para los que parecía interesante desarrollar un programa de prueba que muestre su funcionamiento utilizando las librerías nativas de Arduino.

## 3.1. Modificación de partes del núcleo

Como parte de la adaptación de las librerías es importante modificar las funciones del núcleo de Arduino que proporcionan un 'delay' en el tiempo y las que dan un instante en el tiempo del reloj del microprocesador. Originalmente en esta librería utilizan la implementación nativa de Arduino, en cambio para este proyecto es interesante poder utilizar las funciones de tiempo que proporciona MaRTE OS.

Por tanto se han cambiado las partes del núcleo de Arduino que hacen estas llamadas. En concreto se han modificado las funciones de los ficheros 'wiring_time.c' y 'wiring_time.h', sustituyendo las implementaciones que trae Arduino por llamadas a las funciones que incorpora MaRTE OS[11].

En la nueva implementación de 'delay' y 'delayMicroseconds' por ejemplo se hace una llamada a la función 'usleep'[12], la cual hace la misma tarea de esperar el

[10] Software que se encarga de escribir el programa que cargamos en la memoria de la placa a través del puerto serie. En el caso de este proyecto es el stm32cubeprogrammer explicado anteriormente.
[11] En concreto se ubican en 'time.h'

tiempo indicado por parámetro, pero incluyendo la suspensión del thread que la ejecuta permitiendo liberar el procesador durante la espera.

## 3.2. Problemas en la adaptación de la librería stm32duino

Siguiendo el hilo de la adaptación de libreras se han hecho varios wrappers para distintos tipos de sensores, que se explican posteriormente.

Sin embargo, el hardware timer[13] que utiliza Arduino, ha generado un conflicto con los propios timers que utiliza MaRTE OS. Cuando estas librerías intentan utilizar el timer proporcionado por MaRTE OS no presentan el funcionamiento esperado por lo que sensores con funcionamientos mas complejos como son el Servo, el mando infrarrojo, o una función Arduino como 'analogWrite'[12] no funcionan.

El uso de este recurso es común para la mayoría de dispositivos medianamente complejos, por ejemplo para la generacion continua de pulsos[14].

Como solución para poder seguir utilizando esos dispositivos se ha optado por crear unas nuevas librerías que permitan el uso de ese hardware sin tener que utilizar los timers. A continuación de describirá con detalle cada una de ellas.

[12]Función que escribe un valor analógico en un pin. Esto hace que en ese pin se genere una onda cuadrada con el ciclo especificado hasta la siguiente llamada a esta función en ese pin.

# 4. Creación de nuevas librerías

Debido a las limitaciones detectadas en el análisis del apartado anterior, esta sección aborda el desarrollo de nuevos drivers para aquellos dispositivos cuyos drivers existentes en Arduino no pueden ser utilizados directamente con MaRTE OS.

Un matiz importante a destacar es que a pesar de crearse nuevas librerías, para la creación de estas mismas se han seguido utilizando el core de Arduino y las funciones que proporciona(interrupciones, definiciones, funciones de escritura en pines, etc).

## 4.1. Librería para el MandoIR

Una de las funciones mas interesantes para el desarrollo de un demostrador para prácticas en el laboratorio de la asignatura podría ser controlar remotamente las acciones en el microcontrolador. Para trabajar sobre esto se pretende desarrollar una librería que permita manejar la pareja que conforman un receptor y mando de señales infrarrojas. Dado que la librería que proporciona el fabricante para Arduino hace uso de HardwareTimers, en este punto se pretende el desarrollo de una librería desde cero que cubra una funcionalidad básica del dispositivo.
Para este desarrollo lo primero que se debe tener en cuenta es el funcionamiento del protocolo de datos bajo el que trabaja el dispositivo.

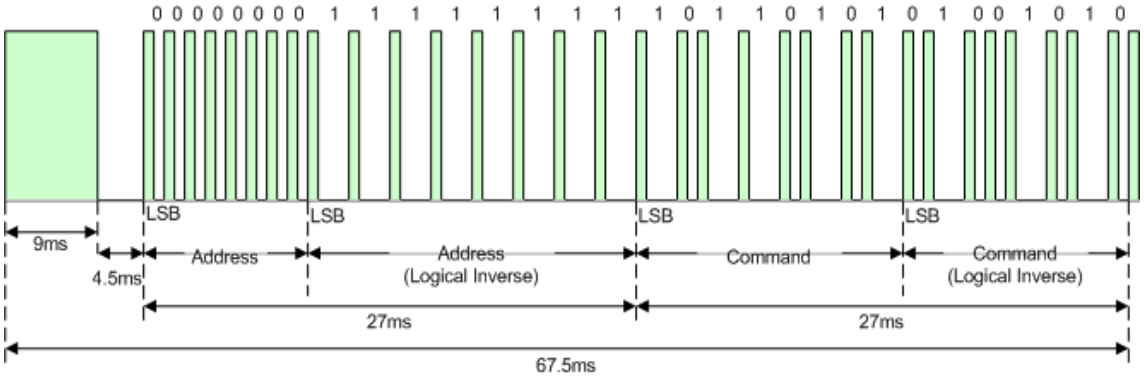



Figura 2: Esquema funcionamiento del protocolo NEC

Un breve resumen de como funciona el protocolo NEC[15] (un dato a destacar es que este protocolo tiene ligeras variaciones dependiendo del fabricante del que provenga) se muestra a continuación. En la Figura 2 se puede ver esquemáticamente como funciona la lógica de este protocolo. La información se transmite en bits lógicos codificados con el siguiente criterio:

- 0 Lógico. Un pulso de flanco alto de 562,5 microsegundos de duración, seguido de un espacio de 562,5 microsegundos de duración. Un total de 1,125 milisegundos.
- 1 Lógico. Un pulso de flanco alto de 562,5 microsegundos de duración, seguido de un espacio de 1,6875 milisegundos de duración. Un total de 2,25 milisegundos.

Siguiendo el esquema de la Figura 2, y obviando la lógica de los bits de datos que se utilizan para las direcciones de dispositivos que no se van a utilizar para simplificar el código(es decir, solo podremos utilizar un único dispositivo de este tipo simultáneamente) y ya que solo vamos a usar un único mando, hay que generar una librería que siga el esquema lógico que se ve en la Figura 3.

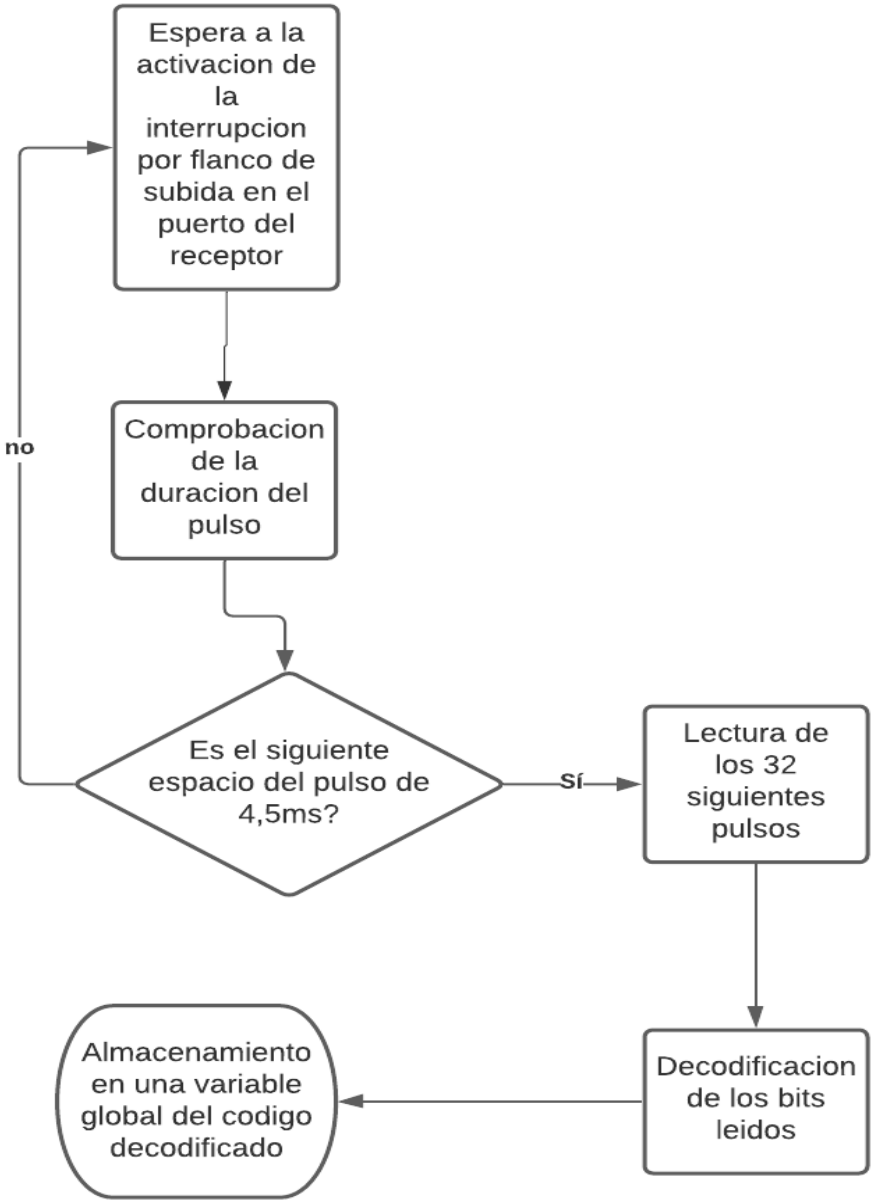



Figura 3: Diagrama Flujo librería Infrarroja

Como se ve en el Diagrama de Flujo, la idea es detectar que ha habido una interrupción antes de que el pulso de 9ms de comienzo termine, de esta manera seremos siempre capaces de detectar el siguiente espacio de 4,5 milisegundos al completo.

Una vez se tenga esta medición se sabe seguro que el protocolo ha empezado.

Como se ve en la Figura 3, el primer elemento de control que se utiliza para el sensor infrarrojo son las interrupciones, que vinculadas a un puerto en concreto, da la posibilidad saber cuando hay un flanco de subida en dicho puerto. En la atención a esta interrupción, se cambia un flag que le indica a la librería que está en el punto de comienzo del protocolo NEC.

Si esta librería fuese el único programa que se está ejecutando en el SO, podríamos comprobar si el flag ha cambiado o no con una simple espera activa. En cambio es mas interesante el poder detectar pulsaciones en el mando infrarrojo mientras el programa ejecuta otras funciones o tareas de forma concurrente, para ello debe de hacer uso de una de las construcciones que proporciona MaRTE OS, la tarea periódica[16]. De esta manera, se puede poner a la tarea un periodo(menor a los 9 milisegundos que tenemos de pulso inicial) para que compruebe el estado del flag, y de esta manera saber si hay que lanzar el proceso para leer los 32 datos siguientes. Con los datos ya leídos, solo hay que decodificar los bits correspondientes a la información de la tecla que se ha pulsado(en este caso unicamente los bits 16 a 23) para obtener un código decimal que el programa pueda comparar con una tabla ya predefinida con los códigos de cada tecla.

En el Anexo A se detallan los prototipos de las funciones de la librería.

### 4.1.1. Mejoras especificas a la librería

Como se ha explicado en puntos anteriores, la librería basa su lectura de los datos que envía el mando infrarrojo de medir el espacio entre pulsos altos. Como hemos visto en la Figura 2, para leer una secuencia completa del protocolo, o lo que es lo mismo, recibir un dato enviado por la pulsación de un botón del mando, desde el pulso de inicio de protocolo hasta que se recibe el ultimo de los 32 'unos' o 'ceros' pasan un total de 67,5 milisegundos que pasamos leyendo el tiempo que duran estos pulsos(o el espacio entre ellos). Esto se hace con una función proporcionada por el core de Arduino que se llama "pulseIn". El problema de esta función es que es bloqueante, por lo que en esos 67 milisegundos el sistema esta exclusivamente haciendo esta tarea. La mejora propuesta es desarrollar una alternativa a esta función que mida el tiempo igualmente pero sea parcialmente no-bloqueante. Sin esta mejora, se aprecia en los demostradores complejos en los cuales hay varias tareas periódicas corriendo, que el resto de tareas no se ejecutan durante el periodo bloqueante que genera la librería con las lecturas de pulsos.

Otra mejora implementada en la librería respecto a la original de Arduino es cambiar el modo en el que mide los tiempos entre cada pulso. En vez de utilizar una función bloqueante de Arduino para conocer la duración del pulso, se ha cambiado la lógica para que se produzca una interrupción[17] cada vez que llegue un pulso al pin que está conectado el sensor.

De esta manera se ejecuta una lectura del reloj del sistema para apuntar cada una de las llegadas. Cuando detecta que ha llegado el ultimo pulso de subida del protocolo y sepamos que ha acabado el mismo se pasa a analizar los tiempos obtenidos igual que se hacia en la primera versión de la librería explicada. La única diferencia es que en vez de tener duraciones de pulsos, se registran instantes temporales de llegada de

cada pulso. Esta mejora implica que la librería es mas eficiente al no ser bloqueante en ningún momento para el microcontrolador cuando se ejecuta concurrentemente con otras tareas.

## 4.2. Librería para el ServoMotor

Figura 4: Imagen de un servo motor

El servomotor[18], el cual se puede ver en la Figura 4 tiene un funcionamiento que se basa esencialmente en generar una onda cuadrada con frecuencia de 50Hz en el pin digital Arduino al que esté conectado su cable de datos. Según la duración del pulso alto en esta onda generada el servo se moverá a una posición determinada. Los valores mas representativos son los siguientes:

- Si el pulso mide 544 microsegundos, el servo se va a la posición de 0 grados, este es el mínimo posible.
- si el pulso mide 2400 microsegundos, el servo se posiciona en 180 grados, este es el máximo posible.

El resto de posibles valores entre el máximo y el mínimo posicionaran al servo en proporción respecto a estos. Para que el servo mantenga la fuerza en la posición especificada, necesita que el pulso generado sea constante, a causa de esto no sirve generar un único pulso en un instante del tiempo y parar. Se necesita un mecanismo para que la librería una vez utilizada la función para mover el servo a una posición determinada, esté generando el pulso correspondiente continuamente hasta que se vuelva a utilizar esa función para mover hacia otra posición. La Figura 5 representa gráficamente el funcionamiento descrito:

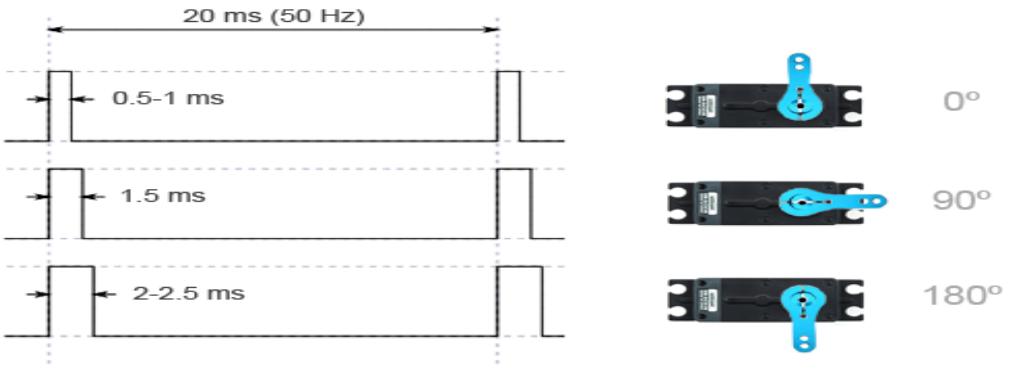



Figura 5: Esquema de funcionamiento de un servomotor

Por lo tanto, la primera tarea para desarrollar una librería que sea capaz de manejar un servomotor debe ser el construir una función que genere un pulso determinado. Una primera aproximación de lo que necesita se muestra en la Figura 5, en ella se ve como la función hace la gestión de poner los valores ALTO o BAJO en el pin al que se ha conectado el servomotor, con un delay entre uno y otro que define el ancho del pulso. Es de destacar que para generar este delay ya se utiliza una de las funciones que proporciona MaRTE OS como es el nanosleep[19] en vez de utilizar un delay básico de Arduino. La diferencia entre estos dos es que el nanosleep suspende el thread durante el tiempo que se le ha especificado en la llamada a la función, por lo que el procesador puede ejecutar otras tareas concurrentemente[20]. Este detalle es importante dado que uno de los objetivos de este proyecto (ya que se utiliza un sistema operativo de tiempo real) es desarrollar aplicaciones que tengan la capacidad de ejecutar múltiples tareas concurrentes en el microcontrolador.

Source Code 2: pseudocódigo de la funcion creaPulso()

```
void creaPulso(){

        digitalWrite(servoPin,HIGH);
        nanosleep(&request,&remaining);
        digitalWrite (servoPin,LOW);

}
```

Esta función por si sola no genera una onda periódica, solo generaría uno de los ciclos de la onda que necesita. Para completar el desarrollo se requiere una segunda función, en este caso otra de las funciones que proporciona MaRTE OS, la tarea periódica. Utilizando una tarea periódica, se puede controlar la frecuencia con la que llama a la función para generar cada pulso, suspendiéndose la misma entre llamada y llamada, y permitiendo también la concurrencia de eventos en el sistema. Para completar la funcionalidad de la librería y que proporcione unas funciones parecidas a la que nos daría la nativa en el core de Arduino se muestra en el Código 3 la cabecera de las funciones. Las cuales se resumen en la configuración del dispositivo

y 4 funciones que hacen uso de la generación de pulsos para dar un control del servomotor.

Source Code 3: Cabecera de la libreria del servomotor.

```
void configuraServo (int pin);

void mueveServo(int pos);

void barrido(int anguloMin, int anguloMax);

void paraBarrido();

int getBarriendo();
```

De esta manera, tenemos una librería que permite el uso de un servomotor en una placa Arduino y con sistema operativo MaRTE OS, que permite manejar el dispositivo como se haría en Arduino nativo e incluso tener un servo barriendo entre ángulos específicos constantemente con una función que es no-bloqueante.

## 4.3. Librería para el Control de motores

En esta sección se va a describir la librería desarrollada para controlar los motores de corriente continua[21], uno de los tipos mas comunes de motores. Se expone primero el esquema general de funcionamiento de dichos motores.

### 4.3.1. Descripción general de los motores utilizados

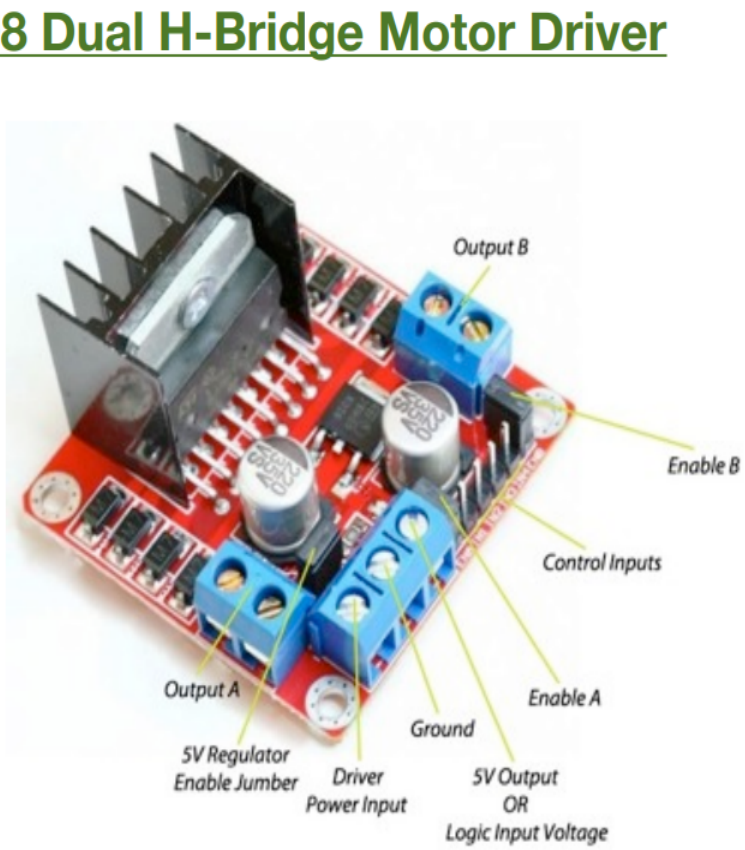



Figura 6: Imagen de un controlador de un motor de corriente continua.

Estos motores son motores DC o de corriente continua,y como intermediario entre la placa y los motores se utilizará un driver hardware L298 Dual Motor Driver Module mostrado en la Figura 6. Con los puertos de este dispositivo son con los que interactuará directamente nuestra librería de la siguiente manera.

- ENA: Permite activar o desactivar el Motor A
- ENB: Permite activar o desactivar el Motor B
- IN1, IN2: Con estos pins controlas la dirección de la rotación del motor A
- IN3, IN4: Con estos pins controlas la dirección de la rotación del motor B

Con esto en mente, las funciones básicas para mover los motores quedarían resumidas en el esquema representado en la Figura 7:

```
//    The direction of the car's movement
//  ENA   ENB   IN1   IN2   IN3   IN4   Description
//  HIGH  HIGH  HIGH  LOW   LOW   HIGH  Car is runing forward
//  HIGH  HIGH  LOW   HIGH  HIGH  LOW   Car is runing back
//  HIGH  HIGH  LOW   HIGH  LOW   HIGH  Car is turning left
//  HIGH  HIGH  HIGH  LOW   HIGH  LOW   Car is turning right
//  HIGH  HIGH  LOW   LOW   LOW   LOW   Car is stoped
//  HIGH  HIGH  HIGH  HIGH  HIGH  HIGH  Car is stoped
//  LOW   LOW   N/A   N/A   N/A   N/A   Car is stoped
```

Figura 7: Resumen del funcionamiento lógico de los motores respecto a las entradas del controlador.

Hasta aquí sigue un paralelismo directo como si fuese a usar el controlador en Arduino nativo. El problema es de nuevo el uso de los Hardware Timers que se utilizan en el control de la velocidad en los motores. En particular, cuando se usa la función AnalogWrite() la cual escribe un valor analógico en el puerto indicado generando una onda rectangular de ciclo especificado hasta que se vuelva a llamar a la misma función. Sin poder hacer uso de esta función, los motores irán o bien al 100 % de potencia o al 0 %, que es lo que permite la función DigitalWrite (o escribimos 0, o 1).

En el Anexo B se detallan los prototipos de las funciones de la librería.

### 4.3.2. Adaptación al problema de los Timers

Se propone el generar una función propia que sustituya al AnalogWrite para tener el control de la velocidad de los motores DC pero sin utilizar los HWTimers. Para esto se utiliza un mecanismo similar al que ya se ha explicado en el Servomotor. Tendremos una tarea periódica configurada en MaRTE OS que se encarga de poner ese pin responsable de la velocidad en posición alta o baja según corresponda, a una frecuencia calculada para que se genere como resultado la onda adecuada.

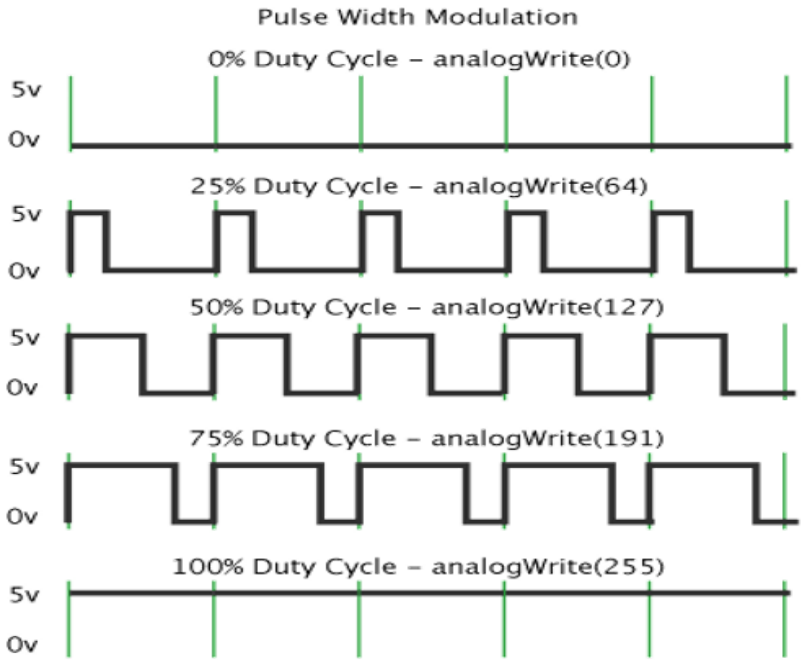



Figura 8: Ejemplos de la onda resultante al usar la función AnalogWrite.

En la figura 8, se ven 5 ejemplos de la onda que la función debe generar(entre otros) para ciertos valores de entrada. Consiguiendo así, que dependiendo del valor de entrada que se le de a dicha función se obtenga una onda equivalente en los puertos ENA y ENB pudiendo controlar así la velocidad de igual manera que si se utilizase la librería nativa de Arduino.

Para que esto funcione es necesario que en la aplicación se produzca la inicialización del thread que controla la generación del pulso, y este deberá continuar activo mientras se necesite el control de velocidad en los motores. Una de las claves para que esta función no acapare toda la CPU de nuestra placa, es utilizar de nuevo un delay diferente a la original de Arduino y basada en la función nanosleep. De esta manera este proceso solo utilizara la cantidad de tiempo de CPU que necesite para poner 'alto' o 'bajo' el puerto. El funcionamiento resumido de la creación de un solo ciclo de pulso se muestra en el siguiente fragmento de pseudocodigo. Siendo el tiempoDelay1 el tiempo en el que la onda permanece en alto, y el tiempoDelay2 el tiempo que permanece en bajo, o lo que es lo mismo en el código, el periodo de nuestra tarea.

Source Code 4: Pseudocódigo para generar un ciclo de onda en un pin determinado.

```
void creaPulso(){
        digitalWrite(Pin,HIGH);
        nanosleep(tiempoDelay1);
        digitalWrite (Pin,LOW);
        nanosleep(tiempoDelay2);
}
```

## 4.4. Librería del motor paso a paso

Un dispositivo hardware que se considera interesante como para incluirlo en el proyecto es el 'stepper' o motor paso a paso[22], el cual se puede ver representado en la Figura 9. Su funcionamiento es simple y obtiene mas precisión para movimientos en ángulos pequeños que el que puede dar el motor de corriente continua o incluso el servoMotor. El modelo en especifico con el que se ha trabajado tiene una precisión de 64 pasos por vuelta, que junto con un reductor interno se convierten en 4096 pasos por un giro completo (360 grados), esto es equivalente a que cada paso tiene una precisión de $0{,}0088^{0}$. La desventaja respecto a los otros tipos de motor vistos anteriormente es que a velocidad máxima unicamente consigue entorno a 1.5rpm, no siendo adecuado si el objetivo es hacer giros rápidos. Este tipo de motor se utiliza junto con un controlador (en este caso el ULN2003) y tanto este como el motor tienen una sencilla conexión a los pines de la placa Arduino.

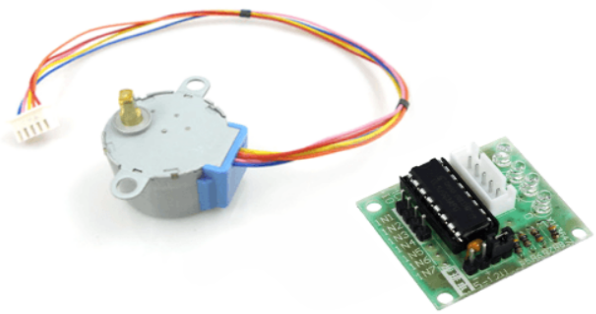


Figura 9: Imagen de un motor paso a paso y su controlador.

El funcionamiento de este motor es relativamente sencillo si se compara con el servomotor y se puede utilizar sin necesidad de importar una librería a pesar de que Arduino tiene una integrada. Después de probar esta librería nativa, se ha detectado que su funcionamiento es muy limitado(produce un movimiento trabado, se calienta mucho y solo gira en un solo sentido) por lo que se ha desarrollado una que recopila las funcionalidades básicas para este motor, ademas de como se verá a continuación se han añadido un par de funciones extra para hacerla mas completa.

En la Código 5 mostramos la cabecera de la librería con las funciones que provee.

Source Code 5: Cabeceras de la libreria del stepper.

```
//funcionalidad básica
void stmMaRTE_stepper_spinReloj(int grados,int velocidad);
void stmMaRTE_stepper_spinAntiReloj(int grados,int velocidad);

//funcionalidad avanzada con threads
void stmMaRTE_stepper_init(int prio,int pin1,int pin2,int pin3,int pin4);
void stmMaRTE_stepper_spin(int velocidad,int sentido);
```

Se puede ver que las dos primeras funciones proporcionadas nos ofrecen una funcionalidad mas básica, en este caso mueven el motor ya sea en sentido horario o

antihorario pudiendo proporcionar dos parámetros configurables que son la velocidad de giro y los grados exactos que se mueva. También en esta librería se decide añadir una funcionalidad extra con la que poder hacer girar el motor continuamente con una única llamada.

Esto se consigue utilizando una tarea periódica, la cual se inicializa en la función 'init' y posteriormente cuando llame a spin configurará su periodo de acuerdo con la velocidad parametrizada en la llamada de la misma. De esta manera en una hipotética aplicación que utilizase este hardware, seria capaz de mantener el motor girando continuamente con una única llamada a la función, y la tarea periódica se despertará en los momentos del tiempo adecuados para hacer que gire a la velocidad configurada dejando el procesador libre el resto del tiempo para otras posibles tareas o acciones.

## 4.5. Librería para el passiveBuzzer

La librería Tone, muy utilizada en los kits básicos de Arduino. Esta permite generar una onda cuadrada a una frecuencia especificada en un pin, con una duración también parametrizable. Si a ese pin se le conecta un 'buzzer'[23] o cualquier otro tipo de altavoz se obtiene un tono audible como salida.

Este sensor parecía interesante integrarlo en el proyecto porque permite de forma sencilla demostrar una concurrencia real en un sistema sencillo como el que se propone mas adelante. Por ejemplo, si una aplicación es capaz de ejecutar concurrentemente varias aplicaciones y generar un tono continuo representando algún tipo de melodía o canción conocida sin desvirtuar el sonido, significa que se consigue que la tarea que se utiliza para generar este tono esta ejecutándose con los periodos especificados en el sistema con MaRTE OS.

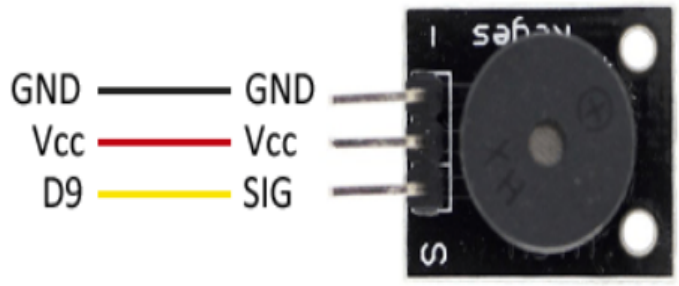



Figura 10: Esquema de conexiones en un zumbador pasivo.

La razón por la que no se puede utilizar esta librería Tone de forma nativa en MaRTE OS es la misma que se ha explicado en anteriores puntos como el Servo o el control de velocidad en los motores, la imposibilidad de utilizar los HardwareTimers. Por lo tanto lo primero que se hace para construir una librería que nos proporcione un funcionamiento similar a la original, es entender como funciona. La generación de sonido en los zumbadores mas simples de Arduino (Figura 10) se produce a través de una onda cuadrada que les llega por el pin digital al que están conectados, cuanto

mas corto sea el ciclo del pulso, mas agudo será el tono producido, y viceversa. Por lo tanto, una vez mas el objetivo será conseguir que la librería genere este pulso.

La diferencia respecto a las anteriores adaptaciones en la que se genera un pulso continuo llega en cuanto a que por la naturaleza del sensor, aquí hay que controlar tanto la duración como la frecuencia continuamente, ya que se quiere conseguir reproducir una melodía, la cual está compuesta de varias notas y estas a su vez, varían tanto en frecuencia, como en duración.

En el siguiente pseudocodigo se muestra otra función para generar un pulso cuadrado pero añadiendo las variantes necesarias para controlar la duración de las notas.

Source Code 6: pseudocodigo de una funcion que genera un tono.

```
void myTone(int pin, int frequency, int duration){

        startTime=millis();
        halfPeriod= 1000000L/frequency/2;
        while (millis()-startTime< duration)
        {
                digitalWrite(pin,HIGH);
                nanosleep(halfPeriod);
                digitalWrite(pin,LOW);
                nanosleep(halfPeriod);
        }
}
```

Con la función anterior se genera una única nota, con su frecuencia y duración correspondiente que se pasan como argumentos. Al igual que en las anteriores adaptaciones que se han hecho con el servo y el control de velocidad se ha programado teniendo en mente que en los tiempos de espera entre subidas y bajadas del pulso se suspenda el thread para dar paso a otras tareas que puedan ejecutar en esos intervalos.

Ahora que se tiene la capacidad de reproducir una nota especifica necesitamos un mecanismo para que dispense nota a nota en los tiempos adecuados a la función myTone. Para este propósito se vuelve a utilizar el mecanismo de las tareas periódicas, cuando se vaya a reproducir una melodía, junto con el array de notas de la melodía que se va a tocar, se hace una inicialización de un thread periódico que se encargará en cada iteración de pasar la siguiente nota correspondiente del array a la función que la reproduce. Es necesario que el thread periódico sea capaz de adaptar su periodo a la necesidad de la canción que se va a tocar, ya que en el array tendremos multitud de notas y cada una puede tener diferente duración, diferente espacio entre notas, etc. Por lo que se irá recalculando en cada iteración para adaptar su propio periodo de ejecución y de esta manera despertarse en el sistema en el momento justo para dispensar la siguiente nota correctamente.

Una observación importante respecto al comportamiento de esta librería, es que si se quiere que la melodía que estamos reproduciendo no se distorsione cuando compite

por la CPU con otras funciones que puedan estar ejecutándose concurrentemente, hay que dotarle de una prioridad mayor de ejecución. En caso contrario cuando en el sistema compitan varios threads con el de la música, puede no ejecutarse en el momento exacto previsto, provocando que coja la CPU algunos instantes de tiempo mas tarde(esto dependerá mucho de la carga de CPU) haciendo que el resultado quede audiblemente distorsionado.

En el Anexo C se detallan los prototipos de las funciones de la librería.

# 5. Demostradores

En esta sección se aborda uno de los aspectos mas importantes y en los que se ha invertido mas tiempo del proyecto, y es que ya se expuso en la introducción al trabajo que uno de los objetivos era conseguir construir un marco de trabajo en el que los alumnos en un hipotético laboratorio de tiempo real tengan las herramientas para construir practicas sencillas que permitan demostrar y entender de una manera visual e intuitiva las teorías explicadas en clase.

Por ello en el proyecto se han desarrollado una serie de demostradores con el objetivo de poner en práctica todo lo explicado anteriormente de las adaptaciones que se han hecho tanto al 'core' de Arduino, como para probar las librerías adaptadas a MaRTE OS, e incluso para integrar en estos demostradores herramientas de medición de tiempo con lo que los dotamos con la posibilidad de sacar mediciones exactas en el tiempo con las que comparar el experimento del laboratorio con los ejercicios teóricos hechos previamente.

Dividimos estos demostradores en varios tipos para ir explicando cada uno independientemente.

## 5.1. Demostradores simples

En este apartado se incluyen los primeros demostradores que se han desarrollado tanto para probar la placa, como medición de tiempo o incluso los sensores mas básicos que conectamos(normalmente los mas sencillos suelen ser los que devuelven un valor analógico a partir de un mecanismo físico, como puede ser un sensor lumínico). También los sensores que aunque su librería sea mas compleja, el programa de prueba con el que se usa tiene un funcionamiento simple:

- blink.c : Este programa de prueba es el ejemplo mas sencillo que podemos hacer con Arduino, y que también es el que se ha utilizado en los primeros pasos del proyecto para comprobar que nuestra placa tenia conectividad con el ordenador(HOST), que el cargador de programas del host a la placa funcionaba y sobre todo que el mapeo de pines y leds estaba funcionando adecuadamente. Lo único que este código hace es ejecutar una y otra vez en bucle un parpadeo de un led que la placa tiene integrado, es decir se pone el pin correspondiente a alto, hace un delay, y luego lo pone en flanco bajo.

- delays-stm32duino.c : Este demostrador se diseñó para medir la granularidad del sistema MaRTE OS sobre el que ejecutamos programas. Es un programa simple en el que se toma el tiempo del reloj del sistema, hace un delay y vuelve a coger el reloj de la CPU para ver si la diferencia de tiempos coincide con el delay que se hizo entre medias. Ejecutando varias muestras de este código, cada vez con delays mas pequeños se llega a un punto en el que vemos que la diferencia de tiempos se estanca en una cantidad de tiempo fija, la cual es la granularidad del reloj en el sistema operativo. En el caso de MaRTE OS son 100 micro-segundos. Adjunto un ejemplo de pseudocódigo de una versión resumida para una de las mediciones en este demostrador.

Source Code 7: Código para medir la granularidad del sistema.

```
ms0 = millis();
clock_gettime(ts0);
delay(MS_DELAY);
clock_gettime(ts1);
ms1 = millis();
printf(" Measured delay (clock_gettime): ts1 - ts0");
printf(" Measured delay (millis): ms1 - ms0" );
```

- lsensor.c Este demostrador hace una sencilla prueba del sensor de intensidad lumínica, el cual puede detectar la intensidad de la luz que hay en un entorno 'indoor'. La salida que produce este sensor es un valor analógico depositado en el pin al que está conectado. Este valor producido se puede traducir a un valor lumínico aproximado utilizando una tabla que el fabricante proporciona adjunta al sensor.

  Este sensor analógico por su sencillez carece de librería en Arduino por lo que en este proyecto se puede usar de igual manera sin utilizar ningún añadido, por lo que se obtiene una portabilidad total.


Figura 11: Imagen de un sensor lumínico.

- sharp.c En este caso se construye un demostrador para el sensor de medición de distancia SHARP GP2DI20 [24] y al igual que en el demostrador explicado en el anterior punto, estamos hablando de otro sensor analógico, el cual emite un rayo de luz infrarroja para devolvernos una tensión proporcional a la distancia medida en el puerto analógico conectado. Por lo tanto al tratarse de otro dispositivo de uso sencillo, no requiere de una librería adicional para su uso. A pesar de esto, en este proyecto este sensor se pretende utilizar en concreto en un demostrador que explicaremos mas adelante por lo que finalmente agregamos

código extra, añadiendo una tarea periódica que con un periodo de ejecución configurable hace una serie de medidas para devolver la media de estas, este cálculo se realiza para evitar errores puntuales en las mediciones ya que este tipo de sensores tienden a dar alguna medida errónea en ciertas ocasiones de mala luz u objetos con refracciones extrañas. También se ha añadido una comprobación para que el resultado de nuestras mediciones esté dentro de un umbral(este umbral lo configuramos a nuestro gusto) para si supera dicho umbral tomar la acción que se quiera (es un disparador).

Figura 12: Imagen de un sensor de distancia óptico.

## 5.2. Demostradores de librerías con wrappers

En esta sub-sección se construyen demostradores de las librerías para las que se ha desarrollado un wrapper en la fase intermedia del proyecto, es decir, son librerías que que utilizan tal y como se utilizarían en Arduino solo que en vez de llamar a sus funciones, se llaman a los envoltorios que se han creado para evitar el C++:

- ultrasonicc.c : Utilizando el sensor 'Ultrasonic Ranger' mostrado en la Figura 13, y basándonos en los ejemplos que proporciona el fabricante para este se construye un demostrador que ejecuta una medición de distancia en un bucle infinito y muestra esta medición por pantalla. Este sensor se conecta a una entrada digital de la placa y una vez conectado también a voltaje tiene una resolución de medida de 1cm permitiendo un rango de medición que va desde los 0 hasta los 4 metros de distancia como máximo.


Figura 13: Imagen de un sensor de distancia de ultrasonidos.

Basándonos en los ejemplos que proporciona el fabricante, solo hay que adaptar a las llamadas que se hacen a la librería que maneja el dispositivo para construir un demostrador adaptado a la placa y sistema operativo. Se muestra en el Código 8:

Source Code 8: Pseudocódigo de uso del wrapper.

```
#include <Ultrasonic_c.h>
int main(void) {
        Ultrasonic_t  sensor= ultrasonic_create(7);
        for (;;) {
                RangeInCentimeters =  medida_centimetros(sensor);
                printf("Centimetros: %ld \n" , RangeInCentimeters);//0~400cm
                sleep(2);
        }
        return 0;
}
```

Como se ve al principio del anterior ejemplo se ha incluido una cabecera que incluye los envoltorios de las funciones en C++(proceso explicado en el capítulo 3 de este proyecto) que proporciona el fabricante en su librería para poder utilizar estas llamadas en C. A partir de este punto, el demostrador construido es simple y similar a cualquier ejemplo que podamos encontrar de la comunidad Arduino.

- teclado.c: Previo al desarrollo de la librería del mando infrarrojo se buscó una alternativa para poder dar ordenes al micro-controlador sin tener que enchufarlo constantemente para cargar otro programa desde el host. En este caso seria un control con cable y con un pequeño teclado como el que muestro en la Figura 14.

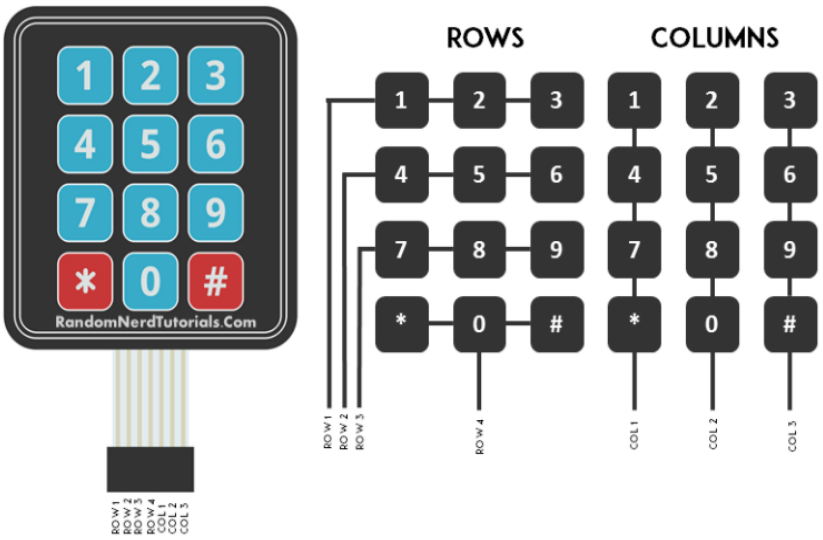



Figura 14: Imagen del Membrane switch module.

También para este dispositivo se ha desarrollado un wrapper para poder manejar las funciones de la librería original(Keypad.h) en este entorno. Una vez conseguido esto utilizando las funciones de la librería y definiendo una matriz para definir los valores de cada tecla del keypad se ha construido un demostrador que la utiliza de manera sencilla en teclado.c. En el Código 9 se ve un ejemplo del pseudocódigo.

Source Code 9: Pseudocódigo del demostrador del teclado de membrana.

```
//initialize an instance of class NewKeypad
Keypad_t customKeypad = keypad_create();

while(true)
{
        char customKey = keypad_getKey(customKeypad);

        if (customKey){
                printf("%c \n",customKey);//0~400cm
        }
}
```

## 5.3. Demostradores de librerías adaptadas

- mandoInfrarrojo.c : En el punto 4.1 se ha visto como y con que premisas se desarrolló la librería para controlar la pareja receptor/mando infrarrojo. Partiendo de este punto, se ha construido un demostrador que utiliza esta librería para leer las pulsaciones del mando infrarrojo(Figura 15).

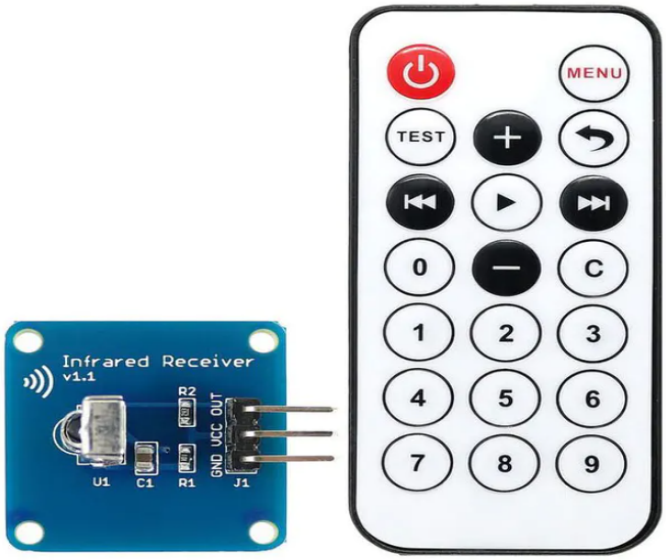



Figura 15: Imagen de la pareja receptor/mando infrarrojos.

Tal y como está desarrollada la librería, cuando se produce una pulsación deja el código de dicha pulsación en una variable global. En el programa de pruebas para saber si se ha pulsado una tecla se debe comprobar esa variable, pero si se utiliza un loop infinito que haga esta función pierde la concurrencia con otras tareas por lo que se ha decidido utilizar otra tarea periódica para esta función.

En este caso se ha configurado a este thread con un periodo de 1 segundo, es decir, detecta la última pulsación que se ha producido en el mando cada segundo en el tiempo. Se muestra en el Código 10.

Source Code 10: Pseudocódigo que se activa cuando ha habido una pulsación.

```
if(getResult(&resultado)){
        accionTecla(resultado);
}
```

El trozo de código que se ve en el Código 10 es parte de la tarea periódica que comprueba si ha habido pulsación solo activa el desencadenante que es la función 'accionTecla' si se ha depositado un resultado en la variable por lo que si no se ha pulsado ninguna tecla, el thread se suspende y no hace nada hasta su próxima ejecución programada. Para este demostrador se ha puesto un caso sencillo de desencadenante al pulsar la tecla, que es imprimir por la pantalla de la placa el código de la tecla pulsada.

- stepper.c : Para comprobar que la librería desarrollada la cual se ha explicado en la sección 5.4, se ha construido un programa de prueba el cual testa que el motor se comporta como esperábamos. Para esto se ha conectado previamente el motor a su controlador y posteriormente al micro-controlador como mostramos en la Figura 16.

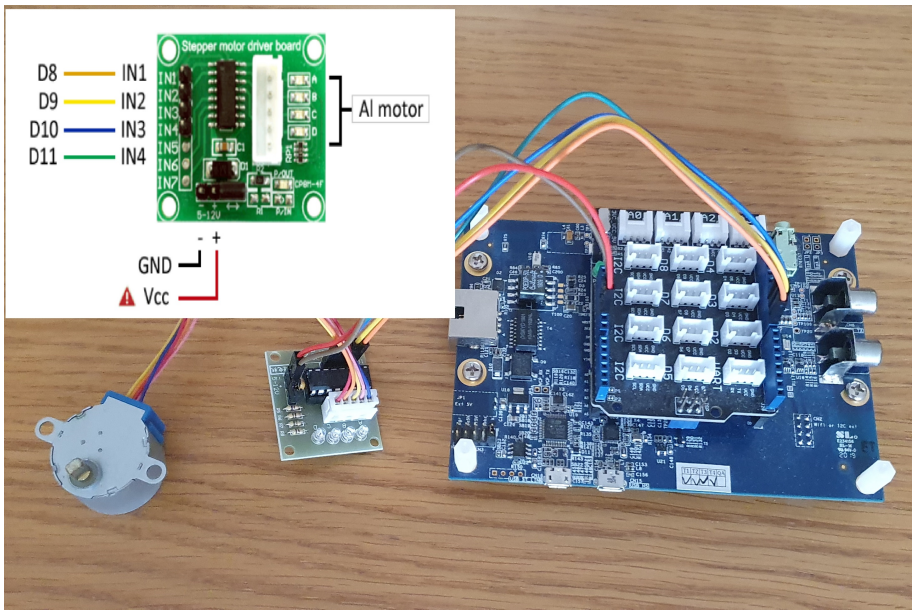



Figura 16: Esquema e imagen de las conexiones de un motor paso a paso.

- buzzer.c : En el punto 4.5 se explicó el funcionamiento de la librería para el control del zumbador pasivo. A partir de este punto se ha creado una aplicación de prueba en la que se utiliza un conjunto de frecuencias definidas para cada nota musical. También se ha definido un 'array'[13] de datos que contiene las notas necesarias para tocar una conocida melodía[25]. Podemos ver un fragmento de ambas definiciones en el Código 11.

Source Code 11: Parte de las definiciones en el demostrador del zumbador.

```
----Notas con su frecuencias predefinidas---------
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
-----------------------------
-------Array con las notas de una melodia----------
int melody[] = {

        NOTE_G4,8, NOTE_C4,8, NOTE_DS4,16, NOTE_F4,16,
        NOTE_G4,8, NOTE_C4,8, NOTE_DS4,16, NOTE_F4,16,
        NOTE_G4,8, NOTE_C4,8, NOTE_E4,16, NOTE_F4,16,
        NOTE_G4,8, NOTE_C4,8, NOTE_E4,16, NOTE_F4,16,
        NOTE_G4,-4, NOTE_C4,-4
};
```

En el programa principal del ejemplo se construye un bucle que recorre el

[13]Array es un tipo de dato estructurado. Permite guardar múltiples elementos de un mismo tipo relacionados.

anterior 'array' para dispensar la nota correspondiente de la melodía a la función de la librería del zumbador que reproduce esa nota.

- testservo.c : Para probar la librería desarrollada en el punto 4.2 se construye una sencilla aplicación que utiliza varias llamadas a la librería para mover el servo a distintas posiciones. También se utiliza la función desarrollada para realizar un barrido continuo de un ángulo a otro.

  En la Figura 17 se muestra un servomotor con sus correspondientes conexiones a la placa.

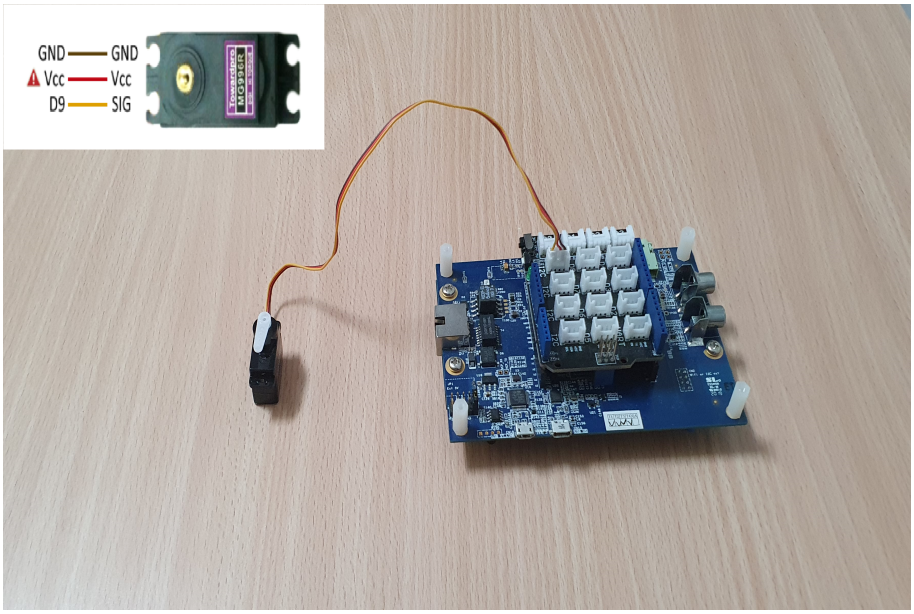



Figura 17: Esquema e imagen de las conexiones de un servoMotor.

## 5.4. Demostradores complejos

Como última fase de desarrollo en el proyecto queríamos materializar dos demostradores que pongan en práctica los objetivos que han sido explicados anteriormente. Programas mas complejos que integren varios sensores y tareas concurrentes para a su vez dar visibilidad conjunta a todas las pequeñas piezas que se han desarrollado y explicado a lo largo de toda la memoria. En el proyecto se han puesto estos dos ejemplos pero la idea es que con el marco de trabajo que se ha desarrollado (y lo que podría extenderse a posterior por alguien interesado) puedan surgir otros demostradores con otras funciones interesantes e incluso incluyendo otros sensores que no se han considerado aquí.

Antes de entrar en detalle se va a presentar el Hardware que se utiliza para ambos demostradores, el kit de Arduino "Smart Robot car kit V3.0"[26] del fabricante Elegoo. Este kit básicamente es un conjunto de motores y sensores en forma de coche sencillo. De estos sensores se utilizan un subconjunto de los que ofrece. De primeras el coche trae incluida una placa Arduino Uno[27] la cual se ha desconectado y conectado el micro-controlador STM32F769I.

Para el movimiento del coche, se utilizan los motores y la librería para controlar a

estos descrita en el apartado 4.3 que son los que vienen incluidos en el kit de Elegoo. En el frontal del coche se ha sustituido el sensor de medición de distancia basado en ultrasonidos y se ha colocado el dispositivo explicado en el apartado 5.1, dado que este sensor mide igualmente distancias pero basándose en emisión de luz.

Este ultimo sensor está colocado sobre una placa adjunta a un servomotor que permite el giro de 180 grados del sensor explicado anteriormente. Esto habilita que seamos capaces de medir la distancia no solo del frente del coche sino también los obstáculos que puedan acercarse a los laterales.

El kit también tiene integrado un receptor de infrarrojos con el objetivo de habilitar ser controlado desde un mando inalambrico. Esta pareja de receptor/mando podremos utilizarla gracias a la librería desarrollada en el apartado 4.1. Todo lo anteriormente descrito se muestra gráficamente en la Figura 18.

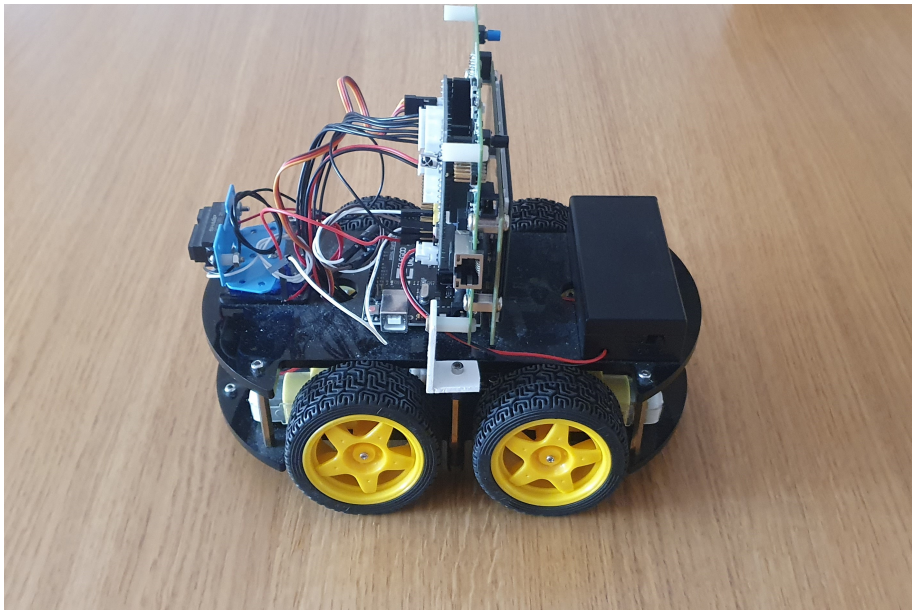


Figura 18: Foto del coche que utilizamos para los demostradores complejos.

### 5.4.1. Coche Autónomo

En este apartado se propone la construcción de un coche que tiene como objetivo el ser suficientemente autónomo como para moverse por el suelo y no chocarse con los obstáculos que le vayan surgiendo. Para esto se utiliza un conjunto de los sensores y librerías desarrollados en apartados anteriores.

La inteligencia que tenga esta demo será limitada ya que para hacer un código que contemple todos los casos para la autonomía del coche nos requeriría una gran cantidad de tiempo. Al fin y al cabo el propósito de esta demo es demostrar como en el sistema interaccionan muchos sensores a la vez en tiempo real trabajando y coordinándose con un mismo propósito, que en este caso es que el coche tenga una cierta autonomía de movimiento en el suelo.

Se detallan a continuación los componentes Hardware que participan y la función que proporciona cada uno:

- Sensor de distancia. Se utiliza el sensor SHARP explicado en la sección 5.1. Este sensor tiene el propósito de proporcionar la distancia del obstáculo mas cercano al movimiento del coche. A partir de esta información podremos decidir si se necesita reducir la velocidad, girar, etc. Para este propósito se podría proponer también la utilización del sensor de distancia basado en ultrasonidos explicado en el apartado 5.2 pero en este caso y dado que vamos a trabajar con varias tareas periódicas a la vez hemos escogido el que mide distancias utilizando un láser ya que al tener que hacer mediciones constantemente el tiempo de bloqueo que se obtiene en cada medida nos interesa que sea el menor posible para que tenga el mínimo impacto de bloqueo en las otras tareas que comparten el sistema.

- Servomotor. Este dispositivo (explicado en la sección 4.2) sostiene en su eje el sensor explicado anteriormente. De esta manera se tiene la posibilidad de elegir la dirección en la que mide la distancia respecto del coche otorgando una mayor información del entorno. Una función interesante para usar en este demostrador es la de 'barrido', con ella se consigue un movimiento constante de lado a lado permitiendo al sensor de distancia medir tanto el frente del coche como los laterales.

- Motores. Con la librería desarrollada que se explica en el apartado 4.3 se obtiene el control de los motores de las ruedas del coche. Pudiendo modificar la dirección y velocidad con la que el coche se mueve.

- Buzzer pasivo. Este dispositivo se ha añadió a la demo para tener una demostración muy clara de la concurrencia con la que conviven varias tareas en el sistema. Utilizando la librería generada en este proyecto explicada en el apartado 4.5 se reproduce una melodía conocida mientras el coche va en movimiento. Escuchar una melodía generada por pulsos que controla una tarea periódica y ver que no se distorsiona demuestra que las tareas en el sistema cumplen el plazo de periodos estimulado.

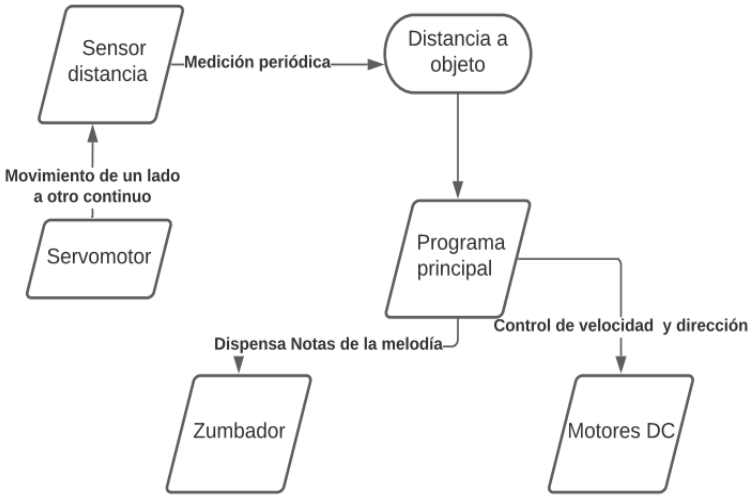



Figura 19: Esquema de los componentes en el coche autónomo.

El funcionamiento general de este programa es el siguiente(representado también en la Figura 19). El servomotor se coloca en una posición orientada hacia delante para que el coche avance en dirección recta mientras medimos la distancia en ese sentido. Mientras no detecte un obstáculo a una distancia suficientemente cercana para que el coche colisione seguimos esta dirección. Cuando esto ocurra, el servomotor ejecutara un barrido de $180^{o}$ (desde el lado izquierdo del coche hasta el derecho) mientras el sensor toma varias medidas para encontrar la mejor ruta. Entonces el coche girara hacia el lado en el que el obstáculo mas cercano este mas lejos y volverá a moverse en esa dirección hasta el próximo obstáculo.

Una añadido que se ha hecho es la variación de velocidad del coche dependiendo de la distancia hasta el objeto. Si no hay nada cerca el coche ira a la máxima velocidad, una vez se vaya acercando a algo, irá disminuyendo.

### 5.4.2. Coche controlado por IR

Para este demostrador se utiliza también el coche utilizado en la sub-sección anterior pero en vez de utilizar los sensores para que se auto-dirija se incluyen el mando y receptor infrarrojos explicados en el apartado 5.1 para dar ordenes en tiempo real al hardware. La lógica del programa será muy simple, con el mando infrarrojo se tiene la posibilidad de dirigir el movimiento del coche, de pararlo, cambiar su velocidad, medir distancias,etc. Adicionalmente este demostrador incluye una funcionalidad de emergencia que evita que el coche sufra daños haciendo que el sensor de distancia detecte un obstáculo que esté tan cerca como para chocar con el coche lo detenga por completo.

Por tanto en este ejemplo existen dos tareas periódicas ejecutando en paralelo en el sistema.

- La primera será una tarea que compruebe periódicamente si el usuario ha pulsado una tecla del mando infrarrojo(utilizando la librería correspondiente), si es así el programa analizará el código que ha retornado la tecla pulsada y ejecutará la acción que se haya decidido para esa tecla.
- La segunda será la tarea que hace una llamada periódica al sensor de distancia para comprobar que el coche no se va a chocar contra ningún obstáculo al frente.

Finalmente en la fase de pruebas de este demostrador se ha detectado un nuevo conflicto entre Arduino y MaRTE OS. Concretamente entre las interrupciones(por parte de Arduino) que se utilizan en la librería del mando infrarrojo y las interrupciones que lanza el SO para hacer la gestión de cambio de contexto entre unas tareas del sistema y otras. Esto hace que en ocasiones en este demostrador el sistema colapse y deje de funcionar, por lo que seria un interesante punto de mejora para una linea de trabajo futura.

# 6. Conclusiones y trabajo futuro

## 6.1. Conclusiones

Este proyecto ha tenido como objetivo asentar un marco de trabajo para futuros desarrollos que tengan como objetivo desarrollar programas en MaRTE OS utilizando las facilidades que ofrece el mundo de Arduino.

Un ejemplo de salida práctica para este proyecto es el poder utilizarlo como herramienta de aprendizaje teniendo un entorno en el que poder desarrollar proyectos software/hardware en un hipotético laboratorio de practicas con el que demostrar una concurrencia de tareas periódicas en un mismo sistema de tiempo real, pudiendo demostrar los casos teóricos de una manera visual.

El proyecto comenzó con unos objetivos mas centrados en la configuración y adaptación de la librería stm32duino en un desarrollo cruzado con MaRTE OS utilizando como entorno de desarrollo el IDE GPS y a partir de ese punto probar y adaptar si fuese necesario librerías de común uso en el hardware de Arduino. Esto ya supondría una notable ventaja al querer hacer uso de dispositivos tales como podrían ser un sensor de ultrasonidos o un servomotor al utilizar librerías ya desarrolladas por la comunidad Arduino.

Pero a lo largo del desarrollo nos hemos encontrado con varias dificultades para cumplir todos los objetivos unicamente con la adaptación de la librería, como la problemática con la mezcla de lenguajes entre el C de MaRTE OS y el C++ de Arduino.

Aunque sin duda el mayor problema encontrado a lo largo del proyecto ha sido la incompatibilidad de los HWTimers de Arduino ya que interfieren con los de MaRTE OS. Debido a este problema, se ha optado por desarrollar nuevas librerías para MaRTE OS que permitan el manejo del hardware que requiera el uso de timers. Asimismo, algunas de las librerías desarrolladas incorporan nuevas funcionalidades respecto a las originales utilizadas en Arduino, como en el caso de la librería para controlar el Mando IR.

Esto ultimo mencionado es probablemente la mayor contribución de este trabajo dado que además de proporcionar librerías funcionales para hardware muy común en Arduino también se muestra una forma de abordar la generación de pulsos en un pin sin utilizar los 'hardware timers' y hacerlo además de forma no bloqueante para el sistema.

Finalmente se han desarrollado demostradores para cada uno de los sensores que se han adaptado, de medición de tiempos de ejecución y por ultimo dos demostradores que utilizan un subconjunto de sensores y dispositivos para poner en practica los objetivos marcados en el TFG. Tanto el código de los demostradores y librerías desarrollados como la integración de la librería STMduino en MaRTE OS está disponible para su visualización y descarga en el gitlab creado específicamente para este proyecto `https://gitlab.com/alexvejo/tfg-alejandro-fernandez`. En el Anexo D se puede encontrar mas información acerca de la estructura del repositorio.

## 6.2. Observaciones y lineas de trabajo futuro

Exponemos también unas cuantas propuestas de mejora y/o expansión de este proyecto con el objetivo de que alguien esté interesado en continuar, ampliar y optimizarlo. Bajo mi punto de vista estos son los puntos mas interesantes para continuar con este trabajo:

- Mejora de las librerías desarrolladas para permitir el control de dos o mas dispositivos del mismo tipo. Las librerías desarrolladas, como por ejemplo la del servomotor o el mando infrarrojo, actualmente están limitadas a controlar un único dispositivo. Una linea de trabajo interesante seria generalizar el concepto, modificar y optimizar el código para que no sea limitante en este sentido, permitiendo el control de múltiples dispositivos en el mismo demostrador.

- Incluir C++ en el entorno de compilación cruzada. Un modo de evitar el trabajo que se realiza desarrollando wrappers para las librerías de Arduino consistiría en una linea de trabajo centrada en incluir los fuentes de C++ en el GNAT Studio e integrarlo también con MaRTE OS para permitir la compilación directa de todos los 'headers' .cpp que componen el núcleo y librerías periféricas de Arduino.

- La investigación de la raíz del problema que existe entre MaRTE OS y Arduino para poder utilizar los Hardware Timers. Resolver este problema nos brindaría la oportunidad de trabajar con todas las librerías que hacen uso de esta herramienta en el ecosistema de Arduino sin tener que desarrollar las nuestras propias, con el ahorro de trabajo y el abanico de posibilidades que esta comunidad ofrece.

- Investigación mas profunda de la problemática que surge al utilizar muchas interrupciones en pines Arduino mientras en MaRTE OS ejecutan múltiples tareas periódicas. Este problema no surge siempre de una manera clara, se manifiesta a veces(como en el demostrador del punto 5.4.2) pero no siempre en el mismo momento ni en la misma situación, por lo que requiere de una futura linea de trabajo para determinar más concretamente la raíz del problema.

# Referencias


[1] *MaRTE OS.* URL: https://marte.unican.es/.

[2] Mario Aldea Rivas y Michael González Harbour. "MaRTE OS: An Ada Kernel for Real-Time Embedded Applications". En: (2001).

[3] *Repositorio del conjunto de librerias stm32duino.* URL: https://github.com/stm32duino/Arduino_Core_STM32.

[4] *Documentacion ofcial de los procesadores ARM.* URL: https://developer.arm.com/documentation/.

[5] *STM32F769.* URL: https://www.st.com/en/evaluation-tools/32f769idiscovery.html.

[6] *STM32cubeprogrammer official site.* URL: https://www.st.com/en/development-tools/stm32cubeprog.html.

[7] *Definición de compilación.* URL: https://developer.arm.com/documentation/.

[8] *GNAT Studio official website.* URL: https://www.adacore.com/gnatpro/toolsuite/gnatstudio.

[9] *Arduino and Arduino IDE official website.* URL: https://www.arduino.cc/en/software.

[10] *Estandar POSIX.* URL: https://www.istr.unican.es/publications/mgh-1993b.pdf.

[11] *Example of build wrapper in C.* URL: https://nachtimwald.com/2017/08/18/wrapping-c-objects-in-c/.

[12] *usleep(3) — Linux manual page.* URL: https://man7.org/linux/man-pages/man3/usleep.3.html.

[13] *Descripcion y funcionamiento de los Timers Hardware en Arduino.* URL: https://www.prometec.net/timers/.

[14] *Pulse Width Modulation in Arduino.* URL: https://www.arduino.cc/en/Tutorial/Foundations/PWM.

[15] *Descripcion y funcionamiento del protocolo NEC.* URL: https://techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol.

[16] *Tarea periódica en un Sistema de Tiempo Real.* URL: https://www.istr.unican.es/asignaturas/STR/apuntes/Tema%2001%20-%20Int.pdf.

[17] *Arduino Interrupt official reference.* URL: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/.

[18] *Controlar un servo con arduino.* URL: https://www.luisllamas.es/controlar-un-servo-con-arduino/.

[19] *Linux nanosleep man page.* URL: https://man7.org/linux/man-pages/man2/nanosleep.2.html.

[20] *Definition of concurrency.* URL: https://www.geeksforgeeks.org/concurrency-in-operating-system/.

[21] *Controlar un motor de corriente continua con Arduino.* URL: https://www.luisllamas.es/arduino-motor-corriente-continua-l298n/.

[22] *Controlar un motor paso a paso con Arduino.* URL: https://www.luisllamas.es/motor-paso-paso-28byj-48-arduino-driver-uln2003/.

[23] *Use a buzzer module.* URL: https://create.arduino.cc/projecthub/SURYATEJA/use-a-buzzer-module-piezo-speaker-using-arduino-uno-89df45.

[24] *Uso de un módulo de medición de distancia SHARP.* URL: https://naylampmechatronics.com/blog/55_tutorial-sensor-de-distancia-sharp.html.

[25] *Melody examples in Arduino system.* URL: https://github.com/robsoncouto/arduino-songs.

[26] *ELEGOO SMART ROBOT CAR KIT V3.0.* URL: https://www.elegoo.com/blogs/arduino-projects/elegoo-smart-robot-car-kit-v3-0-plus-v3-0-v2-0-tutorial.

[27] *Documentacion oficial de Arduino Uno.* URL: https://arduino.cl/arduino-uno/.

# A. Prototipos de las funciones en la librería del mando IR

```
//Inicialización de la librería
//@param pin - Pin en el que está conectado el receptor
void stmMaRTE_IRremote_init(int pin);

//Devuelve un valor solo si se ha pulsado una tecla.
//@return - código de la tecla pulsada, sino 0.
int stmMaRTE_IRremote_getResult();

//códigos mando IR
#define KEY_OK 2
#define KEY_LEFT 34
#define KEY_RIGHT 194
#define KEY_DOWN 168
#define KEY_UP 98
#define KEY_ASTERISC 66
#define KEY_PAD 82
#define KEY_0 74
#define KEY_1 104
#define KEY_2 152
#define KEY_3 176
#define KEY_4 48
#define KEY_5 24
#define KEY_6 122
#define KEY_7 16
#define KEY_8 56
#define KEY_9 90
```

# B. Prototipos de las funciones en la librería de los motores DC

```
//Funciones de movimiento
extern void stmMaRTE_motors_forward();
extern void stmMaRTE_motors_back();
extern void stmMaRTE_motors_left();
extern void stmMaRTE_motors_right();
extern void stmMaRTE_motors_stop();

//Inicializa la librería
//@param prio - valor de la prioridad del thread
//@param tarea - si es 0 no se crea el thread
extern void stmMaRTE_motors_init(int prio,int tarea);

//Cambia la velocidad de movimiento
//@param velocidad - valor de la velocidad del motor
extern void stmMaRTE_motors_changeSpeed(int velocidad);
```

# C. Prototipos de las funciones en la librería del zumbador

```
//Reproduce una unica nota
//@param pin - pin al que se conecta el zumbador
//@param frequency - frecuencia del tono
//@param duration - duracion del tono
void stmMaRTE_buzzer_playTone(int pin, int frequency, int duration);

//Reproduce el array de notas preconfigurado en la librería
void stmMaRTE_buzzer_playMelody();
```

# D. Repositorio gitlab del proyecto

Este anexo es una aclaración para las personas que quieran leer, probar o modificar el código de este proyecto. Dado que todo lo desarrollado está dentro de MaRTE OS es importante proporcionar las ubicaciones de las librerías creadas, las partes de la librería stm32duino modificadas o los demostradores generados.

Tanto las nuevas librerías descritas en la sección 4, como los demostradores de la sección 5 se encuentran en la ruta 'examples/STM32F769I-Discovery'. Las nuevas librerías se rigen bajo la nomenclatura 'stmMaRTE_xxxx'.

Las cabeceras de la librería stm32duino están en 'stm32f_arch/hwi/stm32duino'. Dentro de este directorio se encuentra el script de compilación descrito en la sección 3 junto con código añadido y/o modificado a la librería.

En el directorio 'marte_modified_files' están los cambios descritos en el apartado 3.1.

En el directorio 'marte_added_files' están los wrappers descritos en la sección 3.