Facultad de Ciencias

Evaluación del sistema operativo Android para aplicaciones de tiempo real

(Evalutaion of the Android operating system for real-time applications)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Santiago Sañudo Martínez

Director: Mario Aldea Rivas

Co-Director: Alejandro Pérez Ruiz

Junio – 2018

Índice de contenido

Índice de Figuras

Índice de Tablas

Agradecimientos

Antes de comenzar, me gustaría agradecer a mis familiares más cercanos los cuales han hecho posible que llegue hasta donde estoy. Por ello, un simple gracias es poco para mis padres y mi hermano.

Durante estos largos años, no solo he estado yo luchando por conseguir este objetivo, sino que al igual que yo, había muchos otros. Entre ellos, los grandes amigos que he hecho a lo largo de la carrera y que seguramente sin ellos esto habría sido muchísimo más difícil y por ello solo me queda daros las gracias.

Por último, agradecer también a todos y cada uno de los profesores que en mayor o menor medida han hecho posible que adquiera todos los conocimientos que tengo hasta la fecha sobre este bonito e interesantísimo campo que es la Ingeniería Informática.

Entre ellos tengo que destacar especialmente a Mario Aldea, Alejandro Pérez y Héctor Pérez, ya que gracias a su increíble paciencia y su increíble conocimiento en la materia han conseguido convertir un tema totalmente desconocido como era este trabajo en un primer momento en una experiencia enriquecedora y realmente bonita.

Resumen

En las últimas décadas hemos visto un avance enorme en lo referente al mundo de la tecnología y la informática. Un campo en el que este avance ha resultado especialmente destacable es el relativo a la computación y los dispositivos móviles donde el sistema operativo Android tiene un papel protagonista. La gran popularidad de este sistema operativo y su implantación en gran número de dispositivos móviles ha motivado su utilización en entornos para los que, a priori, no estaba destinado. Entre ellos cabe destacar por la relevancia para este proyecto los entornos industriales, médicos o de automoción donde existen requisitos de tiempo real.

Este proyecto busca desarrollar un entorno que permita la prueba y evaluación de aplicaciones con requisitos de tiempo real en el sistema operativo Android ejecutado sobre un computador Raspberry Pi 3. Para ello nos hemos basado en un trabajo previo que se ha realizado en el grupo de Ingeniería Software y Tiempo Real de la Universidad de Cantabria (ISTR) donde se ha propuesto utilizar mecanismos que están disponibles en Android/Linux para poder aislar un núcleo del procesador. De tal modo que en dicho núcleo aislado se puedan ejecutar aplicaciones de tiempo real reduciendo las interferencias y consiguiendo tiempos de respuesta más acotados.

El entorno desarrollado en este proyecto está pensado para facilitar la implementación de aplicaciones robóticas de tiempo real. Este tipo de aplicaciones requieren utilizar sensores y actuadores con los que interactuar con el entorno físico. Los sensores y actuadores elegidos son los proporcionados por el kit de Lego Mindstorms, los cuales son accesibles desde la Raspberry Pi 3 utilizando el adaptador PiStorms. Para el uso del PiStorms ha sido necesario adaptar al sistema operativo Android una librería existente basada en el protocolo de comunicaciones I2C.

Los tres pilares base sobre los que desarrollamos el proyecto son los siguientes, realización del sistema operativo para la Raspberry, una librería que utiliza el protocolo I2C para establecer la comunicación con el kit de Lego Mindstorms y la demostración de las aplicaciones de tiempo real con el mecanismo de aislamiento de procesos para Android desarrollado en el grupo ISTR. De esta forma dejamos a futuros alumnos un entorno para desarrollo de aplicaciones de tiempo real laxo en Android sobre el que desarrollar nuevos proyectos robóticos.

Palabras Clave: Tiempo Real, Raspberry Pi, PiStorms, Lego Mindstorms, Sistema Operativo Android, protocolo I2C.

Abstract

In the last decades we have seen a huge development in the world of technology and computer science. One field that has experienced this development and it has been especially remarkable is the field related to computation and mobile devices where the Android operating system has a leading role. The great popularity of this operating system and its implementation in many mobile devices has motivated its use in environments for which it was not intended. Among these we must mention some environments such as medical, industrial or automotive environments where there are real time requirements.

This project looks forward to evaluating the viability of the execution of applications with real-time requirements running in a Raspberry Pi 3 b using the Android operating system. To do so, we have work base on a previous project made by the Software Engineering and Real Time group of the University of Cantabria (ISTR) where there've been made different mechanisms available for Android/Linux, so we can isolate a core of the CPU. This isolated core can execute real-time applications reducing interferences and achieving times of response more accurate.

The environment developed in this project is designed to facilitate the implementation of robotics in real-time applications. This type of applications requires the use of sensors and actuators with which to interact with the physical environment. The sensors and actuators chosen are those provided by the Lego Mindstorms kit, which can be use with the Raspberry Pi 3 thanks to the PiStorms adapter. To be able to use de PiStorms adapter we have had to adapt the Android operating system with an existing library based on the I2C communication protocol.

The three main pillars the project is based on are the following, adapting a custom Android operating system for the Raspberry, adapting a library which uses I2C communication protocol, so we can communicate with the Lego Mindstorms kit and a simple demo of a real-time application using the isolation technic for process with Android developed by ISTR. In this way we give future students an environment for development of lax real-time applications in Android on which to develop new robotic projects.

1. Introducción

1.1 Motivación

Desde que en 2008 apareció el primer móvil con Android este sistema operativo ha ido imponiéndose hasta convertirse en el más extendido en el ámbito de la telefonía móvil. Este hecho ha producido que haya ido evolucionando para adaptarse a nuevos dispositivos y ámbitos. Prueba de esto, es que en los últimos años ha crecido el interés por utilizar este sistema operativo en entornos industriales, médicos o de automoción donde existen requisitos de tiempo real. Esto provocó que en el grupo de Ingeniería Software y Tiempo Real (ISTR) de la Universidad de Cantabria se iniciase una línea de investigación para tratar de buscar soluciones para la adaptación de Android en entornos de tiempo real. La solución encontrada se basa en la ejecución de las aplicaciones de tiempo real en uno o más procesadores aislados, de forma que el sistema Linux afecte lo menos posible al comportamiento temporal de dichas aplicaciones, estando esta ya publicada en un trabajo previo [1]. Este trabajo constituye uno de los principales antecedentes del presente Trabajo Fin de Grado, ya que en él se ha querido evaluar la anterior propuesta en un entorno real haciendo uso de la famosa placa de desarrollo Raspberry Pi 3 Modelo B junto al conjunto de sensores y actuadores ofrecidos por Lego Mindstorms [2].

Debido a esto, este proyecto busca plantear un entorno de pruebas demostrando el trabajo previo [1], usando para ello la Raspberry Pi, con un sistema operativo Android, pero que hiciera uso de la plataforma de Lego Mindstorms [2]. Se pensó en utilizar una versión de Android de manera que fuera posible adaptarla a bajo nivel permitiéndonos aplicar los cambios necesarios para adaptarnos a las condiciones necesarias explicadas en el trabajo previo [1] para ejecutar aplicaciones de tiempo real laxo. Este hecho está ampliamente relacionado con nuestra elección de la Raspberry Pi 3 b [3] como nuestra plataforma sobre la que desplegar Android, ya que es un miniordenador de bajo coste y energético muy acotado. Además, la Raspberry Pi 3 cuenta con una CPU multinúcleo, lo que constituye un requerimiento básico para la utilización de la tecnología desarrollada en el grupo ISTR.

Las principales razones por las que se ha elegido trabajar con la Raspberry Pi 3 b son dos. La primera es debido a que está ampliamente extendida en el mundo de la docencia y posee una gran comunidad altamente activa en constante innovación con la aparición de nuevos proyectos, prueba de ello son las numerosas publicaciones [4] elaboradas comentando las distintas posibilidades que esta ofrece. La segunda es el abanico de posibilidades del que disponemos cuando trabajamos con ella debido a los puertos entrada/salida (GPIO) lo cual para este proyecto concreto nos facilita la conexión de sensores y actuadores. Estos dos hechos están ampliamente relacionados con el factor del precio y el reducido tamaño. Por ello consideramos que la Raspberry Pi es una elección ideal ya que puede ser elegida de nuevo en posibles nuevos proyectos donde sea necesario el uso de Android en dicha plataforma.

1.2 Objetivos

El objetivo principal de este Trabajo Fin de Grado consiste desarrollar un entorno que permita la prueba y evaluación de aplicaciones de tiempo real laxo sobre Android utilizando los mecanismos de aislamiento desarrollados en el grupo ISTR [1].

Esta solución propone utilizar una serie de mecanismos que existen en Linux/Android para poder aislar un núcleo de un procesador multinúcleo y así en él poder ejecutar aplicaciones de tiempo real. Las aplicaciones ejecutadas en el núcleo aislado ven reducidas significativamente las interferencias debidas a las actividades del sistema operativo y del resto de aplicaciones ejecutando en los restantes núcleos del procesador.

El entorno de pruebas debe facilitar la utilización de forma sencilla de motores y actuadores con los que construir sistemas de tiempo real sencillos. Por esta razón para desarrollar el objetivo principal se plantean los siguientes objetivos secundarios:

- Adaptación del sistema operativo Android para la Raspberry Pi 3. La razón de emplear este dispositivo no es otra que el hecho de que cuenta con un puerto entrada/salida que nos permite conectar distintos sensores y actuadores de Lego Mindstorms [2].

- Adaptación del módulo PiStorms a Android para facilitar la conexión de sensores y actuadores de Lego Mindstorms [2]. Para ello es necesario adaptar la librería de otro Trabajo Fin de Grado [5] realizado previamente, de forma que esta dependiera únicamente del protocolo I2C, ya que en el trabajo mencionado previamente se dependía de una librería concreta del chipset empleado, BCM2835 [19].

- Evaluación temporal de la tecnología de aislamiento desarrollada en el grupo ISTR. Para ello se plantea un entorno de pruebas de tiempo real donde podamos comprobar de manera correcta que funciona de la forma esperada.

Para la consecución de estos objetivos se plantea la realización de dos prototipos. El primero permite comprobar la funcionalidad de la librería adaptada y el otro evaluar las tecnologías de aislamiento.

1.3 Metodología

En el caso de la metodología empleada, hemos optado por utilizar una metodología incremental. Esta metodología consiste en realizar repetidas iteraciones en las que vamos añadiendo funcionalidad adicional o mejoramos la actual. De esta manera tenemos un seguimiento progresivo de los avances que hemos ido realizando.

La principal ventaja que obtenemos de esta manera es que minimizamos los posibles riesgos ya que los errores se van corrigiendo progresivamente. Sin embargo, su principal desventaja es que los errores se pueden detectar demasiado tarde ya que si un error en las últimas iteraciones requiere modificaciones en algo realizado en las primeras iteraciones supone un riesgo enorme y complejo.

Las funcionalidades o tareas que se han ido realizando a lo largo de las iteraciones son las siguientes:

- Documentación sobre el sistema operativo Android y sus posibles adaptaciones. En esta iteración nos documentamos como está estructurado el sistema operativo Android, y buscamos información sobre posibles adaptaciones de este a la Raspberry Pi.

- Adaptación del sistema operativo Android a la Raspberry Pi. Seguimos los pasos explicados en el repositorio [9] donde encontramos la información necesaria. Realizamos las modificaciones en el kernel necesarias para adaptar el sistema operativo a las condiciones deseadas.

- Adaptación de la librería del Trabajo Fin de Grado previo [5] al protocolo I2C. De forma que añadimos una capa intermedia software para desligarnos de la librería empleada previamente y así hacer uso del protocolo I2C.

- Demostración del correcto funcionamiento de la librería adaptada. Elaboramos un prototipo con Lego Mindstorms para comprobar la funcionalidad de la librería adaptada.

- Demostración de situación de tiempo real con la librería adaptada. Planteamos un modelo de tiempo real, como es el caso de un estabilizador de cámara, el cual es un proyecto planteado numerosas veces en la comunidad de Raspberry Pi. Por último, comprobamos su correcto funcionamiento.

- Elaboración de pruebas y diagnóstico de las mismas. Elegimos las distintas pruebas a elaborar y establecemos modificaciones en el código para poder recoger los datos generados. Analizamos los datos obtenidos y planteamos las conclusiones obtenidas.

2. Tecnología y Herramientas Utilizadas

2.1 Lego Mindstorms

Es una plataforma de robótica orientada al desarrollo de pequeños juguetes para niños. Está pensada para unificar la construcción de objetos o estructuras con la programación. Esto se realiza con el fin de que los niños obtengan una experiencia interactiva respecto a lo que programen para que hagan sus diseños.
Esta plataforma está fabricada por la empresa LEGO, y actualmente consta de 3 generaciones:

1ª Generación, Bloque RCX

Como se puede apreciar por su robusta apariencia en la Figura 1, es la primera generación y cuenta con la posibilidad de lectura de 3 sensores distintos, así como la capacidad de almacenar programas en su memoria interna. El microcontrolador de esta generación era un Hitachi H8/3292, el cual posee una CPU con un solo núcleo.

Figura 1: Bloque RCX

2ª Generación, Bloque NXT

Esta segunda generación, con un aspecto más atractivo y moderno como se puede apreciar en la Figura 2, presenta ciertas mejoras tanto en los sensores aumentado la calidad y la precisión de estos e incluyendo nuevos tipos. Pese a que se cambia el microcontrolador por un modelo más moderno con arquitectura ARM7 se sigue teniendo una CPU de tan solo un núcleo

Esta generación añade una nueva cualidad, la conectividad por Bluetooth además de una nueva interfaz USB, facilitando drásticamente la comunicación con el dispositivo.

Figura 2: Bloque NXT

3º Generación, Bloque EV3

Esta última generación, ver Figura3, es una enorme mejora en todos los aspectos, ya que permite la paralelización de código de manera efectiva, hay 4 conexiones para sensores y 4 para motores e incluye un altavoz integrado. Además, se puede emplear una tarjeta microSD o la conexión USB para realizar la carga de ejecutables en el dispositivo. Cuenta también con conexión WIFI y bluetooth para comunicarse con otros dispositivos. Esta generación cuenta con un sistema operativo basado en Linux además de mejorar su procesador con una arquitectura ARM9. También incluye sensores de color, comunicación mediante infrarrojos, un mando a distancia y servomotores.

Figura 3: Bloque EV3

En nuestro caso, la idea fundamental del proyecto, descrita en la sección 2.2, será sustituir el mini controlador de la 3ª Generación, EV3 por la Raspberry Pi 3 b, con el adaptador de PiStorms de manera que realice la misma función que el mini controlador de Lego. Sin embargo, en lo que respecta a los motores y sensores utilizaremos todos los que están disponibles en un Kit de Lego de Mindstorms.

Figura 4: Kit Lego Mindstorms EV3

Actuadores y sensores

En estos kits, como el de la Figura 4 podemos encontrar dos tipos de actuadores, uno grande y uno pequeño. Ambos poseen la capacidad de detectar la posición que se encuentran actualmente respecto de un punto base inicialmente indicado. Además, es posible también sincronizar las acciones de manera que podamos hacer que dos actuadores avancen a la vez o se paren simultáneamente.

Respecto a las diferencias que podemos encontrar entre ambas versiones de actuadores son tres:

- La rotación que realiza el actuador grande es en el eje transversal de la estructura, mientras que el actuador pequeño es el de la propia estructura.
- La potencia que podemos emplear en el actuador grande es considerablemente mayor que la del menor, llegando en situaciones optimas de rendimiento a ser el doble.
- El actuador grande posee la capacidad de frenarse de manera que sea una parada lenta y suave o una parada brusca, mientras que el actuador pequeño tan solo puede pararse de manera brusca.

Respecto a los sensores encontramos:

- Sensores táctiles, permitiéndonos diferenciar cuando se pulsan.
- Un giroscopio, permitiéndonos saber el grado de inclinación que llevamos respecto del momento inicial de configuración o punto de referencia, así como la velocidad angular que el giroscopio lleva. Estas medidas solo se toman sobre un eje de coordenadas.
- Sensor de color, siendo capaz de distinguir entre 7 colores distintos, aunque la distancia de reconocimiento es bastante corta.
- Sensor ultrasonido, encargado de detectar la distancia con el objeto más próximo siendo muy útil y contando con una alta precisión.

2.2 Adaptador PiStorms Mindstorms

Figura 5: Adaptador PiStorms

El pequeño dispositivo mostrado en la Figura 5 es el responsable de establecer la comunicación de la Raspberry Pi con los sensores y motores EV3 de Mindstorms. Para ello, se conecta a las conexiones GPIO, ilustradas en la Figura 6, de la Raspberry Pi, tanto para alimentarse como para comunicarse con ella. Estas son las conexiones:

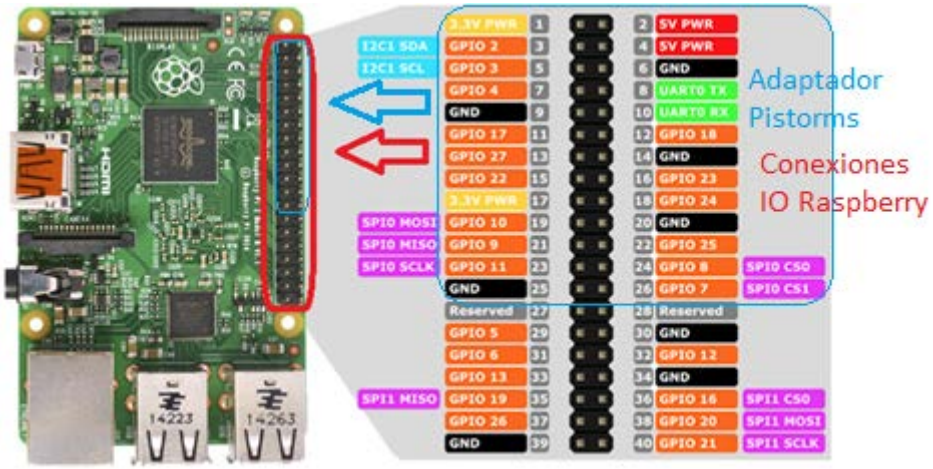

Figura 6: Conexión IO Raspberry

Este dispositivo para realizar la conexión con los sensores y motores de Mindstorms emplea conexiones RJ12. Las conexiones se dividen en dos bancos, el A y el B. En cada banco encontramos dos conexiones para motores y dos para sensores.

Para comunicarse la Raspberry con el Adaptador PiStorms toda la comunicación se realiza empleando el protocolo I2C, explicado extensamente en múltiples publicaciones [6], que además posteriormente comentaremos con más detalle en la subsección 2.7

2.3 Raspberry Pi 3 B

Figura 7: Raspberry Pi 3 B

La Raspberry Pi 3 B, mostrada en la Figura 7, es la tercera generación de un modelo que inicialmente fue ideado como un miniordenador de bajo coste con los recursos mínimos necesarios para centrase en el aprendizaje y la enseñanza. Y así fue como salió al mercado en febrero de 2012 la primera versión. Este dispositivo consiguió con un coste muy bajo presentarse al mercado como un miniordenador de alta conectividad y una capacidad de procesamiento más que suficiente para ejecutar pequeños programas. En la actualidad la versión más moderna es la Raspberry Pi 3 B+, sin embargo, nosotros utilizaremos la 3 B debido a que el lanzamiento de esta versión fue en pleno desarrollo del proyecto, el chipset era modificado ligeramente y no disponíamos del código fuente del kernel de la nueva versión que más adelante comentaremos para que era necesario.

Mencionar que la Raspberry no solo ha servido para el campo de la enseñanza, sino también para que mucha gente pueda realizar pequeños proyectos como pueden ser emuladores de consolas viejas (ver Figura 8), o incluso pequeños "*rigs*" de minado. Actualmente cuenta con una comunidad muy grande que siempre intenta ser altamente creativa con nuevos proyectos y propuestas.

Figura 8: Raspberry Pi emulador Juegos

En nuestro caso la Raspberry Pi 3 B cuenta con las siguientes especificaciones:

- CPU: 1.2GHz 64-bit quad-core ARMv8 (Broadcom BCM2837)
- GPU: Broadcom VideoCore 4.
- RAM: 1 Gb compartido con la GPU.
- Bluetooth: Disponible, versión 4.1
- Wifi: Disponible a 2.4 Ghz, versión 802.11n.
- Puertos USB: 4 Puertos 2.0.
- Salida Video: HDMI.
- Salida Audio: HDMI y conector de 3.5 mm.
- Conexión Ethernet: Conexión RJ45 FastEthernet 10/100.
- Fuente de Alimentación: Mediante Micro USB con 5V o con la conexión GPIO.
- Conectividad Entrada/Salida mediante puertos GPIO.

El motivo fundamental de su elección para este proyecto han sido tres factores. El primero es la gran comunidad y soporte que podemos encontrar dado que se encuentra actualmente muy extendida en el mundo de la docencia, lo cual es un valor añadido enorme a la hora de poder resolver problemas que puedan surgir. El segundo factor clave es sus especificaciones técnicas. Está claro que para que podamos plantear la solución explicada previamente del aislamiento del núcleo del procesador es necesario contar con una CPU multinúcleo de manera que estos sean núcleos físicos y no virtuales, y en el caso de la Raspberry Pi 3 b, no solo contamos con más de un núcleo, sino que tenemos cuatro, lo cual nos da los requisitos mínimos técnicos necesarios para llevar a cabo nuestro plan. Por último, el tercer motivo que hace que la Raspberry Pi 3 sea el dispositivo sobre el que basar nuestro proyecto es el hecho de su alta capacidad de conectividad gracias al GPIO. Esto nos permite conectar de manera sencilla los sensores y actuadores necesario.

Es cierto que podríamos haber elegido otros modelos similares a la Raspberry Pi, como pueden ser la Banana Pi [7] o el Odroid C2[8], que cumplen el requisito de tener un procesador con múltiples núcleos. Sin embargo, estos no poseen esa gran comunidad de entusiastas y desarrolladores que hemos mencionado anteriormente y por ello se descartaron.

2.4 Arquitectura del sistema operativo Android

Para conocer un poco más en que consiste Android y como está formado es necesario referirnos a este como un sistema operativo basado en Linux el cual es un sistema de desarrollo a través de capas software apiladas las cuales se muestran en la Figura 9.

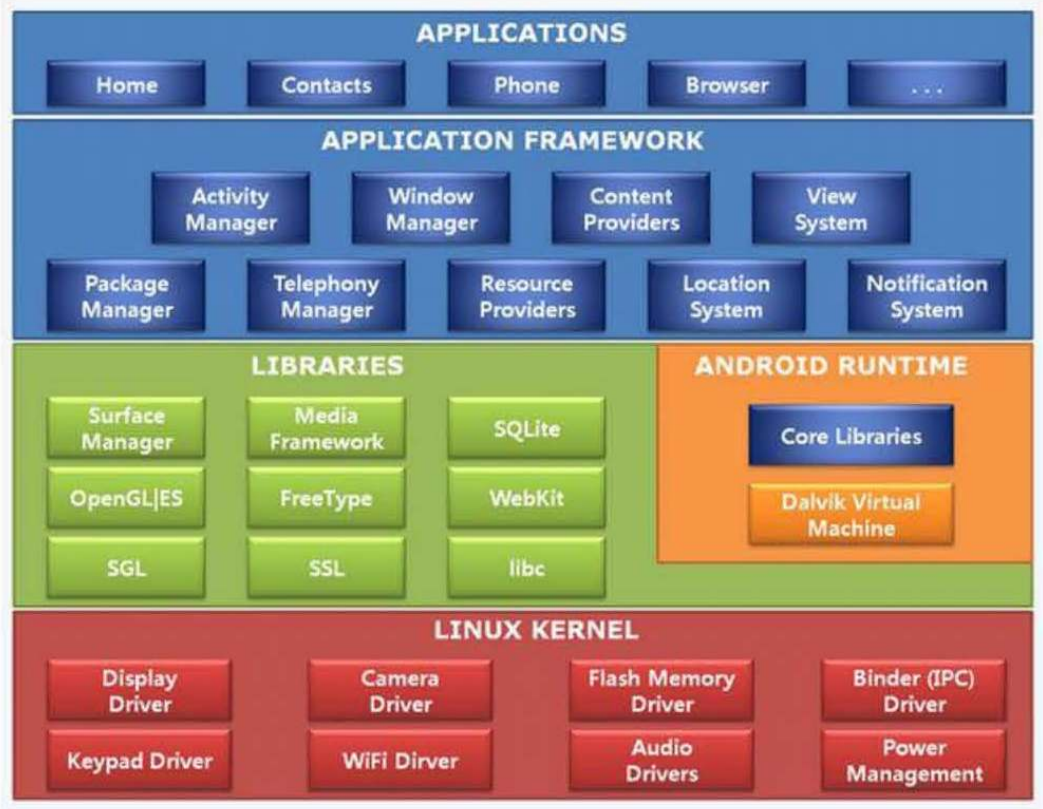

Figura 9: Arquitectura sistema operativo Android

Como se puede observar claramente encontramos cuatro capas software bien diferenciadas. En la capa más superior nos topamos con las aplicaciones Java de usuario y del sistema. Estas utilizan los servicios, las APIs y las librerías de la capa que se encuentra inmediatamente debajo

La capa denominada entorno de aplicaciones (Application Framework en inglés) contiene todo el conjunto de clases y servicios que utilizan las aplicaciones Java del nivel superior para realizar sus funciones en este sistema operativo.

A continuación, encontramos una capa de librerías escritas en principalmente en C/C++ y el software que compone el entorno de ejecución para las aplicaciones Android. El componente principal del entorno de ejecución en Android es la máquina virtual denominada ART (Android RunTime), ya que las aplicaciones de usuario se codifican en Java y son compiladas en un formato específico para que la máquina virtual pueda interpretar.

Dentro del conjunto de las librerías de esta capa nos encontramos con una especialmente importante para el presente proyecto, se trata de la denominada Bionic. Esta librería es una modificación realizada por Google de la librería tradicional glibc. Existen tres razones principales para que Google haya decidido crear su propia versión de glibc:

- Posee un tamaño reducido favoreciendo así su uso en dispositivos móviles con recursos limitados.

- La potencia de los procesadores que emplean los dispositivos Android es menor que la de un ordenador de sobremesa o portátil y esto es un factor importante ya que esta librería se diseñó en un principio para CPUs con frecuencias relativamente bajas.

- Se decidió otorgar al código de esta librería una licencia BSD, la cual permite que sea utilizada por aplicaciones con fines comerciales.

Por último, encontramos la capa más profunda de la estructura del sistema operativo Android, el kernel. En esta capa se gestionan los procesos, la memoria, la energía o los drivers entre otras muchas cosas.

2.5 AOSP (Android Open Source Project)

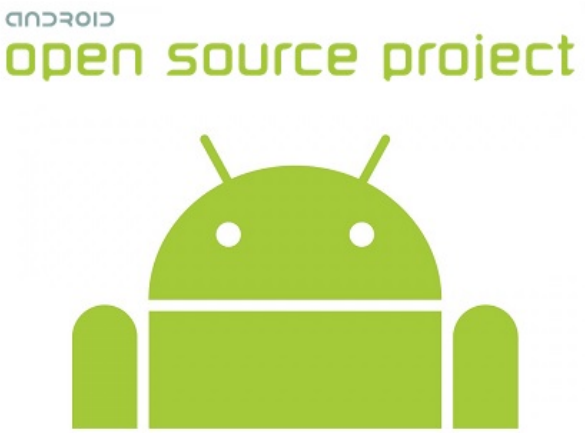

Figura 10: Android AOSP

AOSP es una versión de Android que libera Google para que los desarrolladores y fabricantes de dispositivos Android puedan adaptar dicho sistema operativo a su hardware específico. De este modo podemos encontrar distintos sistemas operativos en el mercado, de forma que, aunque estén en la misma versión de Android, estos varían unos de otros gracias a que los fabricantes han adaptado el software del mismo para explotar al máximo los recursos que ofrecen sus dispositivos y hacer que estos funcionen de la mejor forma posible.

Sin embargo, además de lo previamente mencionado es necesario también saber para qué dispositivo estamos adaptando el sistema operativo Android y, más concretamente, es necesario tener a nuestra disposición el código fuente del kernel de dicho dispositivo. En nuestro caso el código fuente del kernel de la Raspberry Pi 3 B pudimos encontrarlo en un repositorio de GitHub [9], mientras que el código fuente del AOSP proviene de los repositorios oficiales de Google [10].

Una vez obtenidas ambas partes, kernel y sistema operativo, tan solo quedaba realizar las pertinentes modificaciones al kernel y al sistema operativo y compilar ambas obteniendo así la imagen que usaremos para desplegar nuestro sistema Android en la Raspberry Pi 3 b, que en nuestro caso fue la versión 7.1.2.

Sin embargo, realizar estos pasos supone dos grandes desventajas, deberemos descargarnos aproximadamente 50GB de código fuente si sumamos el sistema operativo Android y el kernel de la Raspberry, y además deberemos llevar a cabo la compilación de todo el sistema lo cual tomará una cantidad muy elevada de tiempo en un equipo de trabajo estándar.

Una de las principales modificaciones realizadas al kernel de la Raspberry se realiza en su configuración previa a la compilación donde pudimos activar las opciones necesarias para posteriormente ser capaces de aislar correctamente un núcleo del procesador.

Otra modificación realizada fue la implementación de la librería i2c-dev [11] en el sistema operativo permitiendo así que pudiéramos emplear dicho protocolo para poder comunicarnos con los distintos componentes de Lego.

Dado que nosotros como desarrolladores necesitamos adaptar ciertas opciones del sistema operativo a nuestras necesidades, se ha optado por coger el código fuente del sistema operativo liberado por Google (AOSP) junto al kernel específico para la Raspberry Pi 3 B. Permitiéndonos así alcanzar las condiciones necesarias para realizar nuestra demostración de una aplicación de tiempo real laxo explicada en el trabajo previo [1].

Android es un sistema basado en Linux, sin embargo, este está mucho más limitado respecto a lo que permite o no hacer al usuario no "*root*", por ello era necesario modificar ciertas propiedades como es el sistema de autenticación a la hora de conectarse de forma remota mediante el protocolo ADB, permitiendo que el usuario generado por dicha conexión fuera superusuario. Esto es altamente importante debido a que las conexiones I2C empleadas en el sistema para comunicarnos con el Adaptador de Pistorms, se realizaran mediante lectura y escritura sobre ciertos elementos del sistema que solo son accesibles por el superusuario y por tanto para la ejecución de nuestros programas es totalmente necesario que esto fuera posible.

2.6 ADB (Android Debug Bridge)

ADB es una interfaz software para sistemas Android. Esta puede ser usada mediante USB, WIFI o Ethernet para conectar nuestro dispositivo Android con un ordenador, ofreciendo así una amplia variedad de posibilidades para establecer la conexión entre ambos.

Con el paso del tiempo esta tecnología se ha ido desarrollando hasta tal punto que nos permite ejecutar comandos, de igual modo a cómo funciona una simple función SSH a un ordenador, es decir, que podemos hacer uso del dispositivo Android como haríamos uso de cualquier dispositivo Linux en su terminal. También realizar transferencia de archivos entre ambos, pudiendo así no solo mandar o recibir ficheros simples, sino que también podemos instalar o desinstalar aplicaciones en Java, o incluso montar y desmontar particiones concretas del sistema teniendo los permisos apropiados.

2.7 Protocolo I2C

El protocolo I2C es un bus serie síncrono multiesclavo y multimaestro que nos permite conectarnos entre distintos dispositivos de forma simple y sencilla. Hoy en día podemos decir que se encuentra entre los más usados, entre los que destacan UART, SPI o CAN.

La idea sobre la que está basada no es ningún concepto nuevo ya que la vemos aplicada a numerosos y distintos tipos de protocolos de comunicación, el Maestro/Esclavo. En principio el protocolo de comunicación es bastante simple, los distintos esclavos serán los múltiples sensores o dispositivos capaces de enviar o recibir información mientras que el maestro será generalmente la CPU encargada de determinar cuál de los esclavos tiene permitido realizar la comunicación.

Por esta razón simultáneamente no puede haber dos esclavos ocupando el bus I2C. Si deseáramos mandar la información de un esclavo a otro, esto habría que hacerlo en 2 partes empleando al maestro como intermediario.

En la primera parte, el primer esclavo se comunicaría con el maestro y este almacenaría la información de un buffer. En la segunda parte el maestro mandaría la información acumulada en el buffer al segundo esclavo, completando así el envío de información.

Si se deseara realizar una comunicación constante este protocolo no sería el indicado dado que el cambio de esclavo y el uso de intermediarios supone un retraso en la comunicación muy elevado, sin embargo, la gran ventaja que obtenemos es que poseemos de un método de comunicación sencillo donde en una situación normal es rápido excepto si se plantea el escenario mencionado previamente.

La comunicación en el bus está basada en 2 líneas, denominadas SDA (Serial Data) y SCL (Serial Clock). Como su propio nombre indica, SDA es la línea por la que es enviada y recibida la información en ambos sentidos, mientras que SCL es la línea que emplea el maestro para establecer el reloj para la transmisión de datos, controlada mediante unos bits iniciales en la comunicación y unos finales, dejando reflejado de esta forma cuando hay una comunicación activa o no evitando así cualquier tipo de conflicto. Tanto el maestro como los esclavos están conectados siempre a ambas líneas.

En nuestro caso hemos utilizado las conexiones GPIO de la Raspberry para realizar una conexión con el protocolo I2C con el Adaptador PiStorms y a través de este una conexión a los sensores y los motores.

Un ejemplo de cómo podría ser un bus I2C (ver Figura 11) donde haya múltiples esclavos podría ser de la siguiente, donde SDA es el canal de datos, SCL el canal del reloj y GND el canal de conexión a tierra:

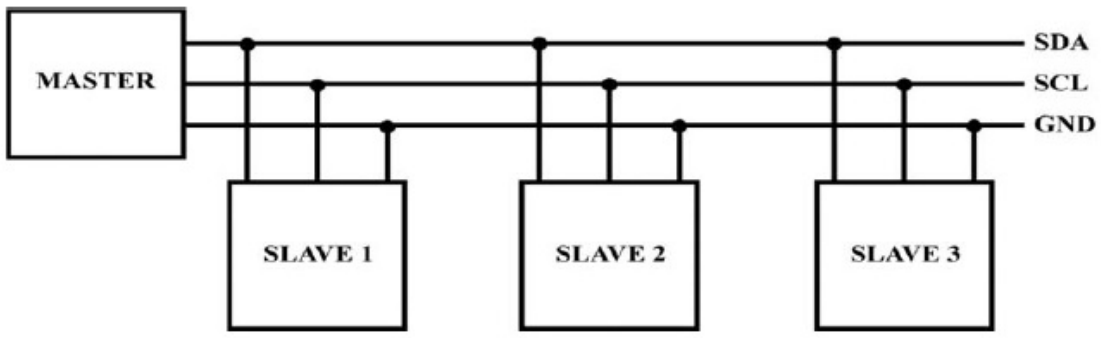

Figura 11: Protocolo I2C

2.8 Compilación Cruzada

La compilación cruzada es una técnica empleada para generar archivos ejecutables para un ordenador, que solemos denominar en inglés *"target"*, con una arquitectura distinta a la de nuestro ordenador de trabajo, denominado habitualmente como *"host"*. Generalmente la solución a este problema sería mandar el código fuente desde "*host*" a "*target*" y hacer que este lo compilase, pero esto no es una opción viable cuando el código a compilar es bastante extenso y el equipo "*target*" no cuenta con demasiada potencia de cómputo para realizar el trabajo sin problemas.

Para guiarnos durante la explicación de cómo funciona un compilador cruzado seguiremos el siguiente esquema, Figura 12:

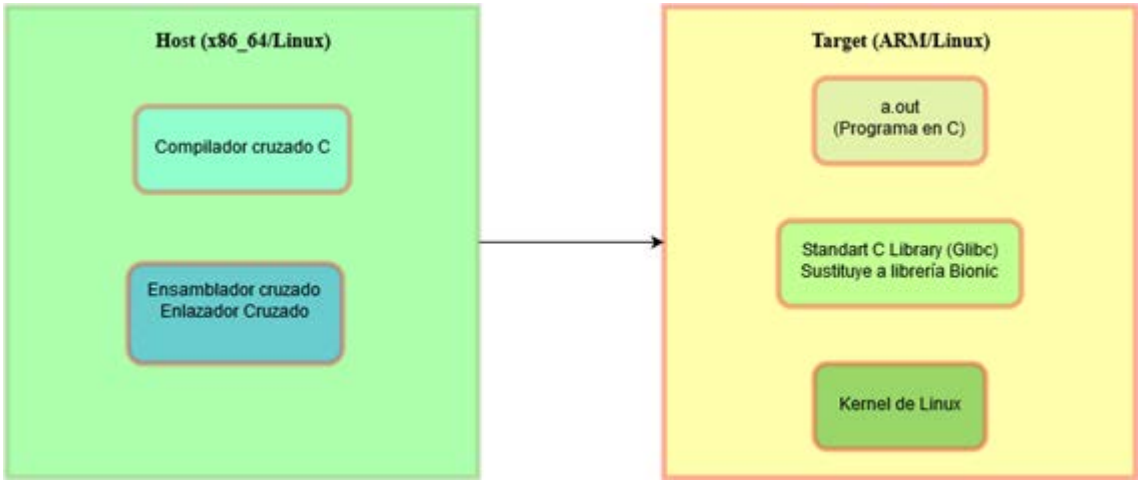

Figura 12: Compilación Cruzada

En nuestro caso el equipo "*host*" se trata de una versión de Linux x86_64. El equipo "*target*" se trata de nuestra Raspberry con un chipset de arquitectura ARM por tanto es necesario hacer la compilación cruzada mencionada previamente.

Como el sistema operativo Android hace uso de manera nativa de la librería Bionic en vez de la glibc, debemos indicar con el comando de compilación que utilice una librería tradicional glibc para ARM/Linux. Previamente deberíamos haber colocado en nuestro "*target*" dicha librería glibc para ARM/Linux y su enlazador dinámico para que el programa en tiempo de ejecución posea todos los recursos necesarios para su funcionamiento. El comando empleado en la compilación es el siguiente.

```
arm-linux-gnueabihf-gcc *.c -o main.o -Wl,--dynamic-
linker=/data/local/lib/ld-linux.so.3 -Wl,-
rpath=/data/local/lib -fPIE -pie
```

Figura 13: Comando de compilación cruzada

2.9 Mecanismos de aislamiento de procesos en núcleos del procesador para Android

A continuación, se detallará brevemente lo explicado en el trabajo previo [1], de manera que comprendamos cómo se deben aplicar los mecanismos que posee Android/Linux para aislar un núcleo de un procesador donde ejecutar aplicaciones con requisitos temporales.

Dado que Android es un sistema operativo muy fragmentado las soluciones encontradas para solventar el problema relacionado con las aplicaciones de tiempo real son muy diversas. Podemos encontrar desde pequeñas modificaciones en el "*kernel*" hasta cambios globales en la estructura, sin embargo, esto no son soluciones a largo plazo debido a que es complicado mantener dichos cambios en siguientes versiones del sistema operativo y la plataforma no se desarrolla específicamente para un tipo de dispositivo, sino que es una plataforma muy variada.

Generalmente cuando hacemos una aplicación de tiempo real hacemos uso de las políticas de planificación de tiempo real disponibles en Android/Linux (SCHED_FIFO y SCHED_RR). Este tipo de planificación nos permite obtener mejoras sustanciales en los tiempos de respuesta, sin embargo, estaremos en competencia con otra aplicación que se encuentre en el sistema ejecutándose haciendo uso de dichas políticas de planificación.

Por ello se buscó una solución portable y que no requiriese de modificaciones a nivel de código en el sistema operativo. Esta solución proponer realizar la ejecución de los procesos de tiempo real en un núcleo aislado del procesador aislado. Esta solución no está sujeta al tipo de dispositivo con el que estemos trabajando y tan solo requiere que poseamos una CPU multinúcleo.

Para poder conseguir esto debemos tener en cuenta cuatro factores:

- Múltiples aplicaciones de tiempo real simultáneamente ejecutándose.
- Interrupciones generadas por los manejadores.
- Cambios dinámicos en la frecuencia de la CPU y apagado automático de los núcleos de la CPU.
- Limitaciones de la librería Bionic, librería C empleada en sistemas Android, en aplicaciones de tiempo real.

Por ello se hace uso de un mecanismo proporcionado para agrupar y restringir recursos de memoria y de CPU en los sistemas Linux, el cual se denomina CPUSET. En el caso de Android esta herramienta debe ser activada a nivel de kernel. Permite asignar procesos a recursos y núcleos concretos de la CPU, dejándonos así asilar procesos concretos en un único núcleo.

También deberemos aislar totalmente posibles interrupciones que pueden ser atendidas potencialmente por los núcleos del procesador, fijar la frecuencia del núcleo aislado y desactivar los demonios que hacen que la CPU apague algunos núcleos automáticamente. Por último, deberemos hacer uso de una librería distinta a Bionic ya que esta no posee

todas las funciones necesarias que necesitaremos en una aplicación de tiempo real, por ello se hace uso de la librería tradicional "*Glibc*".

El script empleado para dejar un núcleo de la CPU totalmente libre de procesos o posibles interrupciones, así como establecer una frecuencia fija y apagar los demonios que apagan automáticamente núcleos de la CPU es el siguiente:

```
#!/bin/sh
#mount -t cpuset cpuset /dev/cpuset/
cd /dev/cpuset
mkdir sys
echo 0-2 > sys/cpus
echo 1 > sys/cpu_exclusive
echo 0 > sys/mems
mkdir rt
echo 3 > rt/cpus
echo 1 > rt/cpu_exclusive
echo 0 > rt/mems
echo 0 > rt/sched_load_balance
echo 1 > rt/mem_hardwall
for i in $(cat tasks); do echo $i > sys/tasks; done

#Disable interrupts on CPU 3 (affinity to CPU 0)
for i in $(find /proc/irq -name "smp_affinity"); do echo 1 >
$i; done
echo 1 > /proc/irq/default_smp_affinity

#Fix frequency CPU 3
echo userspace >
/sys/devices/system/cpu/cpu3/cpufreq/scaling_governor
echo 1200000 >
/sys/devices/system/cpu/cpu3/cpufreq/scaling_setspeed

echo "Haz 'echo $$ > /dev/cpuset/rt/tasks' en el terminal
donde quieras aislar los procesos"
```

Posteriormente a la ejecución de este script tan solo deberemos indicar el PID del proceso padre, en nuestro caso el terminal, de la aplicación de tiempo real y enviarlo al grupo del "*Cpuset*" que se ha quedado sin procesos ni interrupciones, es decir, el grupo aislado.

Las aplicaciones de tiempo real que se ejecutarán en los núcleos aislados deben correr directamente sobre el kernel de Linux/Android y sobre sus librerías nativas de tal modo que evitemos la incertidumbre en los tiempos de respuesta que provocaría ejecutar aplicaciones Java sobre una máquina virtual. Por lo tanto, las aplicaciones del presente trabajo se codificarán en el lenguaje C.

3. Adaptación del Sistema Operativo Android para la Raspberry

3.1 Descripción General

Como producto final del proceso que se describirá a continuación hemos obtenido una versión de Android 7.1.2 que funciona en una Raspberry Pi 3 B. Sin embargo, también es posible desarrollarlo para la Raspberry Pi 2, su predecesora.

Esta versión es un AOSP, o lo que es lo mismo, una versión del sistema que libera Android, y sobre la cual los fabricantes o desarrolladores adaptan su capa de personalización y añaden los drivers necesarios. Por ello, en nuestra versión podemos encontrar ciertas características que no podríamos encontrar en nuestro móvil habitual.

Entre estas destacan el hecho de poseer un acceso a root mediante ADB, por lo tanto, la Raspberry no está totalmente abierta para cualquier usuario, pero sí para los desarrolladores que se comunican mediante dicha tecnología. Por ello decidimos sustituir el código fuente del comando "*su*" que permite modificar el tipo de usuario con el que estamos accediendo empleando el código desarrollado por el código fuente elaborado por Supersu [11] para implementar este comando.

También cabe destacar el hecho que durante la compilación del kernel de la Raspberry se modificaron ciertos parámetros de compilación para permitir que se pudiera acceder al bus I2C y además de esto añadimos la librería i2c-dev de manera que cuando se compilara el sistema operativo pudiéramos hacer uso del bus.

Además de este añadido mencionado de la activación del bus, hicimos posible también que se permitiera el uso del mecanismo denominado "*cpuset*" en el sistema operativo, descrito en el capítulo 2.9. Esto es una herramienta fundamental dado que es la clave para aislar un proceso en un núcleo del procesador. Es decir, permite agrupar procesos en un núcleo deseado del procesador.

Mencionar que dentro de los ya comentados más de 50GB descargado del sistema operativo base desarrollado por Google, apenas ha habido más modificaciones a grandes rasgos, pero cabe destacar que en caso de querer hacerlas podríamos haber cambiado desde las aplicaciones que por defecto vienen con el sistema operativo y no nos deja borrar como son la hora o el calendario, hasta llegar incluso a modificar el código que emplea nuestro teléfono para realizar un emparejamiento bluetooth.

3.2 Proceso de Adaptación de Android para Raspberry Pi 3 b

Para la elaboración de un AOSP tan solo se deben seguir unas pequeñas pautas de guía establecidas por la propia Android, ya que la compañía es la principal propulsora de esta idea de modificar su sistema operativo con el fin de aportar nuevas ideas y mejoras a futuras versiones.

Por ello es tan sencillo como seguir el tutorial que ellos exponen en su página web [16]. Previamente a este paso es necesario obtener dos cosas, el código fuente de la versión del sistema operativo que vayamos a elaborar, el cual está disponible en los repositorios oficiales ofrecidos por Google [10] y el código fuente del kernel del dispositivo en el que

vayamos a utilizar el AOSP. El primer elemento es bastante sencillo hallarlo ya que Google lo libera de manera pública en cada nueva versión de su sistema, para el segundo encontramos un repositorio público en GitHub [9] donde está el código fuente del kernel y los drivers de la Raspberry Pi 3 B.

Una vez obtenidas ambas cosas debemos situarlas en un sistema que ejecute cualquier distribución de Linux ya que facilitara enormemente las cosas. En este caso empleamos una distribución de Ubuntu 16.04 LTS [15], e instalamos en ella las librerías y dependencias necesarias para hacer posible la compilación del sistema.

Completadas las pautas previas proseguimos realizando las modificaciones en el kernel, concretamente en su configuración, la cual posee una serie de variables que determinarán la configuración final del kernel. Aquí es donde activamos el mecanismo denominado "*cpuset*".

Posteriormente añadimos la librería que emplearemos para hacer uso del bus I2C, lib i2c-dev [18]. La incorporación al sistema de la librería requiere añadir la carpeta que contiene el código fuente de esta librería dentro de la sección correspondiente a herramientas externas (directorio *CarpetaDeTrabajo*/tools/external/) que empleará el sistema operativo y a la hora de compilarse se incluirá de manera automática al resto de herramientas nativas de Android.

Adicionalmente a la librería debemos sustituir el actual código fuente que generara el comando "*su*" por uno propio que permita el acceso root. Este código fue obtenido de un grupo, supersu [11], encargado de elaborar kits para liberar el superusuario en múltiples dispositivos con Android. Al sustituir el código fuente (en el directorio *CarpetaDeTrabajo*/system/extras/su) que tenemos originalmente descargado en conjunto con el sistema operativo por el desarrollado por el grupo de supersu, obtenemos los resultados esperados. Debemos tener en cuenta que este paso se debe realizar previamente a la compilación del sistema operativo o de lo contrario no funcionará.

Respecto a las aplicaciones Java instaladas, comúnmente conocidas como APK en dispositivos Android, mencionar que con incluirlas en el directorio *"CarpetaDeTrabajo/package/app/"* correspondiente estarán preinstaladas en la versión inicial del sistema operativo. Entre estas aplicaciones podemos encontrar benchmarks, test de rendimiento, terminal y reproductor de videos.

Por último, solo queda establecer las variables de entorno necesarias e indicar la plataforma objetivo que deseamos desarrollar, en nuestro caso la Raspberry. Una vez realizado este último paso tan solo debemos de indicar que comience el proceso de compilación el cual utilizará todos los recursos del sistema y tomará varias horas.

Una vez generadas las particiones en la tarjeta microSD tan solo es necesario transferir los archivos correspondientes generados al compilar el sistema operativo. Con estos dos pasos previos realizados, la Raspberry será capaz de arrancar el sistema operativo y configurar los dispositivos sin problemas, gracias a un pequeño archivo de texto que incluye una serie de parámetros que se ejecutarán durante el arranque del sistema que permitirán establecer varios tipos de elementos como el tipo de salida de vídeo o la frecuencia de refresco en el bus I2C.

Mencionar que toda esta información viene más detalladamente explicada en un tutorial elaborado y disponible en el Anexo A junto al script encargado de copiar todos los archivos a sus correspondientes particiones en la tarjeta microSD, Anexo B.

3.3 Configuración necesaria

Para poder hacer uso del bus I2C en la Raspberry Pi es necesario activarlo. El proceso de activación consiste en determinar antes de compilar el sistema operativo la configuración correcta del kernel en lo referente al protocolo I2C.

Otro elemento importante a destacar es que por defecto Android desconoce las funciones necesarias para poder hacer uso del bus I2C y por ello es necesario añadir una librería que nos permita hacer uso de este.
Esto se solucionó de la manera más sencilla posible y fue buscando una librería para Android de código abierto que permitirá el uso del protocolo I2C. Tras investigar cuál es la librería [18] empleada por la Raspberry en sus distribuciones de Raspbian, sistema operativo nativo de la Raspberry, se encontró que esta era la librería i2c-dev.h. Tras probar a recompilar todo el sistema operativo incluyendo dicha librería pudimos hacer uso del protocolo I2C como era esperado.

Otra configuración importante que destacar es el hecho de que por defecto nuestro sistema no está "*rooteado*", es decir, no nos permite acceder al superusuario por defecto. Esto suponía que no éramos capaz de desplegar nuestro programa para hacer uso del kit de Lego Mindstorms ya que el acceso al bus I2C por parte de la librería i2c-dev requiere privilegios de superusuario para poder acceder. Debido a esto tuvimos que buscar formas de realizar una modificación en el comando "*su*". Tras investigar cómo funciona el realizar un rooteo o lo que es lo mismo, permitir el acceso a root a todos los usuarios, en versiones de Android disponibles para teléfonos convencionales, encontramos un código fuente del comando "*su*" desarrollado por Supersu [11] que nos permitiría el acceso una vez sustituyéramos el código del comando "su" por el código desarrollado por Supersu [11].

Otra situación problemática encontrada fue que los dispositivos no eran capaces de comunicarse debidamente con la Raspberry a través del bus I2C. Muchas veces llegaban números que no eran correctos o letras en vez de números. Esto se debe a que la frecuencia del bus no estaba correctamente establecida, ya que la Raspberry por defecto hace uso de este bus a unas frecuencias mucho más altas de lo que el kit de Lego Mindstorms es capaz de soportar. Por ello es necesario de reducir o limitar dicha velocidad, la cual finalmente tras investigar distintos métodos se encontró que se podía realizar mediante el archivo de configuración de arranque en la partición de "*system*" de la Raspberry con una simple variable y un valor asociado.

4. Adaptación a Android de la librería de manejo del módulo Pistorms

4.1 Descripción General

En este punto del proyecto contamos con la Raspberry Pi 3 funcionando con el sistema operativo Android y queremos hacer uso de los actuadores y sensores de Lego Mindstorms, sin embargo, la librería elaborada por Lego está desarrollada en Python. Debido a esto se busca adaptar una librería ya desarrollada en otro proyecto [5], la cual está en C, de forma que nos permita utilizarla en cualquier tipo distribución de Linux.

Por ello nuestra librería se ha basado inicialmente en un Trabajo Fin de Grado anterior [5]. La librería original se basa, a su vez, en una librería concreta del chipset de la Raspberry empleada en dicho proyecto, también conocida como BCM2835 [19]. Con ella establecía la conexión directa con el adaptador PiStorms a través del chipset.

En nuestro caso esto es totalmente distinto, ya que queremos inhibirnos de este tipo de dependencias directas con el hardware, haciendo así lo más abierta posible la librería. Sin embargo, podemos utilizar el trabajo previo [5] para elaborar los métodos y funciones encargados de mandar los comandos específicos a los distintos sensores y motores, debido a que estos no cambiarán.

En la Figura 14 se ilustra de manera esquemática cómo estaba planteada la librería original desarrollada en el Trabajo Fin de Grado previo [5].

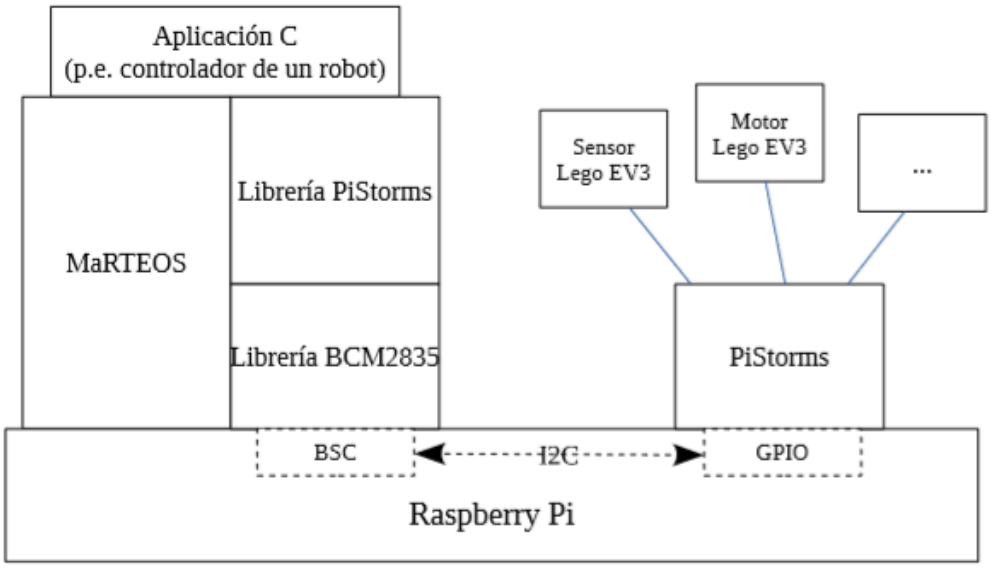

Figura 14: Diagrama de Capas General Original

Como podemos observar este modelo se basaba en un sistema operativo MaRTEOS y hacia uso de la librería BCM2835 para comunicarse con el módulo PiStorms directamente a través del controlador hardware del bus I2C, el Broadcom Serial Controller (BSC).

En nuestro caso sustituiremos el sistema operativo MaRTEOS por Android y haremos que en vez de emplear la librería BCM2835, haga uso de la librería i2c-dev [18]. Esta última es la encargada de establecer la comunicación con el dispositivo. Previamente con la librería BCM2835 la comunicación se realizaba accediendo directamente al controlador hardware del bus I2C mientras que ahora con la librería i2c-dev hacemos uso indirecto del citado controlador a través del fichero de dispositivo de Linux como se ve reflejado en la Figura 15.

Del mismo modo también se adapta la librería original para incluir la funcionabilidad añadida del uso de cámaras como es el caso de la CMUcam5 Pixy [20].

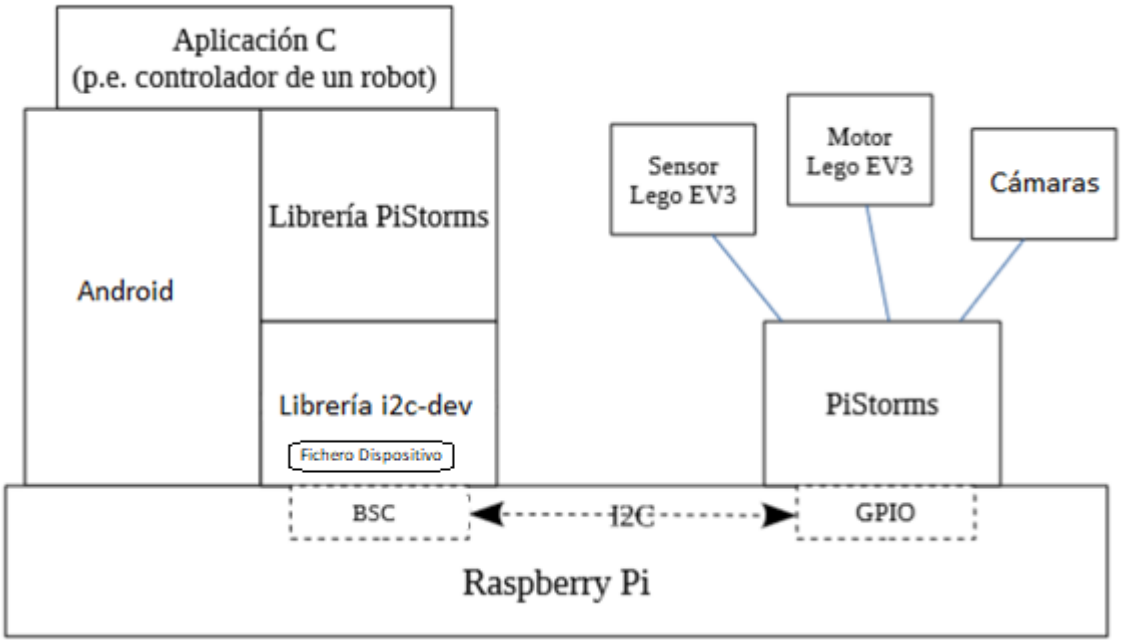

Figura 15: Diagrama de Capas General Actualizado

4.2 Estructura de la Librería

Respecto a la estructura de la librería, como ya hemos mencionado previamente, todo el código del trabajo previo [5] está basado en el chipset BCM2835 [18] y nosotros queremos quitar dicha librería y hacer que se base en la librería i2c-dev disponible en cualquier sistema Linux. Por ello podemos determinar que la estructura de la librería del trabajo previo [5], Figura 16, era de la siguiente forma:

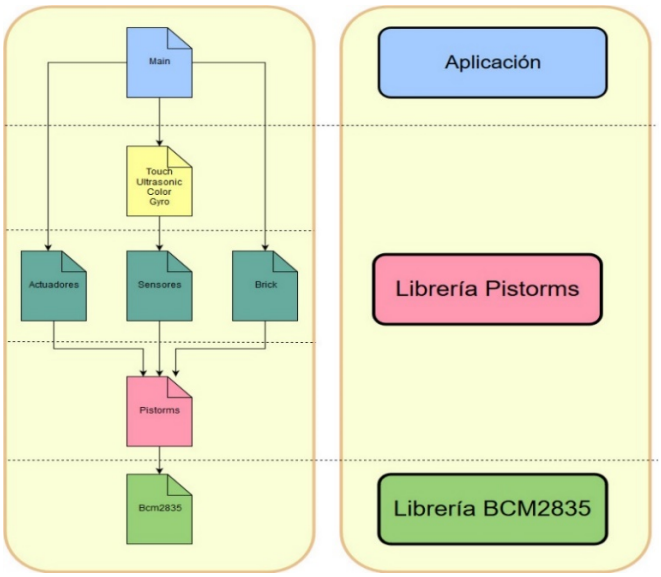

Figura 16: Librería Lego de Carlos Ayerbe

Mientras que la versión nuestra, Figura 17, contará además de con los sensores y motores, con cámaras, que es parte de otro proyecto final de grado [12]. Sustituyendo por completo la capa de Librería BCM2835 y dejando tan solo la librería I2C. Por lo tanto, nuestra estructura tiene la siguiente forma:

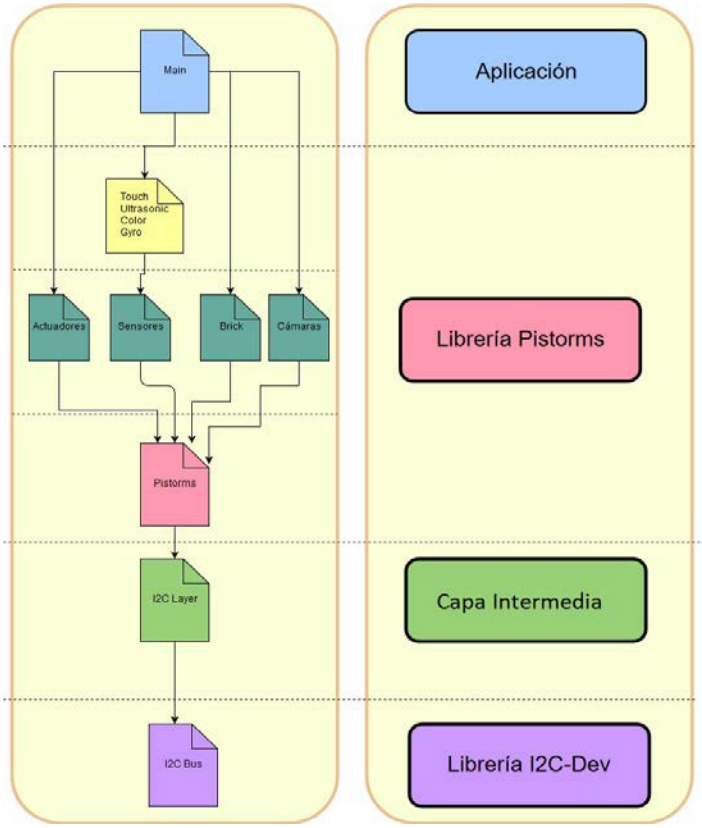

Figura 17: Librería Lego en I2C

4.3 Modificaciones Realizadas

Respecto a las modificaciones realizadas conforme a lo que se hizo en el trabajo previo [5], en dicho caso encontramos que hay distintas operaciones, siendo las funciones de esta lectura, escritura y espera. Todas ellas basadas en la librería BCM2835.

En el proyecto hemos optado por darle un enfoque distinto a la estructura del código planteada en dicho trabajo de manera que podamos controlar todo lo que pasa desde una capa intermedia en el código denominada i2c_layer. En dicha capa hemos elaborado los siguientes métodos:

```
- int getFile();
- char* i2c_read(int file, int reg, int size);
- int i2c_write(int file, int reg, int value);
- int i2c_init(int portNumber);
- int i2c_close(void);
- int i2c_setSlave(int addr);
- void i2c_delay(int millis);
```

La función getFile() permite obtener el identificador de fichero correspondiente al bus I2C para obtener el acceso al bus. Las funciones i2c_read() e i2c_write() permiten leer y escribir a través del bus I2C. Por su parte, la función i2c_init() permite inicializar la conexión con el bus I2C. La función i2c_close() permite cerrar la conexión con el bus I2C. La función i2c_setSlave() consiste en cambiar el dispositivo esclavo con el que el maestro, la CPU, está realizando la comunicación. La función i2c_delay() permite realizar una espera de tiempo determinada.

Respecto a esta capa intermedia elaborada para llevar acabo la adaptación de la librería, i2c_layer, mencionar que es donde encontramos el uso de la librería i2c-dev y aquí podemos encontrar las funciones empleadas de dicha librería. Se han empleado 2, una función para la lectura y otra para la escritura.

La función de lectura es "*i2c_smbus_read_i2c_block_data(file, command, size, output)*". La función de escritura es "*i2c_smbus_write_byte_data(file ,command ,value)*". En ambos casos estas funciones hacen uso de la misma función de la librería i2c-dev, "*i2c_smbus_access(file, read_write, command, size, *data)*".

En lo referente a los parámetros empleados, file es el descriptor de fichero, command es la dirección del registro donde debemos leer o escribir, la cual tiene asociada una acción concreta en el dispositivo. El parámetro value consiste el valor que queremos escribir y el parámetro size es el valor del tamaño de lo que estamos leyendo en bytes. El parámetro output y data hacen la misma función, son buffers donde vamos a almacenar la lectura.

La función *"i2c_smbus_access()"* permite el acceso al bus empleando una llamada IOCTL, la cual es una función empleada en Linux para la comunicación con dispositivos a la hora de desarrollar controladores, de modo que nos permite leer o escribir en un registro concreto en el dispositivo adecuado gracias al descriptor de fichero pasado como parámetro tantos datos como sean necesarios.

Con estos métodos sustituiremos todas las funciones básicas de la Liberia bcm2835 de manera que las sustituciones son 1 a 1:

4.3.1 Lecturas

Sustituimos la función "*bcm2835_i2c_read_register_rs*" por "*i2c_read*". En este caso los parámetros varían totalmente, ya que debemos indicar la dirección del bus, el registro donde queremos leer y el tamaño de la lectura. Además, inicialmente indicábamos donde se debía almacenar la lectura mientras que en el nuevo método se retorna un puntero donde se encuentra la información leída.

4.3.2 Escrituras

Sustituimos la función "*bcm2835_i2c_write*" por "*i2c_write*". En este caso los parámetros varían, ya que debemos indicar además del registro donde vamos a escribir y el valor que queremos escribir el parámetro de file, la dirección del bus I2C.

4.3.3 Inicio

Sustituimos la función "*bcm2835_init*" y "*bcm2835_i2c_begin*" por "*ic2_init*". En este caso tan solo debemos indicar el puerto en el que se encuentra el bus I2C que en nuestro caso es 1.

4.3.4 Fin

Sustituimos la función "*bcm2835_i2c_end*" por "*ic2_close*". En este caso no hay cambios aparentes en los parámetros, pero si en la lógica del método.

4.3.5 Cambio de esclavo

Sustituimos la función "*bcm2835_i2c_setSlaveAddress*" por "*i2c_setSlave*". En este caso, en ambos métodos el parámetro que se pasa es el mismo, siendo la dirección del dispositivo.

4.3.6 Espera

En este caso sustituimos todas las funciones de esperas por "*i2c_delay*" para poder controlar mejor cómo se comporta la espera de manera global en la librería.

5. Prueba de la librería de Lego Mindstorms

5.1 Prueba independiente de sensores y actuadores

En esta primera prueba se comprueba la funcionalidad de los distintos dispositivos y actuadores de Lego. Entre ellos podemos encontrar el “brick” adaptador de PiStorms, los dos motores, el grande y el pequeño, en giroscopio, el sensor de color, el sensor de ultrasonido, el sensor táctil y la cámara [20].

La prueba consistió en verificar que los dispositivos funcionan correctamente y en obtener las medidas de tiempo, tanto para la lectura como para la escritura, de todos los dispositivos teniendo una frecuencia de 80kHz en el bus I2C de la Raspberry Pi asegurándonos de que los valores obtenidos son correctos.

Encontramos que los tiempos a dicha frecuencia son de 0.5ms para la escritura y de 1ms para la lectura aproximadamente para todos los actuadores y sensores excepto para la cámara, la cual requería reducir la frecuencia del bus para que funcionase correctamente a unas cifras muy bajas en comparación, 2,5kHz, provocando esto unos tiempos de lectura y escritura mucho más elevados, aproximadamente 30ms para la escritura y 55 para la lectura.

Teniendo en cuenta que la frecuencia máxima del bus I2C de la Raspberry Pi es de 100kHz podemos asegurar que, a excepción de la cámara, el resto de los sensores y actuadores son capaces de obtener tiempos de respuesta similares a los que obtendríamos empleando el brick EV3 desarrollado por Lego y descrito en el apartado 2.1.

5.2 Demostrador

La demostración elaborada consiste en un robot con una estructura similar a la de un automóvil autónomo, que permitirá un desplazamiento sencillo y la capacidad de localizar obstáculos que se encontraran frente al robot o en las inmediaciones.

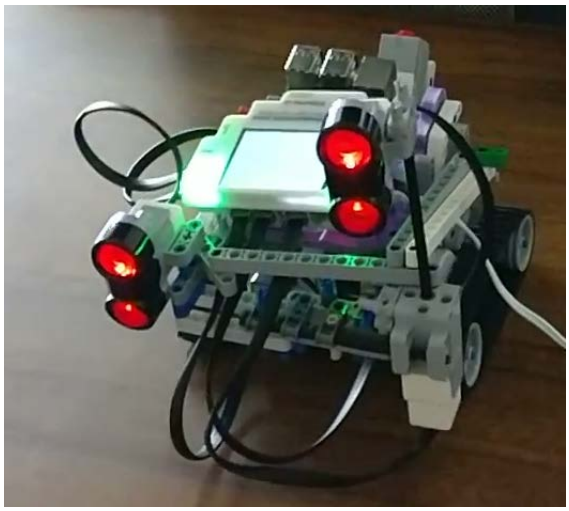

Figura 18: Robot Demo 1

En las figuras 18 y 19 se ilustra cuál es la estructura final que posee el modelo inicial elaborado para comprobar el correcto funcionamiento de las distintas piezas que forman el kit de Lego de Mindstorms.

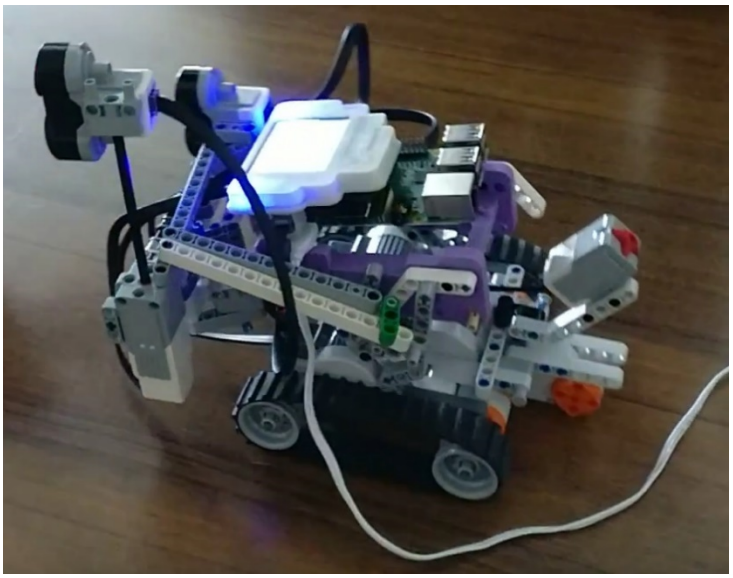

Figura 19: Robot Demo 2

El robot consta de una estructura similar a la de un tanque. Como corazón del prototipo encontramos la Raspberry Pi con el adaptador PiStorms. Posee una tracción de orugas las cuales están provistas de movimiento gracias a dos motores grandes de Lego. Respecto a los sensores, encontramos un sensor táctil en la parte trasera y dos sensores de ultrasonidos en la parte frontal. Uno de ellos se encuentra inmóvil, mientras que el otro puede rotar 360º gracias a la ayuda de un motor pequeño.

Respecto a su funcionamiento, el sensor táctil es el encargado de indicar el inicio y el fin del funcionamiento. Una vez iniciado el funcionamiento el prototipo comienza a avanzar, y se detiene cuando el sensor de ultrasonidos inmóvil detecta un obstáculo. En esta situación, el prototipo se detiene y el sensor ultrasonidos capaz de rotar 360º comienza a determinar los obstáculos en los alrededores y así determina hacia qué lado debe girar el prototipo. Este girará hasta que el sensor de ultrasonidos inmóvil no detecte un obstáculo delante. El proceso se repite indefinidas veces hasta que no pulsemos el sensor táctil para indicar que se finalice el funcionamiento.

5.2 Algoritmo

El programa desarrollado realiza la ejecución periódica del algoritmo de control. La periodicidad se consigue utilizando la función de espera temporizada con tiempo absoluto clock_nanosleep(). En condiciones ideales , el tiempo de ejecución de cada periodo no excede los 10ms gracias a la función de espera temporizada previamente mencionada. Durante la ejecución de cada periodo se repiten las mismas funciones que hacen posible que el prototipo avance y no se choque con obstáculos que pueda encontrar de frente y además los evite de la forma más rápida posible.

El algoritmo es el siguiente:

```
//Comenzamos la comunicación y preparamos sensores y motores.

Iniciamos el brick y el bus
Inicializamos los sensores
Reseteamos todos los parámetros de los motores de desplazamiento


While (true) do

        //Esperamos a que se pulse el sensor táctil para comenzar.

        While (!pulsar sensor táctil) do
                Sleep 5ms
        Fwhile
        Iniciamos el avance de ambos motores de desplazamiento hacia delante.

        //Esperamos a que se pulse el sensor táctil para hacer parar al robot.

        While (!pulsar sensor táctil) do
                While (!detectaObstaculoFrontal) do
                        Sleep 5ms
                Fwhile
                Frenamos los motores de desplazamiento
                Detectar sentido de giro mediante localización de obstáculos
                alrededor del prototipo rotando el sensor.
                Girar al robot sobre sí mismo en el sentido determinado.
                While (detectaObstaculoFrontal) do
                        Sleep 5ms
                Fwhile
                Iniciamos el avance de ambos motores de desplazamiento hacia
                delante.
        Fwhile
Fwhile
```

Como se puede observar a grandes rasgos lo único que realmente hace el robot es entrar y salir de funcionamiento mediante el sensor táctil y una vez se encuentra en funcionamiento, avanza hasta encontrar obstáculos y en ese momento se para y determinar a qué lado se encuentran menos obstáculos y realiza una rotación hacia este hasta detectar que puede volver a reanudar el avance. Esto se repite tantas veces como sea necesario. Los distintos estados por los que va pasando el robot a lo largo del algoritmo se ven reflejados en los colores que toman los leds del adaptador PiStorms.

A pesar de la simplicidad del prototipo, este nos ha servido para demostrar el correcto funcionamiento del entorno y la librería desarrollada. También proporciona un ejemplo sencillo de uso que puede ser utilizado como base para desarrollar otros proyectos robóticos más complicados gracias a la amplia variedad de sensores y motores.

5.3 Funcionalidad Final

Respecto a nuestro objetivo final, que no era otro más que comprobar el funcionamiento correcto de la librería y las velocidades de respuesta frente a la versión original del brick de Lego, pudimos observar muy buenos resultados.

Todos los sensores respondían correctamente como era esperado y además si aumentábamos la frecuencia del bus I2C lo suficiente los tiempos de respuesta eran prácticamente idénticos a los originales, rozando 1ms en lecturas y 0.5ms la escritura.

Dado que esta demo no podía contener todos y cada uno de los dispositivos y sensores de los que dispone y utilizarse de manera útil, los sensores de color y giroscopio fueron comprobados mediante pequeños programas de prueba en los que se comportaban correctamente al igual que las cámaras. Por tanto, pudimos concretar gracias a la demostración que todo funcionaba correctamente.

5.4 Configuración necesaria

Un dato fundamental a tener en cuenta es que debido a que la Raspberry empleada es una versión 3 b, y el adaptador de PiStorms fue pensado originalmente para la Raspberry 1 y 2, el consumo de batería al hacer uso de múltiples motores o sensores es enorme y esto provoca que la Raspberry se quede sin energía en caso de alimentar el sistema a través del propio adaptador PiStorms con baterías, por ello es necesario alimentar al sistema mediante el cable de alimentación tradicional de la Raspberry. Sin embargo, como podemos ver esto suponía que no era posible la entrada en la base acondicionada de Lego preparada para sostener el chip del ordenador, por ello hubo que realizar ciertas modificaciones a dicha base, para permitir entrar el cable si hacer que la Raspberry pudiera salirse.

6. Demostración de Tiempo Real en Android

6.1 Descripción del prototipo

En este caso, el prototipo elaborado en el capítulo 5, no es suficiente para plantear un escenario donde nos encontremos un problema de tiempo real estricto. Por esta razón se plantea desarrollar un estabilizador de cámara. Este dispositivo es empleado en fotografía para mantener la orientación de la cámara estable durante el proceso de captación de imágenes. Existen estabilizadores de dos y tres ejes, pero en nuestro caso fabricaremos uno de un sólo eje, puesto que ya es suficiente para la prueba que queremos realizar.

Para esta demostración vamos a generar un robot, Figura 19 y 20, en el cual podamos colocar un dispositivo de captura de vídeo, en este caso un móvil, y vamos a hacer posible que la imagen generada quede estabilizada gracias a los 2 motores y el giroscopio que controlan la inclinación de la plataforma en la que se encuentra situado el móvil

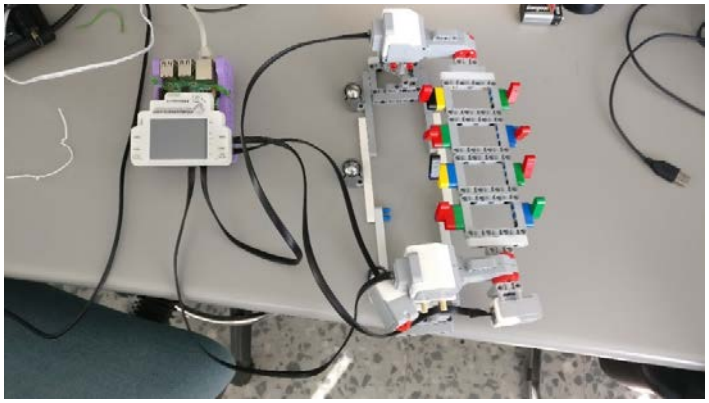

Figura 20: Demostración de estabilizador de cámara 1

La estructura final tomada por el robot es totalmente simétrica. La razón de esto es para garantizar mejor un punto de apoyo, ya que el robot es un estabilizador de cámara, y por tanto en la parte delantera ira colocado el móvil el cual es bastante grande y hace que la estructura ceda hacia delante.

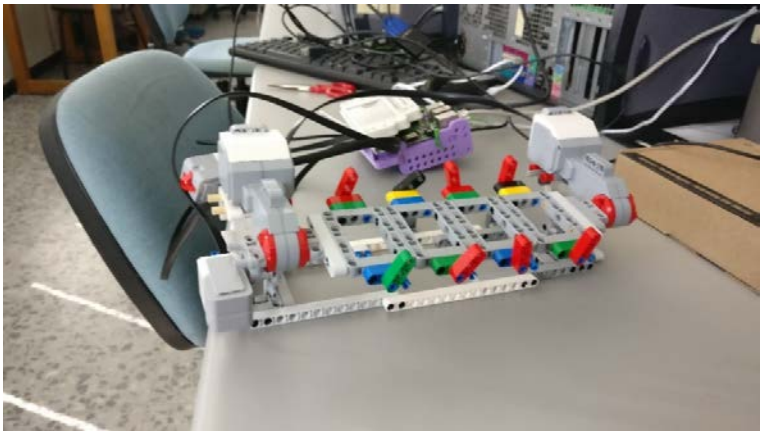

Figura 21: Demostración de estabilizador de cámara 2

La solución a nuestro problema es simple, los contrapesos situados al lado contrario, ya que, pese a que esta herramienta se utiliza en el aire, sujetándolo con ambas manos, el hecho de mantener un punto de apoyo más central nos permite realizar una sujeción más estable y firme.

La idea inicial es que 2 motores sujeten la base sobre la que se coloca el móvil para facilitar la rotación del eje, ya que un solo motor no es capaz de hacer que gire correctamente. Para determinar la diferencia del ángulo hemos empleado el giroscopio el cual está colocado al otro lado de uno de los motores de manera que podemos calcular perfectamente cuando no estamos en la situación deseada.

6.2 Algoritmo

Respecto al algoritmo de control empleado, inicialmente se pensó en utilizar el algoritmo de control proporcional, integral y derivativo (PID). Sin embargo, al realizar las pruebas haciendo uso del algoritmo proporcional, el cual forma parte del algoritmo PID, pudimos ver que los resultados eran satisfactorios y optamos por hacer uso tan solo del control proporcional.

Este algoritmo consiste alcanzar un estado deseado teniendo como parámetro una consigna de entrada. En nuestro caso la consigna de entrada es la inclinación inicial de la plataforma y el error es la diferencia entre la posición actual y la inicial, obtenidas ambas gracias al giroscopio. El valor de error se genera nuevamente a cada periodo del algoritmo de control, siendo en nuestro caso el valor del periodo de control 10ms, duración de tiempo calculada haciendo uso de una espera de tiempo temporizada con tiempo absoluto (clock_nanosleep). En cada periodo haremos que se modifique el estado haciendo uso de los actuadores, de manera que en el siguiente periodo la consigna de entrada cambie haciendo que estemos más cerca de nuestro estado deseado.

En lo que se refiere al cálculo matemático el algoritmo proporcional consiste en multiplicar el error de la orientación por una constante que en nuestro caso es 1. Esto genera que el valor que le pasamos a los actuadores cuanta más diferencia haya de nuestro estado actual con el deseado, mayor será la velocidad a la que los actuadores funcionen dicho periodo.

Este algoritmo de control se ve transformado en el siguiente bloque de código:

```
//Comenzamos la comunicación y preparamos sensores y motores.

Iniciamos el brick y el bus
Inicializamos los sensores
Reseteamos todos los parámetros de los motores de desplazamiento

//Esperamos a que se pulse el sensor táctil para comenzar.
While (true) do
        While (!pulsar sensor táctil) do
                Sleep 5ms
        Fwhile

        Calibramos el giroscopio obteniendo el valor de la inclinación inicial.

        While (!pulsar sensor táctil) do

                Calculamos el error del ángulo actual respecto del deseado.

                Calculamos el valor de potencia que indicaremos a los motores en
                base al algoritmo proporcional.

                Hacemos rotar a los motores con dicha potencia calculada.

                Esperamos hasta alcanzar los 10ms por iteración del bucle.
        Fwhile

Fwhile
```

6.3 Funcionalidad Final

La funcionalidad de un estabilizador de cámara es bastante sencilla, hacer que la imagen sea lo más estable posible mientras grabamos con una cámara. En las versiones profesionales los estabilizadores de cámara poseen esta función no solo en el eje X como en nuestro caso, sino que se pueden configurar para que realice esta misma función en el eje Y, y en el eje Z.

Dado que nosotros tan solo queríamos mostrar un pequeño ejemplo de cómo responde en aplicaciones de tiempo real nuestro sistema Android, esta es una situación ideal, ya que, en caso de no cumplirse correctamente el algoritmo en los tiempos establecidos, el estabilizador de cámara funcionara incorrectamente dejando ver claramente esto en los en los tiempos de ejecución y con menos detalle en los vídeos grabados.

6.4 Datos Recogidos y Conclusión

En lo referente a los experimentos realizados, se ha llevado a cabo la medición de tiempo de ejecución del algoritmo de control del estabilizador de cámara en distintas condiciones para evaluar la influencia del sistema operativo y de las aplicaciones.

Para llevar esto a cabo se ejecutará el algoritmo en un sistema descargado y cargado, a la vez que se ejecutan test de estrés que pretenden cargar al máximo las CPUs de la Raspberry Pi, empleando una espera temporizada de tiempo absoluto de 10ms (clock_nanosleep). Los test de estrés empleados son aplicaciones de minado, MinerGate [17], benchmarks como Antutu [13] y Geekbench [12], y la reproducción de un video en resolución 1920x1080 pixeles a 60 fotogramas.

Esto se realizará sin aplicar y aplicando el mecanismo de aislamiento detallado en el capítulo 2.9. Con ello se pretende comprobar que la ejecución en un procesador aislado sufre muchas menos interferencias viéndose esto reflejado en los tiempos de ejecución, ya que si no aislamos nuestro algoritmo los tiempos de ejecución se verán elevados notoriamente debido a diversas interrupciones generadas tanto por otros procesos como del sistema operativo.

Un dato clave a tener en cuenta durante el experimento es a la frecuencia a la que está trabajando el bus I2C de la Raspberry Pi, siendo en este caso 80kHz. Esto influye en la velocidad de lectura y escritura de nuestro algoritmo. Al establecer 80kHz como frecuencia del bus obtenemos velocidades de lectura y escritura de 1ms y 0,5ms respectivamente, provocando así que el algoritmo tenga una duración inferior a 10ms por periodo.

El experimento se realiza durante cinco minutos en cada una de las situaciones. En cada prueba se registra cuanto ha tardado cada iteración y luego esta queda recogida en un baremo que las distingue según el rango de duración en el que se encuentre, siendo el valor “ALTO” superior a los 35ms el más alto y 11ms el valor más bajo. Además de esto se almacena cual ha sido la medida de tiempo más alta encontrada.

Mencionar que cuando indicamos que cuando aislamos los procesos no solo hacemos uso del *“CPUSET”*, ya que aislamos interrupciones que puedan surgir, establecemos una frecuencia estática a la CPU y desactivamos posibles demonios que puedan apagar núcleos del procesador.

La primera de las pruebas consiste en la ejecución del algoritmo sin carga con aislamiento y posteriormente sin aislamiento:

Con aislamiento y sin carga							
11 ms	15 ms	20 ms	25 ms	30 ms	35 ms	ALTO	Máx Value
24687	0	0	0	0	0	0	10.061614 ms

Tabla 1: Resultados con aislamiento, pero sin carga de trabajo

Sin aislamiento ni carga							
11 ms	15 ms	20 ms	25 ms	30 ms	35 ms	ALTO	Máx Value
22369	69	6	0	0	0	0	16.912917 ms

Tabla 2: Resultados sin aislamiento ni carga de trabajo

Como podemos ver es bastante claro que las interrupciones del sistema operativo únicamente ya afectan mucho a la aplicación de control sin aislamiento incluso cuando el sistema se encuentra descargado, ya que usando el aislamiento no hay ni una medida de tiempo que supere los 10ms mientras que en el caso contrario encontramos múltiples situaciones.

Los resultados posteriores son con carga adicional, como pueden ser test de rendimiento como GeekBench [12] o Antutu [13], una aplicación de minado, MinerGate [17] y la reproducción de un video en 1080p a 60 fotogramas por segundo. De nuevo los primeros resultados son con el aislamiento:

MinerGate							
11 ms	15 ms	20 ms	25 ms	30 ms	35 ms	ALTO	Máx Value
30839	0	0	0	0	0	0	10.111250 ms

Tabla 3: Resultados con aislamiento y ejecutando Minergate

GeekBench							
11 ms	15 ms	20 ms	25 ms	30 ms	35 ms	ALTO	Máx Value
30117	0	0	0	0	0	0	10.063333 ms

Tabla 4: Resultados con aislamiento y ejecutando GeekBench

Antutu							
11 ms	15 ms	20 ms	25 ms	30 ms	35 ms	ALTO	Máx Value
28073	0	0	0	0	0	0	10.236790 ms

Tabla 5: Resultados con aislamiento y ejecutando Antutu

Video 1080@60fps							
11 ms	15 ms	20 ms	25 ms	30 ms	35 ms	ALTO	Máx Value
22014	0	0	0	0	0	0	10.210469 ms

Tabla 6: Resultados con aislamiento y reproduciendo video a 1080@60fps

Como podemos observar, el aislamiento funciona perfectamente ya que ni una iteración supera los 11ms, estando así en el tiempo deseado para cada iteración al no vernos afectados por ningún tipo de interferencia durante la ejecución.

Mientras que estos resultados sin el aislamiento son los siguientes:

MinerGate							
11 ms	15 ms	20 ms	25 ms	30 ms	35 ms	ALTO	Máx Value
27851	109	6	1	0	0	1	41.236875 ms

Tabla 7: Resultados con espera y ejecutando Minergate pero sin aislamiento

GeekBench							
11 ms	15 ms	20 ms	25 ms	30 ms	35 ms	ALTO	Máx Value
27569	23	7	1	0	0	0	22.511510 ms

Tabla 8: Resultados con espera y ejecutando GeekBench pero sin aislamiento

Antutu							
11 ms	15 ms	20 ms	25 ms	30 ms	35 ms	ALTO	Máx Value
26489	215	20	5	1	3	0	29.123852 ms

Tabla 9: Resultados con espera y ejecutando Antutu pero sin aislamiento

Video 1080@60fps							
11 ms	15 ms	20 ms	25 ms	30 ms	35 ms	ALTO	Máx Value
19748	43	4	0	0	0	0	18.565729 ms

Tabla 10: Resultados con espera y reproduciendo video a 1080@60fps pero sin aislamiento

Como podemos ver a simple vista, en todos los casos donde se ejecutó la prueba ya fuera con carga o sin ella, si se empleaba el aislamiento el tiempo era siempre el esperado, aproximadamente 10ms. Y nunca este se salía de dicho rango, dejando claro que es viable la ejecución de aplicaciones de tiempo real ya que no hay riesgo de pérdida de datos, y esto se comprueba fácilmente viendo el video grabado en ambos casos, con y sin aislamiento, la diferencia es notoria.

Por otro lado, podemos ver que en algunas de las pruebas podemos encontrarnos que alguna iteración ha llegado a cuadriplicar el tiempo esperado de una iteración siendo esto una situación insostenible. Con esto podemos concluir fácilmente que los sistemas operativos Android por defecto no están preparados para ejecutar aplicaciones de tiempo real por sí mismos, pero aplicando la solución explicada en el trabajo previo [1] esto se ve solucionado fácilmente.

7. Conclusiones y Trabajos futuros

Podemos concluir una vez finalizado el proyecto que los resultados obtenidos son los esperados. Hemos conseguido configurar una Raspberry Pi funcionando en un sistema operativo Android, lo cual supone innumerables posibilidades en lo que a futuros proyectos se refiere ya que esto permite trabajar en un entorno conocido. También hemos comprobado que en caso de ser necesario realizar modificaciones o personalizar el sistema operativo para cumplir distintos requisitos en dichos posibles proyectos es posible.

En lo referente a la librería adaptada para Lego Mindstorms, podemos decir que es un gran avance, ya que da la posibilidad de elaborar demostraciones o simulaciones con robots de un caso real de manera sencilla con tan solo una interfaz I2C y un sistema operativo basado en Linux. Estos 2 requisitos podemos encontrarlos en abundantes cantidades de dispositivos actualmente y brindan innumerables opciones a explorar en posibles trabajos futuros.

Mencionar que efectivamente somos capaces de hacer uso de aplicaciones de tiempo real empleando la técnica de aislamiento mencionada en el trabajo previo [1]. Esto supone que podemos elaborar numerosos proyectos en este campo haciendo uso de un sistema operativo Android, el cual podemos encontrar ampliamente extendido hoy en día.

El presente proyecto nos deja abiertas algunas líneas exploratorias futuras que consisten en la adaptación de este trabajo para que pueda ser utilizado en otras placas de desarrollo que actualmente se comercializan, como por ejemplo Odroid [8] y Banana Pi [7]. Ambas cuentan con las especificaciones necesarias para poder llevar a cabo este proyecto, es decir, poseen un procesador multinúcleo, una interfaz entrada/salida para realizar la comunicación con el adaptador PiStorms y se pueden inicializar empleando el sistema operativo Android. Por otro lado, sería deseable llevar a cabo una comparación de la solución para la ejecución de aplicaciones de tiempo real en Android planteada en el trabajo previo [1] con respecto a otras soluciones alternativas desarrolladas por otros grupos de investigación.

8. Bibliografía

[1]: **CPU Isolation on the Android OS for running Real-Time Applications.** Alejandro Pérez Ruiz, Mario Aldea Rivas and Michael González Harbour 13th International Workshop on Java Technologies for Real-time and Embedded Systems - JTRES 2015, París (France), October, 2015. DOI: 10.1145/2822304.2822317

[2]: **Lego Mindstorms**
https://www.lego.com/es-es/mindstorms
Visitada por última vez el 22/05/2018

[3]: **Raspberry Pi**
https://www.raspberrypi.org/
Visitada por última vez el 22/05/2018

[4]: **Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux**
Derek Molloy
ISBN: 978-1-119-18868-1
Jun 2016

[5]: **Manejadores de Sensores y Actuadores Lego Mindstorms para Raspberry Pi y Marte OS**.
Carlos Ayerbe González
http://hdl.handle.net/10902/12267
Visitada por última vez el 22/05/2018

[6]: **The I2C Bus: From Theory to Practice**
Dominique Paret, Carl Fenger
ISBN: 978-0471962687
1997

[7]: **Banana Pi**, página oficial
http://www.banana-pi.org/
Visitada por última vez el 22/05/2018

[8] **Odroid C2**, página oficial
http://www.hardkernel.com/main/main.php
Visitada por última vez el 22/05/2018

[9]: **Repositorio de Github Android-Rpi**
Igor Kalkov, Peter Yoon y Sahaj Sarup.
https://github.com/android-rpi
Visitada por última vez el 22/05/2018

[10]: **Repositorio de Google Android**
https://android.googlesource.com/platform/manifest/+/refs/heads/android-7.1.2_r19
Visitada por última vez el 22/05/2018

[11]: **Supersu, grupo encargado de "*unrootear*" dispositivos**
http://www.supersu.com
Visitada por última vez el 22/05/2018

[12]: **Manejador de bajo nivel para cámaras con procesamiento de imágenes integrado**
Ángel Pérez Fuente
http://hdl.handle.net/10902/12613
Visitada por ultima vez el 22/05/2018

[13]: **Geekbench, benchmark**
https://www.geekbench.com/
Visitada por ultima vez el 22/05/2018

[14]: **Antutu, benchmark**
http://www.antutu.com/en/index.htm
Visitada por ultima vez el 22/05/2018

[15]: **Sistema Operativo Ubuntu 16.04 LTS**
https://www.ubuntu.com/desktop/features
Visitada por ultima vez el 22/05/2018

[16]: **Tutorial para elaborar un AOSP**
https://source.android.com/setup/build/building
Visitada por ultima vez el 22/05/2018

[17]: **Minergate**
https://es.minergate.com/
Visitada por ultima vez el 22/05/2018

[18]: **Librería i2c-dev, repositorio**
https://github.com/suapapa/i2ctools
Visitada por ultima vez el 22/05/2018

[19]: **Chipset BCM2835, especificación**
https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf
Visitada por ultima vez el 22/05/2018

[20]: **CMUcam5 Pixy**
http://www.cmucam.org/projects/cmucam5
Visitada por ultima vez el 22/05/2018

Anexo A: Tutorial de elaboración e instalación AOSP

Compilar Android 7.1.2 para Raspberry Pi 3

https://github.com/android-rpi/device_brcm_rpi3

 # Descargamos todas las librerias paquetes y dependecias necesarias:

```
$ cd ~
$ mkdir RaspAndroid712
```

 # Descargamos todas las librerias paquetes y dependecias necesarias:

```
$ sudo apt-get install git-core gnupg flex bison gperf build-essential \
  zip curl zlib1g-dev gcc-multilib g++-multilib libc6-dev-i386 \
  lib32ncurses5-dev x11proto-core-dev libx11-dev lib32z-dev ccache \
  libgl1-mesa-dev libxml2-utils xsltproc unzip
$ sudo apt-get install phablet-tools
$ sudo apt-get install python-mako
$ sudo apt-get install gcc-arm-linux-gnueabihf
```

 # Creamos la carpeta de trabajo y descargamos el tree de android 7.1.2 para la Raspberry Pi 3 (Unos 56GB de descarga aproximadamente)

```
 $ mkdir RaspAndroid712
 $ cd RaspAndroid
 $ repo init -u https://android.googlesource.com/platform/manifest -b android-
7.1.2_r19
 $ cd .repo
 $ git clone https://github.com/android-rpi/local_manifests -b nougat
 $ cd ..
 $ repo sync
```

 # Estabelcemos la configuracion necesaria para nuestro proyecto

```
$ cd ./kernel/rpi
 $ ARCH=arm scripts/kconfig/merge_config.sh
arch/arm/configs/bcm2709_defconfig android/configs/android-base.cfg
android/configs/android-recommended.cfg
```

#Asegurarse que CONFIG_CPUSETS=y, CONFIG_PREEMPT=y #CONFIG_CC_STACKPROTECTOR_STRONG=n, CONFIG_DEVKMEM =y en RaspAndroid/kernel/rpi3/.config

#También nos aseguramos de que todas las variables que contengan i2c están =y

 # Para que los CPUSET estén realmente activos debemos añadir en el fichero cmdline.txt la siguiente línea:

cgroup_enable=cpuset

 # RaspAndroid/prebuilts/sdk/tools/jack-admin en la linea 454 añadimos el siguiente parametron al commando para garantizar que emplea 6GB de ram la máquina de java que se levante.

“-Xmx6G“

 # Compilamos todo

$ cd ./kernel/rpi
$ ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- make -j4 zImage modules dtbs

 # Aplicamos cambios de config indicados en

Patch para el Settings Crash
https://github.com/android-rpi/device_brcm_rpi3/wiki/Runtime-Errors#settings-crash
https://github.com/android-rpi/device_brcm_rpi3/wiki#use-hal_pixel_format_bgra_8888

 # Añadimos la librería I2C
#Encontramos toda la información necesaria en estos enlaces:

https://stackoverflow.com/questions/19763831/building-i2c-tools-on-android

https://boundarydevices.com/i2c-tools-under-android/

Empleamos el repositorio https://github.com/suapapa/i2c-tools , descargando su contenido y descomprimiendolo en:

RaspAndroid712/tools/external/i2c-tools-3.1.0

 # Creamos la imagen de android 7.1.2

$ cd RaspAndroid712
$ source build/envsetup.sh;
$ lunch rpi3-eng;

 # En caso de tener más de 2 cores, hacer -jX donde X es el número de hilos generados

$ make -j8 ramdisk systemimage

 # Preparamos la tarjeta microSD para la instalación de manera que hacemos lo siguiente con Gparted:

1- Elminiamos cualquier partición y dejamos la tarjeta vacia.
2- Creamos 4 particiones, 3 de 512MB y 1 del tamaño restanto, siendo los formatos fat32 la primera particion y ext4 el resto, y los nombres boot, system, cache, userdata respectivamente.
3- Aplicamos los cambios anteriores y tan solo falta añadir a la particion boot la opcion de boot.

#Instalamos las distintas particiones y todos los modulos del kernel necesarios

```
#!/bin/sh
cd ~/RaspAndroid712
echo "Copiando todo a System"
sudo dd if=out/target/product/rpi3/system.img of=/dev/sdb2 bs=1M
cd kernel/rpi
sudo rm -r temp
mkdir temp
sudo mount /dev/sdb2 temp/
echo "Compilando kernel y copiandolo en la raspberry"
sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
INSTALL_MOD_PATH=temp modules_install
sudo umount temp/
sudo rm -r temp/
cd ~/RaspAndroid712
echo "Instalando todo en las distintas particiones"
sudo mount /dev/sdb1 temp/
sudo cp device/brcm/rpi3/boot/* temp/
sudo cp kernel/rpi/arch/arm/boot/zImage temp/
sudo cp kernel/rpi/arch/arm/boot/dts/bcm2710-rpi-3-b.dtb temp/
sudo mkdir temp/overlays
sudo cp kernel/rpi/arch/arm/boot/dts/overlays/vc4-kms-v3d.dtbo temp/overlays/vc4-
kms-v3d.dtbo
sudo cp out/target/product/rpi3/ramdisk.img temp/
sudo umount temp/
rm -r temp
```

 # Añadir al archivo en *la particion de boot config.txt,* el primer comando en la linea 15 y el resto al final del archivo.

gpu_mem=256
dtparam=i2c_arm=on
dtparam=i2c1=on
dtparam=i2c_baudrate=80000 #Este valor es para aumentar la frecuencia del bucle i2c y por tanto reducir el clock de modo que permita leer correctamente.
#http://www.mindsensors.com/blog/how-to/change-i2c-speed-with-raspberry-pi

#La ip de la raspberry la encontraremos en

cat /var/lib/misc/dnsmasq.leases

#Conexion:

#Ahora tan solo basta con instalar adb propiamente

$ apt-get install adb

#Ahora tan solo basta con usar adb en el terminal, por tanto, solo nos queda para realizar una conexión inicial sabiendo la ip:

$ adb connect 10.42.0.190
$ adb shell

 # En este momento nos situamos como si fuera un ssh normal a cualquier pc y por tanto debemos definir nuestra zona de trabajo, que sera en este caso:

/data/local/

#Para instalar un apk bastaria con realizar un:

adb install prueba.apk

#Debido a que queremos realizar compilaciones cruzadas debemos de añadir las librerias necesarias en este caso, compilaremos desde el host empleando el siguiente comando, entendiendo que las librerias que ponemos en la compilacion cruzada asi como el linker ya las hemos colocado en la Raspberry:

arm-linux-gnueabihf-gcc helloWorld.c -o helloWorld -Wl,--dynamic-linker=/data/local/lib/ld-linux.so.3 -Wl,-rpath=/data/local/lib -fPIE -pie

#Mandamos la librería a la raspberry para poder ejecutar todo de manera que se emplee el linker dinamico

En el host hacemos lo siguiente:
cd /usr/arm-linux-gnueabihf
adb connect 10.42.0.204
adb push lib/ /data/local/
adb shell
#Ahora dentro de la rapsberry
cd /data/local/lib

#Posteriormente seria mandar el archivo helloWorld.o generado al dispositivo, esto sería con la ayuda del comando:

adb push helloWorld /data/local/tmp

#Mencionar que el adaptador PiStorms de Mindstorms necesita pulsar el botón go para encederse, aunque este alimentando a la Raspberry. Si no los programas que utilicen el i2c darán fallos de segmentos al no considerar enchufado nada al i2c.

Anexo B: Script copia archivos a tarjeta microSD

```
#!/bin/sh

cd ~/RaspAndroid712
echo "Copiando todo a System"
sudo dd if=out/target/product/rpi3/system.img of=/dev/sdb2 bs=1M
cd kernel/rpi
sudo rm -r temp
mkdir temp
sudo mount /dev/sdb2 temp/
echo "Compilando kernel y copiandolo en la raspberry"
sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
INSTALL_MOD_PATH=temp modules_install
sudo umount temp/
sudo rm -r temp/
cd ~/RaspAndroid712
sudo rm -r temp
mkdir temp
echo "Instalando todo en las distintas particiones"
sudo mount /dev/sdb1 temp/
sudo cp device/brcm/rpi3/boot/* temp/
sudo cp kernel/rpi/arch/arm/boot/zImage temp/
sudo cp kernel/rpi/arch/arm/boot/dts/bcm2710-rpi-3-b.dtb temp/
sudo mkdir temp/overlays
sudo cp kernel/rpi/arch/arm/boot/dts/overlays/vc4-kms-v3d.dtbo temp/overlays/vc4-
kms-v3d.dtbo
sudo cp out/target/product/rpi3/ramdisk.img temp/
sudo umount temp/
rm -r temp
```