Facultad
de
Ciencias

Entorno de desarrollo para la realización de prácticas de sistemas empotrados basado en Raspberry Pi

Development environment for realization of embedded systems practices based on Raspberry Pi

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Gabriel Via Echezarreta

Director: Mario Aldea Rivas

Co-Director: Héctor Pérez Tijero

Octubre - 2017

INDICE GENERAL

INDICE DE FIGURAS

RESUMEN

El proyecto consiste, como indica el título, en el desarrollo de un entorno que permita realizar prácticas de sistemas empotrados utilizando la Raspberry Pi Zero.

Para crear este entorno se han satisfecho tres objetivos principales: la implementación de un entorno de desarrollo cruzado, la creación de una librería que permita la ampliación de las capacidades de entrada/salida de la Raspberry Pi y la implementación de un demostrador que sirva como ejemplo de aplicación que se puede desarrollar con este nuevo entorno.

Antes de la realización de este proyecto, la carga de la aplicación en la Raspberry Pi era un proceso tedioso. Primero, había que compilar dicha aplicación, segundo, extraer la memoria sd de la Raspberry y conectarla al pc y por último desconectar la memoria sd del pc y volver a conectarla a la Raspberry. Por ello el objetivo principal para el entorno de desarrollo consiste en el estudio de los diferentes entornos para la carga remota y la elección del más adecuado y su puesta en funcionamiento.

Por otra parte, otro objetivo tratado en este proyecto es la creación de una librería que sirva de interfaz para el desarrollo de aplicaciones que hagan uso del Explorer Phat. El Explorer Phat es una placa que se conecta al GPIO (General Purpose Input/Output) de la Raspberry y aporta algunas funcionalidades a la misma, como la posibilidad de tener entradas analógicas o el control de motores mediante pulsos (PWM).

Por último, se implementa un demostrador que simula una fábrica en la que una cinta transportadora mueve un objeto y se detiene al llegar a la altura de un brazo mecánico que recoge el objeto y lo suelta detrás suyo.

ABSTRACT

The project consists, as indicated in the title, in the development of an environment that allows to practice embedded systems using Raspberry Pi Zero.

To create this environment, three main objectives have been fulfilled: the implementation of a cross-development environment, the creation of a library that allows the expansion of input / output capabilities of the Raspberry Pi and the implementation of an example demonstrator application that can be developed with this new environment.

Before carrying out this project, loading the application into the Raspberry Pi was a tedious process. First, you had to compile that application, second, extract the SD memory from the Raspberry and connect it to the PC and finally disconnect the SD memory from the PC and reconnect it to the Raspberry. For this reason the main objective for the development environment consists of the study of the different environments for the remote load and the choice of the most suitable one and its putting into operation.

On the other hand, another objective addressed in this project is the creation of a library that serves as interface for the development of applications that make use of Explorer Phat. The Explorer Phat is a board that connects to the GPU (General Purpose Input / Output) of the Raspberry and provides some functionality to it, such as the possibility of having analog inputs or the control of motors using pulses (PWM).

Finally, a demonstrator is implemented that simulates a factory in which a conveyor belt moves an object and stops when it reaches the height of a mechanical arm that picks up the object and looses it behind it.

PALABRAS CLAVE

- Raspberry Pi Zero
- MaRTE OS
- GPIO
- Explorer Phat
- Tiempo real
- Sistema empotrado
- Entorno de desarrollo
- Prácticas de laboratorio

1 INTRODUCCIÓN
1.1 Motivación

La evolución de la tecnología es un hecho constante y en los últimos años el ser humano ha alcanzado a desarrollar cosas impensables no hace mucho tiempo. El primer transistor, pieza fundamental en los microprocesadores, fue creado en el año 1947 por los científicos William Bradford Shockley, John Bardeen y Walter Houser Brattain y medía varios centímetros. Hoy en día los transistores miden del orden de nanómetros y debido a esto el tamaño y peso de los computadores se ha visto reducido enormemente.

Este avance tecnológico ha hecho posible la aparición en el mercado de los microcomputadores, computadores de tamaño muy reducido que se han popularizado debido a su tamaño y sobre todo su bajo coste. Uno de los más populares es la Raspberry Pi. Desarrollada por la Raspberry Pi Foundation se ha convertido en el microcomputador más popular gracias a la gran comunidad de usuarios que posee. Creada con propósito educativo hoy en día es usada con múltiples propósitos distintos aparte del educativo como por ejemplo media center en el hogar, ambientación led, estación meteorológica, robótica, etc.

Esto es debido a que posee un conjunto de pines de entrada y salida llamado GPIO(General Purpose Input/Output). Gracias al GPIO aparecen placas de extensión que conectadas al mismo permiten expandir la funcionalidad de la Raspberry PI con convertidores analógico a digital, digital a analógico o el control de motores. Un ejemplo de estas placas es el Explorer PHat de Pimoroni usado en este proyecto.

Por otro lado, normalmente, todas estas aplicaciones corren bajo el sistema operativo Raspbian. Basado en Debian es el sistema operativo oficial para Raspberry, pero no es el único, hasta Windows 10 tiene su versión entre otros.

Hoy en día existen varias versiones de Raspberry Pi, cada una de ellas con diferentes características. Entre los modelos más nuevos destaca la Raspberry Pi Zero, el modelo más simple, pequeño y barato, pero no por ello menos potente. Por otro lado, posee el mismo chipset que la Raspberry Pi 1, punto de partida de este proyecto, y por ello son compatibles.

La docencia de sistemas empotrados y tiempo real se realiza de una forma más formativa y amena para los alumnos si se puede realizar a base de prácticas. Para poder disponer de un laboratorio de sistemas empotrados y de tiempo real es necesario disponer de un sistema operativo de tiempo real, un entorno de desarrollo cruzado y dispositivos que faciliten la realización de prácticas atractivas para los alumnos como pueden ser las prácticas de robótica.

Los sistemas embebidos o empotrados necesitan de un entorno de desarrollo cruzado para el desarrollo de aplicaciones que corran sobre ellos.

Para ello se hace uso de compiladores cruzados, compiladores que son capaces de generar código ejecutable para una máquina distinta a la que se ejecuta. Además, un sistema de desarrollo cruzado necesita de un cargador que permita la carga remota de aplicaciones y otras herramientas como un depurador de código.

También es necesario un sistema operativo de tiempo real que no es otra cosa que un sistema operativo capaz de satisfacer requisitos de tiempo real y dar soporte para aplicaciones de tiempo real.

El sistema operativo MaRTE OS del grupo de Ingeniería Software y Tiempo Real de la Universidad de Cantabria es un sistema operativo de tiempo real pensado para aplicaciones embebidas. Actualmente posee soporte para aplicaciones para la Raspberry Pi 1 y constituye uno de los puntos de partida de este proyecto.

1.2 Objetivos

El objetivo principal de este Trabajo de Fin de Grado es el desarrollo de un entorno que permita el uso de la Raspberry Pi Zero (y la Raspberry Pi 1 modelo B) en el laboratorio para la docencia de sistemas empotrados y de tiempo real.

En primer lugar, se buscará adaptar a la Raspberry Pi Zero la versión ya existente del sistema operativo MaRTE OS para Raspberry Pi 1. En particular se estudiará el proceso de carga de aplicaciones y el funcionamiento de los periféricos de salida (HDMI y línea serie).

En segundo lugar, se tratará un elemento muy importante para el desarrollo cruzado: un sistema de carga remota de aplicaciones.

Un sistema de carga remota de aplicaciones es necesario porque el proceso de carga de una aplicación en la Raspberry es bastante tedioso. Siendo el objetivo principal del proyecto su uso en docencia, los alumnos necesitan una manera de agilizar el proceso de carga para testear sus aplicaciones.

En tercer lugar, se pretender ampliar las capacidades de entrada/salida de la Raspberry. Para ello se utilizará el Explorer Phat, placa de extensión que se conecta al GPIO de la Raspberry PI y que expande su funcionalidad añadiendo entradas analógicas y digitales, salidas digitales y control para dos motores. Todos estos elementos necesarios para el desarrollo de aplicaciones robóticas de cierta complejidad. Como se pretende utilizar el sistema operativo MaRTE será necesario el desarrollo de una librería que permita el fácil uso de este dispositivo para aplicaciones ejecutadas sobre el citado sistema operativo.

Como último objetivo se planteó la creación de un demostrador de la tecnología desarrollada que permita validar el entorno desarrollado y

proporcionar un ejemplo del tipo de sistemas que se pueden llegar a desarrollar.

1.3 Metodología

La metodología empleada en el desarrollo de este proyecto consistió en el desarrollo en cascada para cada objetivo por separado.

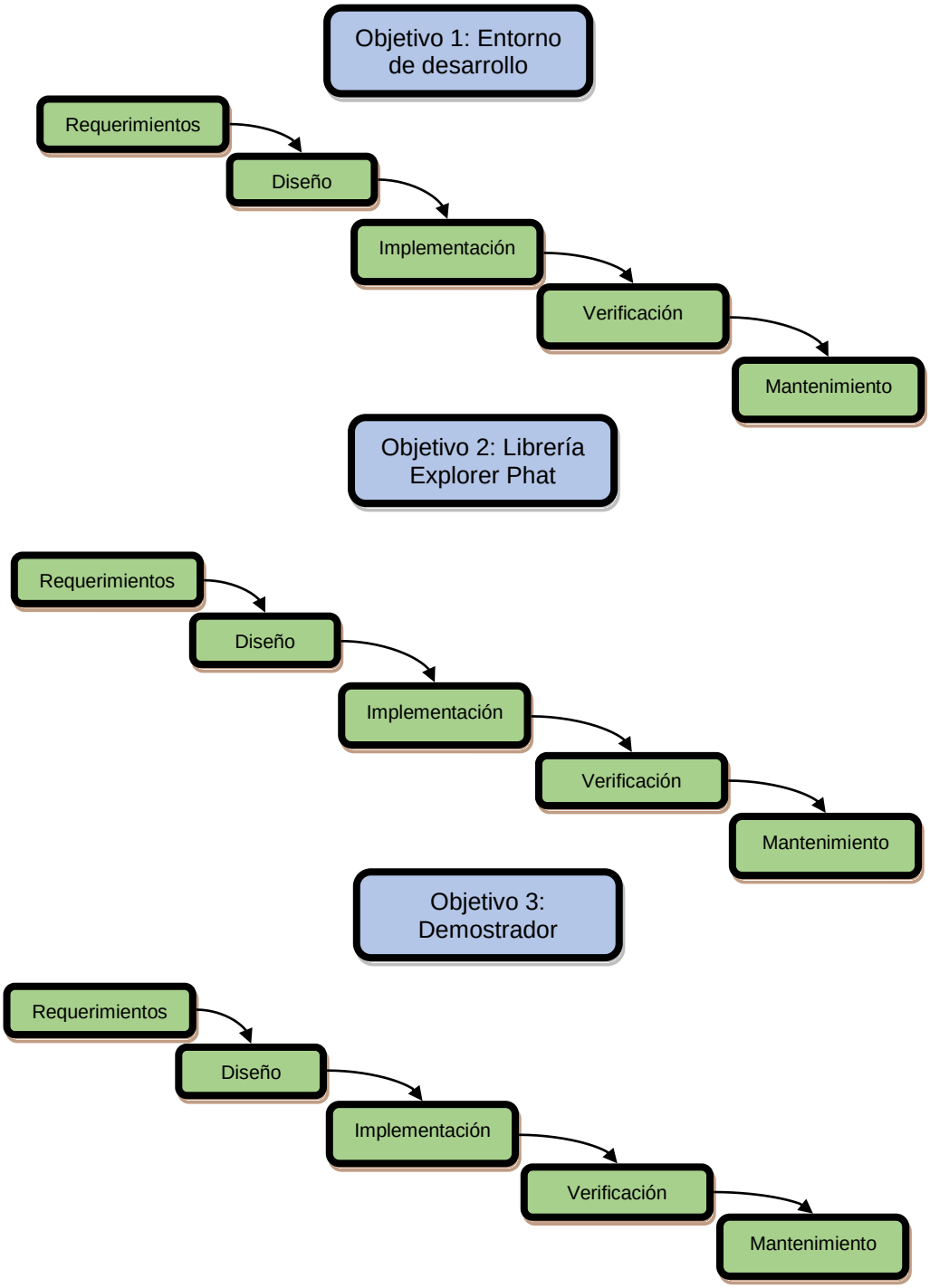

Figura 1: Ejemplo de desarrollo en cascada

El desarrollo en cascada [1] es el enfoque metodológico que ordena rigurosamente las etapas del proceso de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. Al final de cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase. De esta forma, cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo. La palabra cascada sugiere, mediante la metáfora de la fuerza de la gravedad, el esfuerzo necesario para introducir un cambio en las fases más avanzadas de un proyecto.

2 HERRAMIENTAS Y TECNOLOGÍAS UTILIZADAS

2.1 MaRTE OS

El sistema operativo *MaRTE OS* [2][3] es un sistema operativo de tiempo real estricto para aplicaciones embebidas que implementa el subconjunto de Minimal Real-Time POSIX.13. Su principal característica es que proporciona un entorno controlado y sencillo de usar para el desarrollo de aplicaciones de tiempo real multi-thread. Sus principales características son:

- Soporta aplicaciones desarrolladas en los lenguajes de programación Ada, C y pudiendo mezclar entre ellos
- Soporta las siguientes arquitecturas: x86 (32 bits), Raspberry Pi y Linux.
- Soporte de tareas Ada completo.
- Ofrece los servicios definidos en POSIX.13. Como la creación de hilos, mutex, relojes y temporizadores.
- Todos los servicios tienen un tiempo de respuesta acotado.
- Un único espacio de direcciones de memoria compartido por aplicaciones multi thread y el sistema operativo.
- Disponible bajo la licencia GNU General Public License 2.
- Basado en el conjunto de herramientas AdaCore GNU.
- Implementa la mayoría del apéndice de tiempo real Ada2012.

2.2 Raspberry Pi Zero

Para el desarrollo de este proyecto se ha utilizado el computador Raspberry Pi Zero [4]. Se trata de un computador de placa simple (SBC) el cual destaca por su tamaño (65 x 30 mm) y su precio (5 dólares).

Figura 2: Imagen de una Raspberry Pi Zero

La Raspberry Pi Zero hace uso del system-on-a-chip Broadcom BCM2835 el cual incluye un procesador central ARM1176JZF-S a 1GHz, procesador gráfico VideoCore IV y 512 MB de RAM.

También cómo se puede apreciar en la Figura 1.1, la placa posee dos puertos micro USB, uno de energía y otro de datos. Debido a su pequeño tamaño utiliza como salida de video y de audio un puerto mini-HMDI. Para el almacenamiento interno posee una ranura para tarjetas microSD. También posee el GPIO de los modelos más recientes de 40 pines.

En algunas partes del proyecto también su utilizó la Raspberry Pi 1 modelo B debido a que posee el mismo system-on-a-chip (aunque con su procesador a 700 MHz) pero también posee un conector Ethernet (RJ-45).

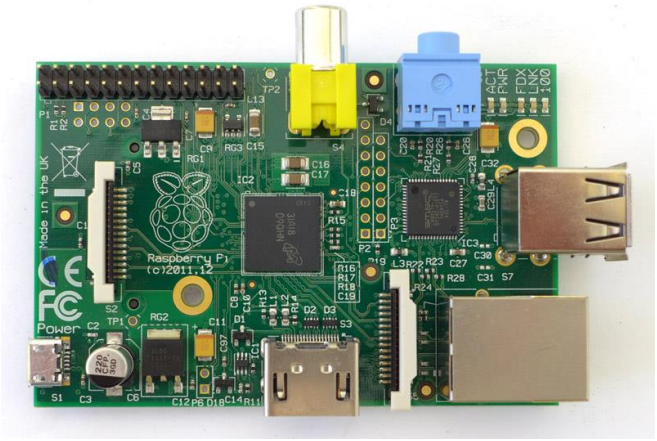

Figura 3: Imagen de una Raspberry Pi 1 modelo B

2.3 Explorer Phat

El Explorer Phat [5] es una placa diseñada para ser conectada a los pines del GPIO de los modelos de Raspberry Pi de 40 pines, y que expande la funcionalidad de la Raspberry.

Fabricado por la empresa Pimoroni el Explorer Phat proporciona cuatro entradas analógicas y otras cuatro entradas digitales, cuatro salidas digitales y dos controladores H-bridge para motores que permiten moverles bidireccionalmente. Además, tiene dos tomas de 5V y dos tomas de tierra.

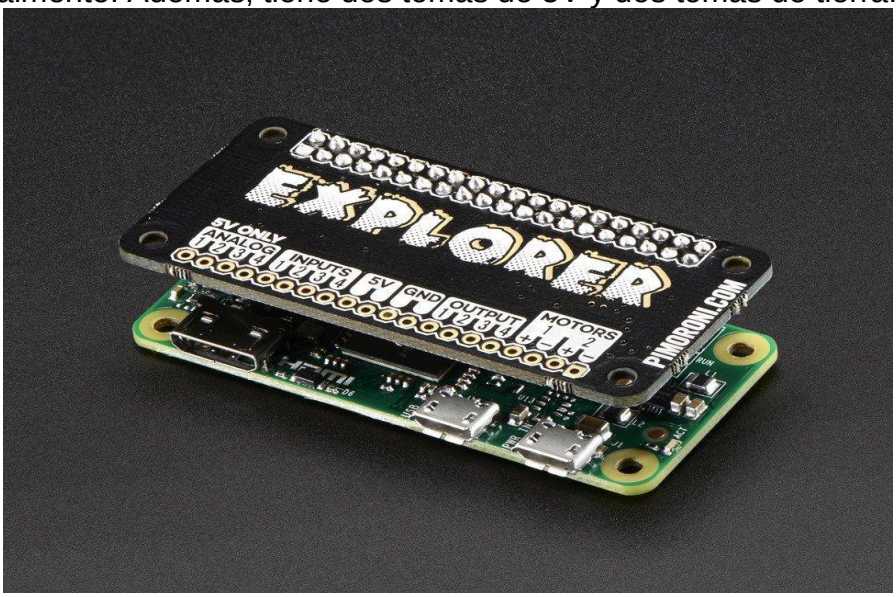

Figura 4: Imagen del Explorer Phat conectado a la Raspberry Pi Zero

Por otro lado, Pimoroni dispone de una API escrita en Python para todos los usuarios de Raspbian que permite el fácil uso de todos los modelos de Explorer.

Su uso principal es el de la robótica, ya que con sus prestaciones es sencillo el montaje de un pequeño robot o de circuitos con sensores aprovechando sus entradas analógicas.

2.4 Micro Metal Gearmotor

Motor fabricado por Pimoroni cuyo uso está pensado junto al Explorer Phat. Funciona con corriente continua y la versión utilizada en el proyecto es el Micro Metal Gearmotor with Push Header Shim 50:1.

Para regular su velocidad se ha utilizado la simulación de pulsos PWM [6] mediante software. Esto consiste en la variación de la energía que recibe el motor durante un ciclo determinado. Cuanto más tiempo recibe el motor corriente durante el ciclo más rápido se mueve, siendo recibir todo el ciclo corriente la velocidad máxima. La proporción de ciclo en el que la señal es positiva es conocida como Duty cycle.

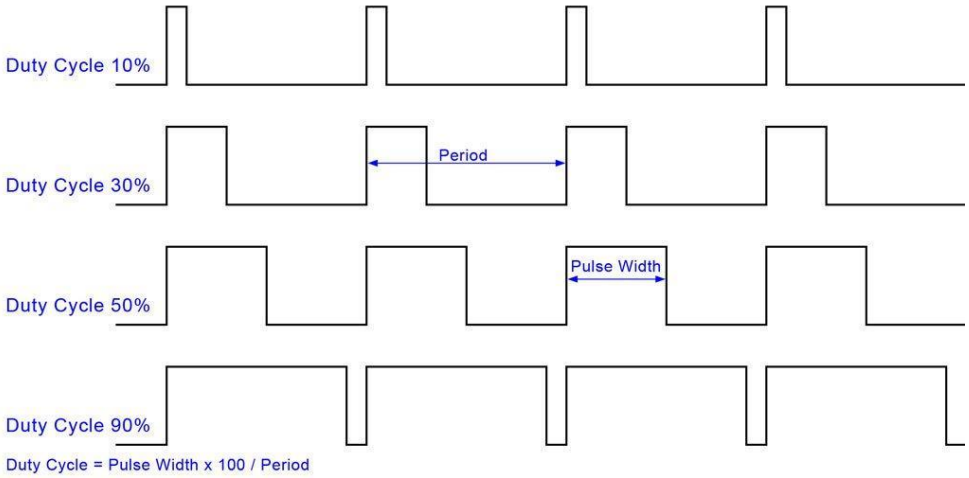

Figura 5: Esquema de modulación por ancho de pulso

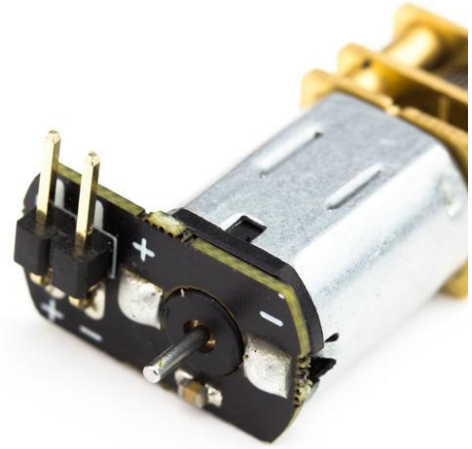

Figura 6: Imagen del Micro Metal Gearmotor junto con el Push Header Shim para su conexión al Explorer Phat

En este proyecto, el Micro Metal Gearmotor será utilizado en el demostrador que se describe en el capítulo 5.

2.5 Hitec HS-422 servo

El HS-422 es un servomotor [7] fabricado por la empresa Hitec. Los servos son dispositivos equivalentes a motores de corriente continua, con la diferencia de que se ubican y mantienen estables en una posición dentro de su rango de acción.

Figura 7: Imagen de un servomotor HS-422

Estos motores hacen uso de los pulsos PWM para controlar su posición. Trabajan a una frecuencia de 50 Hz por lo que tienen un periodo de 20 ms. Según el ancho de pulso dentro de ese periodo el motor se coloca en una posición o en otra.

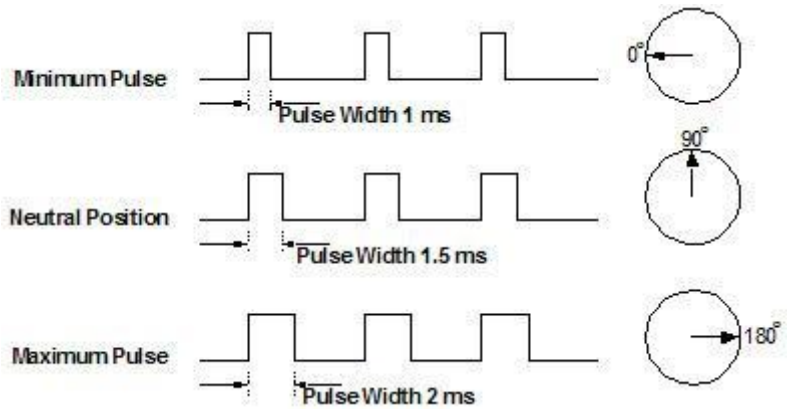

Figura 8: Ejemplo del movimiento de un servomotor según el ancho de pulso.

2.6 Sensor de distancia Sharp GP2Y0A21YK

Para el desarrollo del demostrador se ha utilizado el sensor de distancia Sharp GP2Y0A21YK. Funciona por infrarrojos y proporciona un valor analógico según la distancia a la que se encuentre el objeto.

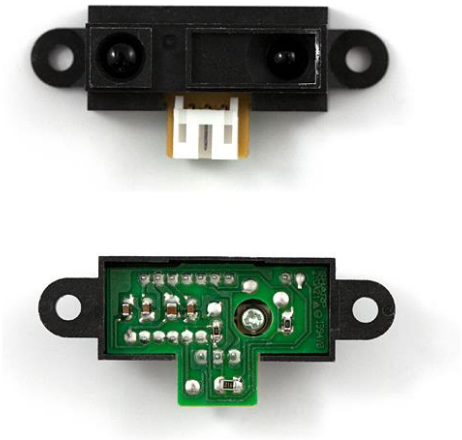

Figura 9: Sensor de distancia Sharp GP2Y0A21YK

2.7 Librería de BCM2835

Para este proyecto se ha utilizado la librería del chipset BCM2835 adaptado a MaRTE Os. Se trata de una librería escrita en C que nos permite una fácil interacción con el GPIO de la Raspberry Pi y que fue adaptada a MaRTE OS en un trabajo de fin de carrera anterior [8].

También incluye las funciones para la utilización del bus I2C [9] para la comunicación con dispositivos a través de dicho bus. En este proyecto se utilizará para el control del GPIO y la comunicación a través del bus I2C con un sensor analógico conectado al Explorer Phat.

2.8 Librería Soft_PWM

Librería proporcionada en el repositorio oficial de Explorer Hat [10], escrita en C, que se utiliza para la simulación de pulsos PWM mediante software. En este proyecto será adaptada para su correcto funcionamiento junto a MaRTE OS usando para ello la librería de BCM2835 para el control de los pines del GPIO asociados a los pines del motor conectado en el Explorer Phat.

2.9 Qemu

Qemu [11] es una herramienta la cual nos permite la creación de máquinas virtuales dentro del sistema operativo. Este emulador nos permite seleccionar el procesador de nuestro computador emulado por lo que es muy útil para simular una Raspberry Pi ya que hacen uso de un procesador ARM.

La limitación de Qemu respecto a su uso como simulador de Raspberry Pi es que no es capaz de simular el chipset BCM2835 completo por lo que el GPIO no está disponible.

2.10 U-boot

Das U-boot normalmente abreviado a U-boot es un bootloader [12] normalmente usado en sistemas embebidos. Permite la carga de un kernel de distintas maneras, entre ellas por red, utilizando TFTP, por comunicación serie, desde un dispositivo de almacenamiento o desde la misma SD en la que esté cargado U-boot.

Permite la configuración de un script automático que agilice el proceso de carga y que se ejecute automáticamente una vez iniciado U-boot.

En este proyecto se estudiará su funcionamiento y configuración para la utilización de todos sus distintos métodos de carga remota.

3 ENTORNO DE DESARROLLO PARA RASPBERRY PI 1 Y ZERO

En este capítulo se describe el entorno de desarrollo para Raspberry Pi 1 existente al comienzo de nuestro proyecto y se presentan las mejoras realizadas a dicho entorno. Dichas mejoras se centran en la simplificación de la carga de la aplicación desde una tarjeta SD y en la posibilidad de realizar la carga de la aplicación de forma remota.

3.1 Situación inicial del entorno de desarrollo para Raspberry Pi 1 y Zero

Para la realización del proyecto se comenzó por el estudio y comprensión del estado inicial del entorno de desarrollo existente para Raspberry Pi 1 y MaRTE OS.

MaRTE Os utiliza como computador de desarrollo un sistema Linux (en nuestro caso utilizamos la distribución Ubuntu). El compilador cruzado utilizado es AdaCore GNAT compiler [13] el cual permite compilar código C, Ada y ensamblador para procesadores ARM.

En el entorno de desarrollo original [14], la carga del ejecutable en la Raspberry Pi se realizaba siguiendo un proceso muy complejo. El primer paso consistía en la instalación de Raspbian con el cargador NOOBS [15], proceso que, aunque solo era necesario hacerlo la primera vez, podía demorarse hasta una hora. Una vez instalado Raspbian era necesario la extracción de la tarjeta SD de la Raspberry, su conexión al computador de desarrollo y la sustitución del kernel.img por el generado en la compilación de nuestra aplicación.

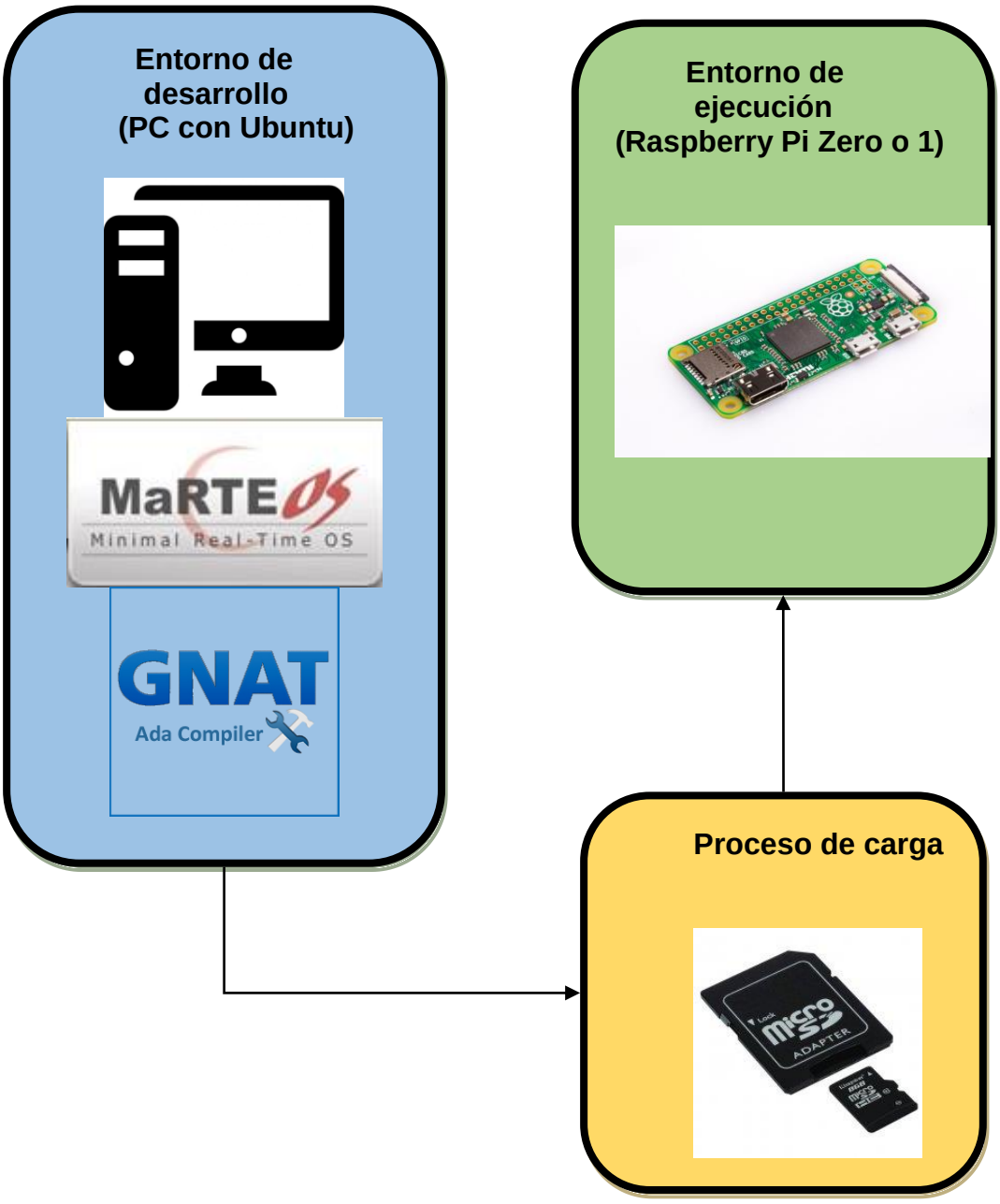

Figura 10: Entorno de desarrollo original

Por otra parte, en cuanto a dispositivos de salida, MaRTE OS para Raspberry Pi 1 posee manejadores para dos dispositivos de salida: HDMI y línea serie.

Un elemento deseable en cualquier sistema de desarrollo cruzado es que exista un emulador que permita la prueba funcional de aplicaciones en el

sistema de desarrollo de una forma mucho más rápida y cómoda que utilizando la placa de ejecución. Se dispone de una versión del emulador qemu que soporta la Raspberry Pi 1.

3.2 Compatibilidad con Raspberry Pi Zero

Una vez probado la situación inicial de MaRTE OS para Raspberry Pi pasamos a comprobar su compatibilidad con la Raspberry Pi Zero. Teóricamente deberían ser totalmente compatibles debido al uso del mismo chipset como ya se ha comentado anteriormente.

Por ello se testeó que:

- Las SDs con NOOBS que se utilizaban con la Raspberry Pi 1 funcionen correctamente con la Zero.
- El correcto funcionamiento de los dispositivos de salida HDMI y línea serie.
- El correcto funcionamiento del emulador qemu utilizado para la Raspberry Pi 1.

Es elegido ese emulador debido a que la Raspberry Pi 1 B y la Zero poseen el mismo chipset, el Broadcomm BCM2835 que posee el procesdor ARM1176. Al tener el mismo chipset y por lo tanto el mismo procesador el emulador nos es válido. Si se desea ejecutar alguna aplicación en qemu solo hay que lanzar la siguiente instrucción:

$ qemu-system-arm -kernal kernel-cpu arm1176 -m 256 -M raspi -serial stdio -s -S

El flag -m es para la memoria RAM del sistema a emular, la Raspberry Pi 1 B tiene 256 MB, pero la Zero posee el doble 512 MB.

Al no sufrir ninguna incompatibilidad, no ha sido necesario ningún cambio en MaRTE OS para su correcto funcionamiento en la Raspberry Pi Zero.

3.3 Mejora de la carga de la aplicación desde la SD.

Como se ha descrito en el apartado 3.1 la generación de una tarjeta SD es un proceso muy engorroso y que consume mucho tiempo. Y no sólo la generación, también el tener que copiar el ejecutable a la tarjeta SD una y otra vez. En este apartado se describe la simplificación realizada en este aspecto.

A grandes rasgos el proceso de arranque de un computador Raspberry Pi 1 o Zero [16] es el siguiente:

- Carga en la GPU (carga un código embebido en la misma GPU que contiene entre otras cosas el driver de la tarjeta SD).
- Carga de bootcode.bin de la memoria SD.
- Carga de start.elf de la memoria SD.
- Carga de kernel.img de la memoria SD.

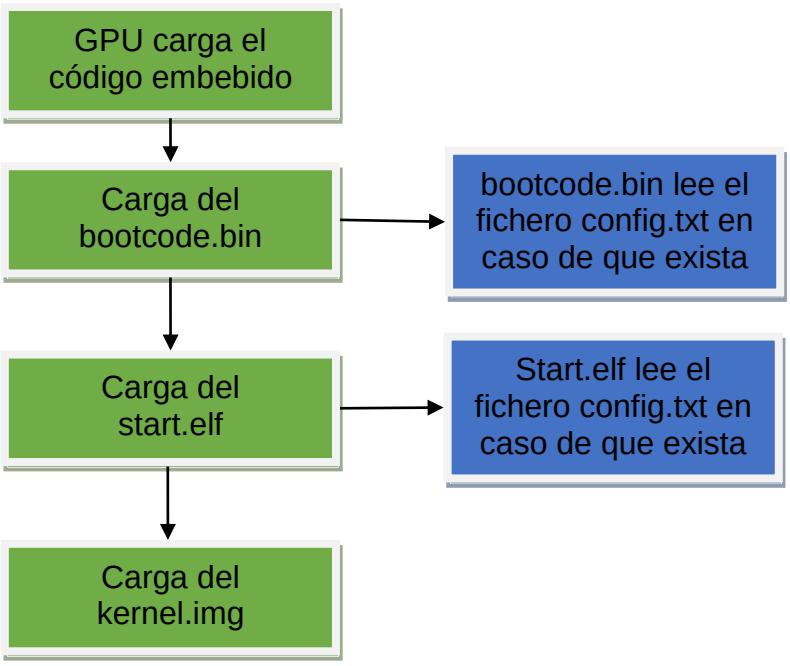

Figura 11: Proceso de carga en la Raspberry Pi

Una vez encendida la Raspberry Pi la GPU carga el código embebido que contiene el controlador de la tarjeta SD y el bootloader del bootcode.bin. Después carga el bootcode.bin de la tarjeta SD en la memoria caché L2. Una vez cargado la GPU transfiere el control a bootcode.bin. Lo primero que hace es habilitar la SDRAM y después, en caso de existir, lee varios parámetros de configuración del fichero config.txt, cómo la cantidad de memoria que se asigna en el particionado. Si el fichero de configuración no existe utiliza los valores por defecto, 64 MB para la GPU y el resto para el core ARM. Luego bootcode.bin particiona la memoria SDRAM para la GPU y el core ARM. Una vez particionada cede el control a star.elf, el cual lee el fichero de configuración config.txt en caso de existir. Por último start.elf carga el fichero kernel.img en la dirección 0x08000 de la memoria SDRAM. Una vez cargado el fichero se transfiere el control al core ARM para que comience la ejecución del código del kernel.img que, como su nombre indica, es típicamente el núcleo del sistema operativo que se desea ejecutar.

Por lo tanto, lo único necesario en el proceso de boot de la Raspberry Pi son los ficheros bootcode.bin y start.elf, disponibles para descarga en el repositorio oficial de Raspberry Pi en Github [17] y el kernel.img que deseemos

ejecutar. De esta manera simplemente formateando la tarjeta como FAT32 y copiando estos ficheros tendríamos todo lo necesario para el uso de la Raspberry, método mucho más rápido que el usado anteriormente de instalar Raspbian con NOOBS y sustituir el kernel.img por el deseado.

3.4 Carga remota de la aplicación por linea serie.

3.4.1 U-boot.

Como se vio en el apartado anterior, hemos simplificado mucho la creación de la SD pero aún así, el proceso de copia del ejecutable a la SD en cada iteración hace muy tedioso el proceso y por eso viene bien un sistema de carga remota. Se escoge U-boot debido a que permite la carga desde USB, ethernet, la misma tarjeta SD, y por línea serie.

Hay varios repositorios online para la descarga de U-boot, para este proyecto se ha utilizado el mainline [18], el principal. Esto es debido a que actualmente posee todo el soporte necesario para Raspberry.

Una vez descargado el repositorio el siguiente paso es su compilación, para ello necesitamos otro compilador cruzado ya que el que usamos para MaRTE no nos vale. El elegido es el compilador arm-linux-gnueabi disponible en los repositorios de Ubuntu.

Cuando compilamos U-boot se genera un archivo llamado u-boot.bin, ese archivo es el que debemos copiar a la memoria SD de la Raspberry Pi y renombrar a kernel.img.

El siguiente objetivo es ahora el estudio de todos los métodos de carga que posee U-boot y cuales son compatibles con las Raspberry Pi Zero y Raspberry Pi 1 B. Pero antes de eso es necesario la comunicación con u-boot.

Para poder comunicarnos con u-boot usaremos cutecom [19], seleccionando en el mismo como device el dispositivo /dev/ttyUSB0 ya que usaremos un adaptador de línea serie a USB.

Ahora que ya podemos ejecutar u-boot y comunicarnos por él pasamos a comprobar el funcionamiento de los distintos métodos de carga.

Primero comprobaremos el funcionamiento por línea serie. Para poder cargar por línea serie nuestras aplicaciones debemos instalar primero el paquete lzrs en Ubuntu, esto nos permitirá enviar nuestras aplicaciones con cutecom. El proceso de carga es el siguiente:

- En u-boot lanzaremos la instrucción `loady 0x1000000`. Siendo 0x1000000 la dirección de memoria en la que queremos guardar nuestra aplicación.
- En cutecom pulsaremos el botón send, habiendo seleccionado antes el modo Ymodem.
- Elegiremos nuestra aplicación a enviar a la Raspberry y esperaremos unos segundos a que se envíe.
- Iniciamos la carga.

El siguiente método que estudiaremos será el de cargar aplicaciones que se encuentren en la memoria SD. Para ello es indispensable haber copiado previamente la aplicación que deseamos cargar a la memoria SD. Entonces desde u-boot utilizaremos la instrucción `fatload mmc 0:1 0x1000000 kernel`. Debido a un conflicto con cutecom esa instrucción es demasiado larga y al enviar el comando a u-boot se corta, para solucionarlo creamos una variable llamada ad en la que guardaremos la dirección de memoria y así acortamos la instrucción. Luego la instrucción que usamos en realiad es `fatload mmc 0:1 $ad kernel`.

El último método que utilizaremos es el de carga de aplicaciones vía ethernet, este método solo es compatible con la Raspberry Pi 1 B ya que la Zero no dispone de puerto ethernet. El proceso que seguiremos para la carga vía ethernet es el siguiente:

- Configuramos en nuestro host PC un servidor DHCP.
- En u-boot tecleamos la siguiente secuencia de instrucciones
    - `Setenv serverip ipdenuestroservdhcp`
    - `Setenv ipaddr nuestraip`
    - `Tftp 0x1000000 kernel`
    - Iniciamos la carga.

Para iniciar la carga hay diferentes métodos y eso depende del formato en el que le enviemos nuestra aplicación. Si lo que le enviamos es un fichero en formato ELF (*Executable and Linkable Format*), utilizaremos la instrucción `bootelf 0x1000000`, si lo que le enviamos es un fichero uImage, utilizaremos `bootm 0x1000000`.

Un fichero uImage es un fichero de imagen que contiene un *wrapper* de U-boot que incluye el tipo de sistema operativo e información para la carga como la dirección de la misma. Es creado mediante las herramientas de U-boot que están disponibles en los repositorios de Linux, en concreto mkimage.

Debido a un conflicto entre MaRTE y U-boot que es desconocido por el momento, en ocasiones el arranque falla y el sistema se reinicia por lo que se plantea como futuro proyecto la solución de dicho problema.

3.4.2 Bootloader06

Para disponer en el momento de finalización de este proyecto de un cargador remoto funcional sin conflictos pasaremos a utilizar una segunda opción. Este es el bootloader06 del repositorio de Raspberry del usuario dwelch67 [20]. Es un bootloader más simple, que funciona solo por línea serie. Para su funcionamiento simplemente copiaremos el binario compilado y renombrado a kernel.img a la memoria SD de nuestra Raspberry Pi y una vez la Raspberry Pi encendida enviaremos por cutecom en modo Xmodem nuestro binario, cuando la carga se haya completado, solo tenemos que enviar una ‘g’ y el sistema se iniciará.

4 LIBRERÍA PARA EL MANEJO DEL EXPLORER PHAT EN MARTE OS

Una vez que ya disponemos de un entorno de desarrollo cruzado para Raspberry Pi Zero pasaremos al siguiente objetivo del proyecto, una librería para MaRTE OS que facilite el desarrollo de aplicaciones de robótica mediante el Explorer Phat.

4.1 Estructura de la librería phat.c

El objetivo principal de la librería es proporcionar una API (Interfaz de programación de aplicaciones) que facilite el acceso a toda la capacidad que posee el Explorer Phat y, por lo tanto, será necesario que aporte control sobre:

- Cuatro entradas digitales
- Cuatro entradas analógicas
- Cuatro salidas digitales
- Dos motores de corriente continua.

Para el desarrollo de la misma ha sido necesario el uso de la librería del chipset BCM2835 adaptada a MaRTE OS para el control del GPIO de Raspberry Pi, ya que principalmente el control del Explorer Phat se basa en el control de los pines GPIO asociados a cada función del Explorer. Para llegar a esta conclusión fue necesario el estudio del código Python de la librería que otorga Pimoroni para el uso del Explorer Phat.

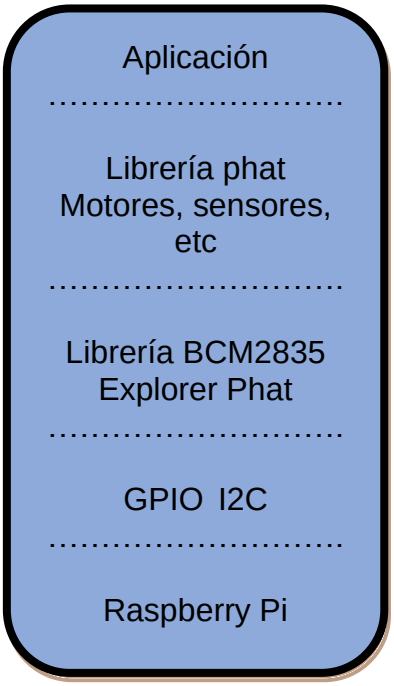

Figura 12: Capas de una aplicación que utiliza la librería

En la figura 12 se muestra el esquema de capas de una aplicación que utilice la librería desarrollada. La aplicación accede a la API proporcionada por la librería que, a su vez, utiliza la librería BCM2835 para acceder al GPIO y el controlador I2C de la Raspberry Pi

También fue necesario la adaptación del código C que usaba Pimoroni para el control de la velocidad de los motores utilizando modulación de ancho de pulso ya que se comunicaba utilizando la librería Python. Fue adaptada usando la librería del BCM2835.

Para la lectura de las entradas analógicas fue necesario el uso del protocolo I2C.

4.2 Control software de la velocidad de los motores por ancho de pulso.

Como se comenta en el apartado 4.1, para el control de la velocidad de los motores se adaptó el código C entregado por Pimoroni para su correcto funcionamiento con MaRTE OS. En esta sección vamos a explicar su funcionamiento y que cambios fueron necesarios para su correcto funcionamiento junto a MaRTE OS.

En el apartado 2.4 se explica cómo se controla un motor por modulación en el ancho de pulso. Para emularlo por software el código lanza un thread por cada motor activo (en nuestro caso dos máximo) que se encarga de poner a nivel alto el pin asociado con el polo positivo del motor durante todo su duty cycle y de ponerlo a nivel bajo durante el resto del ciclo. De esta manera conseguimos que el resultado sea el motor moviéndose a una mayor o menor velocidad que es lo que deseamos.

Para su correcto funcionamiento junto a MaRTE hubo que modificar el código en cuanto a las llamadas al GPIO para el cambio de los valores en los pines. Se sustituyeron las llamadas por las funciones equivalentes de la librería BCM2835 adaptada para MaRTE OS y se añade la posibilidad de poder asignar prioridad a los threads de la librería.

4.3 API de la librería

En este apartado describiremos la Interfaz de programación de aplicaciones (API) de nuestra librería.

4.3.1 Tipos y constantes

Para simplificar las llamadas a las funciones se han definido varios tipos de datos en el fichero `phat.h`.

El tipo de dato enumerado `motor_t` se utiliza en las funciones de control de motores y sus posibles valores son `PHAT_MOTOR_1` o `PHAT_MOTOR_2`.

Para el control de las salidas se utilizan variables del tipo de dato `output_t`. `Output_t` es un tipo de dato enumerado que puede tomar los valores `OUT1`, `OUT2`, `OUT3` y `OUT4` correspondiéndose con las cuatro salidas que posee el Explorer Phat.

Igualmente, para la lectura de las entradas digitales se utilizan variables del tipo de dato `input_t`. `Input_t` es un tipo de dato enumerado que puede tomar los valores `IN1`, `IN2`, `IN3` e `IN4` correspondiéndose con las cuatro entradas digitales del Explorer Phat.

En la lectura de las entradas analógicas se utiliza el tipo de dato enumerado `adc_t` y que toma los posibles valores `ADC1`, `ADC2`, `ADC3` y `ADC4` correspondiéndose con las cuatro entradas analógicas del Explorer Phat.

4.3.2 Función phat_init()

La primera función, `int phat_init()`, es la función que debe ser llamada siempre al comienzo de toda aplicación que quiera usar el Explorer Phat. Es la función encargada de la inicialización. Primero llama a la función de inicialización de la libreria BCM2835, que se encarga de la inicialización de sus variables globales para el funcionamiento del resto de sus funciones. Por otro lado, la función `phat_init` se encarga de la asignación de funciones a cada pin y de la inicialización a cero en caso de los pines de salida.

4.3.3 Funciones de control de motores.

Función phat_motor_start

La siguiente función es `void phat_motor_start(motor_t motor)`, y es la encargada de poner en funcionamiento el motor. Recibe cómo parámetro el motor que se desea poner en funcionamiento.

Función phat_motor_changeSpeed

La función `int phat_motor_changeSpeed(motor_t motor, float speed)` es la utilizada para cambiar la velocidad a la que se mueve el motor deseado. Recibe como parámetros el motor al que deseas cambiar la velocidad y a qué velocidad deseas que vaya. El valor de la velocidad se expresa en porcentaje siendo 100.0 el mayor valor posible y 0.0 el mínimo. Si el valor se expresa en negativo el motor se moverá en dirección contraria. Esta función puede ser llamada tanto con el motor en marcha como quieto.

Función phat_motor_invert

La función `void phat_motor_invert(motor_t motor)` es la encargada de invertir la dirección del motor.

Función phat_motor_stop

Y por último en cuanto a las funciones de control de motores tenemos la función `void phat_motor_stop(motor_t motor)` que es la función encargada de detener el funcionamiento del motor.

4.3.4 Funciones de entrada y salida digitales

Funciones de salida phat_output_on/off

Las funciones `void phat_output_on(output_t pinout)` *y* `void phat_output_off(output_t pinout)` son las encargadas de poner a nivel alto o bajo las salidas del Explorer Phat. Reciben como parámetro la salida que se desea modificar su valor.

Función de entrada phat_input_read

En cuanto a la función `unsigned int phat_input_read(input_t inputpin)` es la encargada de la lectura de los inputs digitales. Recibe como parámetro la entrada que se desea leer.

4.3.5 Entradas analógicas

Si deseamos leer las cuatro entradas analógicas tenemos la función *int* `phat_analog_read(adc_t channel)`. Es la función encargada de la comunicación mediante I2C con los registros en los que se almacenan los valores leídos y es una adaptación a código C, utilizando la librería BCM2835, de cómo se comunica la librería Python con el Explorer Phat. Recibe como parámetro la entrada analógica que se desea leer. La función retorna el valor entero leído.

Para la comunicación I2C se adaptó el código Python de la librería de Pimoroni del que se muestra a continuación el pseudocódigo:

```
Inicializar I2C
Establecer dirección de esclavo

Variable config = valores de configuración por defecto

Según channel Hacer
        Caso = ADC1
                Config |= CHANNEL1
        Caso = ADC2
                Config |= CHANNEL2
        Caso = ADC3
                Config |= CHANNEL3
        Caso = ADC4
                Config |= CHANNEL4

Escribimos en el registro de configuración el valor de config

Esperamos 300 milisegundos

Leemos el valor resultado en el registro conversión
Retornamos el valor
```

Para la inicialización de I2C se llama a la función de la librería BCM2835 `bcm2835_i2c_begin()` cuya función es simplemente configurar los pines del GPIO dedicados al protocolo I2C con dicha función. Para establecer la dirección de esclavo utilizaremos la función `bcm2835_i2c_setSlaveAddress(I2C_ADDRESS)` a la que pasamos como argumento nuestra constante con la dirección. Para escribir en el registro de configuración debemos de crear primero un array de tipo char de 3 elementos en el que el primero es el registro de configuración (constante REG_CONFIG), los 8 bit más significativos de los parámetros de configuración y los 8 bits menos significativos. Luego se llama a la función `bcm2835_i2c_write(configuration, 3)` siendo configuration el array y 3 su número de elementos. Para la lectura del valor llamamos a la función `bcm2835_i2c_read_register_rs(REG_CONVERT, buf, 2)` siendo buf un array de tipo char de 2 elementos. El valor que se retorna es el primer elemento de buf desplazado 8 posiciones a la izquierda haciéndole la operación lógica OR con el segundo elemento de buf desplazado 4 posiciones a la derecha. De esta manera tenemos el valor entero correctamente.

4.4 Prueba realizadas

Para comprobar el correcto funcionamiento de la librería junto a MaRTE OS se realizaron varias aplicaciones con el fin de testear que todas las funciones se ejecutaban correctamente y no había ningún conflicto con MaRTE.

La primera prueba se realizó para comprobar el correcto funcionamiento de los motores y consistía en un programa que probaba todas sus funciones y su correcta ejecución. También se movían los motores a distintas velocidades.

Para comprobar el correcto funcionamiento de las salidas se montó un sencillo circuito con un led y una resistencia y se comprobaba como se encendía el led. Se probó con todas las salidas.

Para las entradas digitales se creó un programa que leía una entrada cada varios segundos y se comprobó como leía un 1 en caso de tener señal y un 0 en caso de no tenerla. Se probó con todas las entradas.

Para las entradas analógicas se creó un programa similar conectando un sensor de distancia a una de las entradas y colocándole un objeto delante a diferentes distancias se observó como variaba el valor leído. Se probó también con todas las entradas analógicas.

5 DEMOSTRACIÓN

Con el objetivo de enseñar la librería implementada en este trabajo de fin de grado se desarrolla una aplicación que a modo de demostración enseñe sus capacidades.

5.1 Descripción del demostrador

La idea del demostrador es simular una cinta transportadora de una fábrica la cual transporta un objeto y se detiene a la altura de un brazo robótico con pinza el cual recoge el mismo y lo saca de la cinta.

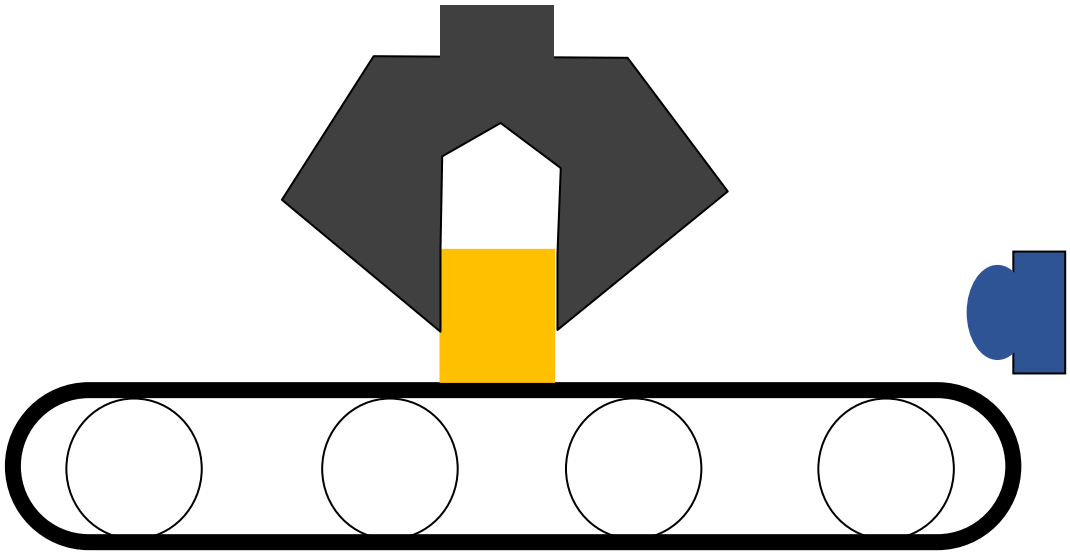

Figura 13: Representación del demostrador

En la figura superior podemos observar los distintos elementos que posee el demostrador. Por un lado, la cinta transportadora movida por un motor. Por otro lado, el brazo mecánico que recoge el objeto transportado por la cinta (bloque amarrillo). Y por último el objeto azul que sería el sensor de distancia que indica cuando el objeto se ha situado a la altura del brazo y debe detenerse.

5.2 Montaje

Para la implementación de la cinta transportadora y del bloque que transporta se utilizaron piezas de LEGO. Con ellas se montó la estructura que simula la cinta, las sujeciones necesarias para acoplar un motor y las sujeciones para el brazo mecánico que cogería el bloque transportado por la cinta.

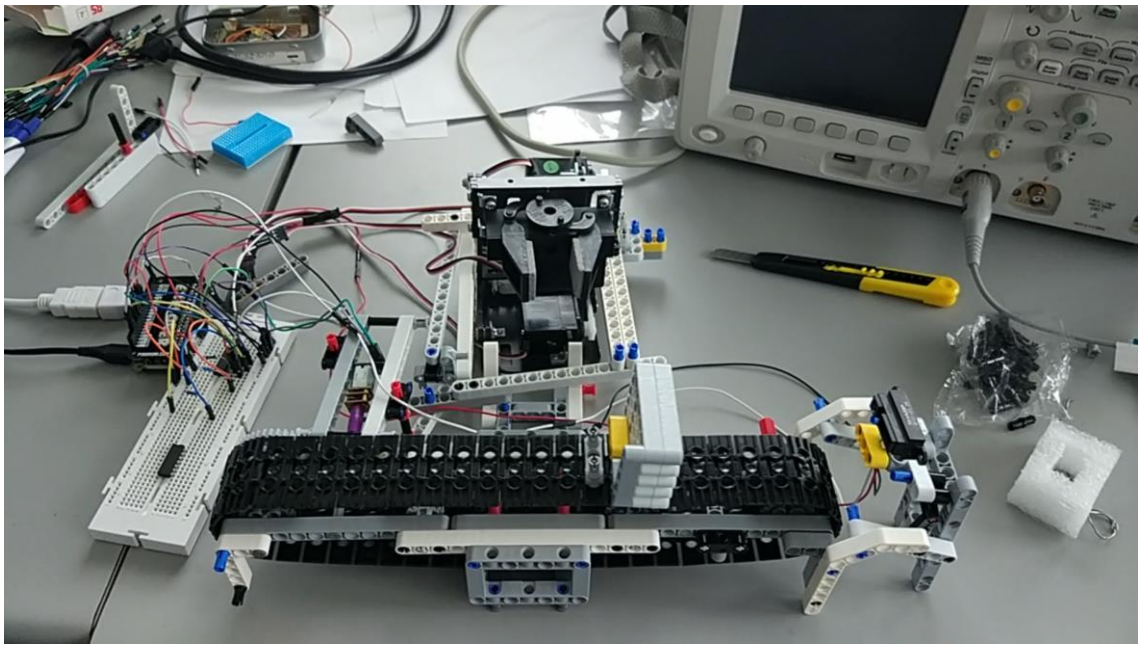

Figura 14: Imagen del demostrador montado

En cuanto al motor se utilizó el mencionado en la sección 2.4 con un adaptador que permite conectar su eje con las piezas de lego.

De brazo mecánico utilizamos un brazo mecánico formado por tres servos HS-422, sección 2.5. Cada servo controla una de las tres partes del brazo, uno controla la base, otro controla el codo y el último controla la pinza que agarra el bloque formado por piezas lego. Para el control del brazo mecánico se crea una sencilla librería que posee tres funciones muy simples, `changePosition`,`start` y `stop`.

Para saber cuándo detener la cinta se ha utilizado un sensor de distancia conectado a una de las cuatro entradas analógicas. El sensor mide la distancia a la que se encuentra el objeto en todo momento y cuando alcanza la posición deseada de la cinta se detiene la misma. Para su manejo se añadió a la librería `phat.h` una función que traduce el valor leído en la entrada analógica a distancia (centímetros).

5.3 Librería de control del brazo robótico

Para el control del brazo robótico se ha creado una librería que proporciona una sencilla API para su utilización. A continuación, se describen sus características principales.

Para las llamadas a las funciones se ha utilizado, al igual que con la librería del apartado 4, un tipo de dato enumerado llamado `servo_t`. Este tipo de dato puede tomar los valores `BASE`, `CODO` y `PINZA` haciendo referencia a los servos situados en dichas posiciones del brazo mecánico.

En cuanto a las funciones, `void hs422_init()` es la encargada de la inicialización del entorno. Realiza la llamada a la función `phat_init()` para la inicialización del phat y de la creación de una lista enlazada que contiene en cada posición la información de cada uno de los servos.

La función `void hs422_changePosition(servo_t servo, int grados)` es la función utilizada para cambiar la posición del servo. Recibe como parámetros el servo del brazo mecánico que deseamos mover y la posición en grados a la que le queremos colocar. Puede ser llamada tanto con el servo en marcha como parado. Si el servo se encuentra parado no se moverá hasta que se ponga en marcha mediante la función `hs422_start`.

La función `void hs422_start(servo_t servo)` es la encargada de poner en funcionamiento el servo. Recibe como parámetro el servo que deseamos mover.

Si deseamos detener el servo del brazo mecánico utilizamos la función `void hs422_stop(servo_t servo)` pasándole como parámetro el servo que deseamos detener.

Cuando el servo está en marcha se crea un hilo que ejecuta la siguiente función, descrita en pseudocódigo:

```
Mientras el servo este funcionando

        Salida a nivel alto
        Esperar el tiempo de señal alta

        Salida a nivel bajo
        Esperar el resto del ciclo

Fin mientras

Salida del thread
```

Para tener las salidas a nivel alto y bajo se llama a las funciones de la librería del Explorer Phat. El tiempo de espera a nivel alto es calculado en función de los grados de la posición del servo, a mayor amplitud, mayor es el pulso enviado al servo. El tiempo de espera a nivel bajo es una espera relativa, espera el resto del ciclo, que son 20 milisegundos.

5.4 Implementación

Para la implementación de nuestro demostrador se ha desarrollado una aplicación cuyo pseudocódigo se encuentra a continuación

```
Inicialización

Configurar velocidad del motor de la cinta
Puesta en marcha del motor de la cinta

Mientras el objeto no se encuentre a la altura del brazo
        Actualizar posición del objeto
Fin mientras

Detener motor de la cinta

Colocar brazo mecánico en posición inicial
Puesta en marcha del brazo

Colocar brazo a la altura del objeto
Cerrar Pinzas

Mover brazo hacia atrás
Soltar Objeto
```

Para la inicialización se llama a la función `hs422_init()` cuyo funcionamiento ha sido explicado en la sección anterior.

Para la configuración del motor se ha llamado a las función `phat_motor_changeSpeed(PHAT_MOTOR_1, 35.0)`. Se le pasa como argumento el motor 1 ya que ahí se encuentra conectado nuestro motor y una velocidad del 35% para que no se mueva la cinta demasiado rápido. A menor velocidad el motor no tiene fuerza suficiente para mover la cinta. También se llama a la función `phat_motor_invert(PHAT_MOTOR_1)` para invertir la dirección del motor y que la cinta se mueva en el sentido deseado.

Como se explica en la sección 4.3.3 para la puesta en marcha del motor se realiza la llamada a la función `phat_motor_start(PHAT_MOTOR_1)`.

En el pseudocódigo podemos observar que se realiza en el bucle *while* en el que se espera a que el objeto se encuentre a una determinada distancia. Para saber la distancia se ha creado una pequeña función que convierte el valor obtenido por la función `phat_analog_read(adc_t channel)` a centímetros.

Una vez alcanzado el objeto la altura del brazo mecánico, se detiene el motor con la llamada a la función `phat_motor_stop(PHAT_MOTOR_1)`.

El siguiente paso mostrado en el pseudocódigo es la colocación del brazo en una posición inicial. Esto se hace para conocer la posición de los tres servos que componen el brazo mecánico y poder realizar luego un movimiento más suave para recoger el objeto. Para ello llamamos a la función `hs422_changePosition(servo_t servo, int grados)` y colocamos la base en 180 grados (atrás), la pinza en 30 grados (abierta) y el codo en 50 (recto). Luego se llama a la función `hs422_start(servo_t servo)` con cada servo para que comiencen el movimiento.

Ahora que ya está el brazo en una posición inicial lo que se hace para moverle hasta la altura del objeto es mover la base poco a poco hasta que las pinzas se posicionen sobre el objeto. Para realizar este movimiento “suave” lo que se hace es un bucle for que comienza en la posición inicial de la base y en cada iteración mueve el servo un grado y espera 50 milisegundos. De esta manera lo que se percibe es un movimiento suave del brazo mecánico.

Para cerrar las pinzas se tiene que colocar el servo de las pinzas en una posición de 180 grados.

Una vez cerradas las pinzas realizamos el mismo movimiento hecho con la base, pero en sentido contrario. También se coloca el codo en una posición de 180 grados que es apuntando hacia atrás para después abrir las pinzas y que suelte el objeto.

6 CONCLUSIONES Y TRABAJOS FUTUROS

Antes de comenzar este trabajo de fin de grado el proceso de desarrollo de aplicaciones bajo MaRTE OS para Raspberry Pi era tedioso, exigía el "baile" de la tarjeta SD entre la Raspberry Pi y el ordenador de desarrollo. Por otro lado, obligaba a la instalación de un sistema operativo que nunca llegaba a ser utilizado (Raspbian) y no había sido testeado con la Raspberry Pi Zero.

Ahora concluido el trabajo de fin de grado poseemos:

- Mejora del entorno de desarrollo: carga remota de la aplicación por línea serie y mejora en el proceso de creación de la tarjeta SD.
- Librería `phat.h`: librería que proporciona una API sencilla para la creación de aplicaciones que den uso del Explorer Phat. También una librería para el control del brazo mecánico.
- Demostrador: aplicación que proporciona un ejemplo de las capacidades que posee el nuevo entorno creado.

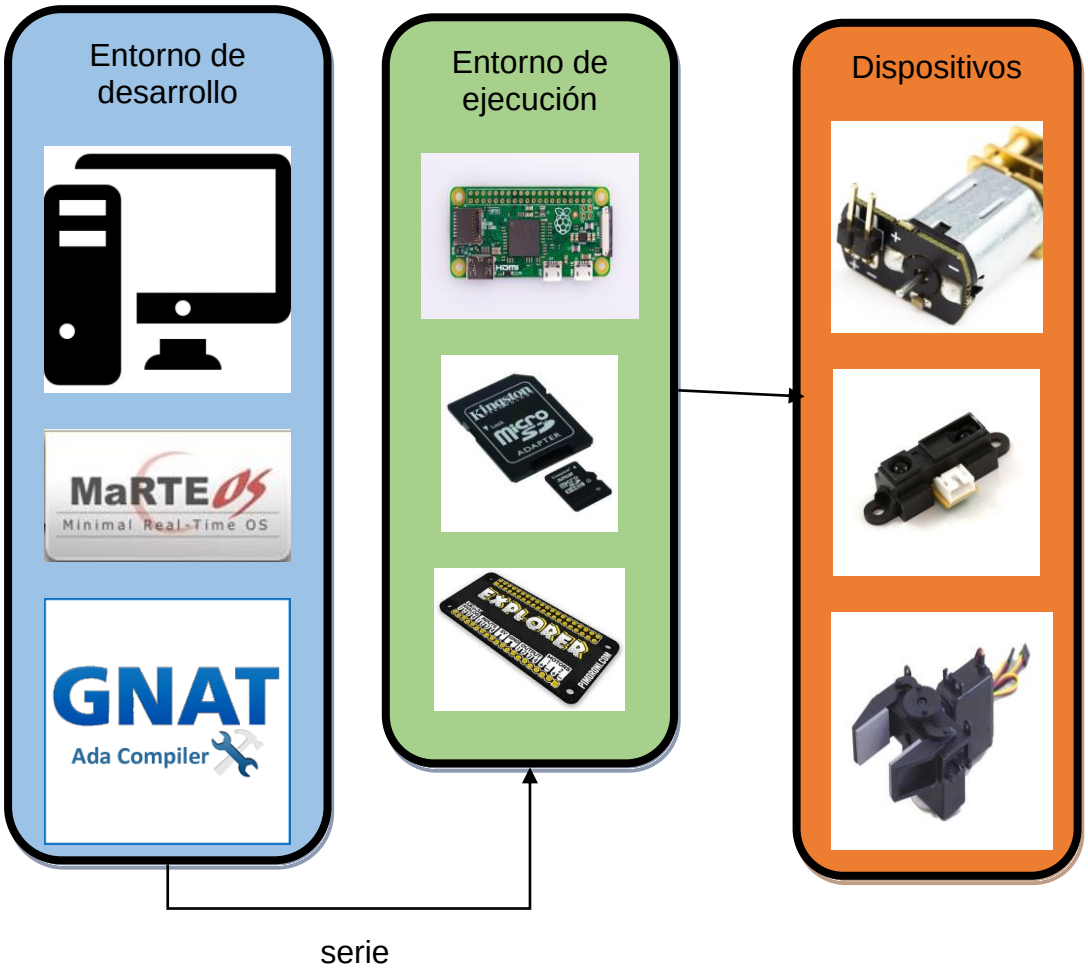

Figura 15: Entorno final

Por otro lado, hay algunos aspectos que deberían ser tratados en un futuro. El primero, la resolución del error que impide que MaRTE funcione con U-boot. También sería deseable la inclusión de alguna herramienta de depuración remota que permita la depuración de código ejecutado directamente en la Raspberry Pi y no en un emulador.

Por último, sería deseable aumentar los sensores y actuadores de los que se pueden utilizar junto al Explorer Phat añadiendo funciones que faciliten su uso como se ha hecho con el sensor de distancia, como podrían ser los motores y sensores de LEGO.

7 Bibliografía

[1] S. Pressman, Roger. Ingeniería del software: Un enfoque práctico, 3.ª Edición, Pag. 26-30

[2] Sitio web del Sistema operative MaRTE
https://marte.unican.es/

[3] Mario Aldea Rivas and Michael González Harbour. "MaRTE OS: An Ada Kernel for Real-Time Embedded Applications". International Conference on Reliable Software Technologies, Ada-Europe-2001, Leuven, Belgium, LNCS, May 2001.

[4] Página web de la Wikipedia con información sobre los modelos de Raspberry Pi
https://es.wikipedia.org/wiki/Raspberry_Pi

[5] Sitio web de la tienda de Pimoroni con la información del Explorer Phat
https://marte.unican.es/

[6] Sitio web de Wikipedia con la información sobre la modulación de ancho de pulso (PWM)
https://es.wikipedia.org/wiki/Modulaci%C3%B3n_por_ancho_de_pulsos

[7] Sitio web de Wikipedia con la definición de un servomotor.
https://es.wikipedia.org/wiki/Servomotor

[8] Ayerbe Gonzáles, C., Pérez Tijero, H. (Director). “Manejadores de sensores y actuadores LEGO Mindstorms para Rasberry Pi y MaRTE OS” (Trabajo de Fin de Grado). Universidad de Cantabria. Junio - 2017

[9] Página de Wikipedia con información sobre el bus I2C
https://es.wikipedia.org/wiki/I%C2%B2C

[10] Repositorio oficial de Explorer Hat
https://github.com/pimoroni/explorer-hat

[11] Página oficial de Qemu
https://www.qemu.org/index.html

[12] Sitio web sobre el bootloader U-boot
http://www.denx.de/wiki/U-Boot

[13] AdaCore GNAT compiler
http://libre.adacore.com/

[14] García Villaescusa, D., Aldea Rivas, M. (Director). "Portado de MaRTE OS a la arquitectura ARM" (Proyecto Fin de Carrera).Universidad de Cantabria. Julio - 2014.
https://repositorio.unican.es/xmlui/handle/10902/5537

[15] Documentación del administrador de instalación NOOBS
https://www.raspberrypi.org/documentation/installation/noobs.md

[16] Página web con la información del proceso de arranque de una Raspberry Pi
http://exileinparadise.com/raspberry_pi_boot

[17] Repositorio oficial Raspberry Pi
https://github.com/raspberrypi

[18] Repositorio mainline de U-boot
http://git.denx.de/u-boot.git/

[19] Página oficial de cutecom
http://cutecom.sourceforge.net/

[20] Repositorio de Raspberry del usuario dwel67
https://github.com/dwelch67/raspberrypi