UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

PROGRAMACIÓN DE INTERFAZ HOMBRE MÁQUINA EN SISTEMA OPERATIVO ANDROID PARA CONTROL Y MONITOREO INALÁMBRICO DE SISTEMAS A BASE DE MICROCONTROLADOR

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO MECÁNICO ELECTRICISTA

P R E S E N T A:

JUAN LUIS GONZÁLEZ BELLO

ASESOR: JOSÉ LUIS BARBOSA PACHECO

CUAUTITLÁN IZCALLI, ESTADO DE MÉXICO, 2017

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
UNIDAD DE ADMINISTRACIÓN ESCOLAR
DEPARTAMENTO DE EXÁMENES PROFESIONALES

U. N. A. M.
FACULTAD DE ESTUDIOS
SUPERIORES CUAUTITLAN

ASUNTO: VOTO APROBATORIO

M. en C. JORGE ALFREDO CUÉLLAR ORDAZ
DIRECTOR DE LA FES CUAUTITLAN
PRESENTE

ATN: I.A. LAURA MARGARITA CORTAZAR FIGUEROA
Jefa del Departamento de Exámenes Profesionales
de la FES Cuautitlán.

DEPARTAMENTO DE
EXAMENES PROFESIONALES

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos el: **Trabajo de Tesis**

Programación de intefaz hombre máquina en sistema operativo Android para control y monitoreo inalámbrico de sistemas a base de microcontrolador

Que presenta el pasante: **JUAN LUIS GONZÁLEZ BELLO**
Con número de cuenta: **40900726-6** para obtener el Título de la carrera: **Ingeniería Mecánica Eléctrica**

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO**.

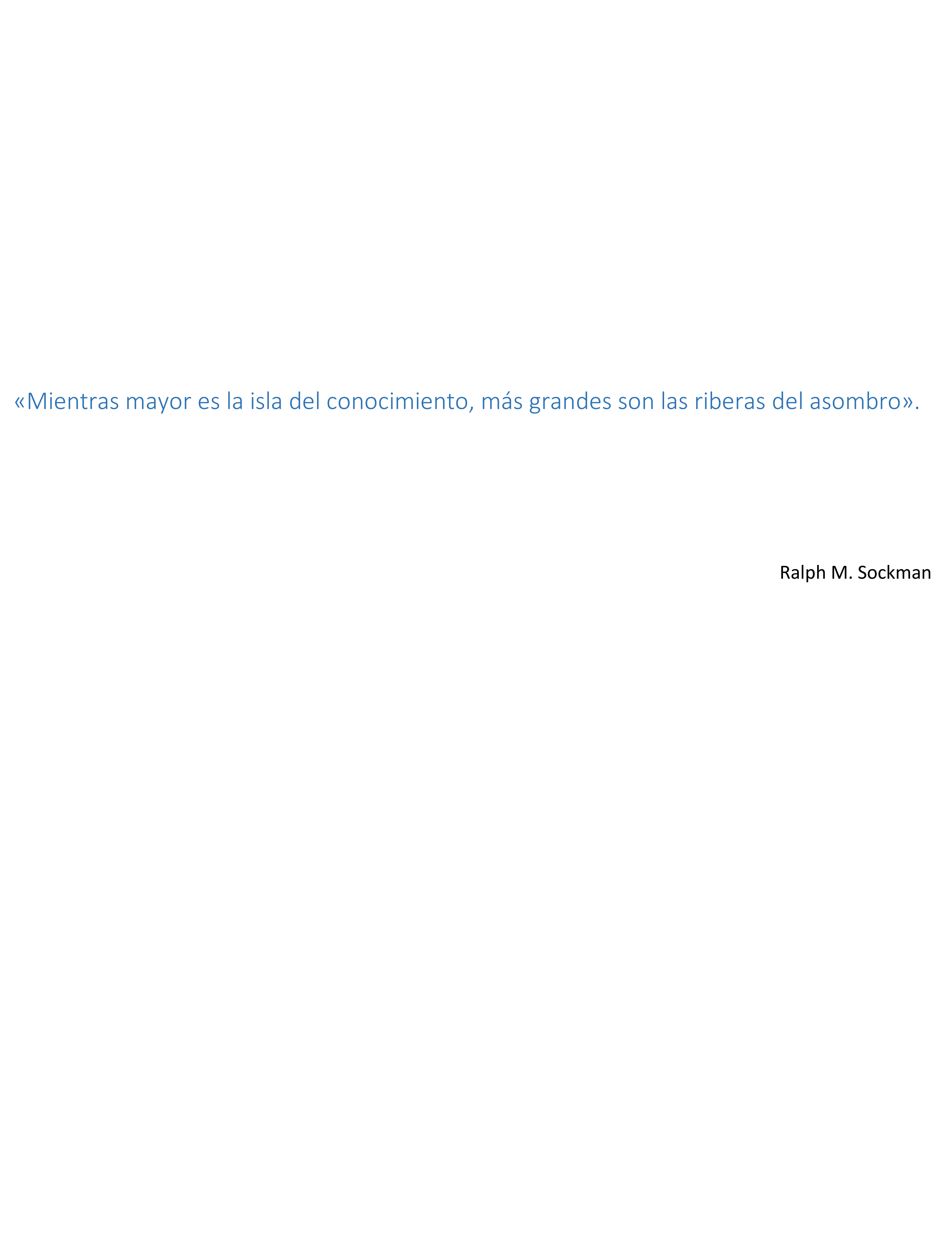
ATENTAMENTE
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 08 de agosto de 2017.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	M. en I. Jorge Buendía Gómez	
VOCAL	Ing. José Luis Barbosa Pacheco	
SECRETARIO	Ing. Marcelo Bastida Tapia	
1er. SUPLENTE	M. en C. Leopoldo Martín Del Campo Ramírez	
2do. SUPLENTE	Dr. David Tinoco Varela	

NOTA: los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).

LMCF/ntm*



«Mientras mayor es la isla del conocimiento, más grandes son las riberas del asombro».

Ralph M. Sockman

Agradecimientos

"A la Universidad, por haberme dotado de las herramientas necesarias para desarrollarme como profesionista. Deseo también expresar todo mi agradecimiento a mi familia, por su apoyo incondicional a lo largo de este trayecto. Por último, dar las gracias a mis amigos y compañeros de trabajo que, de un modo u otro, han respaldado este esfuerzo."

Tabla de contenido

Objetivo 7

Introducción 8

Capítulo 1. Antecedentes y Marco Teórico 11

Antecedentes 11

Interfaz Hombre Máquina 11

Sistema Operativo Android 12

El microcontrolador 16

Disponibilidad de software y hardware en el mercado 19

Marco Teórico 24

Modelo OSI 24

Bluetooth 30

Wi-Fi 33

Aspectos fundamentales de una aplicación en Android 36

Capítulo 2. La Interfaz Gráfica en Arduino Total Control 41

Clases de la Aplicación Arduino Total Control 42

La actividad principal en ATC 43

Elementos constructivos 45

La función BuildLayout 45

Botones de activación 45

Textos 47

Barras deslizantes 49

Imágenes 50

Displays analógicos 56

Herramientas de edición primarias 60

Herramientas de edición secundarias 62

Exportar un layout 62

Importar un layout 63

Capítulo 3. Conectividad en ATC 65

Filosofía de control 66

Bluetooth 66

Emparejar un dispositivo 66

Conexión de un dispositivo Bluetooth 69

Enviar información 71

Recibir información 71
Bluetooth Low Energy (BLE) 72
El demo Bluetooth LEGATT 73
Descubrimiento de dispositivos 73
Conexión 73
Recepción de información 74
Envío de información 75
Wi-Fi (TCP/IP) 75
Configuración de dirección y puerto 76
Conexión 77
Recibir información 78
Enviar información 79
Protocolo de Capa de Aplicación ATC 79
Envío de información a nivel de Capa de Aplicación 79
Recepción de información a nivel de Capa de Aplicación 80
Raw Data 80
ATC Tags 81
Capítulo 4. El Sistema a base de microcontrolador. 83
Hardware mínimo 83
Bluetooth Clásico (BR/EDR) / Bluetooth Low Energy 85
Wi-Fi 86
Estructura de un programa en Arduino 87
Software mínimo 88
Inicialización, envió y recepción de información Bluetooth EDR/BR y Bluetooth LE 91
Inicialización, envío y recepción de información Wi-Fi 92
Capítulo 5. Pruebas funcionales y prototipos. 98
Pruebas funcionales 98
Pruebas de comunicaciones inalámbricas 98
Pruebas de protocolo de Capa de Aplicación ATC 107
Prototipos compatibles con ATC 117
ATC RC Car (Módulo de comunicaciones: HC05) 117
Robot caminante ATC (Módulo de comunicaciones: HC05) 125
Tarjeta de entradas y salidas de propósito general (Módulo de comunicaciones: HM10) 127
Cerradura electrónica ATC (Ethernet Shield) 130
Discusión de resultados 132

Conclusiones **138**

Apéndices **139**

Apéndice A. Objeto Text to Speech en ATC 139

Apéndice B. Conexión fallida a módulo kc-4114 141

Apéndice C. Comandos AT para módulos HC-05 y HC-06 144

Apéndice D. Comandos AT para HM10 146

Apéndice E. Comandos AT para ESP8266 147

Bibliografía **148**

Anexos **152**

Anexo 1. Sketch bt_firmware.ino 152

Anexo 2. Sketch eth_firmware.ino 155

Anexo 3. Sketch esp_firmware.ino 159

Anexo 4. Sketch yun_firmware.ino 164

Anexo 5. Sketch ethernet_shield_test.ino 167

Anexo 6. Sketch eth_firmware_ap_layer_test.ino 168

Anexo 7. Sketch bt_ATC_RC_CAR.ino 171

Anexo 8. Sketch bt_ATC_Walker.ino 178

Anexo 9. Sketch ble_generalIO.ino 184

Anexo 10. Sketch eth_doorlock.ino 190

Glosario **194**

Índice de tablas y figuras **196**

Objetivo

El objetivo de este trabajo es documentar el desarrollo de una Aplicación Interfaz Hombre Máquina (*Human Machine Interface*, o “HMI” por sus siglas en inglés), programada en Java sobre la plataforma de dispositivos móviles con Sistema Operativo Android; así como el diseño y construcción de prototipos compatibles con la aplicación, para el monitoreo y control de sistemas a base de microcontrolador (*Microcontroller Unit*, o MCU de sus siglas en inglés). La Figura 1 representa las interacciones entre el sistema a base de microcontrolador y la aplicación desarrollada en este trabajo.

Objetivos Secundarios:

- Desarrollar una aplicación en Android para el control y monitoreo inalámbrico de sistemas digitales compatible con Bluetooth, Bluetooth LE y Wi-Fi.
- Desarrollar un protocolo de comunicación de Capa de Aplicación lo suficientemente sencillo y robusto para transmitir información y comandos entre el dispositivo inteligente Android y el sistema a base de microcontrolador.
- Desarrollar una aplicación personalizable, con los elementos suficientes para controlar los sistemas físicos más comunes (Iluminación, Control de Acceso, Vehículos, Nivel de contenedores, Motores, Bombas, etc.).
- Diseñar y construir (para cada uno de los estándares de comunicación inalámbrica soportados) un circuito a base de microcontrolador capaz de codificar y decodificar la información proveniente de y hacia la aplicación, con la función de operar entradas y salidas digitales y analógicas.
- Implementar funciones para exportar e importar las pantallas o “Layouts” creados en la aplicación, con la finalidad de compartir o archivar determinada configuración de HMI.
- Publicar un repositorio en GitHub, como fuente principal para obtener los *Sketches* compatibles con *Arduino* para la implementación de los usuarios de la Aplicación.

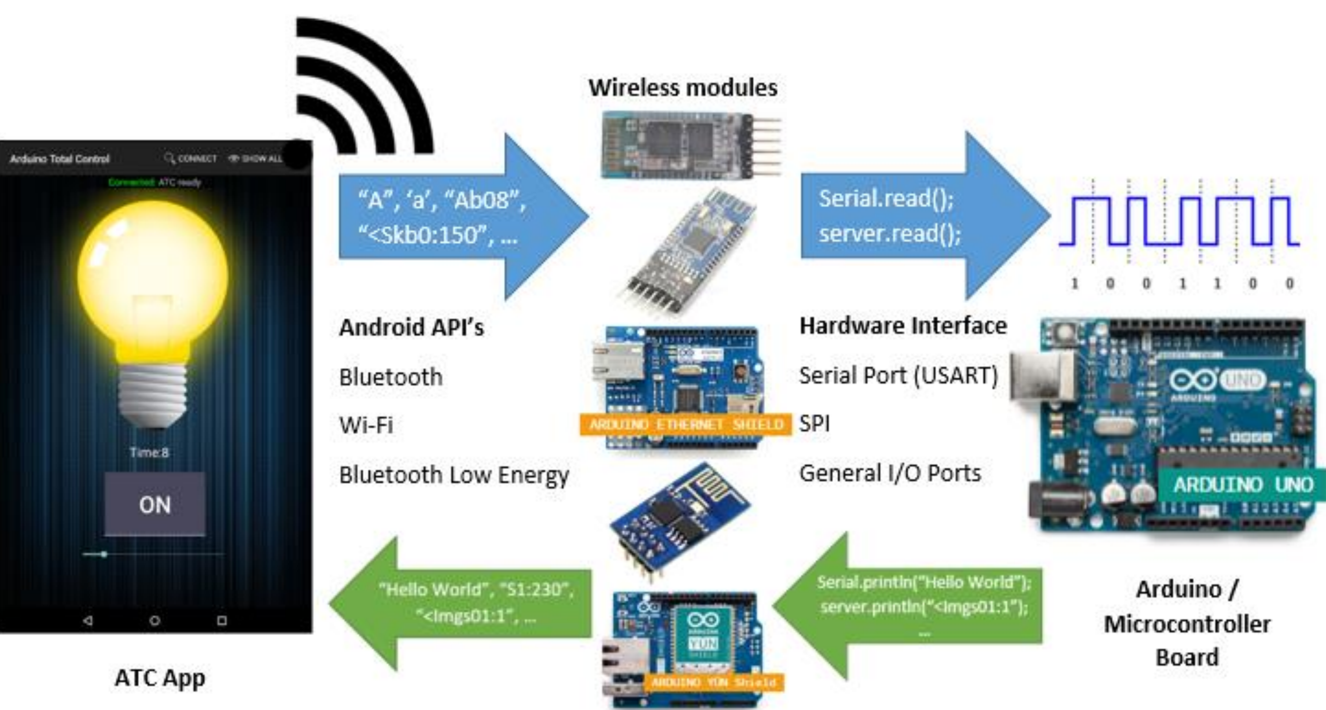


Figura 1. Esquema general de funcionamiento de la aplicación.

Introducción

El cuerpo de esta tesis está estructurado en cinco capítulos. El primero expone los antecedentes, las tecnologías y estándares en las que se basa este trabajo. El segundo consiste en una serie de apartados acerca de cómo se realiza una interfaz gráfica personalizable o HMI; desde el uso de objetos clase “View” estándar en la plataforma Android, hasta la creación de objetos nuevos, descendientes de la misma clase. También en este capítulo, el lector encontrará el desarrollo de las herramientas principales de edición de Layout (como edición de propiedades, agregar objetos, mover objetos, etc.), así como el de las herramientas secundarias de personalización (como Importar / Exportar Layout, fondo de pantalla, selección de Layout, etc.).

El tercer capítulo profundiza en la conectividad de la aplicación, donde el lector encontrará apartados relacionados a la implementación de Bluetooth, Bluetooth LE y Wifi en Android para lograr la comunicación entre el dispositivo Móvil Inteligente y el MCU. En esta sección también se incluye un apartado dedicado al protocolo de Capa de Aplicación utilizado en ATC, así como los métodos que convierten los datos recibidos del microcontrolador en información o acciones valiosas para el usuario.

En el cuarto capítulo, el lector encontrará información relativa al hardware y software mínimo requerido en el sistema a base de MCU para su funcionamiento con ATC.

Por último, en el quinto capítulo, se expondrán las pruebas funcionales de las comunicaciones inalámbricas y del protocolo de Capa de Aplicación ATC; así como el diseño y construcción de prototipos a base de MCU compatibles con la HMI desarrollado en este trabajo, seguido de las conclusiones del mismo.

Justificación

En el mundo moderno, los dispositivos inteligentes (*smartphones* y *tabletas electrónicas*) controlan gran parte del flujo de información, a través de redes sociales, internet, etc. La motivación para realizar este trabajo de tesis surge de la necesidad de controlar y obtener información en tiempo real del mundo físico desde la comodidad de un dispositivo móvil.

Las ventajas de usar una aplicación para controlar un sistema digital, en lugar de construir un Control Remoto o HMI específico son las siguientes:

- Ahorro en materiales que conformarían el control Remoto o HMI en cuestión, así como la energía que éstos consumirían en comparación con un módulo de comunicaciones inalámbrico.
- Se cuenta con todo el poder de procesamiento del dispositivo inteligente, además de otros servicios como una interfaz gráfica táctil, acceso a internet, diversos sensores e interfaces inalámbricas, entre otros elementos. En pocas palabras, se le pone al sistema en cuestión todo el poder de un Smartphone o Tablet actual.
- Se tiene una interfaz personalizable no limitada por hardware.

Materiales empleados y metodología

Para la realización de la Aplicación en Android, se utilizó Android Studio para programar, depurar y liberar el paquete de aplicación “.APK”, el cual es cargado a la tienda de Google mediante el software en línea, *Developer Console*, que proporciona las herramientas necesarias para administrar una aplicación en la tienda de Android. Además, se hace uso extensivo de las referencias a las API (Interfaz de Programación de Aplicaciones, por sus siglas en inglés “Application Programming Interface”) de Android, para analizar y adaptar las distintas funcionalidades requeridas por la aplicación.

Para la programación del microcontrolador, se utiliza la plataforma de firmware libre, Arduino, debido a su popularidad y facilidad de uso. El monitor serial fue ampliamente utilizado para observar la información saliente y entrante al microcontrolador.

Otra herramienta importante fue el desarrollo de la función “ATC Logger”, la cual permite almacenar el historial de determinados comandos provenientes del MCU en la forma de un archivo de texto en el dispositivo móvil. Lo anterior, con fines de monitoreo y/o depuración.

Arduino Total Control App

Herramientas primarias
- Editar propiedades de elementos constructivos
- Agregar objeto
- Mover objeto
- Limpiar *layout*

Herramientas secundarias
- Orientación de *layout*
- Cambiar de *layout*
- Exportar / Importar *Layout*
- Fondo de pantalla
- *Scroll Enable*

Layout - Interfaz Gráfica Personalizable

Elementos constructivos

Temp Button
Touch Pad
Analog Bar
Texto
Seek bar
Toggle Button
Speech Recognition
Acelerómetro
Imagen

- **Text to speech**
- **Vibración**
- **Alarma auditiva**

- **Auto Conectar**
- **ATC Logger**
- **In app support**

Conectividad - Comunicación inalámbrica con el MCU

Servicios de descubrimiento de dispositivos
- Bluetooth
- Bluetooth LE

Servicios de conexion
- Bluetooth
- Bluetooth LE
- Wifi

Capa de Aplicación (ATC tags / Raw data)

Bluetooth
- *Serial profile*

Wifi
- TCP/IP

Bluetooth LE
- *Manufacturer especific Characteristic UUID*

Figura 2. Diagrama de bloques de ATC (arriba), ícono de aplicación ATC (abajo).

La estructura general de la aplicación (app) desarrollada en este trabajo, que desde este punto se hará referencia a ella como "***Arduino Total Control***" **(ATC) o como "La aplicación",** se plantea mediante un diseño Top-Down, en donde se definieron las partes constitutivas del sistema, tanto en la Aplicación ATC como en el MCU, como se muestra en la Figura 2.

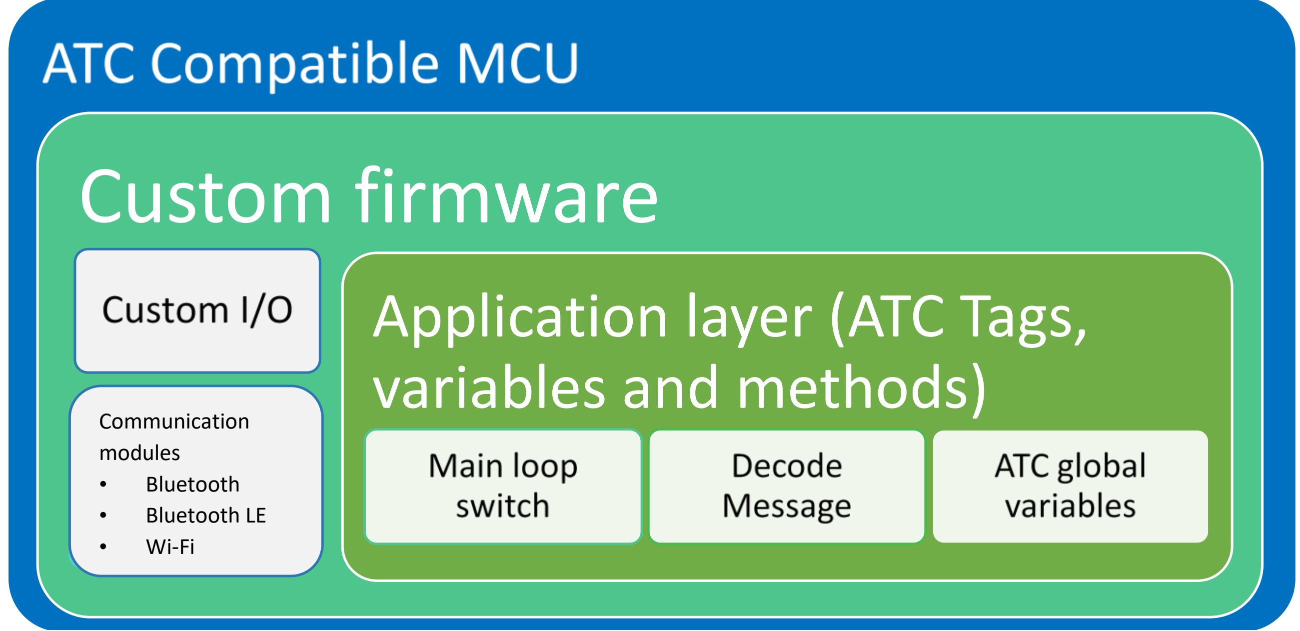


Figura 3. Diagrama de bloques de sistema a base de microcontrolador compatible con ATC.

El diseño del sistema a base de microcontrolador es dependiente del uso específico que el usuario final decida asignarle, sin embargo, éste deberá contar con el software y hardware mínimo para comunicarse con la aplicación ATC. La estructura general de un sistema a base de microcontrolador compatible con ATC se muestra en la Figura 1.

La metodología para realizar este trabajo consiste en los pasos listados en la Tabla 1, desde en análisis de factibilidad hasta la puesta en marcha, es decir, la publicación de la aplicación en la Play Store de Google.

Etapa	Descripción general
Factibilidad	Definición del marco teórico y disponibilidad en el mercado de software y hardware similar.
Definición	Selección del hardware y del software a usar (Android API), abastecimiento de materiales y hojas de datos, y definición del protocolo de comunicación ATC.
Diseño y Construcción	Diseño del circuito, programación de la aplicación y firmware en Android y Arduino respectivamente y pruebas de funcionamiento generales.
Puesta en marcha	Pruebas extendidas de funcionamiento y desplegado de aplicación en tienda virtual.

Tabla 1. Metodología del diseño y puesta en marcha para aplicación ATC.

Capítulo 1. Antecedentes y Marco Teórico

Antecedentes

Interfaz Hombre Máquina

ISO 9241-110, define el término interfaz de usuario como "todas las partes de un sistema interactivo (software o hardware) que proporcionan la información y los controles necesarios para que el usuario lleve a cabo una tarea específica con el sistema interactivo" [1]. Una Interfaz hombre máquina es el espacio donde las interacciones entre humanos y máquinas ocurren. El objetivo de estas interacciones es permitir la eficiente operación y control de la máquina, mientras ésta retroalimenta información que ayuda al proceso de toma de decisiones del operador.

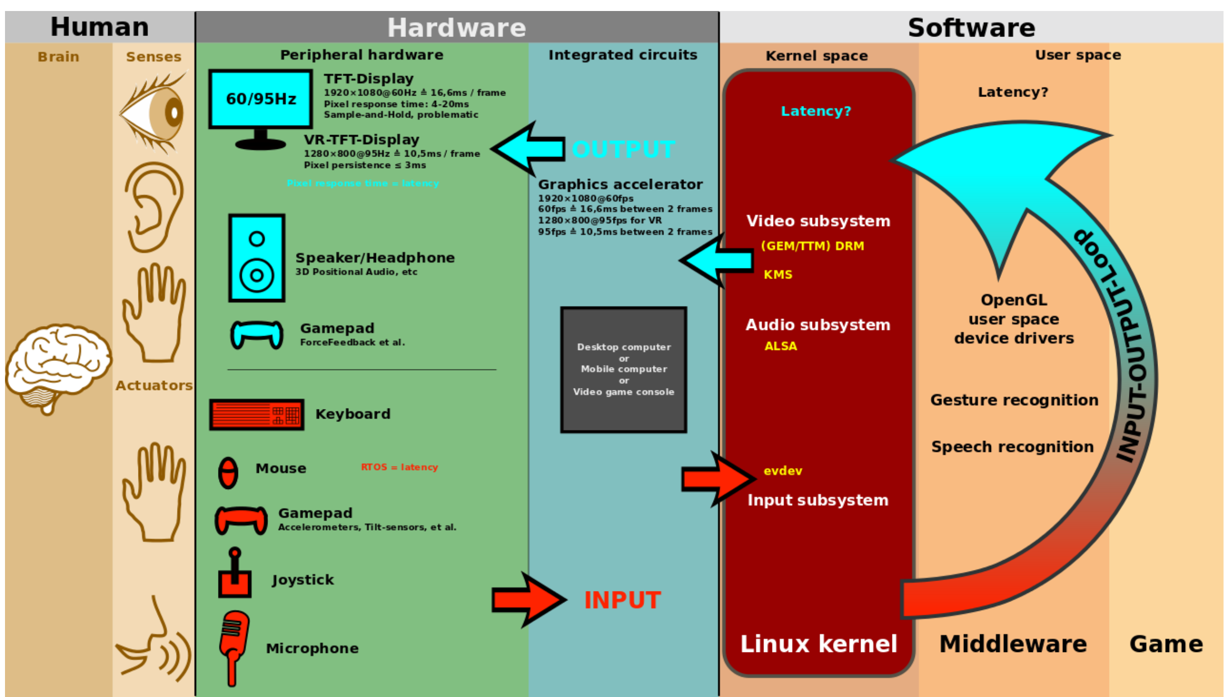


Figura 1.1. Una interfaz de usuario está conformada por software y hardware [2].

“El éxito de un producto técnico depende de más factores aparte del precio, la fiabilidad y el ciclo de vida; también depende de factores como la capacidad de manipulación y la facilidad de uso para el usuario” [3].

Figura 1.2. HMI industrial moderno [4].

La historia de las interfaces de usuario para sistemas computarizados puede ser dividida en las fases mostradas en la Figura 1.3, de acuerdo con el tipo dominante de Interfaz Hombre Máquina.

1945 –1968: Tarjetas perforadas. No existia interaccion en tiempo real.

1969 – Presente: Interfaz de línea de comandos. Comandos especializados para interactuar con el ordenador.

1984 – Presente: Interfaz gráfica de usuario. Ventanas, menús, íconos, punteros, etc.

Figura 1.3. Línea de tiempo de las interfaces de usuario.

Sistema Operativo Android

Android es un Sistema Operativo basado en el núcleo Linux (sistema operativo libre y de código abierto) utilizado en más de mil millones de teléfonos inteligentes y tabletas en más de 190 países [5]. Inicialmente fue desarrollado por Android Inc., empresa que Google respaldó económicamente y más tarde, en 2005, la compró [6]. Android fue presentado en 2007 junto con la fundación del Open Handset Alliance (un consorcio de compañías de hardware, software y telecomunicaciones) para avanzar en los estándares abiertos de los dispositivos móviles. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes y tabletas; y posteriormente también para relojes inteligentes, televisores y automóviles.

Cada día, más de un millón de nuevos dispositivos Android son activados a nivel mundial. Con más de 300 colaboradores de hardware, software y proveedores de telefonía móvil, Android se ha vuelto el sistema operativo de mayor crecimiento [5].

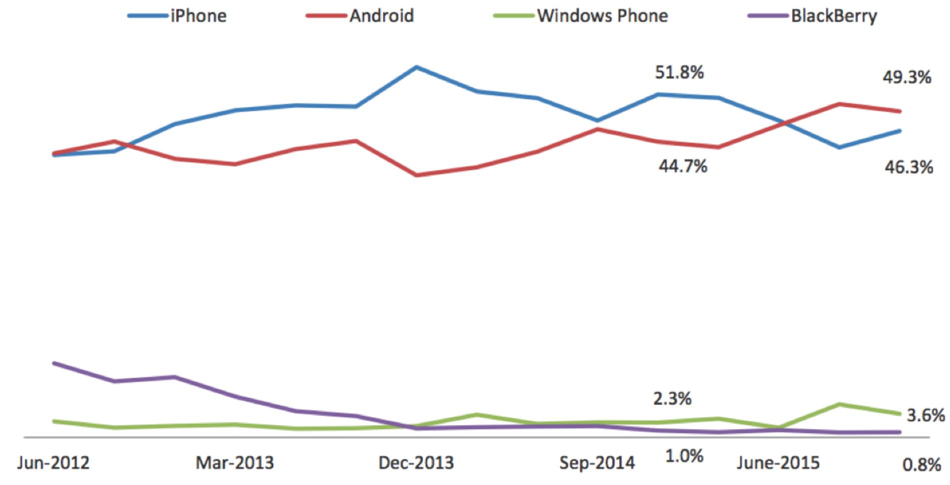


Figura 1.4. IOS vs Android OS Market Share [7].

Interfaz

La interfaz de usuario esta principalmente basada en manipulación directa [8], usando gestos táctiles que corresponden a acciones del mundo real. Controladores de videojuegos, teclados, y apuntadores son soportados vía Bluetooth o USB. La respuesta de la interfaz está diseñada para ser en tiempo real, usualmente se usan las capacidades de vibración para proveer retroalimentación háptica al usuario. Hardware interno, como acelerómetros, giroscopios y sensores de proximidad, son usados por algunas aplicaciones para responder a acciones de los usuarios.

En Android se pueden crear aplicaciones que toman ventaja de las capacidades de hardware disponibles en cada dispositivo. Además de adaptar automáticamente la interfaz gráfica en cada uno de ellos en tiempo de ejecución, aplicando el recurso correcto basado en el tamaño de la pantalla, densidad, y localización entre otros. [5]

Figura 1.5. Interfaz “Home” en Android.

Aplicaciones en Android

Las aplicaciones (“apps”), son programas que extienden la funcionalidad de los dispositivos, son escritas generalmente en Java, usando las Herramientas de Desarrollo de Software de Android (“SDK” por sus siglas en inglés). El SDK está compuesto por herramientas de depuración, librerías de software, documentación, código fuente de muestra, tutoriales, y un emulador de Dispositivos Android. Al principio, el Entorno de desarrollo integrado (“IDE” por sus siglas en inglés) soportado por Google, era Eclipse, utilizando el *plug-in* de las Herramientas de Desarrollo de Software de Android. Eclipse fue utilizado en las primeras iteraciones de “**La aplicación**” desarrollada para este trabajo. En diciembre de 2014, Google libera Android Studio, como su IDE (*Integrated Development Environment*) primario para el desarrollo de aplicaciones en Android.

Android Studio es la herramienta principal utilizada en el desarrollo de **ATC**. Otros entornos de desarrollo incluyen el Kit de Desarrollo Nativo (NDK por sus siglas en inglés) para aplicaciones en C o C++, así como Google App Inventor, un entorno visual para programadores sin experiencia.

Figura 1.6. Logo de Android Studio. IDE oficial para desarrollar aplicaciones para cualquier dispositivo Android [9].

El repertorio de Android contiene más de 2,800,000 aplicaciones [10] las cuales pueden ser descargadas por los usuarios usando un programa de tienda, que les permite instalar, actualizar y remover aplicaciones de sus dispositivos. Google Play Store es la aplicación principal instalada en los dispositivos Android que cumple con los requerimientos de compatibilidad y de licencia de Servicios Móviles de Google. La tienda de Google les permite a los usuarios buscar, descargar y actualizar aplicaciones publicadas por Google y terceros (Véase https://play.google.com/store). Las aplicaciones son descargadas e instaladas en el formato **“.APK”**.

Figura 1.7. Ícono del formato “.APK”.

Un archivo con extensión ".APK" (de las siglas en inglés de *Android Application Package*) es una variante del formato JAR de Java [11] y se usa para distribuir e instalar componentes empaquetados para la plataforma Android para teléfonos inteligentes y tabletas.

Un archivo .APK normalmente contiene los siguientes ficheros:

- AndroidManifest.xml.
- classes.dex.
- resources.arsc.
- res (carpeta).
- META-INF (carpeta).
- lib (carpeta).

El formato .APK es básicamente un archivo comprimido en *ZIP* con diferente extensión por lo cual pueden ser abiertos e inspeccionados usando un software archivador de ficheros como *7-Zip, WinZip, WinRAR* o *Ark*. Para abrirlo como aplicación debe usarse un dispositivo o emulador de Android.

Manejo de la memoria

Debido a que los dispositivos Android son comúnmente energizados por baterías, Android está diseñado para administrar sus procesos con un consumo mínimo de energía. Cuando una aplicación no está en uso, el sistema suspende su operación, lo cual no consume batería o recursos de procesamiento, de forma que está disponible para reanudar su uso inmediatamente después de que es pausada.

Android administra la memoria de las aplicaciones automáticamente, cuando la memoria esta baja, el Sistema empezará a terminar los procesos inactivos, empezando por aquellos que han estado en pausa por el mayor tiempo.

Hardware

La principal plataforma de hardware para Android es la arquitectura ARM (ARMv7 y ARMv8-A), con las arquitecturas x86 y MIPS también oficialmente soportadas. Desde Android 5.0 "Lollipop", variantes de 64 bits de todas las plataformas son soportadas.

Los requerimientos mínimos para dispositivos corriendo Android 5.1 varían desde los 512 MB de RAM para pantallas de densidad normal, hasta 1.8 GB para pantallas de alta densidad. Android soporta unidades de procesamiento de gráficos OpenGL ES 1.1, 2.0, 3.0 y 3.1.

Los dispositivos Android incorporan diferentes componentes de hardware opcional, incluyendo Cámaras de Video e imágenes, GPS, sensores de orientación, acelerómetros, giroscopios, barómetros, magnetómetros, sensores de proximidad, sensores de presión, termómetros y pantallas táctiles. Algunos componentes de hardware no son obligatorios, pero se convirtieron en el estándar en cierto tipo de dispositivos, como los teléfonos inteligentes.

Desarrollo

Android es desarrollado en privado por Google hasta que los últimos cambios y actualizaciones están listos para ser liberados, al punto que el código se hace disponible públicamente. Este código fuente sólo correrá en determinados dispositivos, usualmente los de la serie *Nexus*. El código fuente es entonces adaptado por los Fabricantes de Equipo Original (OEM, "Original Equipment Manufacturer") para correr en su hardware, debido a que el código fuente de Android no contiene los controladores propietarios de cada dispositivo.

Google provee actualizaciones mayores aproximadamente cada 6 o 9 meses, con nombres basados en confitería, que la mayoría de los dispositivos son capaces de recibir *en el aire*. La última actualización de Android es la 6.0 "Marshmallow". [12]

Linux kernel

El kernel de Android está basado en el kernel de Linux. Desde abril 2014 los dispositivos Android usan principalmente las versiones 3.4, 3.10 o 3.18 del kernel de Linux, dependiendo de la plataforma de hardware del dispositivo.

El almacenamiento flash en los dispositivos Android está separada en diversas particiones, como "/system" para el sistema operativo mismo, y "/data" para información del usuario y aplicaciones instaladas. En contraste con distribuciones Linux de escritorio, los dueños de los dispositivos Android no tienen acceso al directorio raíz del sistema operativo.

Software Stack

Android es una pila de software de código abierto creado para una variedad amplia de dispositivos con diferentes factores de forma, proveedores de servicios móviles, fabricantes y desarrolladores [9].

Sobre el kernel de Linux, se encuentran librerías nativas escritas en C, API's, y software de aplicaciones corriendo en un espacio de trabajo de aplicaciones que incluye librerías compatibles con Java.

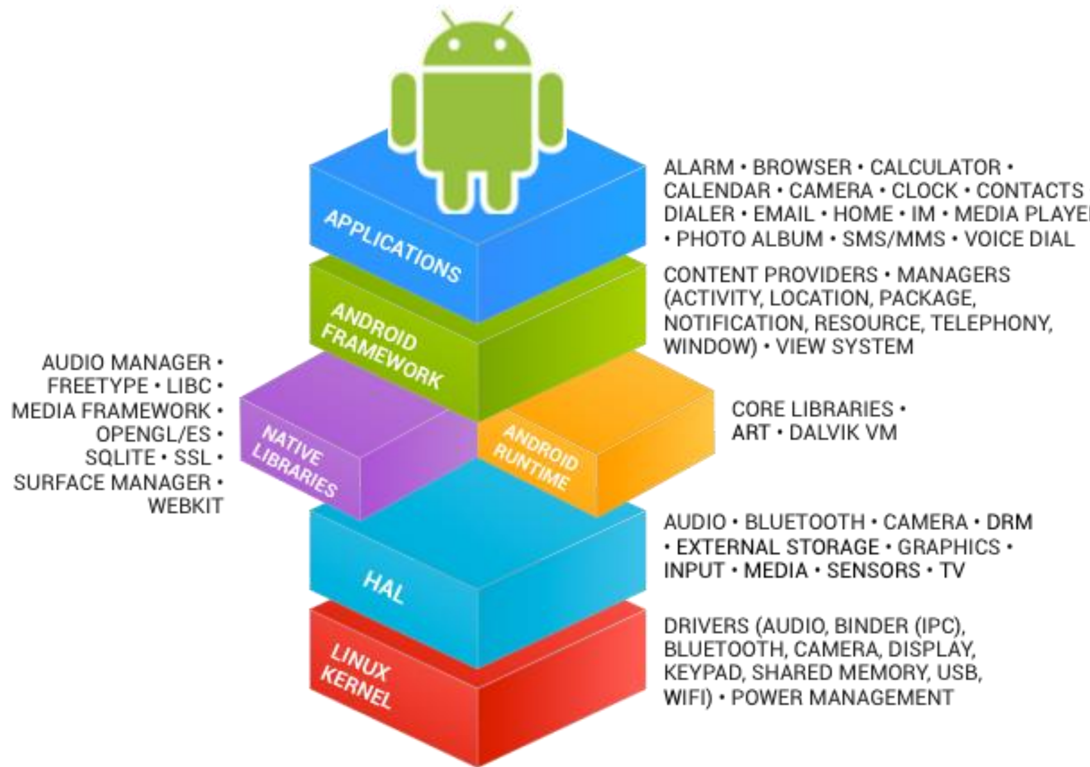


Figura 1.8. Diagrama de Arquitectura de Android [9].

Hasta la versión 5.0, Android usaba *Dalvik* como máquina virtual. A partir de Android 4.4, *Android Runtime* (ART) fue introducido como entorno de ejecución, la cual utiliza la compilación de las aplicaciones al momento de la instalación de las mismas. En Android 4.4, ART era una opción experimental no habilitada por defecto; se convirtió en el único entorno de ejecución en la siguiente versión de Android, 5.0.

Comunidad de código abierto

Android cuenta con una comunidad activa de desarrolladores y entusiastas que usan el código abierto de Android para desarrollar y distribuir sus propias versiones modificadas del sistema operativo. Los desarrollos de estas comunidades comúnmente traen consigo nuevas características y actualizaciones a los dispositivos más rápido que el fabricante oficial, con un nivel comparable de calidad; proveen soporte continuo a dispositivos antiguos que ya no son soportados por el fabricante o importan Android a dispositivos que originalmente tenían otro sistema operativo. Generalmente estos desarrollos vienen con privilegios de acceso al directorio raíz, y contienen modificaciones no proporcionadas por el vendedor original, como la habilidad de forzar la velocidad de reloj del dispositivo. *CyanogenMod* es el firmware de este tipo más usado, y es la base de muchos otros [13].

El microcontrolador

Un microcontrolador (o MCU, por sus siglas en inglés "Microcontroller Unit") es una pequeña computadora en un solo circuito integrado que contiene un núcleo procesador, memoria y entradas / salidas periféricas programables. Es normal encontrar memoria de programa en forma de RAM ferro eléctrica, NOR Flash, o PROM incluida en el chip, así como una cantidad limitada de memoria RAM.

Los microcontroladores son usados en productos automáticamente controlados, como en automoción, en equipos de comunicación y telefonía, en instrumentos electrónicos, en equipos médicos e industriales de todo tipo, electrodomésticos, juguetes, controles remotos, herramientas eléctricas, y otros sistemas embebidos.

Los microcontroladores están concebidos fundamentalmente para ser utilizados en aplicaciones puntuales, es decir, aplicaciones donde el microcontrolador debe realizar un pequeño número de tareas, al menor costo posible [14]. En estas aplicaciones, el microcontrolador ejecuta un programa almacenado permanentemente en su memoria, con el cual interactúa con el exterior a través de las líneas de entrada y salida de que dispone. Algunos microcontroladores usan palabras de 4 bits y operan a frecuencias tan bajas como los 4kHz, para bajo consumo de energía [15].

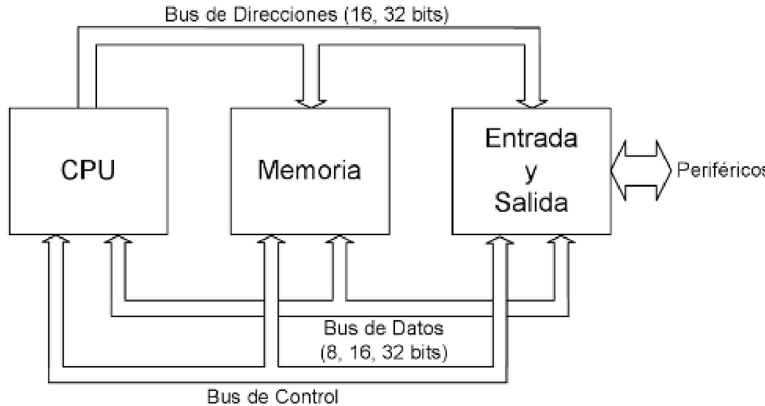


Figura 1.9. Esquema básico general de un microcomputador [14].

Historia

El primer microprocesador fue el Intel 4004, con 4bits, lanzado en 1971. El Intel 8008 (8 bits) y otros microprocesadores más capaces se volvieron disponibles en los siguientes años. Sin embargo, éstos necesitaban chips externos para implementar un sistema funcional, elevando el costo total del sistema, y haciendo imposible el digitalizar los aparatos de uso común.

Los Ingenieros de Texas Instruments Gary Boone y Michael Cochran son galardonados por el Instituto Smithsoniano en 1971 por la creación del primer microcontrolador. El resultado de su trabajo fue el TMS 1000, que se convirtió comercialmente disponible en 1974. Éste combinaba ROM, RAM, un procesador y circuito de reloj en un solo chip.

En 1977 Intel lanza el 8048, un circuito optimizado para controlar aplicaciones. Éste combinaba RAM y ROM en un mismo circuito y fue instalado en más de un billón de teclados y otras aplicaciones.

En este tiempo, la mayoría de los microcontroladores contenían una memoria de programa EPROM, con una ventana de cuarzo que permitía borrar la memoria con luz ultravioleta, usados para prototipos. Para producción en masa los microcontroladores poseían memoria PROM, la cual era idéntica a la EPROM, pero sin la ventana de cuarzo, o bien memoria programable una sola vez u OTP ("One Time Programable" por sus siglas en inglés).

En 1983, la introducción de la memoria EEPROM (comenzando con el Microchip PIC16x84) les permitió a los microcontroladores ser eléctricamente reprogramables, permitiendo acelerar el proceso de realización de prototipos. El mismo año, Atmel introduce el primer microcontrolador con memoria Flash, un tipo especial de EEPROM. Otras compañías rápidamente adoptaron ambos tipos de tecnologías.

En el 2002, cerca del 55% de los microprocesadores vendidos en todo el mundo fueron microcontroladores de 8 bits. Semico, una compañía de marketing y consultoría en semiconductores, aclama que el mercado de los MCU creció 12% en 2011 [16], año en que los microcontroladores de 16 bits gobernaron el mercado. Se espera que para el 2017, los microcontroladores de 32 bits representen el 55% de las ventas de MCUs debido a la demanda de niveles más altos de precisión en sistemas embebidos y conectividad a internet.

En un país desarrollado, un hogar promedio tiene tan solo cuatro microprocesadores de uso general (PC o Laptop) pero cerca de una docena de microcontroladores. Un automóvil promedio contiene alrededor de 30 microcontroladores.

Características de los microcontroladores

Además de contener los elementos básicos como el CPU, el cual es el "cerebro" del microcomputador que actúa bajo el control del programa almacenado en la memoria, la memoria de programa y la memoria de datos, los microcontroladores incluyen hardware que aumenta sus capacidades y reducen la carga del microprocesador.

Entradas y Salidas de uso general (*General Purpose Input / Output, GPIO*)

Los microcontroladores usualmente contienen múltiples puertos de entradas y salidas, los cuales son configurables por software para leer y escribir su estado lógico. En algunos microcontroladores, existe la opción de habilitar resistencias de *pull-up* o *pull-down* cuando el puerto es de entrada, o bien, drenaje o colector abierto, cuando estas se configuran como salidas.

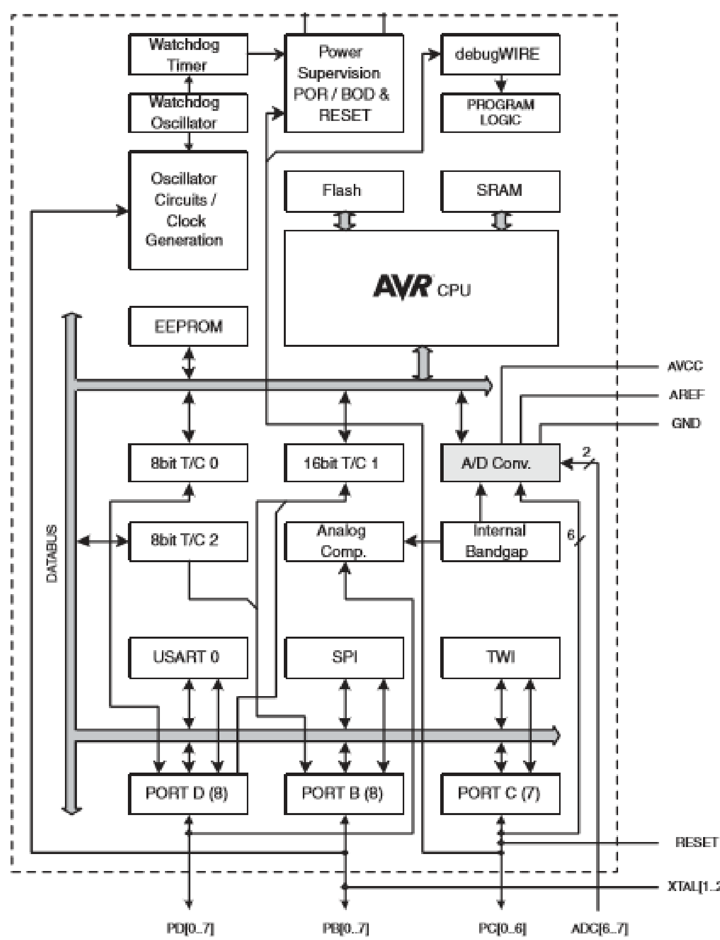


Figura 1.10. Diagrama de bloques del microcontrolador ATMEGA328 [17].

Convertidor Analógico Digital (ADC)

Muchos sistemas embebidos necesitan leer sensores que producen señales analógicas. Éste es el propósito del ADC. El convertidor analógico digital es usado para convertir información analógica a una forma que el procesador pueda interpretar.

Convertidor Digital Analógico (DAC)

Algunos Microcontroladores cuentan con DAC, el cual le permite al procesador proporcionar señales analógicas o niveles de voltaje en sus salidas.

Temporizadores

Muchos sistemas embebidos incluyen una variedad de temporizadores (o *timers*), los cuales pueden variar en frecuencias, número de bits, sentido de conteo (ascendente, descendente), interrupciones, etc. A menudo, estos timers se encargan de controlar características de Modulación de Ancho de Pulso (PWM), lo cual le permite al microprocesador controlar convertidores, cargas resistivas, motores, entre otros sin usar recursos de procesamiento considerables.

Módulos de comunicaciones seriales

Permite enviar y recibir datos por medio de diversos protocolos de comunicación serial, como RS232, RS485, I2C, SPI, USB y Ethernet con una carga de procesamiento mínima para el CPU.

Arduino

Arduino es una plataforma de código abierto para la realización de prototipos, basada en hardware y software fácil de usar. Las tarjetas Arduino son capaces de leer entradas como la luz incidiendo en un sensor, un dedo presionando un botón, o un mensaje de Twitter, y convertirlo en una salida, activando un motor, prendiendo un LED o publicando algo en línea [18]. Para lograr esto, se utiliza el lenguaje de programación de Arduino (basado en *Wiring*, el cual a su vez está basado en C++), y el software Arduino IDE.

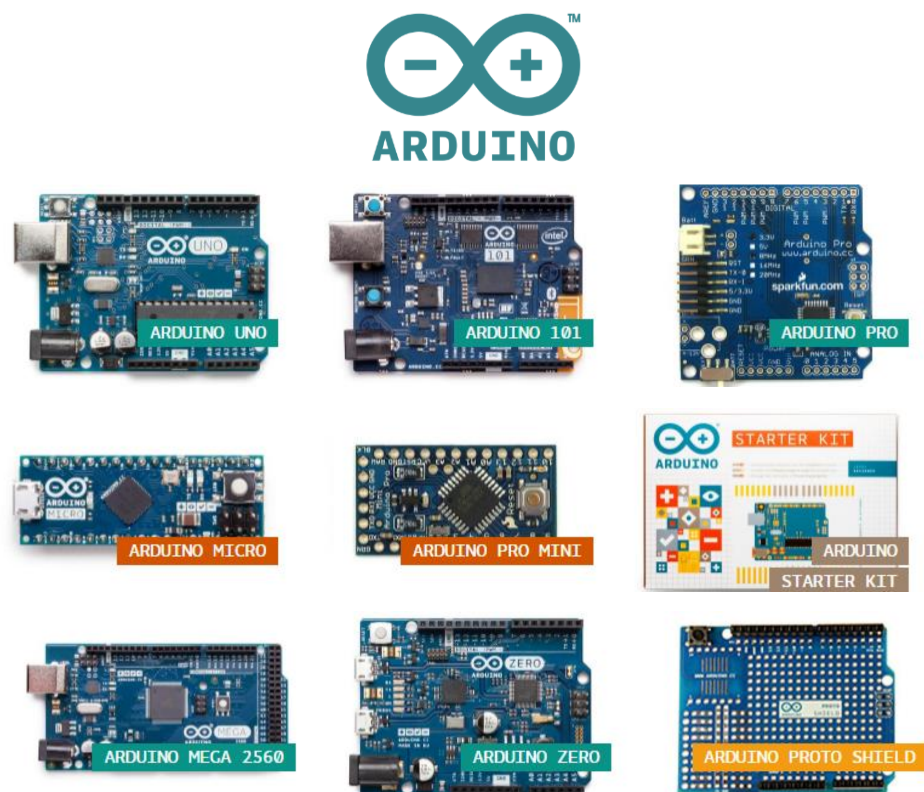


Figura 1.11. Logo oficial de Arduino y Placas Arduino [19].

Arduino nació en el *Interaction Design Institute Ivrea* (Italia) como una herramienta para la creación de prototipos rápidos, dirigida a estudiantes sin conocimientos en electrónica y programación. Tan pronto éste alcanzó comunidades más extensas, la tarjeta Arduino empezó a cambiar para adaptarse a nuevas necesidades y desafíos como impresión 3D, *Internet of Things* (IOT), *Wearable Devices* y entornos embebidos. Todas las tarjetas Arduino son de código abierto, permitiendo a los usuarios adaptarlas a sus necesidades particulares. [18]

¿Por qué Arduino?

- **Simple y fácil de usar**. Arduino encapsula los detalles y la complejidad de programar un microcontrolador en un paquete fácil de usar. Su entorno de desarrollo, Arduino IDE, es fácil de usar para los principiantes, pero lo suficientemente flexible para los programadores más avanzados.
- **De bajo costo**. Las tarjetas de desarrollo de Arduino son relativamente baratas comparadas contra otras plataformas de microcontrolador. Además, al ser una plataforma abierta (tanto en hardware como en software), el usuario puede obtener tarjetas de desarrollo “clones” (Arduino.cc, 2016) por menos de 10USD.
- **Multi-plataforma**. El software Arduino IDE corre en Windows, Mac OSX, y Linux, en contraste con otros entornos de programación de microcontroladores que están limitados a Windows.
- **Plataforma abierta de software y hardware**. Tanto el software como el hardware son publicados como herramientas abiertas bajo la licencia de *Creative Commons*, por lo que diseñadores y programadores experimentados pueden hacer sus propias versiones y mejoras de los módulos Arduino que componen.

Trabajo previo, la aplicación BT Relay Control

En sus primeras etapas, “**La Aplicación**” consistía en una simple interfaz con 8 botones, 8 imágenes y 8 textos dispuestos en una matriz fija en la pantalla del usuario (Figura 1.12). Las únicas funciones de BT Relay Control eran enviar y recibir información de 1 carácter (8 bits) de y hacia un solo controlador remoto para encender o apagar salidas digitales. BT Relay Control era compatible con Bluetooth convencional únicamente.

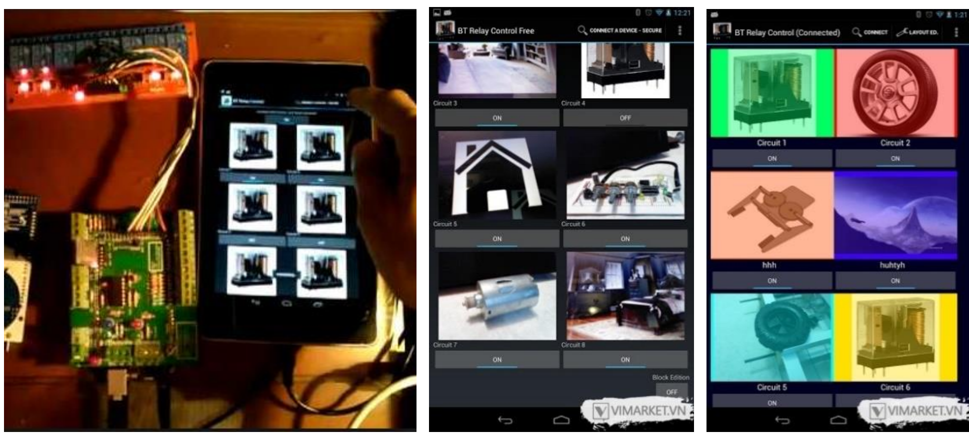


Figura 1.12. Primeros pasos de la aplicación, BT Relay Control.

Disponibilidad de software y hardware en el mercado

Software

Las plataformas de dispositivos móviles y el internet facilitan y propician la competencia directa de productos de diferentes desarrolladores, y las interfaces de usuario personalízales para dispositivos portátiles no son la excepción. Entre las aplicaciones tipo HMI para microcontroladores más populares en la tienda de Android se pueden encontrar ***Blynk y Arduino Commander***.

Blynk

Es una aplicación creada para controlar Arduino, ESP8266, *Raspberry Pi* y otros microcontroladores con un Smartphone a través de internet [20].

Arduino Commander

Le permite al usuario controlar tarjetas Arduino desde el dispositivo Android, por medio de Bluetooth, Ethernet o USB, usando una interfaz WYSIWYG (“lo que vez es lo que obtienes”, de sus siglas en inglés), sensores de Android o JavaScript [21].

Característica	Blynk	Arduino Commander
Interfaz	Personalizable y estética	La Interfaz asemeja una tarjeta Arduino
Microcontroladores compatibles	*Arduino Uno, Nano, Mini, Pro Mini, Pro Micro, Mega, YÚN (Bridge), Due, 101, Raspberry Pi, Particle Core, Particle Photon, ESP8266, TinyDuino (CC3000), Wicked WildFire (CC3000)*	*Diecimila, Duemilanove, Uno r1/r2/r3, Mega, Leonardo, Nano*
Bluetooth	No	Si
Bluetooth LE	Si	No
Wifi	Si	Si
USB	Si	Si
Facilidad de uso	Código autogenerado dependiendo de los objetos habilitados en la interfaz gráfica.	Código Autogenerado y cargable directamente desde el dispositivo Android a la tarjeta Arduino.
Soporte en línea	Si	Si
Compartir proyectos	Si	No
Servidor propio en línea para control sobre internet	Si	No
Sensores (Acelerómetro, barómetro, etc.)	Si	Si (costo adicional)
Reconocimiento de voz	No	Si
Graficador de datos	Si	Si
Texto a voz	No	No
Comercialización	Aplicación Gratis con opciones de compra dentro de la aplicación	Aplicación Gratis con opciones de compra dentro de la aplicación

Tabla 1.1. Características y comparativa de aplicaciones HMI para Android.

Hardware

Así como en el software, el comercio electrónico facilita la adquisición de distintos productos para construir prototipos a bajo costo. Los módulos de comunicación inalámbrica Bluetooth y Wi-Fi más populares se describen a continuación.

Módulos Bluetooth

Los módulos Bluetooth más populares en el mundo "Maker" (véase http://www.youngmarketing.co/la-cultura-del-maker-movement-y-como-esta-cambiando-el-mundo/) son el HC-05 (módulo maestro-esclavo) y el HC-06 (esclavo) debido a su facilidad de uso, ya que emulan un puerto serial que es transparente para el microcontrolador. Su bajo costo, que ronda alrededor de los $4USD los hace populares entre los entusiastas.

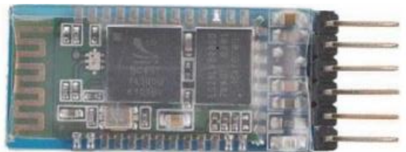


Figura 1.13. Módulo HC-05.

Tanto el módulo HC-05 como el HC-06 están basados en el Bluecore 04-Ext de CSR (adquirido por Qualcomm en el 2015), la única diferencia es el software cargado en cada uno de ellos. El módulo Bluecore 04-Ext es un sistema de Bluetooth en un circuito que implementa las siguientes características [22]:

Características Generales

- *Enhanced Data Rate* (EDR) Bluetooth v2.0 con módulos de 2Mbps y 3Mbps.
- Sistema en un Chip que integra radio, banda base y microcontrolador.
- Interfaz para memoria flash externa (8Mbits).
- Soporte para *Piconet.*
- Soporte para coexistir con 802.11 (Wi-Fi).

Radio

- Soporta modulación Π/4 DQPSK y 8DPSK.
- Transmisor
  - Potencia de transmisión RF +6dBm, con control de nivel vía convertidor digital analógico de 6 bits integrado con un rango dinámico mayor a 30dB.
  - Compatible con clase 2 y clase 3 sin necesidad de amplificadores externos.
- Receptor
  - Filtros de canal integrados.
  - Clasificación de canales integrada.

Software

- Memoria RAM interna de 48kbytes.
- Compatible con Bluetooth v1.2.
- Control de errores, CRC, demodulación, encriptado y generación de tren de pulsos.
- Soporte para codificación *ley A* y *ley µ.*

Interfaces físicas

- Interface serial síncrona para depuración con una tasa de transferencia de hasta 4Mbit/s.
- Interface serial asíncrona con una tasa de transferencia programable de hasta 1.5Mbit/s.
- Interface Full Speed USB v2.0 que soporta interfaces *OHCI* y *UHCI*.
- Interface de audio programable bidireccional serial.

Módulos BLE

Entre los módulos BLE más populares en la comunidad de entusiastas se encuentra el HM10. Éste está basado en el circuito CC2540 de Texas Instruments. El módulo HM10 tiene un costo aproximado de 8USD en el mercado. El CC2540 es un sistema en un chip para aplicaciones Bluetooth LE que cuenta con las siguientes características [23]:

- Microcontrolador 8051 integrado.
- 128/256 kB de memoria flash.
- 8 kB de memoria RAM.
- *Stack* Bluetooth LE integrado.
- Transceptor RF.
- Modos de bajo consumo de energía.
- *Baud Rate* 1000 kbps.

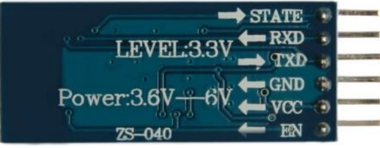


Figura 1.14. Módulo HM10 basado en el TI CC2540.

Módulos Wi-Fi/Ethernet

En cuanto a comunicación Wifi, ésta se puede lograr de formas diferentes:

- Usando una tarjeta Arduino habilitada para comunicación Wi-Fi, como el **Arduino Yun** (**50USD**)
  Arduino Yún es una tarjeta basada en el microcontrolador ATmega32u4 y el procesador Atheros AR9331. El procesador Atheros soporta una distribución de Linux basada en OpenWrt llamada Linino OS [24]. La tarjeta Arduino Yún cuenta con las siguientes características:

  - Soporte para Ethernet y Wifi.
  - Conexión micro USB y Puerto USB-A.
  - Slot para tarjeta micro-SD.
  - 20 salidas/entradas digitales.
  - Cristal Oscilador de 16Mhz.
  - Puerto ICSP.

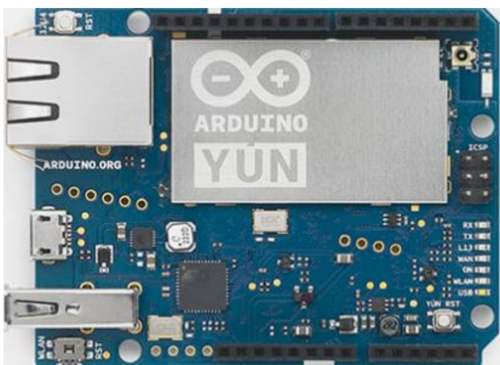
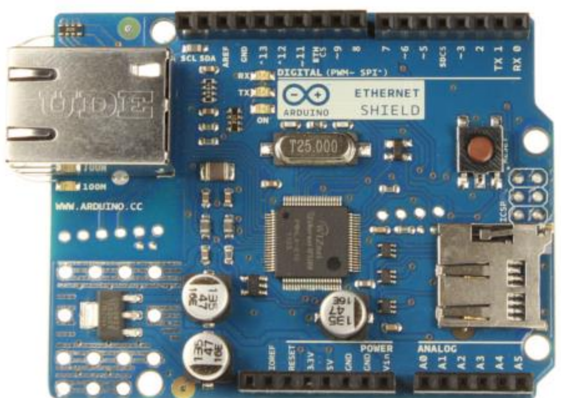


Figura 1.15. Arduino Yún.

- Utilizando un ***Ethernet Shield*** (**6USD**)
  El *Ethernet Shield* permite a una tarjeta Arduino conectarse al internet. Está basada en el circuito Wiznet W5100 (en su versión V1) o en el circuito Wiznet W5500 en su versión V2. El primero puede manejar hasta cuatro conexiones simultaneas, mientras el segundo hasta ocho conexiones o sockets TCP/IP [25]. Sus principales características son:

- Layout de pines compatible con tarjetas Arduino.
- Voltaje de operación: 5V.
- Controlador Ethernet con 16kb de buffer.
- Velocidad de conexión: 10/100Mb.
- Conexión con Arduino mediante Puerto SPI.



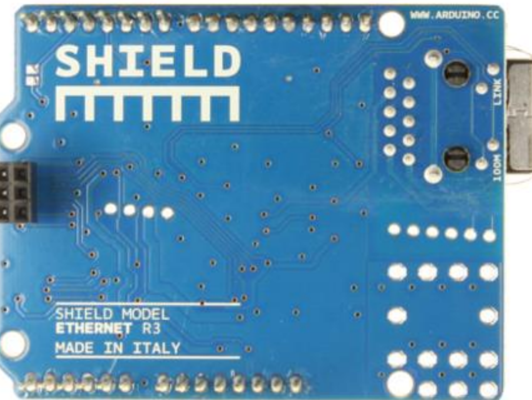


Figura 1.16. Ethernet Shield.

- O bien, utilizando un circuito independiente de Wi-Fi, como el **ESP8266** (**2USD**)
  El circuito ESP8266 es un sistema en un solo chip que integra el protocolo 802.11 b/g/n y la pila de protocolos TCP/IP y UDP/IP. Además de ser de bajo costo, tiene un *footprint* menor a 2cm^2. Sus características principales son [26]:

  - Protocolo 802.11 b/g/n.
  - Soporta Wi-Fi Direct (P2P), soft-AP.
  - Pila de protocolo TCP/IP integrada.
  - Amplificador de potencia integrado.
  - PLL, reguladores y unidades de control de potencia integrados.
  - Potencia de salida en modo 802.11b de +19.5dBm.
  - CPU de 32-bit de baja potencia integrado.
  - Interfaces físicas: SDIO 2.0, SPI, UART.

Figura 1.17. Módulo ESP8266.

Marco Teórico

A continuación, se presentan los conceptos necesarios para una mejor comprensión del trabajo general.

Modelo OSI

De acuerdo con el estándar internacional ISO/IEC 7498-1, el modelo básico de Interconexión de Sistemas Abiertos (OSI por sus siglas en inglés) es un modelo de referencia en capas que provee las bases comunes para la coordinación de desarrollo de estándares para interconexión de sistemas, mientras permite que los estándares existentes se pongan en perspectiva contra el modelo de referencia [27]. Su objetivo es la interoperabilidad de diversos sistemas de comunicación con protocolos estándar. El modelo OSI secciona un sistema de comunicación en siete capas, que definen las fases por las que deben pasar los datos para viajar de un dispositivo a otro sobre una red de comunicaciones.

Los protocolos o estándares de comunicación son conjuntos de normas para formatos de mensaje y procedimientos que permiten a las máquinas y los programas de aplicación intercambiar información. Cada máquina implicada en la comunicación debe seguir estas normas para que el sistema principal de recepción pueda interpretar el mensaje [47].

Elementos básicos del modelo OSI

a) Sistemas abiertos. Es un conjunto de uno o más procesadores, el software asociado, periféricos, terminales, operadores humanos, procesos físicos, etc., que conforman una entidad capaz de procesar y/o transferir información que cumple con los estándares de OSI en cuanto a su comunicación con otros sistemas abiertos concierne. Para "**La aplicación**", el dispositivo móvil inteligente y el sistema a base de microcontrolador representan los sistemas abiertos.
b) Entidades de aplicación. Es un elemento en el sistema abierto que realiza el procesamiento de la información para una determinada tarea. Éste puede representar procesos manuales, computarizados, o físicos. "**La aplicación**" cumple con esta descripción.
c) Asociaciones. Relación de intercambio de información entre elementos de la misma capa de comunicación. En "**La aplicación**", el enlace de la actividad principal con el microcontrolador es un ejemplo.
d) Medio físico. Define el, o los medios físicos por los que va a viajar la información. Para "**La aplicación**", éste es la señal de radio propagándose en el aire.

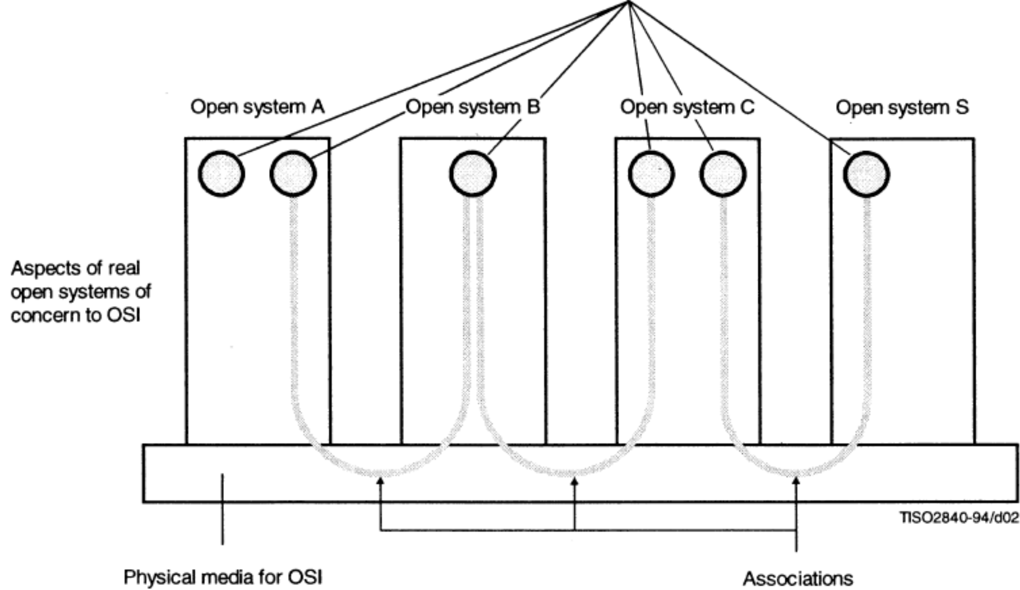


Figura 1.18. Elementos básicos de OSI [27].

El modelo de capas

De acuerdo con el modelo OSI, cada sistema abierto es visto como un conjunto ordenado de (N) subsistemas, como se representa en la Figura 1.19. Los subsistemas de la capa (N) se comunican con sus iguales bajo los mismos límites. Solo hay un y solo un subsistema (N) para la capa (N) en el modelo OSI. Nótese que las entidades en la misma capa (N) son denominados unidades igual a igual (N).

No todas las unidades de la misma capa (N) necesitan comunicarse. Puede haber condiciones que prevengan esta comunicación (por ejemplo: las entidades no son parte de un sistema de abierto, o no soportan el mismo protocolo).

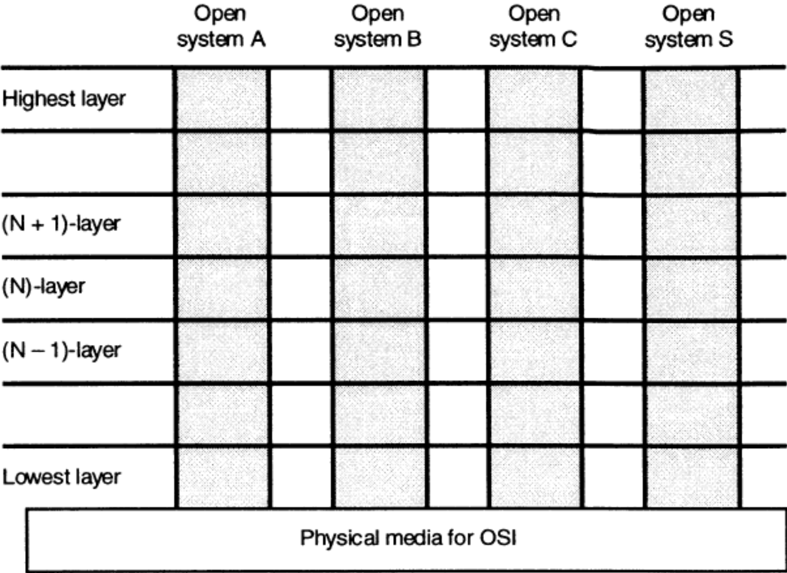


Figura 1.19. Capas en un sistema abierto [27].

Comunicación entre capas de igual a igual

Para que exista un intercambio de información entre dos o más entidades (N+1), una asociación debe ser establecida entre ellas en la capa (N) usando un protocolo (N). Las reglas y formatos de un protocolo (N) son definidas por una entidad (N). Una entidad (N) puede soportar protocolos (N) que pueden ser en modo de desconexión, modo de conexión o ambos. Las entidades (N+1) se comunican únicamente usando protocolos de la capa (N).

Modos de comunicación

Una capa (N) puede ofrecer un servicio en modo de desconexión, en modo de conexión o en ambos, para la capa (N+1). Toda transmisión entre entidades (N+1) debe usar el mismo modo del servicio (N).

Modo de conexión

Una conexión es una asociación establecida para la transferencia de datos entre dos o más entidades iguales de capa (N). Esta asociación vincula las entidades iguales (N) con las entidades (N-1) de la siguiente capa. El uso de un servicio en modo de conexión sigue las siguientes fases:

a) Establecimiento de la conexión.
b) Transferencia de datos.
c) Liberación de la conexión.

El modo de conexión tiene las siguientes características:

- Involucra el establecimiento y mantenimiento de un acuerdo de transmisión de datos entre dos o más partes.
- Provee identificación de conexión.
- Provee un contexto en el cual unidades de datos sucesivas transmitidas entre las entidades iguales están relacionadas lógicamente, lo que permite mantener una secuencia y proveer control de flujo para esas transmisiones.

Estas características son particularmente atractivas en aplicaciones donde se requieren conexiones de larga duración, orientadas a un flujo de datos entre entidades con configuraciones estables, por ejemplo, transferencia de archivos y terminales remotas.

Modo de desconexión

Es la transmisión de una unidad de información desde una fuente a uno o más destinos sin establecer una conexión. En contraste a una conexión, un servicio en modo de desconexión no tiene una duración claramente definida. Además, cuenta con las siguientes características:

a) Solo requiere una pre-asociación entre las unidades iguales (N) involucradas en donde se determinan las características de los datos a ser transmitidos.
b) Toda la información requerida para enviar una unidad de datos es presentada a la capa junto con la unidad de datos a ser transmitida. La capa que provee el servicio en modo de desconexión no requiere relacionar este acceso con uno anterior.
c) Cada unidad de datos transmitida es enrutada independientemente por la capa que provee el modo de desconexión.
d) Copias de la unidad de datos pueden ser transmitidas a diferentes direcciones destino.

Unidades de datos

La información es transmitida en varios tipos de unidades de datos por las entidades (N). Un (N)-PDU ("Protocol Data Unit" por sus siglas en inglés) está constituido por información de control de protocolo (N) y datos de usuario (N). El primero es información compartida entre las entidades (N) para coordinar su operación conjunta, el segundo es la información transmitida entre las capas (N) proveniente de las entidades (N+1).

Capas del modelo OSI

El modelo de referencia tiene 7 capas como se muestra en la Figura 1.20. La capa más alta es la de Aplicación, y consiste de entidades de aplicación que cooperan en el entorno OSI. Las capas más bajas proveen los servicios con los cuales las entidades de aplicación cooperan.

a) La Capa de Aplicación (capa 7).
b) La Capa de Presentación (capa 6).
c) La Capa de Sesión (capa 5).
d) La Capa de Transporte (capa 4).
e) La Capa de Red (capa 3).
f) La Capa de Enlace de datos (capa 2).
g) La Capa Física (capa 1).

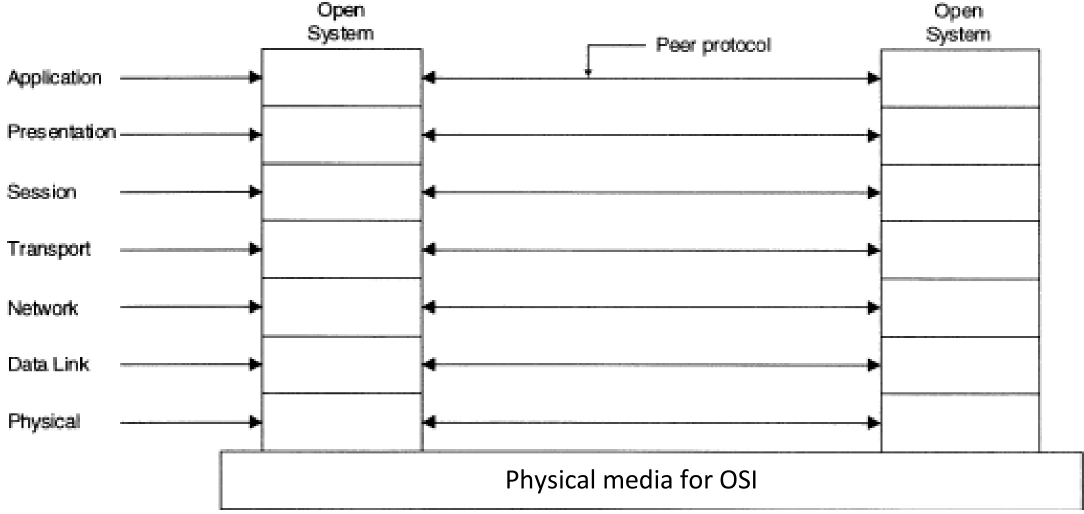


Figura 1.20. Capas del modelo OSI [27].

No todos los sistemas abiertos son las fuentes iniciales o los destinos finales de la información. Algunos sistemas abiertos actúan como “relevadores”, pasando datos a otros sistemas abiertos como se muestra en la Figura 1.21.

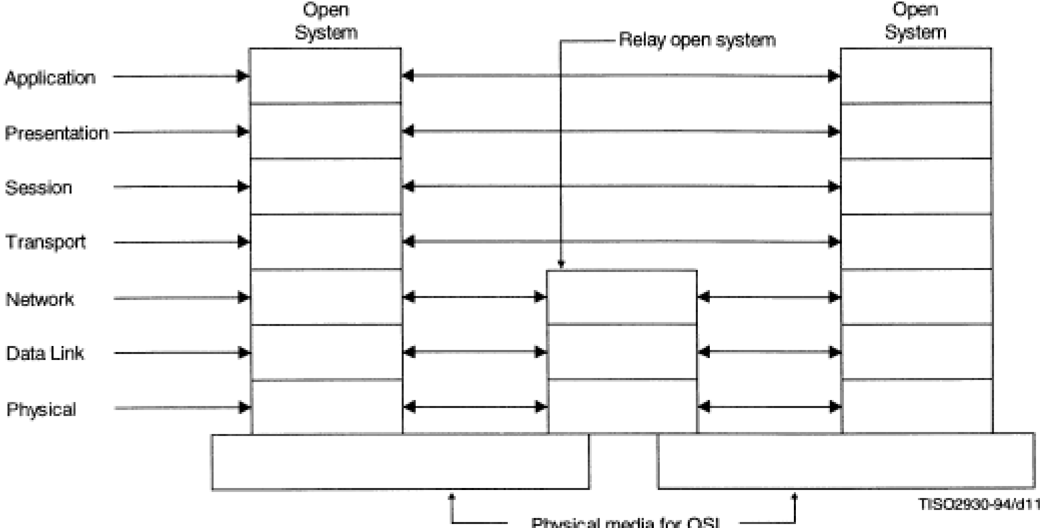


Figura 1.21. Sistema abierto actuando como relevador [27].

Capa de Aplicación

Al ser la última capa, la Capa de Aplicación provee los únicos medios para que el proceso de aplicación acceda al entorno OSI. Los procesos de aplicación intercambian información por medio de entidades de aplicación, los cuales usan protocolos de aplicación y servicios de la Capa de Presentación.

La Capa de Aplicación contiene todas las funciones que implican comunicación en cualquier modo entre sistemas abiertos que aún no son ejecutados por las capas anteriores. Esto incluye funciones realizadas por los programas, así como seres humanos. En particular, las entidades de aplicación mantienen información sobre el uso de los modos de transmisión (conexión o desconexión) con las entidades con las cuales se tienen que comunicar.

Capa de Presentación

El propósito de la Capa de Presentación es proveer una representación común de la información transferida entre entidades de aplicación. Esto libera a las entidades de aplicación del problema de representar la información. La Capa de Presentación asegura que el contenido de información de la Capa de Aplicación es preservado durante la transmisión.

La Capa de Presentación provee los siguientes servicios:

a) Identificación de las sintaxis de transferencia.
b) Selección de la sintaxis de transferencia.
c) Acceso a los servicios de sesión.

La Capa de Presentacion cumple con las siguientes funciones que ayudan a llevar a cabo los servicios de presentacion:

a) Negociación de la sintaxis de transferencia.
b) Representación de la sintaxis abstracta escogida por las entidades de aplicación.
c) Restauración de sintaxis previamente negociadas cuando ciertos eventos ocurren.

Capa de sesión

La Capa de Sesión provee los medios necesarios para la cooperación entre entidades de presentación para organizar y sincronizar su diálogo y administrar su intercambio de información. La Capa de Sesión permite el intercambio ordenado de datos, y la liberación de la conexión de forma ordenada.

En modo de desconexión, la única función de la Capa de Sesión es el mapeo de direcciones de transporte a direcciones de sesión.

Los servicios que provee la Capa de Sesión en modo de conexión son:

a) Establecimiento de la sesión-conexión.
b) Finalización de la sesión-conexión.
c) Transferencia de datos normal.
d) Transferencia de datos expedita.
e) Administración de *token.*
f) Sincronización de sesión.
g) Administración de actividad.
h) Reporte de excepciones.

Capa de Transporte

La Capa de Transporte provee una transferencia transparente de datos entre las entidades de sesión y las libera de los detalles acerca de cómo ésta se lleva a cabo de forma confiable y eficiente. La Capa de Transporte optimiza el uso de los servicios de red variables para proveer el desempeño requerido por cada entidad de sesión al costo mínimo.

En modo de desconexión, segmentación y re-ensamble de la información no son proveídos en la Capa de Transporte. Por lo tanto, el tamaño de las unidades de datos de transporte en modo de desconexión está limitado por el tamaño de las unidades de datos y control del protocolo. En modo de desconexión, las funciones de la Capa de Transporte son:

a) Mapeo entre las direcciones de Capa de Transporte y Capa de Red.
b) Detección de errores punto a punto y monitoreo de la calidad del servicio.
c) Delimitación de las unidades de datos de transporte.
d) Funciones de supervisión.

En modo de conexión, las funciones de la capa transporte incluyen:

a) Mapeo de las direcciones de transporte a direcciones de red.
b) Multiplexado de las conexiones de transporte a conexiones de red.
c) Establecimiento y Cierre de conexiones de transporte.
d) Control de secuencia entre conexiones punto a punto.
e) Detección de errores en conexiones punto a punto.
f) Recuperación de errores en conexiones punto a punto.
g) Segmentación, concatenación y reagrupación en conexiones punto a punto.
h) Control de flujo en conexiones punto a punto.
i) Funciones de supervisión.
j) Transferencia de Unidades de Datos de Transporte expedita.
k) Suspensión y reinicio de la comunicación.

Capa de Red

La Capa de Red provee los medios para llevar a cabo transmisiones en modo de conexión y modo de desconexión entre entidades de la Capa de Transporte de forma que el enrutado de los paquetes es transparente para la anterior. Esto incluye casos donde la red está formada por múltiples subredes, o redes en paralelo.

En general, las funciones de la Capa de Red permiten soportar una gran variedad de conexiones de red, desde una conexión punto a punto a configuraciones complejas de combinaciones de subredes con diferentes características. Las funciones generales de la Capa de Red se listan a continuación:

a) Enrutado.
b) Multiplexado de conexiones de red.
c) Segmentación y agrupamiento.
d) Detección, notificación y recuperación de errores.
e) Parámetros de calidad del servicio.
f) Secuenciado.

g) Control de flujo.
h) Selección de servicio.
i) Mapeo de direcciones.
j) Conversión de servicios de modo de conexión de la Capa de Enlace de Datos a servicios en modo de desconexión en la Capa de Red.

En modo de conexión, la Capa de Red además proporciona las siguientes características:

a) Establecimiento de Conexiones de red.
b) Transmisión de Unidades de Datos de red.
c) Envió de Unidades de Datos de Forma expedita.
d) Recepción de confirmación.
e) Reseteo.

En modo de desconexión, la Capa de Red además proporciona los siguientes recursos:

a) Transmisión de unidades de datos de red de un tamaño máximo definido.

Capa de Enlace de Datos

La Capa de Enlace de Datos detecta y posiblemente corrige errores que podrían ocurrir en la Capa Física. Además, la Capa de Enlace de Datos habilita la Capa de Red para controlar la interconexión de circuitos de datos en la Capa Física.

Las funciones generales de la Capa de Enlace de Datos se enlistan a continuación:

b) Mapeo de Unidades de Datos con servicios de enlace de datos.
c) Identificación y cambio de parámetros.
d) Control de interconexiones de circuitos de datos.
e) Detección de errores.
f) Recuperación de errores (solo en modo de conexión).
g) Ruteo.
h) Control de secuencia (solo en modo de conexión).
i) Reset (solo en modo de conexión).

Unidades de datos de la Capa de Enlace de Datos

Dependiendo de la aplicación, el tamaño de una Unidad de Datos de servicio de la Capa de Enlace de Datos puede estar limitada por la relación entre la tasa de errores en la conexión física y la capacidad de detectar errores en la Capa de Enlace de datos.

Capa Física

La capa física provee los medios mecánicos, eléctricos y funcionales para activar, mantener y desactivar conexiones físicas para transmisión de bits entre entidades de enlace de datos.

Los servicios que provee la capa física no se pueden definir como modos de conexión, debido a que pueden ser muy diversos. Los servicios que provee la capa física son:

a) Activación y desactivación de conexiones físicas.
b) Terminales de conexión física.
c) Identificación de circuitos de datos.
d) Secuenciado y multiplexado.
e) Notificación de condiciones de falla.
f) Parámetros de calidad de servicio.

Unidades de datos de la capa física

Una unidad de datos de servicio de la capa física consiste en un bit.

Bluetooth

Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,402 a los 2,483.5 MHz. Bluetooth fue inventado en 1994 por la compañía Ericsson como una alternativa a la comunicación serial cableada RS-232 [28].

Figura 1.22. Logo Bluetooth.

Bluetooth es administrado por el Bluetooth SIG (*Special Interest Group*) el cual tiene más de 30,000 miembros en áreas de telecomunicaciones, cómputo, redes y electrónica. La IEEE estandarizó Bluetooth en la IEEE 802.15.1 pero el estándar no es mantenido en la actualidad. El Bluetooth SIG vela por el desarrollo, mantenimiento, y marcas de la especificación. Las versiones más relevantes del estándar se resumen en la Tabla 1.2.

Un dispositivo Bluetooth usa ondas de radio en lugar de cables para conectar a un computador o teléfono. Un producto habilitado con esta tecnología contiene una pequeña computadora con un circuito Bluetooth y software que le facilita conectarse. [29]

Bluetooth es un protocolo basado en paquetes con una estructura maestro-esclavo. Sólo un maestro puede comunicarse con siete esclavos en una *piconet* (red de área personal Bluetooth). Cada paquete es transmitido en uno de los 79 canales designados para Bluetooth, cada canal tiene un ancho de banda de 1MHz. El intercambio de paquetes está basado en el reloj definido por el maestro (con intervalos de tiempo de 312.5us), el cual rige en qué intervalos los esclavos y el maestro comparten información con un tamaño de paquete definido. Todos los esclavos comparten el reloj del maestro.

Los dispositivos Bluetooth son clasificados en tres clases de acuerdo con su potencia como se muestra en la Tabla 1.2.

Clase	Nivel máximo de potencia		Rango típico (m)
	(mW)	(dBm)	
1	100	20	~100
2	2.5	4	~10
3	1	0	~1

Tabla 1.2. Clases de Bluetooth de acuerdo a potencia de salida. [31]

Bluetooth 5

De acuerdo al Bluetooth SIG [30], el Bluetooth 5 es una actualización que incrementará significativamente el rango, la velocidad, y la capacidad de *broadcasting* de las aplicaciones Bluetooth. Bluetooth 5 cuadruplicará el rango y doblará la velocidad de transmisión de las conexiones de Baja Energía (BLE). Mientras que, en la trasmisión punto-multipunto de datos en modo de desconexión, incrementará la capacidad en ocho veces.

Bluetooth Core Especificación

La especificación *Bluetooth Core* [31] define los bloques constructivos que los desarrolladores utilizan para crear dispositivos interoperables que construyen el ecosistema Bluetooth. Las implementaciones más importantes de la especificación son Bluetooth BR/EDR ("Basic Rate/Enhanced Data Rate" por sus siglas en inglés) y Bluetooth LE ("Low Energy" por sus siglas en inglés). Cada implementación usa un circuito integrado diferente para cumplir con los requerimientos mínimos de hardware.

Bluetooth BR/EDR

Establece una conexión inalámbrica continua de rango corto, lo cual lo hace ideal para casos como transmisión de audio.

Bluetooth 1.1	Bluetooth 1.2	Bluetooth 2.0	Bluetooth 2.1	Bluetooth 3.0	Bluetooth 4.0	Bluetooth 5
2002 Se corrigen errores de la versión 1.0 Se convierte en el estándar IEEE 802.15.1-2002 Se agrega el indicador de potencia de señal (RSSI)	2003 Mejoras en velocidad de conexión y descubrimiento Se mejora la resistencia a la interferencia utilizando espectro ensanchado por salto de frecuencia adaptable con una velocidad de transmisión 721 kbit/s	2004 Introducción de EDR (*Enhanced Data Rate*) para mejorar la velocidad de trasmisión a 3Mbit/s Uso de π/4-DQPSK y 8DPSK para la transferencia de datos	2007 Mejora la conexión y la seguridad entre dispositivos Bluetooth Bluetooth SIG se hace cargo de mantener la especificación	2009 Se introduce el modo de alta velocidad o “HS” (*High Speed* de sus siglas en inglés) donde el enlace Bluetooth solo es usado para establecer la conexión y un enlace Wi-Fi es usado para transmitir la información Mejoras en el control del consumo de potencia	2010 Adición del protocolo de Bluetooth de Baja Energía (BLE) Definición de Bluetooth Smart (Bluetooth clásico + HS + LE)	2016 Rango x4 Orientado a *Internet of Things* (IOT) BLE transmite a 2Mbits/s BLE de largo alcance

Tabla 1.3. Versiones de la especificación de Bluetooth.

Bluetooth LE

Permite enviar cadenas cortas de información sobre una conexión de largo alcance con un consumo menor de energía, lo que lo hace ideal para aplicaciones de Internet de las Cosas (o IoT), donde generalmente no se necesita una conexión continua, pero de bajo consumo eléctrico.

Modo dual

Los Circuitos integrados en modo dual están disponibles para dispositivos como smartphones o tabletas que necesitan conectarse a dispositivos BR/EDR (como auriculares inalámbricos) o dispositivos LE de bajo consumo (como “wearables” y sensores inalámbricos).

La pila de protocolos Bluetooth

La especificación Bluetooth permite la interoperabilidad entre los sistemas abiertos al definir los mensajes de protocolo que son intercambiados entre las capas equivalentes. También, habilita la interoperabilidad entre los diferentes subsistemas Bluetooth definiendo la interfaz común entre los Controladores y Anfitriones Bluetooth. Los protocolos de Radio (capa física), Link Control y Link Manager (Capa de Enlace) son comúnmente agrupados en un subsistema conocido como el Controlador Bluetooth. Esta es una implementación común que utiliza la interfaz opcional *Host Controller Interface* (HCI), la cual permite una comunicación full dúplex con lo restante del sistema Bluetooth, llamado el Bluetooth Host o Anfitrión.

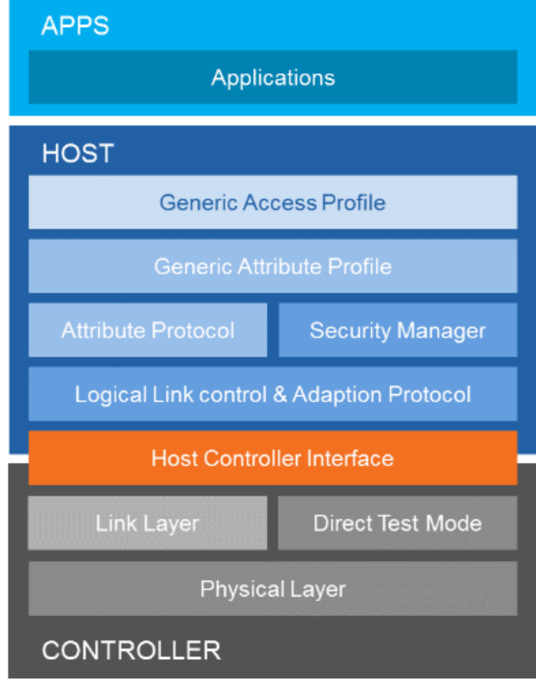


Figura 1.23. *Stack* de la especificación Bluetooth [33].

La capa Física
Controla la transmisión y recepción de los canales de comunicación Bluetooth en la banda de los 2.4GHz.

Capa de Enlace
Define la estructura de los paquetes y canales, procedimientos de descubrimiento y conexión, así como la recepción y envió de datos.

Direct Test Mode
Les permite a los desarrolladores instruir a la capa física para que envíe o reciba una secuencia de paquetes definida.

Capa L2CAP (Logical Link Control and Adaptation Protocol)
Protocolo basado en paquetes que transmite información al HCI o directamente a la Capa de Enlace en sistemas sin un anfitrión. Soporta multiplexado, segmentación, re-ensamblaje, y parámetros de calidad del servicio.

Attribute Protocol (ATT)
Define el protocolo para intercambiar información entre el cliente/servidor una vez que la conexión ha sido establecida.

Security Manager
Define el protocolo que administra el emparejamiento, la autentificación y el encriptado entre los dispositivos Bluetooth.

Generic Attribute Profile (GATT) y Generic Access Profile (GAP)
El GATT agrupa los servicios que encapsulan el comportamiento de un dispositivo en particular (como descubrimiento, lectura, escritura y notificación). GATT es solo usado en dispositivos Bluetooth LE.

El GAP define los procedimientos y los roles relacionados al descubrimiento de dispositivos Bluetooth y el intercambio de información.

Bluetooth LE

Como parte de la especificación Bluetooth Core versión 4.0, lanzada el 30 de junio del 2010, la especificación Bluetooth LE o de Baja Energía tiene la intención de reducir el costo y la potencia consumida mientras se mantiene un rango de comunicación similar. STMicroelectronics, AMICCOM, CSR, Nordic Semiconductor y Texas Instruments son algunos ejemplos de las compañías que tienen soluciones para Bluetooth LE. [32]

La eficiencia energética de la tecnología Bluetooth LE la hace perfecta para dispositivos que corren por largos periodos sobre fuentes de poder como baterías de moneda, o por cosechamiento de energía. El soporte nativo de Bluetooth en los sistemas operativos predominantes habilita el desarrollo de una gran variedad de dispositivos conectados, desde electrodomésticos y sistemas de seguridad hasta bandas deportivas y sensores de proximidad. [33]

Entre las principales características del Bluetooth LE se encuentran:

- Consumo de energía ultra-bajo.
- Encriptado 128-bit AES.
- Arquitectura de desarrollo estandarizada.

Bluetooth versus Wi-Fi

Bluetooth y Wi-Fi (IEEE 802.11) tienen aplicaciones similares: configurar redes y compartir archivos. Sin embargo, mientras Wi-Fi está intencionado como un remplazo a Ethernet (cableado de área local de alta velocidad), Bluetooth está orientado a equipos portátiles personales y sus aplicaciones (electrodomésticos, sensores de campo, etc.).

Wi-Fi y Bluetooth se complementan en el sentido de las aplicaciones que los utilizan. Wi-Fi usualmente está centrado en un punto de acceso, donde las conexiones cliente servidor son asimétricas, con todo el tráfico de datos enrutado a través del punto de acceso; mientras que, en caso de Bluetooth, las conexiones son simétricas entre dos dispositivos Bluetooth maestro y esclavo. Wifi se ajusta mejor en aplicaciones donde se requiere cierto grado de configuración cliente-servidor y altas velocidades. [28]

Wi-Fi

Wi-Fi es una tecnología para redes inalámbricas de área local (WLAN) para dispositivos basados en el estándar IEEE 802.11. El estándar IEEE 802.11 es un conjunto de especificaciones de Capa Física (PHY) y Control de Acceso al Medio (MAC por sus siglas en inglés) para la implementación de redes de comunicación inalámbricas de área local en las bandas de frecuencia de 900MHZ y 2.4, 3.6, 5 y 60 GHz. Wi-Fi es una marca registrada de la Alianza Wi-Fi, la cual restringe el uso del término *Certificado para Wi-Fi* a productos que cumplen satisfactoriamente con la certificación de interoperabilidad. [34]

La Alianza Wi-Fi es una red mundial de compañías que define la tecnología Wi-Fi. Además, certifica que los productos Wi-Fi cumplan con los estándares, desempeño, seguridad y calidad. [35]

Figura 1.24. Logo de la Wi-Fi Alliance.

Historia

La tecnología basada en 802.11 tiene sus orígenes en 1985 cuando la Comisión Federal de Comunicaciones de EE. UU. libera la banda ISM para su uso sin licencia.

En 1991 la AT&T y NCR (una compañía de cajas registradoras) inventaron el precursor del 802.11 en Países Bajos. Los inventores inicialmente tenían la intensión de usar la tecnología para sistemas de cajas registradoras. Los primeros productos fueron traídos al mercado bajo el nombre de *WaveLAN,* con tasas de transferencia de 1Mbit/s y 2Mbit/s.

La primera versión del protocolo 802.11 fue liberada en 1997, la cual proveía velocidades de enlace de hasta 2Mbit/s. Esta fue actualizada en 1999 con la 802.11b para permitir velocidades de hasta 11Mbit/s.

En 1999 la Alianza Wi-Fi fue formada para mantener la marca Wi-Fi, sobre la cual la mayoría de los productos son vendidos.

El estándar IEEE 802

Las redes IEEE 802 usan comunicaciones basadas en *frames* o tramas (Unidades de Datos asíncronas con direcciones de fuente y destino) sobre diferentes medios físicos para conectar diversos aparatos digitales, sin importar su tecnología o tipo de información. En un sistema basado en frames, el formato de la información es una secuencia de octetos de longitud variable. En contraste, una comunicación basada en células transmite Unidades de Datos de una longitud fija en intervalos de tiempo específicos. El objetivo principal de la familia de estándares IEEE 802 es definir la Capa Física (PHY) y la Capa de Enlace de Datos (DLL).

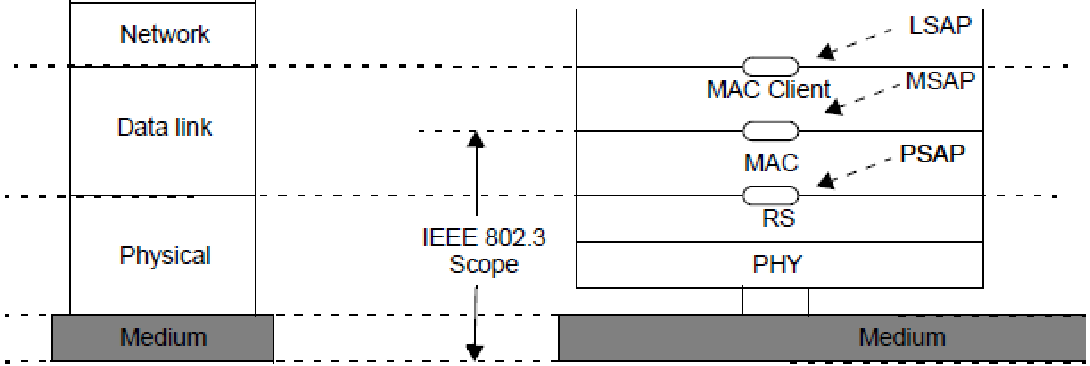


Figura 1.25. Modelo básico de referencia IEEE 802 [36].

El estándar IEEE 802.11 Wi-Fi

El grupo de estándares 802.11 es un miembro a su vez de la familia de estándares del IEEE 802. El estándar 802.11 se apoya en la especificación de la Capa de Acceso al medio común a las tecnologías LAN, es decir, al control de enlace lógico (LLC), incluyendo además la capa MAC las cuales conforman la Capa de Enlace de Datos; así como la Capa Física la cual está dividida a su vez en PLCP (*Physical Layer Convergence Procedure*) y PMD (*Physical Medium Dependent*) como se muestra en la Figura 1.26.

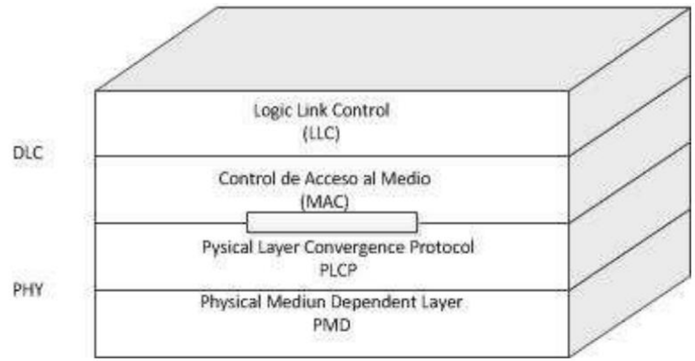


Figura 1.26. Pila de protocolos IEEE 802.11 [37].

La versión original el estándar 802.11 tenía el propósito de proveer tasas de transmisión de 1 a 2 Mbps operando en la banda no licenciada ISM (*Industrial, Scientifical and Medical band*) de 2.45 GHz, la cual requería el uso de técnicas de espectro disperso, por lo cual, se definieron dos modalidades: salto en frecuencia (*Frequency-Hopping Spread-Spectrum*, FHSS) y esparcimiento en secuencia directa (*Direct Sequence Spread-Spectrum*, DSSS), los cuales resultaban ser incompatibles.

Conforme los usuarios requerían mayores tasas de transmisión, se formaron dos grupos que investigaban diferentes esquemas: DSSS para el 802.11b que proponía 11 Mbps en el canal de 20 MHz; y *Orthogonal Frequency Division Multiplexing* (OFDM) para el 802.11a que especificaba una capa física alterna, brindando tasas de hasta 54 Mbps en la banda de 5 GHz, la cual está menos saturada.

Estándar	Alcances	Ancho de banda [Mbps]	Técnica de difusión
802.11	Define estándar para WLAN para capa PHY y MAC	2	FHSS, DSSS, IR
802.11a	Define una capa física de alta velocidad en la banda de 515-5.825 GHz.	6,12,24 opcional 54	OFDM con 24 subportadoras
802.11b	Define una capa física de alta velocidad en la banda de 2.4 GHz.	11	HR/DS, HR/DSSS
802.11e	Mejora del estándar original para implementar QoS (aplica a 802.11a/b/g)		
802.11g	Define una tasa de datos más alta en la capa física de 2.4 GHz.	22-54	ERP
802.11h	Define funciones MAC para permitir a equipos 802.11a cumplir con los requerimientos europeos.	6,12,24 opcional 54	OFDM
802.11i	Mejora de la capa MAC para proveer seguridad en 802.11a/b/g.		

Tabla 1.4. Variantes del IEEE 802.11 [37].

Las técnicas de difusión que resultaron del esfuerzo de mejorar las tasas de transmisión se denominan de acuerdo con la variante del estándar bajo el cual se definió:

- 802.11a: Orthogonal Frequency Division Multiplexing (OFDM) PHY (hasta 54 Mbps).
- 802.11b: High-Rate Direct Sequence (HR/DS or HR/DSSS) PHY (11 Mbps).
- 802.11g: Extended Rate PHY (ERP) (hasta 24 Mbps efectivos). Esta versión surgió como una mejora para las versiones ya existentes y su principal característica es proveer compatibilidad con los sistemas previos.

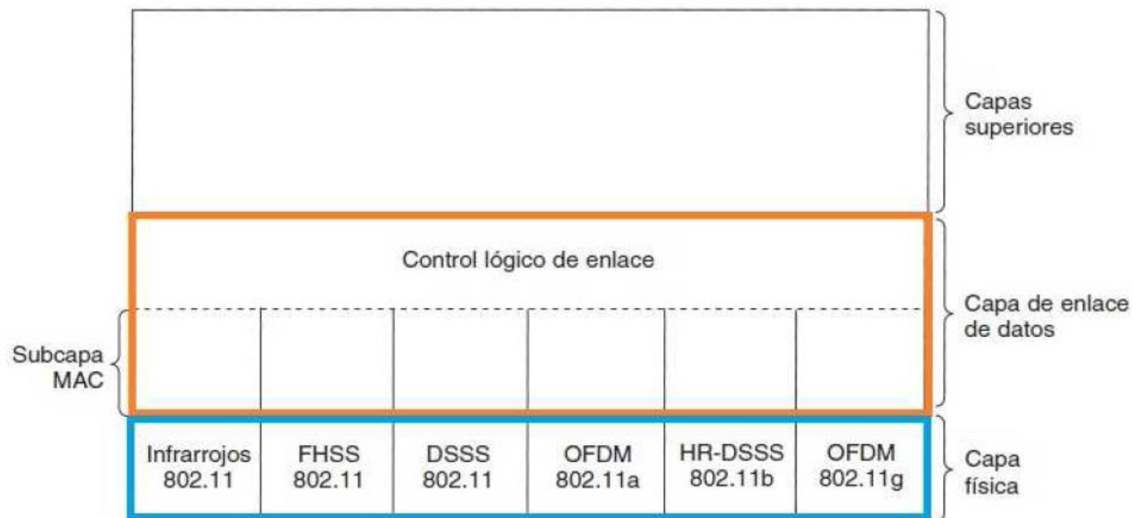


Figura 1.27. Técnicas de transmisión de la Capa Física en Wi-Fi [37].

Canales y Frecuencias

802.11b y 802.11g utilizan el espectro de 2.4-2.5 GHz, el cual es una de las bandas ISM que operan en los Estados Unidos bajo la Parte 15 de las Reglas y Regulaciones. Debido a esta elección de banda de frecuencia, 802.11b y g ocasionalmente sufren interferencia por parte de hornos de microondas, teléfonos inalámbricos, y dispositivos Bluetooth. 802.11a y 802.11n usan la más estrictamente regulada banda de los 4.915 – 5.825 GHz. Cada espectro es sub-dividido en "canales" con una frecuencia central y un ancho de banda como se muestra en la Figura 1.28 para la banda de 2.4GHz.

La banda de 2.4GHz es dividida en 14 canales espaciados 5MHz, empezando por el canal 1, el cual tiene frecuencia central en 2.412 GHz. Estos canales pueden tener restricciones adicionales dependiendo de las regulaciones de cada país. [38]

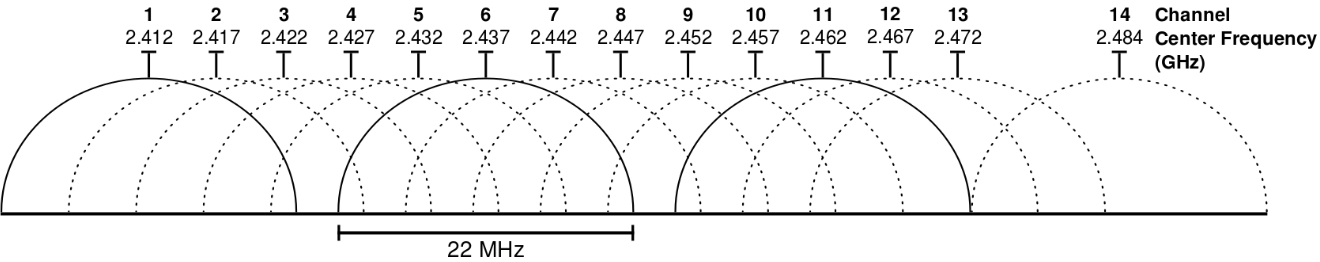


Figura 1.28. Representación gráfica de los canales de la banda 2.4GHz [38].

Las asignaciones de espectro electromagnético y las limitaciones operacionales no son consistentes en todo el mundo: Australia y Europa permiten dos canales adicionales (12,13) además de los 11 permitidos en EE. UU. para la banda de 2.4GHz, mientras que Japón tiene tres más (12 – 14). [34]

Una señal Wi-Fi ocupa cinco canales en la banda de 2.4GHz. Cualquier par de canales espaciados entre sí por cinco o más canales, como los canales 2 y 7, no se traslapan.

La banda de los 5GHz ofrece al menos 23 canales que no se traslapan, en comparación con la banda 2.4GHz donde los canales adyacentes se sobreponen. [34]

Aspectos fundamentales de una aplicación en Android

Las aplicaciones de Android se escriben en lenguaje de programación Java. Las herramientas de Android SDK compilan el código, junto con los archivos de recursos y datos, en un *paquete de Android*, que es un archivo de almacenamiento con el sufijo ".apk". Un archivo de APK incluye todos los contenidos de una aplicación de Android y es el archivo que usan los dispositivos con tecnología Android para instalar la aplicación. [39]

Una vez instalada en el dispositivo, cada aplicación de Android se aloja en su propia zona de seguridad:

- El sistema operativo Android es un sistema Linux multiusuario en el que cada aplicación es un usuario diferente.
- De forma predeterminada, el sistema le asigna a cada aplicación una ID de usuario de Linux única (solo el sistema utiliza la ID y la aplicación la desconoce). El sistema establece permisos para todos los archivos en una aplicación de modo que sólo el ID de usuario asignado a esa aplicación pueda acceder a ellos.
- Cada proceso tiene su propio equipo virtual (EV), por lo que el código de una aplicación se ejecuta de forma independiente de otras aplicaciones.
- De forma predeterminada, cada aplicación ejecuta su proceso de Linux propio. Android inicia el proceso cuando se requiere la ejecución de alguno de los componentes de la aplicación, luego lo cierra cuando el proceso ya no es necesario o cuando el sistema debe recuperar memoria para otras aplicaciones.

De esta manera, el sistema Android implementa el *principio de mínimo privilegio*. Es decir, de forma predeterminada, cada aplicación tiene acceso sólo a los componentes que necesita para llevar a cabo su trabajo y nada más. Esto crea un entorno muy seguro en el que una aplicación no puede acceder a partes del sistema para las que no tiene permiso.

Sin embargo, hay maneras en las que una aplicación puede compartir datos con otras aplicaciones y en las que una aplicación puede acceder a servicios del sistema:

- Es posible disponer que dos aplicaciones compartan la misma ID de usuario de Linux para que puedan acceder a los archivos de la otra. Para conservar recursos del sistema, las aplicaciones con la misma ID de usuario también pueden disponer la ejecución en el mismo proceso de Linux y compartir el mismo EV (las aplicaciones también deben estar firmadas con el mismo certificado).
- Una aplicación puede solicitar permiso para acceder a datos del dispositivo como los contactos de un usuario, los mensajes de texto, el dispositivo de almacenamiento (tarjeta SD), la cámara, Bluetooth y más. El usuario debe garantizar de manera explícita estos permisos.

Componentes de una aplicación

Los componentes de una aplicación son bloques de creación esenciales para Android. Cada componente es un punto diferente a través del cual el sistema puede ingresar a una aplicación. No todos los componentes son puntos de entrada reales para el usuario y algunos son dependientes entre sí, pero cada uno existe como entidad individual y cumple un rol específico.

Hay cuatro tipos diferentes de componentes. Cada tipo tiene un fin específico y un ciclo de vida diferente que define cómo se crea y se destruye el componente.

Actividades

Una *actividad* representa una pantalla con interfaz de usuario y se implementa por medio de una subclase ***Activity***. Por ejemplo, una aplicación de correo electrónico tiene una actividad que muestra una lista de los correos electrónicos nuevos, otra actividad para redactar el correo electrónico y otra actividad para leer correos electrónicos. Si bien las actividades trabajan juntas para proporcionar una experiencia de usuario consistente en la aplicación de correo electrónico, cada una es independiente de las demás. De esta manera, una aplicación diferente puede iniciar cualquiera de estas actividades (si la aplicación de correo electrónico lo permite). Por ejemplo, una aplicación de cámara puede iniciar la actividad en la aplicación de correo electrónico que redacta el nuevo mensaje para que el usuario comparta una imagen.

Servicios

Un *servicio* es un componente que se ejecuta en segundo plano para realizar operaciones prolongadas o tareas para procesos remotos. Un servicio se implementa como una subclase de ***Service***. Un servicio no proporciona una interfaz de usuario. Por ejemplo, un servicio podría reproducir música en segundo plano mientras el usuario se encuentra en otra aplicación, o podría capturar datos en la red sin bloquear la interacción del usuario con una actividad. Otro componente, como una actividad, puede iniciar el servicio y permitir que se ejecute o enlazarse a él para interactuar.

Proveedores de contenido

Un *proveedor de contenido* administra un conjunto compartido de datos de la app. Un proveedor de contenido se implementa como una subclase de ***ContentProvider*** y debe implementar un conjunto estándar de API que permitan a otras aplicaciones realizar transacciones. A través del proveedor de contenido, otras aplicaciones pueden consultar o incluso modificar los datos a los que el dispositivo puede acceder (si el proveedor de contenido lo permite). Por ejemplo, el sistema Android proporciona un proveedor de contenido que administra la información de contacto del usuario.

Receptores de mensajes

Un *receptor de mensajes* es un componente que responde a los anuncios de mensajes en todo el sistema. Un receptor de mensajes se implementa como una subclase de ***BroadcastReceiver*** y cada receptor de mensajes se proporciona como un objeto ***Intent***. Muchos mensajes son originados por el sistema; por ejemplo, un mensaje que anuncie que se apagó la pantalla, que la batería tiene poca carga o que se tomó una foto. Las aplicaciones también pueden iniciar mensajes; por ejemplo, para permitir que otras aplicaciones sepan que se descargaron datos al dispositivo y están disponibles para usarlos. Si bien los receptores de mensajes no exhiben una interfaz de usuario, pueden crear una notificación de la *barra de estado* para alertar al usuario cuando se produzca un evento de mensaje. Aunque, comúnmente, un receptor de mensajes es simplemente una "puerta de enlace" a otros componentes y está destinado a realizar una cantidad mínima de trabajo. Por ejemplo, podría iniciar un servicio para que realice algunas tareas en función del evento.

Un aspecto exclusivo del diseño del sistema Android es que cualquier aplicación puede iniciar un componente de otra aplicación. Por ejemplo, si se requiere que el usuario tome una foto con la cámara del dispositivo, se puede usar una aplicación existente. En lugar de desarrollar una actividad para tomar una fotografía, simplemente se inicia la actividad en la aplicación de cámara que toma la foto.

Cuando el sistema inicia un componente, inicia el proceso para esa aplicación (si es que no se está ejecutando) y crea una instancia de las clases necesarias para el componente. A diferencia de lo que sucede con las apps en la mayoría de los demás sistemas, las apps de Android no tienen un solo punto de entrada **(no existe la función *main ()*).**

Como el sistema ejecuta cada aplicación en un proceso independiente con permisos de archivo que limitan el acceso a otras aplicaciones, una aplicación no puede activar directamente un componente de otra aplicación. Sin embargo, el sistema Android sí puede hacerlo. Por lo tanto, para activar un componente en otra aplicación, se debe enviar un mensaje al sistema que especifique el *intent* de iniciar un componente específico.

Activación de componentes

Tres de los cuatro tipos de componentes (actividades, servicios y receptores de mensajes) se activan mediante un mensaje asincrónico llamado *intent*. Los intents enlazan componentes individuales en tiempo de ejecución independientemente de a que aplicación corresponda el componente. Un intent se crea con un objeto ***Intent***, que define un mensaje para activar un componente específico o un *tipo* específico de componente; un intent puede ser explícito o implícito, respectivamente.

Para actividades y servicios, un intent define la acción a realizar (por ejemplo, "ver" o "enviar" algo) y puede especificar el URI de los datos en los que debe actuar (entre otras cosas que el componente que se está iniciando podría necesitar saber). Por ejemplo, un intent podría transmitir una solicitud para que una actividad muestre una imagen o abra una página web. En algunos casos, se puede iniciar una actividad para recibir un resultado; en cuyo caso, la actividad también devuelve el resultado en un Intent.

Para los receptores de mensajes, el intent simplemente define el anuncio que se está transmitiendo (por ejemplo, un mensaje para indicar que la batería del dispositivo tiene poca carga incluye sólo una cadena de texto o “String” de acción conocida que indica “batería baja”).

Un proveedor de contenido no se activa mediante intents, sino a través de solicitudes de un ***ContentResolver***. El solucionador de contenido aborda todas las transacciones directas con el proveedor de contenido, de modo que el componente que realiza las transacciones con el proveedor no deba hacerlo y, en su lugar, llame a los métodos del objeto ContentResolver. Esto deja una capa de abstracción entre el proveedor de contenido y el componente que solicita información (por motivos de seguridad).

El archivo de manifiesto

Para que el sistema Android pueda iniciar un componente de la app, el sistema debe reconocer la existencia de ese componente leyendo el archivo *AndroidManifest.xml* de la APP (el archivo de “manifiesto”). Una aplicación debe declarar todos sus componentes en este archivo, que debe encontrarse en la raíz del directorio de proyectos de la aplicación.

Además de declarar los componentes de la aplicación, un archivo de manifiesto:

- Identifica los permisos de usuario que requiere la aplicación, como acceso a Internet o acceso de lectura para los contactos del usuario.
- Declara el nivel de API mínimo (o versión de Android) requerido por la aplicación en función de las API que usa la aplicación.
- Declara características de hardware y software que la aplicación usa o exige, como una cámara, servicios de Bluetooth o una pantalla multitáctil.
- Bibliotecas de la API a las que la aplicación necesita estar vinculada (además de las *Android framework API*).

Declaración de componentes

La tarea principal del manifiesto es informarle al sistema acerca de los componentes de la aplicación. Por ejemplo, un archivo de manifiesto puede declarar una actividad de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

Figura 1.29. Ejemplo de archivo de manifiesto.

- En el elemento <**application**>, el atributo ***android:icon*** señala los recursos para un ícono que identifica la app.
- En el elemento <**activity**>, el atributo ***android:name*** especifica el nombre de clase plenamente calificado de la subclase Activity y el atributo ***android:label*** especifican una string para usar como etiqueta de la actividad visible para el usuario.

Declaración de capacidades de los componentes

Se puede iniciar un componente de aplicación al denominar explícitamente el componente objetivo (usando el nombre de clase del componente) en el intent. Sin embargo, el poder real de los intents se encuentra en el concepto de ***intents implícitos***. Un intent implícito simplemente describe el tipo de acción a realizar (y, opcionalmente, los datos en función de los cuales quieres realizar la acción) y le permite al sistema buscar un componente en el dispositivo que pueda realizar la acción e iniciarla. Si hay múltiples componentes que pueden realizar la acción que describe la intent, el usuario selecciona la que quiere usar.

El sistema identifica los componentes que pueden responder a una intent comparando la intent recibida con los *filtros de intents* que se proporcionan en el archivo de manifiesto de otras apps en el dispositivo.

Cuando se declara una actividad en el manifiesto, se tiene la posibilidad de incluir filtros de intents que declaran las capacidades de la actividad de modo que pueda responder a intents de otras aplicaciones.

El siguiente ejemplo muestra el manifiesto que se utilizaría para que una actividad que redacta un correo electrónico responda a los intents "enviar" (***android.intent.action.SEND***):

```
<manifest ... >
    ...
    <application ... >
        <activity android:name="com.example.project.ComposeEmailActivity">
            <intent-filter>
                <action android:name="android.intent.action.SEND" />
                <data android:type="*/*" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Figura 1.30. Ejemplo de filtros intent.

Declaración de requisitos de la aplicación

Existen muchos dispositivos con tecnología Android, pero no todos ofrecen las mismas funciones y capacidades. Para evitar que se instale una aplicación en dispositivos que no tienen las funciones que ésta necesita, es importante definir claramente un perfil para los tipos de dispositivos que admite la aplicación al declarar los requisitos de dispositivos y software en el archivo de manifiesto. La mayoría de esas declaraciones son solo informativas y el sistema no las lee, pero servicios externos como Google Play (tienda en línea de aplicaciones) sí lo hacen para ofrecerles a los usuarios opciones de filtrado cuando buscan aplicaciones desde sus dispositivos.

Por ejemplo, si tu aplicación requiere una cámara y usa API introducidas en Android 2.1 (nivel de API 7), debes declarar esto como requisitos en tu archivo de manifiesto de la siguiente manera:

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera.any"
                  android:required="true" />
    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />
    ...
</manifest>
```

Figura 1.31. Manifiesto que requiere una cámara y las API introducidas en Android 2.1 (nivel 7) como mínimo.

Recursos de la aplicación

En Android, una aplicación está compuesta por más que simplemente código; requiere recursos independientes del código fuente, como imágenes, archivos de audio y otros elementos relacionados con la presentación de la aplicación. Por ejemplo: animaciones, menús, estilos, colores y el diseño de las interfaces de usuario. El uso de recursos de la aplicación facilita la actualización de varias características de una aplicación sin necesidad de modificar el código y, al proporcionar conjuntos de recursos alternativos, es posible optimizar la aplicación para una variedad de configuraciones de dispositivos (como diferentes idiomas y tamaños de pantalla).

Para cada recurso que se incluye en un proyecto para Android, el SDK asigna un número entero de identificación o "ID" que se puede usar para hacer referencia al recurso desde el código de la aplicación o desde otros recursos. Todos los ID pertenecen al directorio "R".

Uno de los aspectos más importantes de proporcionar recursos independientes del código fuente es la capacidad de ofrecer recursos alternativos para diferentes configuraciones de dispositivos. Por ejemplo, al definir strings o cadenas de texto en XML, es posible traducir las strings a otros idiomas y guardarlas en archivos independientes sin necesidad de alterar el código en Java.

Capítulo 2. La Interfaz Gráfica en Arduino Total Control

La interfaz gráfica en Arduino Total Control (ATC) se plantea mediante un diseño Top-Down, como se muestra en la Figura 2.1, en donde se definen los elementos que la constituyen. La interfaz gráfica o *Layout* en ATC está constituida por botones, imágenes, textos, barras accionables y paneles táctiles. Un Layout en ATC es personalizable, retroalimentado, adaptable, y de orientación fija. La aplicación ATC proporciona hasta 4 Layouts independientes.

Características de un Layout en ATC:

- **<u>Personalizable</u>**. El usuario puede personalizar la cantidad, el tamaño, el color y el contenido de los elementos constructivos.
- **<u>Retroalimentado</u>**. El sistema a base de microcontrolador puede retroalimentar el estado de los elementos constructivos en tiempo real.
- **<u>Adaptable</u>**. Se adapta dimensionalmente a cualquier tamaño de pantalla.
- **<u>Orientación Fija</u>**. La interfaz no cambia de orientación si el usuario rota la pantalla físicamente. Para cambiar de orientación, el usuario debe solicitarlo explícitamente utilizando las herramientas de edición secundarias.

Elementos constructivos de un Layout:

- **Imagen**. Es un elemento visual para la creación de un Layout. El usuario puede usarlo para relacionarlo con la acción de otro elemento constructivo.
- **Botón de activación** (o *Toggle Button*). Es un botón que cambia de estado y envía información cada vez que es presionado. Un *toggle button* se mantiene en el estado anterior hasta que es presionado nuevamente.
- **Botón temporal** (o *Temporary Button*). Es una imagen con el método *onTouchListener* habilitado. Este tipo de botón envía información cuando es presionado, así como cuando se deja de presionar.
- **Paneles táctiles** (o *Touch Pad***)**. Es una imagen con el método *onTouchListener* habilitado para seguir la coordenada donde se “tocó” el objeto. Este elemento envía información de posición en “X” y “Y” al microcontrolador mientras se mantenga presionado.
- **Reconocimiento de voz (**o *Speech Recognition***)**. Es una imagen con el método *onTouchListener* habilitado. Cuando se hace “clic” en este elemento, se inicia la actividad de reconocimiento de voz. El texto reconocido se envía al microcontrolador.
- **Texto**. Es una cadena de texto personalizable que el usuario puede usar para relacionarla con otro elemento constructivo.
- **Acelerómetro**. Es un texto que muestra y envía el valor del acelerómetro del dispositivo móvil en “X”, “Y” o “Z” al microcontrolador.
- **Barra deslizante** (o Seek Bar). Es una barra de acción que le permite al usuario enviar un número entero de 0 a 255 al microcontrolador al deslizar el dedo sobre ella.
- **Barra analógica** (o *Analog Bar*). Es un objeto personalizado descendiente de la clase View. Muestra un nivel de llenado dependiendo de su valor actual. El valor de una barra analógica es definido en “tiempo real” por el microcontrolador.

Herramientas de edición primarias

- **Editar propiedades**. Permite cambiar color, tamaño, contenido (texto o datos a enviar) y función (en caso de imágenes y textos) de los elementos constructivos de un Layout.
- **Agregar Objeto**. Es la herramienta utilizada para agregar objetos nuevos al Layout.
- **Mover Objeto**. Le permite al usuario trasladar un objeto en el plano del Layout.
- **Limpiar Layout**. Elimina todos los objetos presentes en el Layout.

Herramientas de edición secundarias

- **Orientación de Layout**. Cambia la orientación de la interfaz relativa al dispositivo móvil en vertical u horizontal.
- **Cambiar Layout**. Permite seleccionar uno de los 4 Layouts personalizables que proporciona la aplicación.
- **Exportar / Importar Layout**. Es la herramienta utilizada para compartir un Layout. Empaqueta los contenidos y configuraciones de un Layout en una carpeta exportable a cualquier dispositivo móvil Android.
- **Editar fondo de pantalla**. Le permite al usuario cambiar el fondo de pantalla del Layout activo.
- ***Scroll Enable***. Habilita el deslizamiento vertical de la pantalla o Layout.

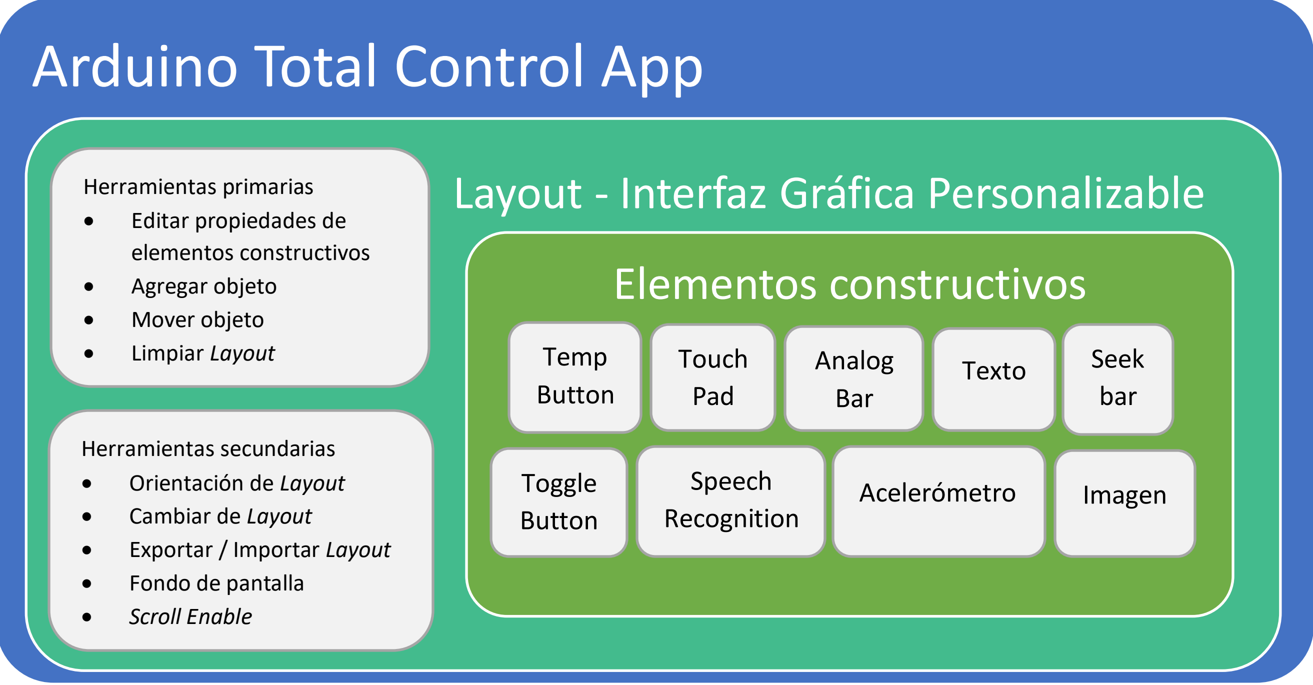


Figura 2.1. Diseño Top-Down de la Interfaz Gráfica en ATC.

Clases de la Aplicación Arduino Total Control

Al mismo tiempo de utilizar el Kit de Desarrollo de Software de Android Nivel 21(Android SDK), el cual incluye la plataforma y las librerías pertenecientes al nivel de la API 21 (Android 5.0 “Lollipop”), la aplicación define las clases [40] listadas en la Tabla 2.1.

Clase	Superclase	Uso (interfaz Gráfica o comunicaciones)
Main	Activity	Interfaz Gráfica / Comunicaciones
AnalogBarView	View	Interfaz Gráfica
CodeAdapter	RecyclerView	Interfaz Gráfica
CodeDrawer	Activity	Interfaz Gráfica
CodeGeneratorClass	Activity	Interfaz Gráfica
ColorPickerDialog	Dialog	Interfaz Gráfica
CustomScrollView	ScrollView	Interfaz Gráfica
DecodedView	N/A	Interfaz Gráfica
EditName	Activity	Interfaz Gráfica
EProp	Activity	Interfaz Gráfica
ExProp	N /A	Interfaz Gráfica
SelectLayout	Activity	Interfaz Gráfica

BLEScanActivity	List Activity	Comunicaciones
BLeService	Service	Comunicaciones
BTScanActivity	Activity	Comunicaciones
BTService	Thread	Comunicaciones
ChannelSelector	Activity	Comunicaciones
IPset	Activity	Comunicaciones
TCPClient	Thread	Comunicaciones

Tabla 2.1. Las clases de java definidas en ATC.

La Tabla 2.1 muestra la superclase de la cual las clases en ATC descienden, así como categoriza el uso que le da a estas. Las clases de Interfaz Gráfica definen el comportamiento de las interacciones con el usuario. Las clases de Comunicaciones se encargan de la conexión inalámbrica con el microcontrolador (Capitulo 3).

Además, para mantener compatibilidad de la interfaz gráfica con versiones anteriores de Android, la aplicación incorpora las librerías de compatibilidad que se muestran en la Tabla 2.2.

com.android.support:support-v13:21.0.2
com.android.support:appcompat-v7:21.0.2
com.android.support:recyclerview-v7:21.0.2
com.android.support:cardview-v7:21.0.2

Tabla 2.2. Librerías de compatibilidad usadas en ATC.

La actividad principal en ATC

La actividad principal, llamada *Main*, es el único punto de entrada a la aplicación ATC (ya que no se definieron *intent-filters* para que otras actividades de la aplicación fueran posibles puntos de entrada). Como se muestra en la Figura 2.2, Main se define en el manifiesto como la actividad que se *lanza* cuando la aplicación es iniciada.

```
<activity
    android:name="com.apps.emim.btrelaycontrol.Main"
    android:configChanges="orientation|keyboardHidden"
    android:label="ATC PRO" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Figura 2.2. Declaración de la aplicación principal en el Manifiesto.

La actividad Main, es la pantalla o interfaz en la que el usuario realiza la mayor parte de la interacción con la aplicación. Main se encarga de iniciar, administrar y terminar: los datos de sesiones anteriores, el Layout, los servicios de comunicaciones inalámbricas y el hardware del dispositivo móvil. El ciclo de vida detallado de la actividad Main se presenta en el diagrama de flujo de la Figura 2.3. Todas las actividades en Android tienen un ciclo de vida bajo el mismo esquema [41].

Debido a que la mayoría de las interacciones microcontrolador-aplicación-usuario se llevan a cabo en la actividad Main, se puede considerar el diagrama de flujo de la actividad Main como un diagrama simplificado de funcionamiento de la aplicación en general.

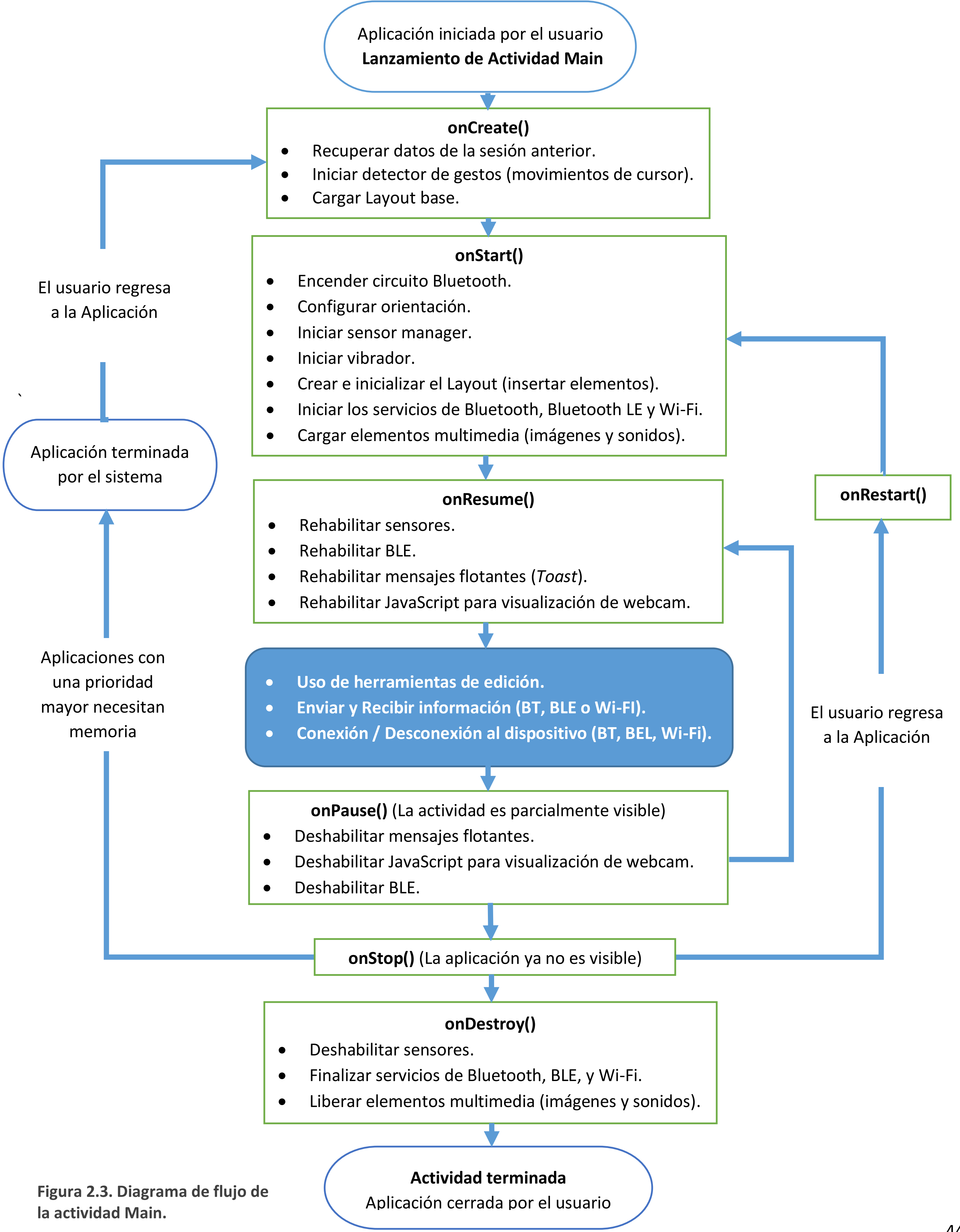


Figura 2.3. Diagrama de flujo de la actividad Main.

Elementos constructivos

Android utiliza objetos que descienden de la clase *View* para presentar textos, imágenes, botones, barras, o cualquier tipo de elemento de interfaz para el usuario. Para lograr la aplicación ATC, se utilizaron clases que extienden la clase View de tres formas distintas; uso de la subclase para "lo que fue diseñada", como en el caso de los ***ToggleButton*** (usado en los botones de activación), ***TextView*** (usado en los textos) y ***SeekBars***; uso de la subclase habilitando métodos que la hacen "parecer un objeto diferente", como en el caso de la clase ***ImageView***, que es utilizada para simular *TouchPads*, *Temporary Buttons*, y Botones de reconocimiento de voz, además de simples Imágenes; o bien, se puede crear una subclase de View completamente nueva, como es el caso de los Displays Analógicos.

Para crear un Layout, es necesario uno o más objetos descendientes de la clase ***ViewGroup***. Un *ViewGroup* es una subclase *View* especial que puede contener a otros objetos *View* (llamados hijos). La clase *ViewGroup* es la base para la creación de Layouts. Esta clase también define los parámetros de Layout (como largo, ancho, orientación, etc.) [42].

Los *ViewGroup* le permiten a la aplicación ATC organizar los elementos de un Layout de forma lineal (o lista), o bien, arbitrariamente en la pantalla, como es el caso de las subclases ***LinearLayout*** y ***RelativeLayout*** respectivamente.

La función BuildLayout

La función ***BuildLayout()*** es la sección de código responsable de crear un Layout "virgen" o base, es decir un Layout con todos los elementos expuestos al usuario organizados en forma de matriz. *BuildLayout()* permite crear un Layout virgen en cualquier dispositivo sin importar el tamaño u orientación de la pantalla. Para lograrlo, calcula programáticamente el tamaño y posición de cada elemento en el Layout. Cada elemento del Layout es parte de un arreglo de objetos del mismo tipo.

Para crear un Layout, el método *BuildLayout* sigue las siguientes condiciones:

Objeto	Orientación Vertical		Orientación Horizontal		Tamaño
	Filas	Columnas	Filas	Columnas	
Imágenes	4	6	3	8	Ancho = Alto = Ancho de la pantalla / columnas
Textos	4	6	3	8	Tamaño de fuente = Ancho de la pantalla / (6 *columnas)
Toggle Button	4	6	3	8	Ancho = Ancho de la pantalla / columnas Alto = Ancho * 2 /3
SeekBars	4	2	3	3	Ancho = Ancho de la pantalla / columnas Alto = no editable
Barras analógicas	4	3	3	4	Ancho = Ancho de la pantalla / columnas Alto = Alto de un Toggle Button

Tabla 2.3. Condiciones para dimensionar un Layout en la función *BuildLayout*.

Botones de activación

Un botón de activación en ATC es un objeto *ToggleButton* que le permite al usuario activar el envío de un comando al microcontrolador en dos estados. La imagen 38 muestra un *ToggleButton* en su estado "on" y "off". Cada Layout de la aplicación puede contar con hasta 24 Botones de activación.

Figura 2.4. Estados de un ToggleButton.

Lectura de cambio de estado del botón

Para que la aplicación pueda responder a los cambios de estado de un *ToggleButton* se siguieron los siguientes pasos:

1. Registrar la actividad Main para recibir el *Callback* "onClick".

```
// Set listeners for ToggleButtons
for (i = 0; i < EProp.RELAY_NO; i++) {
    tbRelay[i].setOnClickListener(this);
}
```

Figura 2.5. Registro de Callback "onClick" en la clase principal (this = Main.class).

2. Implementar el método *onClick* en la clase Main.

```
public class Main extends Activity implements View.OnClickListener, OnTouchListener,
        OnColorChangedListener, OnLongClickListener, SensorEventListener,
        OnSeekBarChangeListener, OnInitListener, GestureDetector.OnGestureListener {
…
@Override
synchronized public void onClick(View v) {
…
}
```

Figura 2.6. Implementación del método onClick en la actividad Main.

3. Cuando se haga clic en cualquier objeto con un *onClickListener*, el método "onClick" será invocado en la actividad principal. Para discriminar el origen del clic, es necesario filtrar el View v por su ID.
   Debido a la gran cantidad de *IDs* de objetos que utilizan el método "onClick" que puede manejar la aplicación (24 Imágenes + 24 Botones + 24 Textos + 8 *SeekBar* + 12 Analog Bar) se creó la Clase ***DecodedView.*** La clase *DecodedView* contiene funciones para cada tipo de objeto, las cuales retornan "true" si el ID forma parte del arreglo del objeto en cuestión, así como el índice que el ID ocupa dentro del arreglo.

```
boolean DecodeButtonID(View v, ToggleButton tbs[]) {
    for(int i = 0; i < tbs.length; i++) {
        if(v.getId() == tbs[i].getId()){
            index = i;
            return true;
        }
    }
    return false;
}
```

Figura 2.7. Función para filtrar el ID de un supuesto Togglebutton.

```
@Override
synchronized public void onClick(View v) {
    DecodedView selectedView = new DecodedView();
    …
    if (selectedView.DecodeButtonID(v, tbRelay))
        ButtonSendCommand(selectedView.index, v);
}
```

Figura 2.8. Filtrado del ID utilizando la función DecodeButtonID.

4. Una vez filtrado el ID del objeto que recibió el "clic", es posible determinar si se enviará un dato correspondiente a un botón encendido ("on") o a un botón apagado ("off") utilizando el método "isChecked()", el cual retorna "true" si el botón está encendido. Esta funcionalidad es encapsulada en el método *ButtonSendCommand* como se puede observar en la Figura 2.9.

Conocer el índice que el View v tiene dentro del arreglo, le permite a la aplicación saber qué cadena de texto enviará a la función *MainSend* (ya que las cadenas de texto *relaycodesBTNON* y *relaycodesBTNOFF* son personalizables y diferentes para cada botón).

```
// Sends the button command via bluetooth and wifi storing button status
void ButtonSendCommand(int selector, View v) {
    ToggleButton localButton = (ToggleButton) v;

    if (localButton.isChecked()) {
        states[selector] = true;
        MainSend(EProp.TYPE_RELAY_BTN, selector,
                relaycodesBTNON[selector][CurrentDisplay]);
    } else {
        states[selector] = false;
        MainSend(EProp.TYPE_RELAY_BTN, selector,
                relaycodesBTNOFF[selector][CurrentDisplay]);
    }
}
```

Figura 2.9. Envío de una cadena de texto a MainSend.

La función *MainSend* "rutea" la información a la interfaz de comunicación activa (Bluetooth, Bluetooth LE o Wi-Fi).

Propiedades editables

Las propiedades editables de un *ToggleButton* se definen en la clase *EProp* (Editar Propiedades, Figura 2.10) y son: tamaño (largo y ancho), comandos "on", comandos "off", color, visibilidad y posición en el Layout (en "X" y "Y").

```
public static final String[][][] BtnSizeFiles  = new String[RELAY_NO][DISPLAY_NO][2];
public static final String[][] CmdBtnOnFiles   = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] CmdBtnOffFiles  = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] BtnColorFiles  = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] BtnVisibilityFiles = new String[RELAY_NO][DISPLAY_NO];
public static final String FILE_BTNX     = "buttonViewX";
public static final String FILE_BTNY     = "buttonViewY";
```

Figura 2.10. Propiedades editables en ATC para un ToggleButton.

Textos

Los "textos" en ATC son implementados usando la clase *TextView*. Cada Layout de la aplicación puede contar con hasta 24 textos.

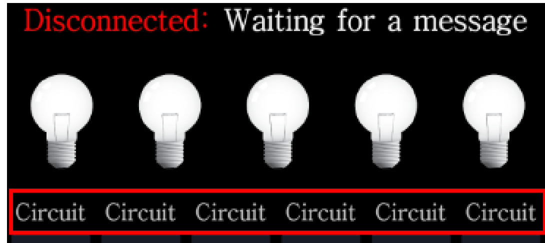


Figura 2.11.En rojo, ejemplos de TextView.

Propiedades editables

Las propiedades de un texto se definen en la clase *EProp* (Figura 2.12) y son: tamaño de fuente, Opciones (para uso como fuente de acelerómetro), contenido, visibilidad y posición en el Layout (en "X" y "Y");

```
public static final String[][] TxtSizeFiles  = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] TextOptionFiles = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] RelayNameFiles  = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] TxtVisibilityFiles = new String[RELAY_NO][DISPLAY_NO];
public static final String FILE_TXTX     = "textViewX";
public static final String FILE_TXTY     = "textViewY";
```

Figura 2.12. Propiedades editables en ATC para un Texto.

Acelerómetro

El elemento constructivo "acelerómetro" es derivado de un texto, cuya opción funcional ha sido habilitada. Un elemento constructivo acelerómetro sólo muestra un eje de aceleración a la vez, es decir, si se requieren lecturas de los tres ejes ("X", "Y" y "Z"), es necesario configurar tres textos como acelerómetros.

Configuración del acelerómetro en Android

Los pasos para habilitar un sensor en la aplicación ATC se muestran a continuación:

1. Declarar las variables tipo "Sensor" y "Sensor Manager".

```
private Sensor Accelerometer;
private SensorManager mSensorManager;
```

Figura 2.13. Declaración de variables tipo sensor.

2. Iniciar el servicio SENSOR_SERVICE y asignar el sensor TYPE_ACCELEROMETER al objeto sensor. Este código se coloca dentro del *callback "onStart()"* ya que sólo se necesita llamar una vez cuando la aplicación inicia.

```
// get sensors
mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
Accelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Figura 2.14. Inicialización del servicio SensorManager y objeto sensor.

3. Rehabilitar el sensor si éste estaba activo (registrando el *callback* o evento en Main). Esta sección de código se coloca dentro *del callback "onResume()"*, ya que es necesario registrar el *callback* de cambio de valor en sensor cada vez que la aplicación vuelve a estar "al frente de la pantalla".

```
if (RunSensor) {
    mSensorManager.registerListener(this, Accelerometer,
            SensorManager.SENSOR_DELAY_UI);
}
```

Figura 2.15. Registro del callback de sensor en Main (this = Main). SENSOR_DELAY_UI es la frecuencia de muestreo del sensor adecuada para una interfaz de usuario.

4. Implementar el método *SensorEventListener* en Main y la función *onSensorChanged.*

```
public class Main extends Activity implements View.OnClickListener, OnTouchListener,
        OnColorChangedListener, OnLongClickListener, SensorEventListener,
        OnSeekBarChangeListener, OnInitListener, GestureDetector.OnGestureListener {
                                        …
    @Override
    public void onSensorChanged(SensorEvent accel) {
                                        …
    }
```

Figura 2.16. Implementación del método onSensorChanged en la actividad Main.

5. Extraer la información de aceleración de la variable *accel*. Los valores de aceleración se retornan en el arreglo *values[ ]*, en donde los elementos 0, 1 y 2 corresponden a los ejes "X", "Y" y "Z" respectivamente.

```
public void onSensorChanged(SensorEvent accel) {
    /*
     * All values are in SI units (m/s^2)
     * values[0]: Acceleration minus Gx on the x-axis
     * values[1]: Acceleration minus Gy on the y-axis
     * values[2]: Acceleration minus Gz on the z-axis
     */
    for (int i = 0; i < EProp.RELAY_NO; i++) {
        String withFormat = null;

        try { // Avoid null pointer exceptions
            switch (TxtSensor[i][CurrentDisplay]) {
                case EProp.OPTION_GX:
                    withFormat = "<AccX:" + get16BitStringNumber(accel.values[0] * 100) + '\n';
                    tvText[i].setText("Gx: " + String.format("%.2f", accel.values[0]));
                    break;
                case EProp.OPTION_GY:
                    withFormat = "<AccY:" + get16BitStringNumber(accel.values[1] * 100) + '\n';
                    tvText[i].setText("Gy: " + String.format("%.2f", accel.values[1]));
                    break;
                case EProp.OPTION_GZ:
                    withFormat = "<AccZ:" + get16BitStringNumber(accel.values[2] * 100) + '\n';
                    tvText[i].setText("Gz: " + String.format("%.2f", accel.values[2]));
                    break;
                default:
                case EProp.OPTION_NONE:
                    break;
            }
            if (withFormat != null) {
                MainBSend(withFormat);
```

Figura 2.17. Extracción de los valores del acelerómetro. Si el texto "i" del "CurrentDisplay" está habilitado como acelerómetro, aplicar formato y enviar a la función MainBSend para procesamiento adicional.

Barras deslizantes

Una barra deslizante en ATC es un objeto *SeekBar* que le permite al usuario enviar un valor entero de 0 a 255 al microcontrolador, dependiendo del "progreso" de la barra. Cada Layout de la aplicación puede contar con hasta 8 *seekbars*.

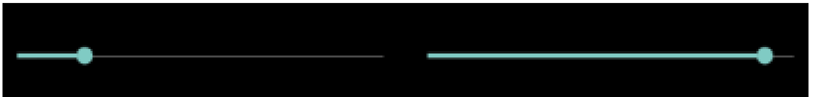


Figura 2.18. Ejemplos de SeekBar con diferentes niveles de progreso.

Lectura de cambio de estado de una *SeekBar*

Los pasos para responder al cambio de estado de una *SeekBar* se muestran a continuación:

1. Configurar el valor máximo de la *SeekBar* y registrar el *callback onSeekBarChangeListener* en Main.

```
for (i = 0; i < EProp.SEEK_BAR_NO; i++) {
    sbBars[i].setMax(255);
    sbBars[i].setOnSeekBarChangeListener(this);
}
```

Figura 2.19. Configuración del valor máximo y registro de evento de SeekBar.

2. Implementar el método *OnSeekBarChangeListener* en la actividad principal.

```java
public class Main extends Activity implements View.OnClickListener, OnTouchListener,
        OnColorChangedListener, OnLongClickListener, SensorEventListener,
        OnSeekBarChangeListener, OnInitListener, GestureDetector.OnGestureListener {
…
 @Override
 public void onProgressChanged(SeekBar seekBar, int progress,
                               boolean fromUser) {
…
}
```

Figura 2.20. Implementación del método onProgressChanged en la actividad principal.

3. Filtrar el ID, aplicar formato y enviar a *MainSend* para procesamiento adicional.

```java
@Override
public void onProgressChanged(SeekBar seekBar, int progress,
                              boolean fromUser) {
    DecodedView selectedView = new DecodedView();
    selectedView.DecodeSeekBarId(seekBar, sbBars);

    if (EditEnable || ChannManagerEnable || MoveEnable || QHideShowEnable) {
        seekBar.setProgress(lastProgress);
        return;
    }
    // Apply format and send
    String withFormat = "<Skb" + selectedView.index + ":" + get16BitStringNumber(progress) + "\n";
    MainSend(EProp.TYPE_RELAY_SB, selectedView.index, withFormat);
}
```

Figura 2.21. Lectura del nuevo valor de la Barra deslizante y envío a función MainSend.

Propiedades editables

Las propiedades de una barra deslizante se definen en la clase *EProp* (Figura 2.22) y son: tamaño, rotación (vertical u horizontal), visibilidad y posición en el Layout (en “X” y “Y”).

```java
public static final String[][][] SbSizeFiles    = new String[SEEK_BAR_NO][DISPLAY_NO][2];
public static final String[][] SbRotateFiles    = new String[SEEK_BAR_NO][DISPLAY_NO];
public static final String[][] SbVisibilityFiles = new String[SEEK_BAR_NO][DISPLAY_NO];
public static final String FILE_SBX       = "sbViewX";
public static final String FILE_SBY       = "sbViewY";
```

Figura 2.22. Propiedades editables en ATC para una SeekBar.

Imágenes

Una imagen en ATC es un objeto *ImageView*, que puede operar como una simple imagen, un botón temporal, un botón con retardo, un botón de reconocimiento de voz o un panel de toque. Cada Layout puede tener hasta 24 *ImageView*.

Para cada imagen ATC define 3 estados: *Default*(0), *Pressed*(1), y *Extra*(2). Una imagen en modo botón (temporal o con retardo) mostrará por defecto el estado “0” o *Default*, y cuando sea presionada, la imagen cambiara a estado “1” o *Pressed*. El microcontrolador puede también controlar el estado de la imagen enviando los comandos de Capa de Aplicación ATC correctos (ver capítulo 3).

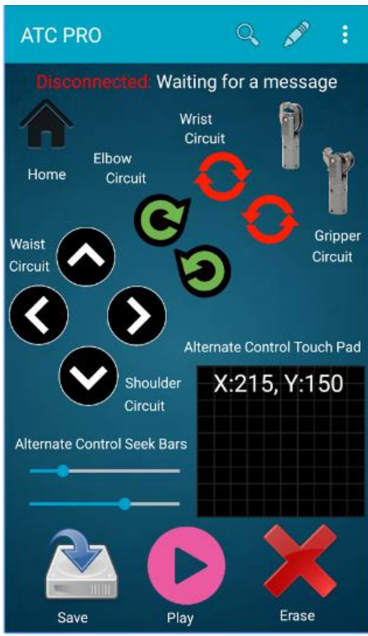


Figura 2.23. Uso de ImageView como botones y panel de toque.

Carga de imágenes usando *AsyncTask*

Debido a la demanda de procesamiento que conlleva cargar 73 imágenes (24 imágenes por Layout x 3 estados + 1 fondo de pantalla) al arranque de la aplicación, se utiliza un *AsyncTask* para preparar las imágenes en un "segundo plano". Si se intentaran cargar las imágenes en el *thread* [43] de la Actividad principal (Main.java), la aplicación dejaría de responder y se cerraría.

```
// Load pictures using async task
new loadPictures().execute();
```

Figura 2.24. AsyncTask loadPictures() ejecutada en callback "onStart( )".

Un AsyncTask permite el uso adecuado y fácil del *thread* de la Actividad principal. Esta clase permite realizar operaciones cortas (con una duración ideal de un par de segundos) en segundo plano y publicarlas en la actividad principal sin tener que manipular otros *threads*. [44]

Cuando una tarea asíncrona (*AsyncTask*) es ejecutada, ésta pasa por cuatro etapas:

1. ***onPreExcecute().*** Invocada por la actividad principal antes de empezar a ejecutar la tarea. Normalmente se usa para mostrar una barra de progreso de la tarea en cuestión. En ATC, *loadPictures()* no lo implementa.

2. ***doInBackground().*** Este paso ejecuta las tareas de procesamiento "pesadas". *loadPictures()* carga las imágenes del almacenamiento interno del dispositivo móvil a un arreglo de *bitmaps,* tanto para la imagen de fondo de pantalla, como para cada estado de cada objeto *ImageView.*

3. ***onProgressUpdate().*** Es utilizado para actualizar la barra de progreso de la tarea en cuestión. En la aplicación, loadPictures*()* no la implementa.

4. ***onPostExecute().*** En esta etapa el resultado del procesamiento en segundo plano se pasa a la actividad principal. Las imágenes que se encuentran hasta ahora en *bitmaps* se asignan al objeto de interfaz de usuario correspondiente (fondo de pantalla o *ImageViews*).

Propiedades editables

Las propiedades de una imagen se definen en la clase *EProp* (Figura 2.25) y son: contenido de la imagen, acción como botón temporal, acción como panel de toque, acción como botón con retardo, acción como botón de reconocimiento de voz, comandos "on", comandos "off", visibilidad, tamaño, y posición en el Layout (en "X" y "Y").

```java
public static final String[][] SavedPicsFiles  = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] PicActionFiles  = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] PicPadFiles     = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] PicLongActionFiles = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] PicSpeechReconFiles = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] CmdPicOnFiles   = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] CmdPicOffFiles  = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] ImgVisibilityFiles = new String[RELAY_NO][DISPLAY_NO];
public static final String[][] ImgSizeFiles  = new String[RELAY_NO][DISPLAY_NO];
public static final String FILE_IMGX     = "imageViewX";
public static final String FILE_IMGY     = "imageViewY";
```

Figura 2.25. Propiedades editables de una Imagen.

Botones Temporales

Un botón temporal en ATC es una imagen modificada con el método *onTouchListener* habilitado. Cuando un botón temporal es presionado, el estado de la imagen cambia de "0" o *Default*, a "1" o *Pressed* de forma instantánea, y se envía una cadena de texto definida en el arreglo de Strings ***relaycodesIMGON[][].*** Cuando el usuario deja de presionar el botón, es decir separa su "dedo" de la pantalla o desliza su "dedo" fuera del botón temporal, éste cambiará a estado "0" o *Default*, y enviará una cadena de texto definida en el arreglo de Strings ***relaycodesIMGOFF[][].***

Lectura de cambio de estado de un botón temporal

Los pasos para realizar la lectura de cambio de estado (*Default* o *Pressed*) de un botón temporal se citan a continuación:

1. Si la imagen fue configurada como botón temporal por el usuario, registrar el evento o *callback* en Main.

```java
// Is picture on touch action enabled?
if (Memory.ReadByte(EProp.PicActionFiles[i][CurrentDisplay]) == 1) {
    PicAction[i][CurrentDisplay] = true;
    ivImages[i].setOnTouchListener(this);
} else
    PicAction[i][CurrentDisplay] = false;
```

Figura 2.26. Registro del evento onTouch en actividad Main.

2. Implementar onTouchListener en actividad principal

```java
public class Main extends Activity implements View.OnClickListener, OnTouchListener,
        OnColorChangedListener, OnLongClickListener, SensorEventListener,
        OnSeekBarChangeListener, OnInitListener, GestureDetector.OnGestureListener {
…
@Override
public boolean onTouch(View v, MotionEvent event) {
…
}
```

Figura 2.27. Implementación del método onTouchListener en la actividad principal.

3. Filtrar el ID del objeto "tocado" y enviar el comando correspondiente a la acción de toque, como se muestra en la Figura 2.28.

```java
if (isPicture && PicAction[selectdObject.index][CurrentDisplay]) {
    mVibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);

    switch (event.getActionMasked()) {
        case MotionEvent.ACTION_UP:
            ImageSendCommand(selectdObject.index, false);
            break;

        case MotionEvent.ACTION_MOVE:
            float x, y;
            x = event.getX(0);
            y = event.getY(0);

            if (((x) > v.getWidth()) || ((y) > v.getHeight()) || (y < 0) || (x < 0)) {
                ImageSendCommand(selectdObject.index, false);
            }
            break;

        case MotionEvent.ACTION_DOWN:
            if (!PicLongAction[selectdObject.index][CurrentDisplay]) {
                ImageSendCommand(selectdObject.index, true);
                mVibrator.vibrate(50);
            }
```

Cuando el usuario levanta el dedo o deja de tocar el objeto, enviar un comando "OFF".

Cuando el usuario mueva el dedo fuera del área del objeto, enviar un comando "OFF".

Cuando el usuario toque el objeto, y éste no esté habilitado como un botón con retardo, enviar un comando "ON" y vibrar por 50ms.

…

Figura 2.28. Switch de decisión para el evento de toque para un botón temporal.

4. Cambiar el estado de la imagen de acuerdo a la acción de toque.

```java
void ImageSendCommand(int selector, boolean on) {
    if (on) {
        ivImages[selector].setBackgroundColor(ImEnphasisColor[sele
        ivImages[selector].setAlpha(210);
        ivImages[selector].setImageBitmap(Pics[selector][1]);
        MainSend(EProp.TYPE_RELAY_IMG, selector,
                relaycodesIMGON[selector][CurrentDisplay]);
    } else {
        ivImages[selector].setBackgroundColor(Color.TRANSPARENT);
        ivImages[selector].setAlpha(255);
        ivImages[selector].setImageBitmap(Pics[selector][0]);
        MainSend(EProp.TYPE_RELAY_IMG, selector,
                relaycodesIMGOFF[selector][CurrentDisplay]);
    }
}
```

Si es un comando "ON", Cambiar imagen a estado "1", sobreponer un color de énfasis de toque y enviar ***relaycodesIMGON[][]*** a *MainSend*.

Si es un comando "OFF", cambiar imagen a estado "0", eliminar color de énfasis de toque y enviar ***relaycodesIMGOFF[][]*** a *MainSend*.

Figura 2.29. Cambio de estado de la imagen de acuerdo al comando a enviar.

Paneles táctiles

Los paneles táctiles o *touchpad* en ATC son imágenes modificadas con el método *onTouchListener* habilitado, pero, a diferencia de los botones temporales, un *touchpad* extrae la coordenada del "toque" relativa al ImageView en cuestión.

Un *touchpad* reemplaza el *bitmap* del *ImageView* correspondiente por una cuadricula. Además, cuando está habilitado como touchpad, ATC genera un cursor en el *ImageView* en cuestión. Otra característica especial de un *touchpad*, es que sobrescribe el contenido del texto con el mismo índice (de 0 a 23) con los valores de "X" y "Y" relativos al *touchpad*. Los valores de "X" y "Y" van de 0 a 255 (la coordenada (0, 0) se encuentra en la esquina inferior izquierda del panel táctil).

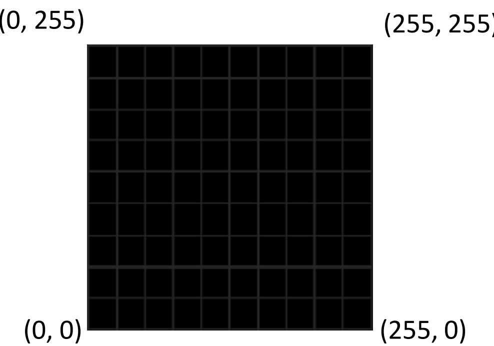


Figura 2.30. Cuadrícula de paneles táctiles.

Lectura de coordenadas de un panel de toque

Los pasos para crear un *touchpad*, y realizar la lectura de la posición del "toque" se listan a continuación:

1. Si la Imagen fue configurada como *touchpad* por el usuario, registrar el evento OnTouchListener en la actividad principal, cambiar la imagen por la cuadricula (Figura 2.31) y modificar el contenido del texto con el mismo índice.

```
// Is picture pad on touch enabled?
if (Memory.ReadByte(EProp.PicPadFiles[i][CurrentDisplay]) == 1) {
    PicPadAction[i][CurrentDisplay] = true;
    ivImages[i].setImageResource(R.drawable.planexyblack);
    ivImages[i].setOnTouchListener(this);
    tvText[i].setText("X:0,Y:0");
} else
    PicPadAction[i][CurrentDisplay] = false;
```

Figura 2.31. Inicializando un touchpad.

2. Implementar *touch listener* en la actividad principal (ver botones temporales).

3. Filtrar el ID del objeto tocado y enviar el comando correspondiente a la acción de toque. Como se muestra en la Figura 2.33, *ACTION_UP* y *ACTION_DOWN* solo proporcionan retroalimentación háptica al usuario (vibrar).

   Los métodos *getX*() y *getY*() retornan el valor de la coordenada para el puntero dado por *event*. Este valor es relativo al objeto *View* en cuestión y tiene su origen en el vértice superior izquierdo del objeto. Para trasladar estos valores a las coordenadas que se muestran en la Figura 2.30, que son más fáciles de entender o relacionar con un control inalámbrico, se aplican las operaciones mostradas en la Figura 2.32.

```
x = (event.getX(0) * 255) / v.getWidth();
y = 255 - ((event.getY(0) * 255) / v.getHeight());
```

Figura 2.32. Escala y traslación de ejes en un touchpad.

   La información de las coordenadas extraídas es enviada a la función *TouchPadSendCommand()* para procesamiento adicional.

```
if (isPicture && PicPadAction[selectdObject.index][CurrentDisplay]) {
    switch (event.getActionMasked()) {
        case MotionEvent.ACTION_UP:
            mVibrator.vibrate(30);
            break;

        case MotionEvent.ACTION_MOVE:
            //Create a new image bitmap and attach a brand new canvas to it
            Bitmap tempBitmap = Bitmap.createBitmap(ivImages[selectdObject.index].getWidth(),
                    ivImages[selectdObject.index].getHeight(), Bitmap.Config.RGB_565);
            Canvas cn = new Canvas(tempBitmap);
            Rect destRect = new Rect(0, 0, ivImages[selectdObject.index].getWidth(),
                    ivImages[selectdObject.index].getHeight());
            cn.drawBitmap(bmPlaneBlack, null, destRect, null);
            cn.drawCircle(event.getX(0), event.getY(0),
                    ivImages[selectdObject.index].getHeight() / 12, dotPaint);
            ivImages[selectdObject.index].setImageDrawable(new BitmapDrawable(getResources(),
                    tempBitmap));

            float x, y;
            // getx and gety are relative to v
            x = (event.getX(0) * 255) / v.getWidth();
            y = 255 - ((event.getY(0) * 255) / v.getHeight());
            if (!((x < 0) || (y < 0) || (x > 255) || (y > 255))) {
                tvText[selectdObject.index].setText("X:" + (int) x + ", Y:" + (int) y);
                TouchPadSendCommand(selectdObject.index, (int) x, (int) y);
            }
            break;

        case MotionEvent.ACTION_DOWN:
            mVibrator.vibrate(60);
            break;
```

Dibuja un circulo blanco (cursor) con un diámetro de 1/12 de la altura del touchpad cuyo centro coincide con el punto de toque.

Extracción de coordenadas relativas al touchpad en un rango de 0 a 255 y desplegado del valor en el texto correspondiente.

Figura 2.33. Switch de decisión para el evento de toque para un touchpad.

Reconocimiento de voz

Los botones de reconocimiento de voz en ATC son imágenes modificadas con el método *onTouchListener* habilitado. Cuando un botón de reconocimiento de voz es activado, se envía un *intent* de la clase *RecognizerIntent.ACTION_RECOGNIZE_SPEECH* para iniciar dicha actividad. El sistema Android buscará una aplicación compatible y la desplegará ante el usuario.

```
// If speech recon enabled
if (isPicture && PicSpeechRecon[selectdObject.index][CurrentDisplay]) {
    mVibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);

    switch (event.getActionMasked()) {
        case MotionEvent.ACTION_DOWN:
            mVibrator.vibrate(50);
            Intent reconIntent = new Intent(
                    RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
            SpeechReconSelector = selectdObject.index;
            if (SpeechRecognizer.isRecognitionAvailable(this))
                startActivityForResult(reconIntent, REQUEST_SR);
            else
                DisplayToast("Speech Recognizer missing");
            break;
    }
    return !PicLongAction[selectdObject.index][CurrentDisplay];
}
```

Figura 2.34. Lanzamiento de intent de reconocimiento de voz.

Cuando el usuario ha terminado de hablar y la aplicación de reconocimiento de voz ha procesado la información y generado la cadena de texto equivalente, se enviará el resultado de vuelta a la aplicación ATC. Como se muestra en la Figura 2.35, la aplicación recibe el resultado del *SpeechRecognizer* con el código de solicitud *REQUEST_SR*.

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
                                    …
  case REQUEST_SR:
      if (resultCode == RESULT_OK) {
          ArrayList<String> thingsYouSaid = data.getStringArrayListExtra(
                  RecognizerIntent.EXTRA_RESULTS);
          String withFormat = STT_TAG + ":" + thingsYouSaid.get(0) + "\n";
          MainSend(EProp.TYPE_RELAY_IMG, SpeechReconSelector, withFormat);
          if (D) Log.i(TAG, withFormat);
      }
      break;
                                    …
```

Figura 2.35. La aplicación recupera la cadena de texto del reconocimiento de voz con el código de solicitud REQUEST_SR y utilizando el método data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS).

Botones con retardo

Un botón con retardo en ATC es una imagen modificada con el método *onLongClickListener* habilitado. Un botón con retardo responde a una pulsación mantenida por un par de segundos. Cuando el tiempo de pulsación pasa ese umbral de tiempo, la aplicación enviará un comando “ON” definido por ***relaycodesIMGON[][].*** Cuanto el usuario deje de presionar la imagen, la aplicación enviará un comando “OFF” definido por el arreglo ***relaycodesIMGOFF[][].***

Displays analógicos

Un display analógico en ATC le permite al usuario observar variaciones continuas de alguna variable de forma gráfica. El indicador analógico más simple, relativo al esfuerzo implicado en la programación, es un simple rectángulo, cuyo ancho y alto varían de acuerdo a las necesidades del usuario como se muestra en la Figura 2.36.

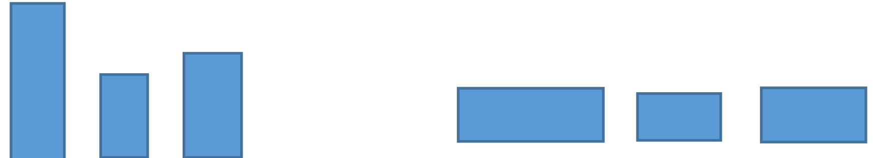


Figura 2.36. Display analógico simple con variación vertical (izquierda) y con variación horizontal (izquierda).

Programación de la clase *AnalogBarView*

Para crear los displays analógicos se realiza la programación de una clase que extienda la funcionalidad de la clase View. Los pasos a seguir para desarrollar una nueva subclase View se listan a continuación:

1. Crear una nueva clase llamada *AnalogBarView.*

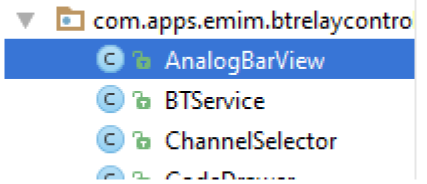


Figura 2.37. Creación de una nueva clase.

2. Extender la clase View.

```java
package com.apps.emim.btrelaycontrol;

import android.content.Context;
import android.graphics.Canvas;
import android.util.AttributeSet;
import android.view.View;

/**
 * Created by J.Luis on 5/22/2016.
 */
public class AnalogBarView extends View {
```

Figura 2.38. Clase AnalogBarView heredando los métodos de la clase View.

3. Definir e inicializar un objeto de la clase "Paint". Según la guía de desarrolladores de Android, para dibujar cualquier figura bidimensional se necesitan al menos dos objetos, uno clase Paint, el cual dicta las propiedades de lo que se quiere dibujar (color, tipo de contorno, tipo de relleno, etc.) y un objeto clase "Canvas" el cual define la forma del objeto a dibujar. El objeto canvas es pasado a la función onDraw, y el objeto Paint se inicializa en la clase AnalogBarView al llamar el método *analogBarPaintInit()*.

```java
private void analogBarPaintInit(){
    barPaint = new Paint(Paint.ANTI_ALIAS_FLAG); // Antialiasing = improves visuals
    barPaint.setColor(Color.WHITE);              // default color will be white
    barPaint.setStyle(Paint.Style.FILL);         // fill the form with color
}
```

Figura 2.39. Configuración del objeto paint en el método analogBarPaintInit().

4. Sobrecargar la función onSizeChanged. Cuando este método es llamado, es "obligación" de la clase AnalogBarView recalcular las posiciones relativas, así como los márgenes de los elementos que conforman al objeto en cuestión. En ATC, el método onSizeChanged es usado para redefinir la posición relativa del texto de nivel de llenado dependiendo si la barra se encuentra en posición vertical u horizontal.

```java
@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
    super.onSizeChanged(w, h, oldw, oldh);
    if(isVerticalBar) {
        textPaint.setTextSize(getWidth() / 5);
        textStartY = 5 + getWidth() / 5;
        textStartX = getWidth() / 10;
    }
    else {
        textPaint.setTextSize(getHeight() / 3);
        textStartX = getHeight() / 6;
        textStartY = 5 + getHeight() / 3;
    }
}
```

Figura 2.40. Sobrecarga de onSizeChanged.

5. Declarar variables para controlar el estado de la barra analógica.

```
// Bar parameters
private static final int MAX_VALUE = 255;
public static final int DEFAULT_COLOR = Color.LTGRAY;
private boolean isVerticalBar = false;
private int BarValue = 255;                        // my default value 255
private static final int myPaddingX = 8;     // my default padding in pixels
private static final int myPaddingY = 5;     // my default padding in pixels
// Text value parameters
private boolean ShowValue = true;
private String sBarValue = "255";
private int textStartX = 0;
private int textStartY = 0;
```

Figura 2.41. Variables locales para control de estado de la barra.

6. Sobrecargar la función onDraw(). En esta función se va a dibujar el rectángulo de acuerdo con las medidas especificadas por el usuario (ancho y largo del objeto) y en función del valor de la barra(BarValue) y la orientación de la misma. onDraw() también dibuja el texto con el valor de la barra analógica, si esta función está habilitada para el objeto específico.

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    if(isVerticalBar){
        // constant width
        canvas.drawRect(myPaddingX, getHeight()- myPaddingY - (((getHeight() -
                        2 * myPaddingY) * BarValue) / MAX_VALUE),
                getWidth() - myPaddingX, getHeight() - myPaddingY, barPaint);
        // Show bar value if set
        if(ShowValue)
            canvas.drawText(sBarValue, textStartX, textStartY, textPaint);
    }
    else{
        // constant height
        canvas.drawRect(myPaddingX, myPaddingY, myPaddingX + (((getWidth() -
                2 * myPaddingX) * BarValue) /
                MAX_VALUE), getHeight() - myPaddingY, barPaint);
        // Show bar value if set
        if(ShowValue)
            canvas.drawText(sBarValue, textStartX, textStartY, textPaint);
```

Figura 2.42. Sobrecarga de la función onDraw.

El método canvas.drawRect dibuja un rectángulo a partir de la esquina superior izquierda del canvas o "lienzo". Por lo que, cuando la barra es vertical, es necesario ajustar el punto de inicio del rectángulo en cuestión.

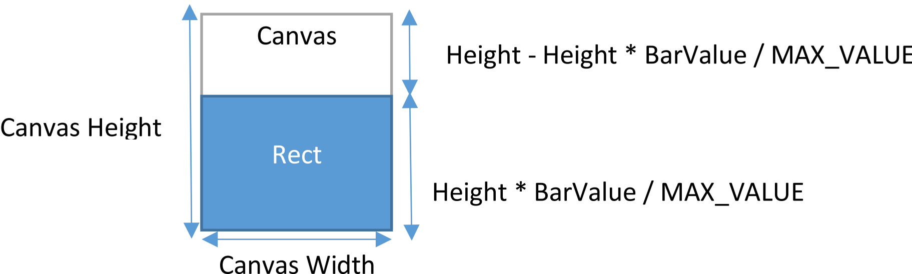


Figura 2.43. Manejo de la barra vertical.

7. Definir los métodos para controlar la orientación (vertical u horizontal), para leer, y para escribir el valor de la barra analógica.

   a. **Control del modo de la barra, setVerticalBar().**
   Como se muestra en la Figura 2.44, la función setVerticalBar recalcula la posición del texto que indica el nivel de la barra dependiendo si éste es vertical u horizontal. Ya que solo onDraw puede modificar la apariencia de la barra, la combinación de los métodos invalidate(); y requestLayout(); es usada para provocar que la barra se vuelva a dibujar y se muestre en la interfaz de usuario.

```
public void setVerticalBar (boolean isVertical){
    isVerticalBar = isVertical;
    if(isVerticalBar) {
        textPaint.setTextSize(getWidth() / 5);
        textStartY = 5 + getWidth() / 5;
        textStartX = getWidth() / 10;
    }
    else {
        textPaint.setTextSize(getHeight() / 3);
        textStartX = getHeight() / 6;
        textStartY = 5 + getHeight() / 3;
    }
    invalidate();
    requestLayout();
}
```

   Figura 2.44. Método para control de orientación de la barra analógica.

   b. **Lectura del valor de la barra analógica, getBarValue().**
   `public int getBarValue() { return BarValue; }` simplemente retorna el valor actual de la barra.

   c. **Escritura del valor de la barra analógica, setBarValue().**
   Como se muestra en la Figura 2.45, setBarValue asigna el valor de la barra (BarValue) si ésta no está fuera de los limites (entre 0 y 255); y genera la cadena de texto que representa el valor de la barra para el usuario.

```
public boolean setBarValue(int newBarValue){
    if((newBarValue > -1) && (newBarValue < 256)) {
        BarValue = newBarValue;
        if(BarValue < 10)
            sBarValue = "00" + BarValue;
        else if(BarValue < 100)
            sBarValue = "0" + BarValue;
        else
            sBarValue = "" + BarValue;
        invalidate();
        requestLayout();
        return true;
    }
    else
        return false;
}
```

   Figura 2.45. Asignación del valor de la barra.

8. Definir los métodos para cambiar la apariencia de la barra analógica (color y texto). Como se muestra en la Figura 2.46, *setBarColor* configura el color de la barra; y *setShowText*, habilita presencia del texto que muestra el valor de la barra.

```
/**
 * Set the bar color
 */
public void setBarColor(int theColor){
    barPaint.setColor(theColor);
    invalidate();
    requestLayout();
}

/**
 * Shows or hides barvalue from bar
 * @param showValue
 */
public void setShowText(boolean showValue){
    ShowValue = showValue;
    invalidate();
    requestLayout();
}
```

Figura 2.46. Métodos para cambiar la apariencia de la barra.

9. Implementar los objetos AnalogBarView en la función BuildLayout, encontrada en la actividad principal de la aplicación (Main). La versión 8.5.0 de la aplicación Arduino Total Control contará con 12 objetos clase AnalogBarView.

10. Incluir a la clase *EProp* las propiedades editables de las Barras analógicas como se muestra en la Figura 2.47.

```
// Analog Bar Files
public static final String[][] AbRotateFiles   = new String[A_BAR_NO][DISPLAY_NO];
public static final String[][][] AbSizeFiles   = new String[A_BAR_NO][DISPLAY_NO][2];
public static final String[][] AbVisibilityFiles = new String[A_BAR_NO][DISPLAY_NO];
public static final String FILE_ABX      = "abViewX_";
public static final String FILE_ABY      = "abViewY_";
public static final int[] MIN_SIZE_AB   = {40, 40};
```

Figura 2.47. Propiedades de una barra analógica.

Herramientas de edición primarias

En ATC, las herramientas de edición primarias son aquellas que modifican a los elementos constructivos (botones, imágenes, textos, barras analógicas y de acción, etc.) en cuanto a su contenido, color, tamaño, posición, función y número de elementos concierne. Estas herramientas son mostradas cuando el usuario selecciona el icono de edición (lápiz) en el menú de la actividad Main como se muestra en la Figura 2.48.

Las herramientas de edición primaria se definen en la clase ***EProp.java,*** donde también se pueden encontrar las constantes que definen las propiedades editables de cada elemento constructivo. La clase EProp utiliza estas constantes y el índice que ocupa cada elemento constructivo en su arreglo para generar las interfaces de edición específicas para cada elemento constructivo, como se muestra en la Figura 2.49.

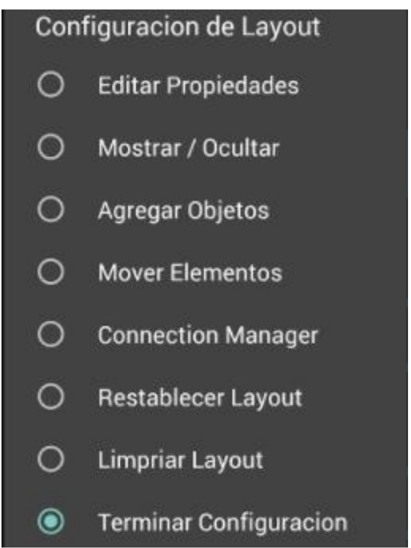


Figura 2.48. Herramientas de edición primaria.

Figura 2.49. Interfaces de edición de propiedades de los elementos constructivos.

Herramientas de edición secundarias

Las herramientas de edición secundaria no afectan directamente a los elementos constructivos. Estas se usan, por ejemplo, para cambiar la orientación del layout, editar el fondo de pantalla y habilitar el deslizamiento vertical de la pantalla o layout como se muestra en la Figura 2.50. Dentro de las herramientas de edición secundarias, destacan las herramientas para importar y exportar el contenido de un layout.

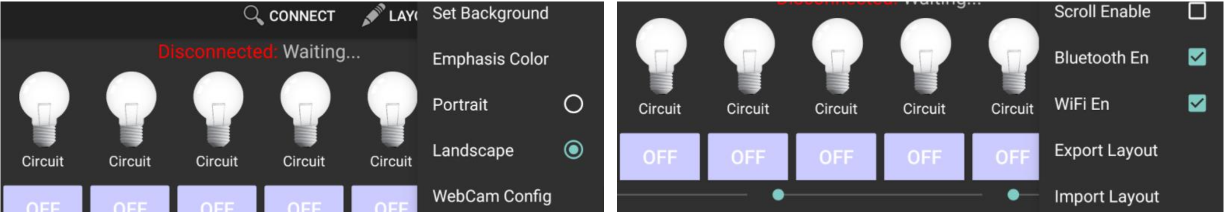


Figura 2.50. Despliegue de herramientas de edición secundarias.

Exportar un layout

En ATC, exportar un layout es crear una carpeta con el nombre del layout en cuestión dentro del directorio */ArduinoTotalControl*. Dicha carpeta contiene un archivo de texto con la información de color, tamaño, posición, modo, contenido, comandos "ON" y comandos "OFF" de cada elemento constructivo visible en el layout. Este archivo de texto también contiene información acerca del tamaño de la pantalla del dispositivo origen (aquel que exporta el layout), la cual será utilizada por el dispositivo objetivo (aquel que importa el layout) para recalcular los tamaños de los elementos constructivos.

Además del archivo de texto que define el comportamiento del layout, la carpeta exportada contiene las imágenes de fondo de pantalla y de cada elemento constructivo tipo *ImageView* como se muestra en la Figura 2.51.

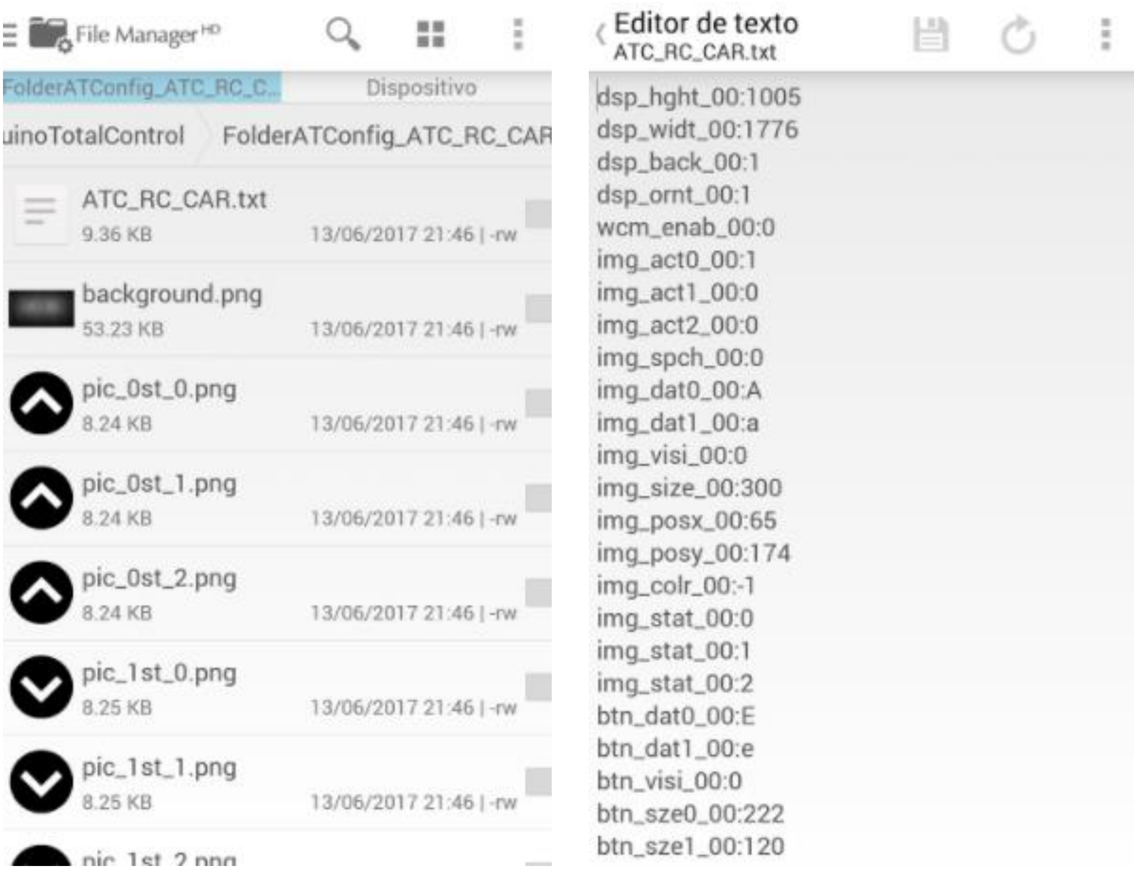


Figura 2.51. Carpeta del layout exportado (izquierda) y archivo de configuración (derecha).

Las rutinas para exportar un layout se encuentran en la clase ExProp (exportar propiedades). Estas rutinas convierten las propiedades editables de un elemento constructivo en una cadena de texto que identifica cada característica con un tag, como los mostrados en la Figura 2.52 para exportar las propiedades de una imagen.

```
public final static String BTN_DATA0 = "btn_dat0_";
public final static String BTN_DATA1 = "btn_dat1_";
public final static String BTN_SIZE0 = "btn_sze0_";
public final static String BTN_SIZE1 = "btn_sze1_";
public final static String BTN_VIS   = "btn_visi_";
public final static String BTN_X     = "btn_posx_";
public final static String BTN_Y     = "btn_posy_";
public final static String BTN_COLOR = "btn_colr_";
```

Figura 2.52. Tags para exportar propiedades de un Imagen.

Cada tipo de elemento constructivo tiene su propia función para exportar propiedades. Cuando el usuario presiona el botón de exportar layout, la actividad principal llama las funciones de ExProp para cada uno de los elementos constructivos del layout. El resultado de estos métodos es guardado en una sola cadena de texto, *configurations, la cual después es pasada como argumento a la función generateTextOnSD* para generar el archivo de configuración .txt del layout como se muestra la sección de código de la Figura 2.53. La función *ExportImageProperties* además de generar la cadena de texto con las configuraciones de cada imagen, guarda en el directorio destino las tres representaciones pictóricas de los tres estados de la imagen.

```
// Export actual settings
for (int i = 0; i < EProp.RELAY_NO; i++) {
    configurations = configurations + ExProp.ExportImageProperties(CurrentDisplay,
            getApplicationContext(), i, PicAction[i][CurrentDisplay],
            PicLongAction[i][CurrentDisplay], PicPadAction[i][CurrentDisplay],
            PicSpeechRecon[i][CurrentDisplay], relaycodesIMGON[i][CurrentDisplay],
            relaycodesIMGOFF[i][CurrentDisplay],ImEnphasisColor[i][CurrentDisplay]);
    configurations = configurations + ExProp.ExportButtonProperties(i, CurrentDisplay,
            relaycodesBTNON[i][CurrentDisplay], relaycodesBTNOFF[i][CurrentDisplay]);
    configurations = configurations + ExProp.ExportTextProperties(i, CurrentDisplay,
            TxtSensor[i][CurrentDisplay], tvText[i].getText().toString());
}
for (int i = 0; i < EProp.SEEK_BAR_NO; i++) {
    configurations = configurations + ExProp.ExportSeekBarProperties(i, CurrentDisplay);
}
for (int i = 0; i < EProp.A_BAR_NO; i++) {
    configurations = configurations + ExProp.ExportABarProperties(i, CurrentDisplay);
}

// Create the file on the card, named after the layout name
// Be careful, do not use backspaces!
File txtFile = generateTextOnSD(
        Memory.ReadString(SelectLayout.FileName + CurrentDisplay) + ".txt",
        configurations);
```

Figura 2.53. La actividad Main, llama los métodos de la clase *ExProp* para generar el contenido del archivo de configuración ".txt".

Importar un layout

El proceso de importación de un layout es a la inversa de la exportación del mismo. Este inicia cuando el usuario toca el botón de importar layout, el cual a su vez, inicia una actividad de búsqueda de archivos, donde el usuario debe seleccionar el archivo de configuración con extensión ".txt". Para que se realice una importación exitosa, es necesario

que la carpeta exportada desde el dispositivo origen se encuentre en el directorio /*ArduinoTotalControl* de la memoria externa del dispositivo objetivo. Esto es debido a que cada dispositivo Android puede retornar la dirección del directorio objetivo en diferentes formatos, lo cual podría provocar errores de lectura en la aplicación.

Una vez seleccionado el archivo ".txt", la actividad de búsqueda de archivos retornará el directorio donde se encuentra este. Este directorio es pasado como argumento a la función *setConfiguration,* como se muestra en la Figura 2.54, la cual leerá los contenidos del archivo ".txt" y las imágenes que se encuentren dentro del directorio.

```
case REQUEST_IMPORT:
    if (resultCode == RESULT_OK) {
        String directory = data.getData().getSchemeSpecificPart();
        /*
        Each device may return a different URI, the following code
        is used to find the ArduinoTotalControl in
        abcde//:xyz/ArduinoTotalControl/FolderATConfig_New_ATC_Arm/example.txt
        */
        int index = directory.indexOf("ArduinoTotalControl");
        if (index == -1) {
            DisplayToast(R.string.main_CouldNotGetDirectory);
            break;
        }
        // knowing the ArduinoTotalControl index in the URI, adding the absolute path
        // will result in the real address.
        String subdir = Environment.getExternalStorageDirectory().getAbsolutePath()
                + "/" + directory.substring(index);

        try {
            // First clear layout
            ClearLayoutSettings(false);
            // Open file input stream on sub-dir path
            InputStream in_s = new FileInputStream(subdir);
            ExProp.setConfiguration(in_s, CurrentDisplay,
                    subdir, this, Display_Height, Display_Width);
            in_s.close();
            RefreshLayout();
```

Figura 2.54. Lectura de contenido de un layout exportado.

Capítulo 3. Conectividad en ATC

La aplicación Arduino Total Control tiene la capacidad de enlazarse con cualquier dispositivo con conectividad Bluetooth, Bluetooth Low Energy o Wi-Fi como se ejemplifica en la Figura 3.1. En el caso de los dos primeros, la aplicación usa un adaptador Bluetooth en Android que le permite enviar datos al Bluetooth Host donde la información es transmitida de acuerdo al protocolo Bluetooth. Cuando la aplicación se comunica usando Wi-Fi como capa física y de enlace de datos, lo hace enviando información al protocolo TCP sobre IP.

La comunicación inalámbrica entre el dispositivo móvil inteligente y el MCU cumplen con las siguientes características:

- La comunicación se realiza en modo de conexión.
- Comunicación punto-punto (Bluetooth LE y Wi-FI) y punto-multipunto (Bluetooth).
- La aplicación "recuerda" el último dispositivo conectado. Y, si está habilitada la opción, la aplicación puede auto conectarse al mismo.
- Los módulos Bluetooth y Wi-Fi pueden convivir, es decir, la aplicación puede enviar datos a un módulo Bluetooth y a un módulo Wi-Fi simultáneamente.

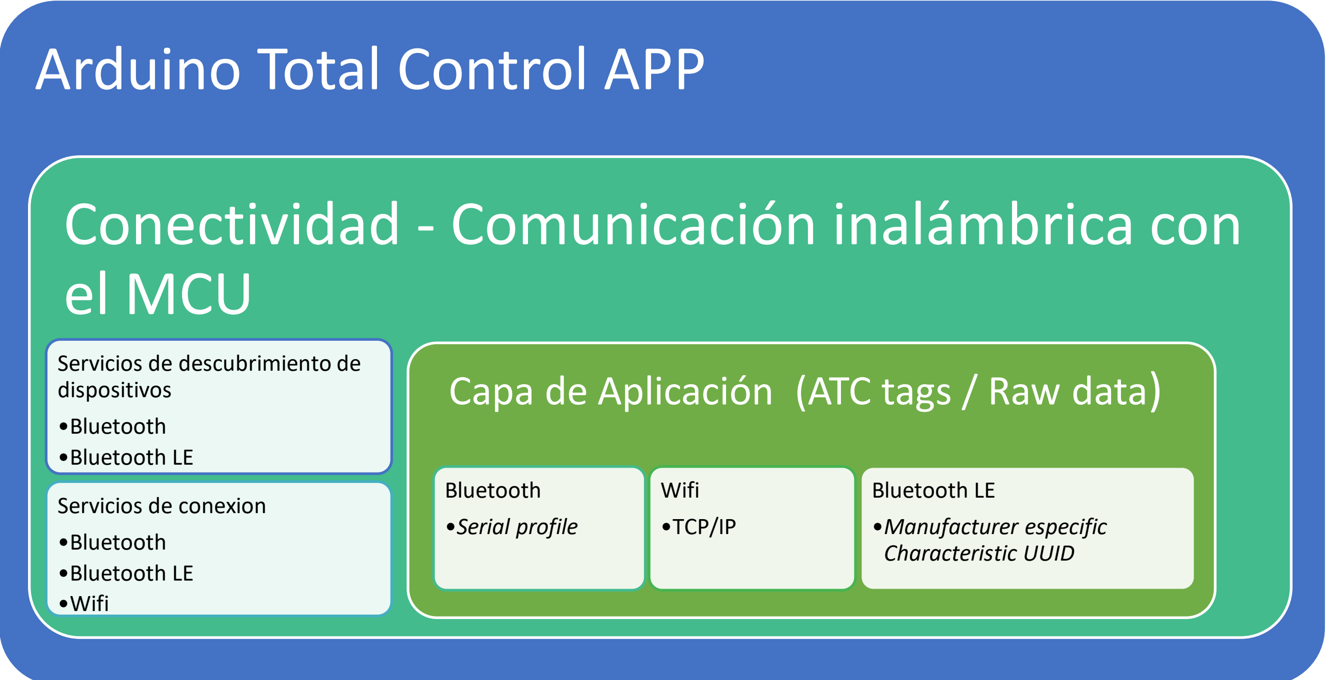


Figura 3.1. Diseño Top-Down de conectividad en ATC.

Como se muestra en la Figura 3.1, hay servicios de descubrimiento de dispositivos y servicios de conexión. En la aplicación, hay una clase en java para cada uno de estos servicios, como se puede ver en la Tabla 2.3. Cuando la aplicación se comunica por Wi-Fi, no hay un servicio de descubrimiento, pero si una actividad para configurar la dirección IP y el puerto del módulo Wi-Fi o Ethernet conectado al MCU.

Clase	Superclase	Uso (interfaz Gráfica o comunicaciones)
Main	Activity	Interfaz Gráfica / Comunicaciones
BLEScanActivity	List Activity	Comunicaciones
BLeService	Service	Comunicaciones
BTScanActivity	Activity	Comunicaciones
BTService	Thread	Comunicaciones

ChannelSelector	Activity	Comunicaciones
IPset	Activity	Comunicaciones
TCPClient	Thread	Comunicaciones

Tabla 2. Las clases de java definidas en ATC para comunicaciones.

Filosofía de control

La filosofía de control (o de operación propiamente dicho) de ATC limita a la aplicación a únicamente enviar y recibir información de y hacia el sistema a base de microcontrolador. La aplicación ATC no está diseñada para llevar a cabo procesos sensitivos en **tiempo y seguridad** que afecten al usuario o al sistema en cuestión. Esto es debido a que, si el enlace de comunicación se interrumpiera (e. g. el usuario está fuera del área de alcance de la antena del dispositivo móvil), el lazo de control se rompería, comprometiendo la seguridad o calidad del sistema embebido.

El sistema a base de microcontrolador debe mantener sus procesos sensitivos locales (temporizadores, sistemas de control de lazo cerrado, componentes del circuito de seguridad, etc.) y el usuario debe conceptualizar la aplicación como una ventana de información entre su dispositivo móvil y el MCU.

Bluetooth

La plataforma Android incluye compatibilidad con la pila de red Bluetooth, la cual permite que un dispositivo Android intercambie datos de manera inalámbrica con otros dispositivos Bluetooth. El *framework* de la aplicación proporciona acceso a la funcionalidad Bluetooth mediante las Android Bluetooth API. [45]

Con las Bluetooth API, una aplicación de Android puede realizar lo siguiente:

- Buscar otros dispositivos Bluetooth.
- Consultar el adaptador local de Bluetooth en busca de dispositivos Bluetooth sincronizados.
- Establecer canales RFCOMM.
- Conectarse con otros dispositivos mediante el descubrimiento de servicios.
- Transferir datos hacia otros dispositivos y desde éstos.
- Administrar múltiples conexiones.

El proceso para conectar o asociar un dispositivo Bluetooth con la aplicación ATC se muestra en el diagrama de flujo de la Figura 3.4. Antes de programar una aplicación con conectividad Bluetooth, el manifiesto debe contar con los permisos *android.permisision.BLUETOOTH_ADMIN* y *android.permisision.BLUETOOTH* como se muestra en la Figura 3.2.

```
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH" />
```

Figura 3.2.Permisos necesarios para implementar Bluetooth en ATC.

Emparejar un dispositivo

Para iniciar una conexión Bluetooth es necesario conocer la dirección física del dispositivo a conectar. Cada dispositivo Bluetooth tiene una dirección única de 48-bits. Sin embargo, generalmente estas direcciones no se muestran por sí solas en los “escaneos”, en cambio, se muestran junto con nombres más amigables para el usuario como se muestra en la Figura 3.3. [28]

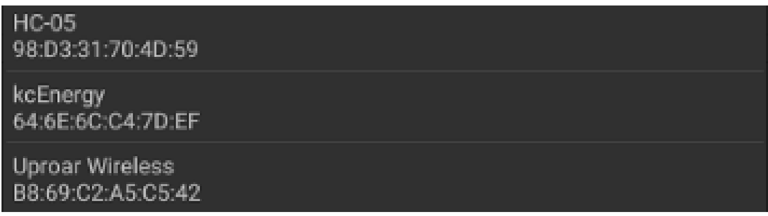


Figura 3.3. Ejemplo de nombres de dispositivos Bluetooth con su dirección física.

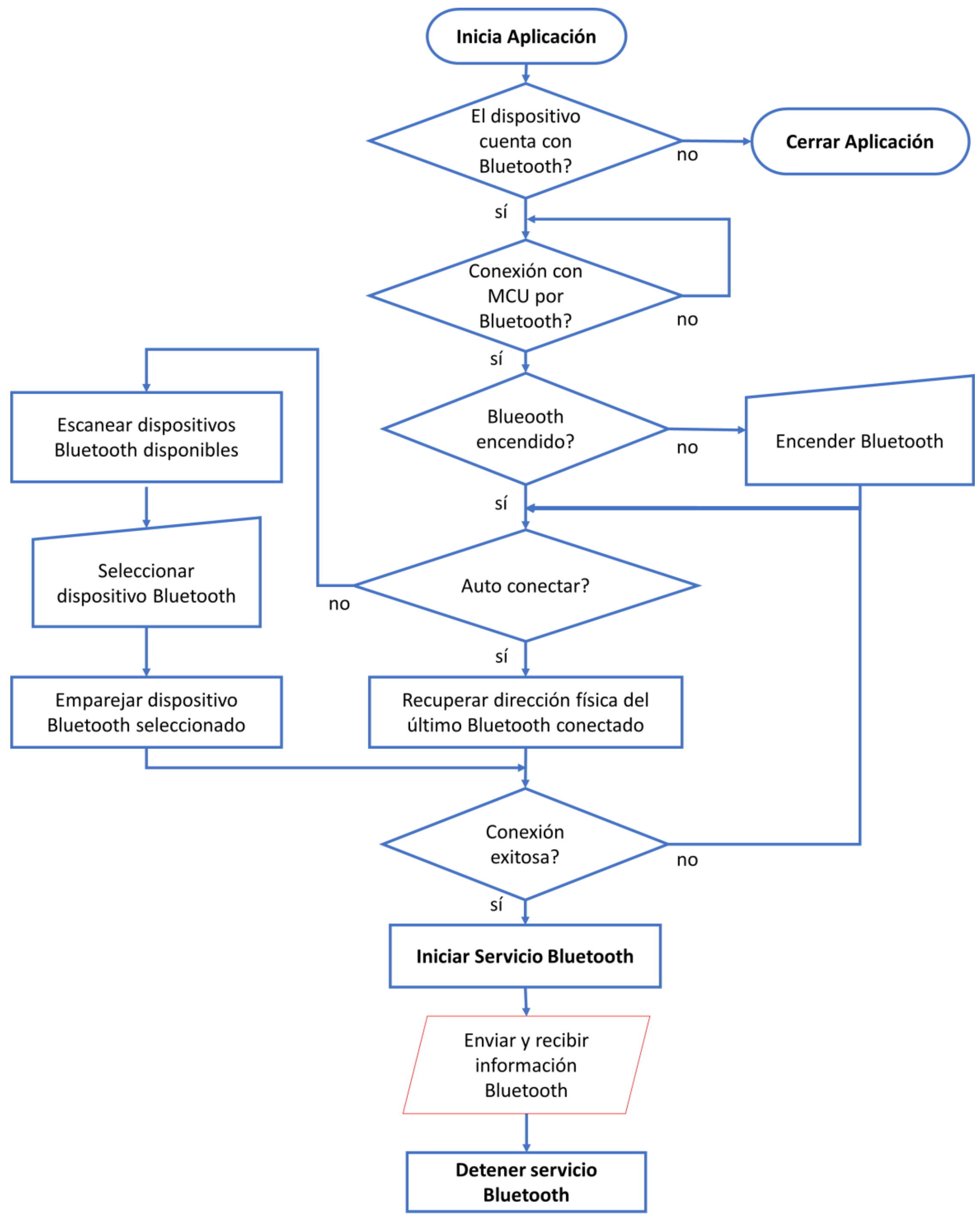


Figura 3.4. Diagrama de flujo de una conexión Bluetooth en ATC.

Para emparejarse con un dispositivo Bluetooth, como se muestra en la Figura 3.4, la aplicación tiene dos opciones: recordar la dirección física del último dispositivo Bluetooth conectado (si éste existe, y si está habilitada la función de auto-conectar); o bien, escanear en busca de un nuevo dispositivo. El último involucra la interacción con el usuario para seleccionar el dispositivo a conectar e introducir la contraseña de emparejamiento (si la hay).

BTScanActivity

La clase que se encarga de escanear en busca de nuevos dispositivos Bluetooth y listar los dispositivos Bluetooth previamente emparejados es *BTScanActivity.java. BTScanActivity* es iniciada por *Main* cada vez que el usuario requiere conectar un dispositivo Bluetooth nuevo, y retorna la dirección física del dispositivo seleccionado como una cadena de texto de 17 caracteres (e. g. "00:11:22:AA:BB:CC").

BTScanActivity también es la encargada de configurar la opción de "auto conectar" y "dejar Bluetooth encendido" como se muestra en la parte superior de la Figura 3.5. Las cuales permiten conectar de forma automática con el último dispositivo Bluetooth emparejado y evitar que la aplicación apague el adaptador Bluetooth al finalizar su operación respectivamente.

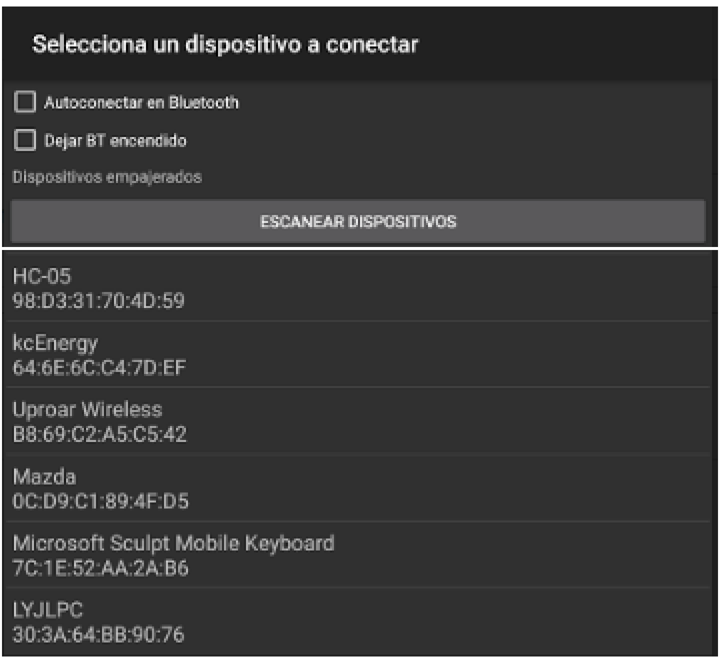


Figura 3.5. Captura de pantalla de actividad BTScanActivity.

Listado de dispositivos Bluetooth previamente emparejados

Para obtener el listado de los dispositivos Bluetooth previamente emparejados, se hace uso de la clase *BluetoothAdapter*, la cual representa el adaptador local de radio Bluetooth [45], y la función *getBondedDevices* para obtener el conjunto de *BluetoothDevice* registrados en la memoria del adaptador Bluetooth. Un objeto *BluetoothDevice* contiene el nombre, dirección, clase y estado de conexión del dispositivo remoto Bluetooth.

```
// Get the local BT adapter
mBtAdapter = BluetoothAdapter.getDefaultAdapter();

// Get a set of currently paired devices
Set<BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();
```

Figura 3.6. Obtención de dispositivos Bluetooth previamente conectados.

Escaneo de nuevos dispositivos

Para escanear por nuevos dispositivos se utiliza un *receptor de mensajes* o *BroadcastReceiver* que "escucha" cuando un nuevo dispositivo es encontrado. Como se muestra en la Figura 3.4, el adaptador de Bluetooth, previamente definido, inicia el proceso de descubrimiento. Como se muestra en la Figura 3.5, cuando el adaptador descubre un nuevo dispositivo, el receptor de mensajes agrega el nombre y la dirección del dispositivo encontrado a la lista.

```
// Register for broadcasts when a device is discovered
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
this.registerReceiver(mReceiver, filter);

// Request discover from BluetoothAdapter
mBtAdapter.startDiscovery();
```

Figura 3.7. Registro e inicialización del descubrimiento de nuevos dispositivos Bluetooth.

```java
// The BroadcastReceiver that listens for discovered devices and
// changes the title when discovery is finished
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        // When discovery finds a device
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            // Get the BluetoothDevice object from the Intent
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            // If it's already paired, skip it, because it's been listed already
            if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
                mNewDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
            }
```

Figura 3.5. Receptor de mensajes para escaneo de dispositivos nuevos.

Una vez listados todos los dispositivos disponibles en la red, el usuario puede seleccionar uno de la lista, y, si éste está protegido por contraseña, el sistema Android solicitará la introducción de la contraseña de emparejamiento (esto lo realiza de forma independiente a la aplicación).

BTScanActivity termina su operación cuando el usuario ha seleccionado un elemento de la lista de dispositivos Bluetooth disponibles, retornando la dirección, la configuración de auto-conectar y la configuración de dejar Bluetooth encendido bajo los identificadores *EXTRA_DEVICE_ADDRESS, EXTRA_AUTOCONNECT* y *EXTRA_LEAVE_BT_ON* respectivamente.

```java
// Create the result Intent and include the MAC address
Intent intent = new Intent();
intent.putExtra(EXTRA_DEVICE_ADDRESS, address);
intent.putExtra(EXTRA_AUTOCONNECT, cbAutoconn.isChecked());
intent.putExtra(EXTRA_LEAVE_BT_ON, cbLeaveBtOn.isChecked());

// Set result and finish this Activity
setResult(Activity.RESULT_OK, intent);
finish();
```

Figura 3.6. Fin de BTScanActivity.

Conexión de un dispositivo Bluetooth

ATC implementa la clase *BTService* para manejar las conexiones por Bluetooth (clásico). La clase *BTService* realiza la conexión, desconexión, envío y recepción de datos con el *Bluetooth Host*. Además, esta clase administra los siete canales disponibles para comunicación con dispositivos Bluetooth remotos.

El estándar Bluetooth permite conectar un maestro con siete esclavos, o "canales" para lograr una comunicación punto-multipunto. Sin embargo, la administración de estos canales debe realizarse por software.

Cuando la función Main recibe la dirección física del dispositivo a conectar por medio de la actividad BTScanActivity o por medio de una auto-conexión, la función *ChannelInit* de la clase *BTService* es llamada para iniciar una conexión Bluetooth. Como se muestra en la Figura 3.7, *ChannelInit* toma como argumentos; un objeto tipo *Handler*, el cual es un método que permite enviar y procesar mensajes entre dos *threads* diferentes (en este caso el *thread* del servicio Bluetooth y el *thread* de la actividad principal); y una cadena de texto que representa la dirección física del dispositivo Bluetooth.

```java
public boolean ChannelInit(Handler handler, String address){
    // Get the BluetoothDevice object
    BluetoothDevice device = mAdapter.getRemoteDevice(address);

    // Check for an available channel(check for null or none status)
    for(int id = 0; id < MAX_CHANNELS; id++){
        if((mChannels[id]== null)
                ||(mChannels[id].getState() == STATE_NONE)){
          mChannels[id] = new Channel(handler);
          mChannels[id].start();
          // Attempt to connect to the device
          mChannels[id].connect(device);
          return true;
        }
```


Figura 3.7. Inicio de una conexión Bluetooth.

Es esta misma función, ChannelInit, la que administra los canales disponibles (MAX_CHANNELS = 7) para conexión con diferentes dispositivos Bluetooth remotos, como se explica en la Figura 3.7. El objeto *mChannels[]* es un arreglo de siete objetos tipo *BTService.Channel*. Cada objeto BTService.Channel contiene dos *thread* o subprocesos, uno para iniciar conexión (*ConnectThread*) y uno para enviar y recibir información (*ConnectedThread*).

ConnectThread

El miembro ConnectThread de cada canal consiste en dos métodos, ConnectThread (el inicializador) y run (el proceso de conexión). El primero crea un BluetoothSocket RFCOMM o SPP (“Serial Port Profile" o perfil de puerto serial por sus siglas en inglés) listo para iniciar una conexión con el dispositivo Bluetooth remoto. Un BluetoothSocket es el objeto que maneja la conexión de lado del cliente [45] (el dispositivo móvil inteligente), mientras un BluetoothServerSocket maneja la conexión del lado del servidor (módulo Bluetooth conectado al MCU).

```java
device.createRfcommSocketToServiceRecord(UUID_SPP)
```

La especificación del UUID (Universal Unique Identifier) del Serial Port Profile (UUID: "00001101-0000-1000-8000-00805F9B34FB") en la función *createRfcommSocketToServiceRecord* le permite a la aplicación conectarse con un módulo Bluetooth-Serial como lo es el HC-05 y el HC-06 mencionados en el capítulo 1.

Para conectarse a un dispositivo Bluetooth determinado, la aplicación debe correr un nuevo thread (función “run”) para iniciar la conexión con el método *Socket.connect()*. Este método bloquea su thread hasta que la conexión se completa o falla. Si ATC no implementara ConnectThread, la aplicación se congelaría y sería finalizada por el sistema operativo Android. Como se muestra en la Figura 3.8, antes de iniciar una conexión es necesario cancelar cualquier tarea de descubrimiento de dispositivos activa.

```java
public void run() {
    if (D) Log.i(TAG, "BEGIN mConnectThread SocketType:" + mSocketType);
    setName("ConnectThread" + mSocketType);

    // Always cancel discovery because it will slow down a connection
    mAdapter.cancelDiscovery();

    // Make a connection to the BluetoothSocket
    try {
        // This is a blocking call and will only return on a
        // successful connection or an exception
        mmSocket.connect();
```

Figura 3.8. Conexión del BluetoothSocket.

ConnectedThread

Una vez conectado el dispositivo Bluetooth remoto, el subproceso *BTService.Channel.ConnectedThread* se encarga de enviar y recibir información entre el dispositivo Bluetooth remoto y la actividad principal, *Main*. Antes de iniciar el proceso de envío y recepción de información, *ConnectedThread* asigna los buffers de flujo de entrada y salida a *mmInStream* y *mmOutStream* respectivamente como se muestra en la Figura 3.9.

```
// Get the BluetoothSocket input and output streams
try {
    tmpIn = socket.getInputStream();
    tmpOut = socket.getOutputStream();
} catch (IOException e) {
    if (D) Log.e(TAG, "temp sockets not created", e);
}

mmInStream  = tmpIn;
mmOutStream = tmpOut;
```

Figura 3.9. Asignación de los flujos de entrada y salida de un BlkuetoothSocket.

Enviar información

El envío de información se realiza mediante la escritura del *OutputStream mmOutStream* como se muestra en la Figura 3.10. Si el envío de la información es exitoso (no se dispara ninguna excepción o falla), BTService retorna los datos enviados a Main como confirmación, a través de un mensaje MESSAGE_WRITE utilizando el Handler previamente asignado por Main.

```
public void write(byte[] buffer) {
    try {
        mmOutStream.write(buffer);

        // Share the sent message back to the UI Activity
        mHandler.obtainMessage(Main.MESSAGE_WRITE, -1, -1, buffer)
                .sendToTarget();
    } catch (IOException e) {
        if (D) Log.e(TAG, "Exception during write", e);
    }
}
```

Figura 3.10. Envío de un buffer de datos a un dispositivo Bluetooth[1].

Recibir información

La recepción de información Bluetooth se realiza en el método "run" del *mConnectedThread* del canal en cuestión (recordar que puede haber hasta siete canales activos, cada uno con su propio subproceso mConnectedThread). En ATC, el objetivo del subproceso mConnectedThread es leer una línea completa de caracteres antes de enviar la cadena a Main como se explica en la Figura 3.11.

[1] Buffer es cualquier arreglo de bytes, e. g. una cadena de texto.

```java
// Read from the InputStream..
singleByte = mmInStream.read();
if (singleByte == -1)
    continue;

// ... and make a text line
if(singleByte != 0x0A){
    buffer[index++] = (byte) singleByte;

    // If no valid string received!!!
    if (index >= 1024){
        index = 0;
        String st = "String too large!";
        if (D) Log.i(TAG, "Out of bounds!");
        mHandler.obtainMessage(Main.MESSAGE_READ, 5,
          -1, st).sendToTarget();
    }
}
else{
  String st = new String(buffer, 0, index);

  // Debug data
  if (D) Log.i(TAG, "Received data: " + st);

  // Send the obtained string to the UI Activity
  mHandler.obtainMessage(Main.MESSAGE_READ, 5, -1, st)
                .sendToTarget();
```


Figura 3.11. Recepción de información Bluetooth.

Bluetooth Low Energy (BLE)

Desde su versión 4.3, Android es compatible con Bluetooth Low Energy (Bluetooth de Baja Energía). Esta tecnología de Bluetooth tiene la ventaja de consumir considerablemente menos energía que el Bluetooth convencional según el Bluetooth SIG [33]. BLE en Android es soportado en rol central (modo maestro), y proporciona las APIs que les permiten a las aplicaciones descubrir dispositivos, servicios, leer y escribir características. Esto les permite a las aplicaciones Android comunicarse con dispositivos BLE de como sensores de proximidad, marcapasos y bandas deportivas, entre otros [46].

Términos principales	
Generic Attribute Profile (GATT)	Todos los dispositivos BLE están basados en GATT. El perfil GATT es una especificación general para enviar y recibir paquetes pequeños de información conocidos como "atributos" sobre un enlace BLE, estos están identificados por un UUID y pueden ser "características" o "servicios". Bluetooth SIG define distintos perfiles para dispositivos BLE. Un perfil es una especificación de cómo debe funcionar un dispositivo en una aplicación particular. Un dispositivo puede implementar más de un perfil (e. g. marcapasos y detección de nivel de batería).
Characteristic (Characteristic en inglés)	Una característica contiene un solo valor y un número determinado de "descriptores" que describen la característica (como una descripción textual, unidades de medida, rango aceptable del valor de la característica, etc.).
Servicio (Service en inglés)	Un servicio es una colección de características.

Tabla 3.1. GATT, *characteristic* y *service*.

El demo Bluetooth LEGATT

Una herramienta esencial para la implementación de BLE en ATC fueron los ejemplos de uso de APIs que provee Google a los desarrolladores. El demo Bluetooth LEGATT se utilizó extensamente para soportar este desarrollo para realizar pruebas de conectividad y funcionamiento. Las clases de la Aplicación Bluetooth LEGATT contienen el software mínimo para conectar un dispositivo Android a un módulo BLE y se muestran en la Tabla 3.2.

Clase	Funciones
BluetoothLeService.java	a) Conecta el dispositivo Bluetooth BLE. b) Envía y recibe información del dispositivo Bluetooth BLE. c) Notifica a la actividad principal el estado de la conexión.
DeviceControlActivity.java	a) Ejecuta el servicio *BluetoothLeService.* b) Contiene un Broadcast receiver que maneja diferentes eventos disparados por el BluetoothLeService.
DeviceScanActivity.java.	a) Es la actividad principal, es decir, es el programa que corre cuando la aplicación es iniciada. b) Verifica que el dispositivo sea BLE *capable.* c) Enciende el Bluetooth si es que está apagado. d) Busca (Escanea dispositivos BLE). e) Enlista los dispositivos BLE encontrados. f) Cuando un dispositivo BLE es seleccionado, crea un intento a *DeviceControlActivity* con el nombre y la dirección del dispositivo seleccionado.
SampleGattAttributes.java	a) Contiene un *hashmap* con los Identificadores Universales (UUID) de los servicios y características soportadas.

Tabla 3.2. Clases de la aplicación demo Bluetooth LEGATT.

Descubrimiento de dispositivos

Como se menciona al principio de este capítulo, para manejar la conexión de un dispositivo Bluetooth LE en la aplicación, se implementó la clase BLEScanActivity, la cual busca y enlista los dispositivos Bluetooth LE disponibles. Esta actividad le proporciona al usuario una lista con los nombres y direcciones de los dispositivos BLE dentro del rango.

Conexión

En ATC, la clase BLeService.class es la encargada de la conexión, envió y recepción de información de un dispositivo remoto Bluetooth Low Energy. Una vez que el usuario ha seleccionado el dispositivo a conectar (clase BLEScan), se iniciara el servicio BLeService. La función bindService inicia o "conecta "el servicio interno de aplicación BLeService como se muestra en la Figura 3.12a.

El argumento mServiceConnection contiene los callbacks que serán llamados cuando la conexión al servicio se finalice. Cuando el servicio de aplicación esté activo, éste conectará el adaptador bluetooth a la dirección del dispositivo seleccionado antes por el usuario como se muestra en la Figura 3.12b.

```
Intent gattServiceIntent = new Intent(getApplicationContext(),
        BLeService.class);
getApplicationContext().bindService(gattServiceIntent, mServiceConnection,
        BIND_AUTO_CREATE);
```

a)

```java
// Code to manage Service lifecycle. After init, device is connected
private final ServiceConnection mServiceConnection = new ServiceConnection() {

    @Override
    public void onServiceConnected(ComponentName componentName, IBinder service) {
        mBLeService = ((BLeService.LocalBinder) service).getService();
        if (!mBLeService.initialize()) {
            if (D) Log.e(TAG, "Unable to initialize Bluetooth");
        } else {
            // Automatically connects to the device upon successful start-up init.
            mBLeService.connect(BLEAddress);
        }
    }
```

b)

Figura 3.12. Conexión de un dispositivo Bluetooth LE.

Finalmente, cuando el dispositivo remoto esté conectado, la aplicación solicitará entrada del usuario para seleccionar la característica de la cual se recibirán y enviaran los comandos de la aplicación. La Tabla 3.3 muestra el servicio y la característica para enviar y recibir información para un módulo HM10.

Service	UUID
COMS	0000ffe0-0000-1000-8000-00805f9b34fb
Characteristic	**UUID**
Data Transfer	0000ffe1-0000-1000-8000-00805f9b34fb

Tabla 3.3. Característica objetivo de un módulo HM10.

Recepción de información

Una vez conectada al dispositivo remoto BLE, la aplicación recibirá los datos de entrada, así como el estado de la conexión por medio del BroadcastReceiver mGattUpdateReceiver. Como se muestra en la Figura 3.13, mGattUpdateReceiver llama la función decodeMessage cuando recibe información del dispositivo remoto para el procesamiento adicional de los datos recibidos en la actividad Main.

```java
// ACTION_GATT_CONNECTED: connected to a GATT server.
// ACTION_GATT_DISCONNECTED: disconnected from a GATT server.
// ACTION_GATT_SERVICES_DISCOVERED: discovered GATT services.
// ACTION_DATA_AVAILABLE: received data from the device.  This can be a result of read
//                        or notification operations.
private final BroadcastReceiver mGattUpdateReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        if (BLeService.ACTION_GATT_CONNECTED.equals(action)) {
            mBLEConnected = true;
            SetConnectedState();
        } else if (BLeService.ACTION_GATT_DISCONNECTED.equals(action)) {
            mBLEConnected = false;
            SetConnectedState();
        } else if (BLeService.ACTION_GATT_SERVICES_DISCOVERED.equals(action)) {
            // Display services discovered
            displayGattServices(mBLeService.getSupportedGattServices());
        } else if (BLeService.ACTION_DATA_AVAILABLE.equals(action)) {
            // Process information according aplication data layer
            ConnectionCounter++;
            decodeMessage(intent.getStringExtra(BLeService.EXTRA_DATA));
        }
```

Figura 3.13. BroadcastReceiver mGattUpdateReceiver.

Envío de información

El envío de información se rea liza en la clase BLeService, mediante la función writeCharacteristic. Esta función envía una cadena de texto (data) a la característica previamente seleccionada por el usuario (mTargetCharacteristic) como se muestra en la Figura 3.14.

```
/**
 * Writes data into the target characteristic
 * @param data
 */
public void writeCharacteristic(String data){
    if (mBluetoothAdapter == null || mBluetoothGatt == null
            || (mTargetCharacteristic == null)) {
        Log.v(TAG, "BLE Cannot send data");
        return;
    }
    mTargetCharacteristic.setValue(data.getBytes());
    mBluetoothGatt.writeCharacteristic(mTargetCharacteristic);
}
```

Figura 3.14. Envío de información desde ATC al dispositivo BLE.

Wi-Fi (TCP/IP)

Android provee soporte nativo con la suite de protocolos de Internet. ATC es capaz de conectarse inalámbricamente utilizando el protocolo TCP/IP, de forma que el protocolo Wi-Fi (estándar 802.11) es transparente para la aplicación.

Protocolo TCP/IP

El Protocolo de Control de Transporte (TCP) provee transmisión de datos confiable y secuenciada en un sistema full-dúplex. De acuerdo al *Intenet Engineering Task Force*, TCP es usada por aquellas aplicaciones que necesitan un servicio de transporte confiable y en modo de conexión (e. g. email (SMTP), file transfer (FTP), y terminal virtual (Telnet)) [46].

El conjunto de protocolos TCP/IP puede interpretarse en términos de capas (o niveles). La Figura 3.15 muestra las capas del protocolo TCP/IP en la aplicación Arduino Total Control. Empezando por la parte superior son: Capa de aplicación, Capa de Transporte, Capa de Red, Capa de Interfaz/Enlace de red y hardware.

Figura 3.15. Pila de protocolos TCP/IP [47] (izquierda) y su relación con ATC (derecha).

TCP/IP define cómo se mueve la información desde el remitente hasta el destinatario. En primer lugar, los programas de aplicación envían mensajes o corrientes de datos al protocolo de Capa de Transporte. Este protocolo recibe los datos de la aplicación, los divide en partes más pequeñas llamadas *paquetes*, añade una dirección de destino y, a continuación, pasa los paquetes a la siguiente capa de protocolo, la Capa de Red de Internet.

La Capa de Red de Internet pone el paquete en un datagrama de IP (Internet Protocol), coloca la cabecera y la cola de datagrama, decide dónde enviar el datagrama (directamente a un destino o a una pasarela) y pasa el datagrama a la Capa de Enlace de red.

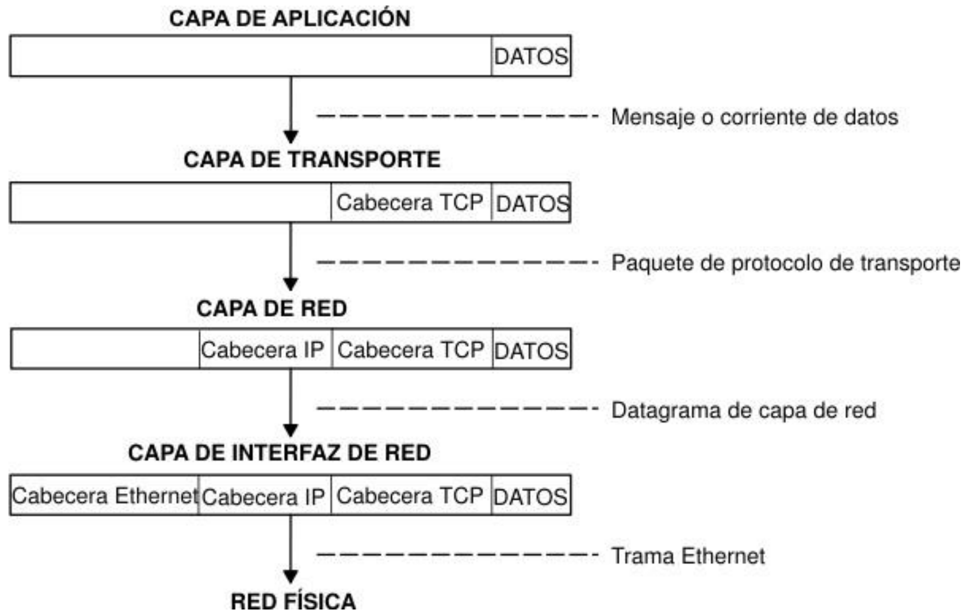


Figura 3.16. Movimiento de información desde la aplicación remitente hasta el sistema destinatario. [47]

Las tramas recibidas por un sistema principal pasan a través de las capas de protocolo en sentido inverso. Cada capa quita la información de cabecera correspondiente, hasta que los datos regresan a la Capa de Aplicación.

Permisos

Para usar la suite de protocolos de internet en Android, es necesario declarar en el manifiesto los permisos de conexión a internet y acceso a estado de red, definidos por INTERNET y ACCESS_NETWORK_STATE respectivamente como se muestra en la Figura 3.17.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

Figura 3.17. Permisos en el manifiesto para uso de Internet.

Configuración de dirección y puerto

Antes de iniciar una conexión TCP/IP es necesario conocer el puerto y la dirección IP del dispositivo a conectar. Además, este dispositivo debe encontrarse en la misma subred que el dispositivo móvil inteligente[2]. La aplicación actúa como un Cliente TCP/IP, mientras que el sistema a base de microcontrolador actúa como un servidor TCP/IP.

IPSet

La aplicación implementa la actividad *IPSet* para configurar el puerto y la dirección IP de destino como se muestra en la Figura 3.18. Además, esta clase configura la función opcional “Auto conectar en Wi-Fi”, que le permite a la aplicación conectarse automáticamente con el dispositivo remoto una vez que se encuentra en la misma red.

[2] Si el MCU con conectividad a internet se encuentra fuera de la red local a la que se encuentra conectado el dispositivo Android, el usuario puede usar un servicio DYN DNS para conectarse al MCU a través de internet. Sin embargo, esto no forma parte del alcance de este trabajo.

Figura 3.18. Actividad IPSet con valores por defecto para dirección IP y puerto.

Cuando el usuario presiona el botón “OK!” en la actividad IPSet, ésta retorna a *Main* una cadena de texto, con la dirección IP; un número entero, con el puerto; y un byte, cuyo valor es “1” si la casilla de “Auto conectar en Wi-FI” está seleccionada, o un “0” si está vacía. Con esta información, *Main* puede empezar la conexión con el servidor embebido.

Conexión

La clase que se encarga de administrar la conexión con un servidor TC/IP es *TCPClient*. *TCPClient* es iniciada por Main cuando el usuario ha configurado el puerto y la dirección IP destino, o cuando la función auto conectar ha sido habilitada previamente.

Como se puede observar en la Figura 3.19, un cliente TCP toma como argumentos un Handler, el cual le permitirá al subproceso TCPClient comunicarse con el proceso de la actividad principal, una cadena de texto que representa la dirección IP, y el puerto del dispositivo a conectar.

```
public TCPClient(Handler handler, String serverIp, int port) {
    mHandler = handler;
    SERVERIP = serverIp;
    SERVERPORT = port;
}
```

Figura 3.19. Inicializador de TCPClient.

El método “run” de un objeto TCPClient en ATC se divide en dos partes, el establecimiento de la conexión y la recepción de datos. El primero configura el “socket” (un socket es un punto terminal para la comunicación entre dos máquinas), notifica a Main del estado de la conexión y obtiene los *InputStream* y OutputStream para recibir y enviar datos respectivamente. El proceso de conexión se detalla en la Figura 3.20.

```java
// Connect
try {
    // Convert from string to IP
    InetAddress serverAddr = InetAddress.getByName(SERVERIP);
    if (D) Log.i("TCP Client", "C: Connecting...");
    // Create a socket to make the connection with the server
    socket = new Socket(serverAddr, SERVERPORT);
} catch (Exception e) {
    if (D) Log.e("TCP", "C: Cannot Connect");
    mHandler.obtainMessage(Main.MESSAGE_TCP, DISCONNECTED, -1).sendToTarge
    return;
}

try {
    // send the message to the server
    out = new PrintWriter(new BufferedWriter(
            new OutputStreamWriter(socket.getOutputStream())), true);

    // receive the message which the server sends back
    in = new BufferedReader(new InputStreamReader(socket.getInputStream()

    // Tell the main device is connected over TCP
    mHandler.obtainMessage(Main.MESSAGE_TCP, CONNECTED, -1).sendToTarget(
    if (D) Log.i("TCP Client", "Connection success");
```

Inicializa el socket.

Notifica a Main si la conexión falló.

Asigna el buffer de salida de datos.

Asigna el buffer de entrada de datos.

Notifica conexión exitosa a Main.

Figura 3.20. Proceso de conexión a un servidor TCP.

Recibir información

Dentro de la función "run" de TCPClient, ATC "escucha" por bytes entrantes provenientes de la conexión TCP/IP con el dispositivo remoto. La función *in.ready()* es llamada constantemente para verificar si hay información disponible para leer del buffer de entrada. De la misma forma que ConnectedThread lee una línea de información proveniente del dispositivo Bluetooth remoto, TCPClient lee una línea de texto, terminada por el carácter '/n,' proveniente del MCU.

Si la línea de texto contiene más de 1024 caracteres, TCPClient enviará el mensaje de error "String too large" a *Main*, de lo contrario la información será pasada "tal cual" a Main por medio de un mensaje identificado por MESSAGE_READ utilizando el Handler previamente definido.

```java
if (in.ready())
    serverMessage = in.readLine();
if ((serverMessage != null) && (mHandler != null)) {
    if (serverMessage.getBytes().length > 1024)
        serverMessage = "String too large!";

    // Send the obtained characters to main
    mHandler.obtainMessage(Main.MESSAGE_READ, -1,
            8, serverMessage).sendToTarget();

    if (D) Log.i("RESPONSE FROM SERVER", "S: Received Message: '"
            + serverMessage + "'");
}
serverMessage = null;
```

Figura 3.21. Recepción de información del servidor TCP/IP.

Enviar información

Para enviar información al dispositivo conectado por TCP/IP, el objeto TCPClient utiliza un *AsyncTast* para correr en un segundo plano el envío de información como se muestra en la Figura 3.22.

```
// Used to send message in another thread
private class sendMessageAsync extends AsyncTask<String, Integer, String> {
    @Override
    protected String doInBackground(String... arg0) {
        if (out != null && !out.checkError()) {
            out.println(sDataToPrint);
            out.flush();
        }
        return null;
    }

    protected void onPostExecute(String result) {

    }
}
```

Figura 3.22. Envió de datos por TCP/IP.

Protocolo de Capa de Aplicación ATC

Sin importar si la comunicación se realiza por Bluetooth, Bluetooth LE o Wi-Fi, la aplicación y el dispositivo remoto se comunican utilizando el protocolo de Capa de Aplicación ATC. Este protocolo tiene las siguientes características:

- Las unidades de datos son delimitadas por el carácter de fin de línea '\n'.
- La información enviada puede ser "raw data" (datos sin procesar) o "tags" (identificadores).
- Un comando estilo "hearthbeat" (latido, *CMD_ALIVE*) es enviado cada 2.5s para confirmar que el MCU sigue conectado y activo.
- Es un protocolo fácil de usar, pero lo suficientemente potente para permitir el intercambio de información entre el dispositivo móvil y el MCU.

Envío de información a nivel de Capa de Aplicación

El envío de datos a nivel de la Capa de Aplicación en ATC se realiza con las funciones *MainSend* o *MainbSend* de Main. Como se muestra en la Figura 3.23, el método *MainBSend* llama las funciones de envió de datos de cada tecnología de comunicación inalámbrica que esté habilitada.

La diferencia entre MainSend y MainBSend, radica en que MainSend no utiliza el método *mBTService.bwrite(),* el cual es usado para enviar un comando a todos los canales activos del adaptador Bluetooth clásico (*broadcast write*), sino que enruta la información al canal correspondiente dependiendo del elemento constructivo que inicia el envío de información. Para esto, MainSend, además de la información a enviar, toma como argumentos el tipo de objeto constructivo (botón, imagen o *seekbar*) y el índice que éste ocupa dentro de su arreglo.

```
private boolean MainBSend(String message) {
    String messageBLE = message;
    if(message.equals("["))
        messageBLE = "[\n";

    // Check that there's actually something to send and its a valid address
    if (message.length() == 0) {
        return false;
    }
    // Send wifi TCP
    if (mTCPClient != null)
        mTCPClient.sendMessage(message);

    // Send Bluetooth LE
    if ((mBLeService != null) && EnableBLE)
        mBLeService.writeCharacteristic(messageBLE);

    // Send normal Bluetooth
    if (mBTService != null) {
        mBTService.bwrite(message.getBytes());
    }
    return true;
}
```

Figura 3.23. Función Main Broadcast Send (MainBSend).

Recepción de información a nivel de Capa de Aplicación

Toda la información que es recibida por la aplicación es pasada como argumento al método *decodeMessage* en Main, como se puede observar en la Figura 3.24. La variable *ConnectionCounter* le indica a Main que el microcontrolador sigue conectado y trabaja en conjunto con el comando *CMD_ALIVE*.

```
case MESSAGE_READ:
    myMain.ConnectionCounter++;
    // Message to display is at object
    String message = (String) msg.obj;
    if (!message.equals("\r") && !message.equals(""))
        myMain.decodeMessage(message);
    break;
```

Figura 3.24. Todos los datos entrantes se envían a Main.decodeMessage().

El objetivo de la función decodeMessage es extraer la información importante de las cadenas de texto entrantes. decodeMessage identifica un ATC tag por iniciar con un ‘<’. Toda cadena que no inicie con ‘<’ es considerada como raw data.

Raw Data

Un raw data (información sin procesar) es una cadena de texto terminada por un carácter de fin de línea (“\n” o “0x0A”) que no contiene un ATC tag (e. g. “Hola mundo\n”, “A\n”, “b\n”, “c13\n”, etc.).

Cuando la aplicación envía raw data al MCU lo hace mediante los diferentes tipos de botones (*toggle button, temporary button y long touch button*). El microcontrolador selecciona la acción correspondiente para cada raw data recibido (ya sea un solo carácter o una cadena de texto).

Cuando el microcontrolador envía raw data a la aplicación, la cadena de texto enviada es mostrada en el texto de la parte superior de la aplicación como se muestra en la Figura 3.25.

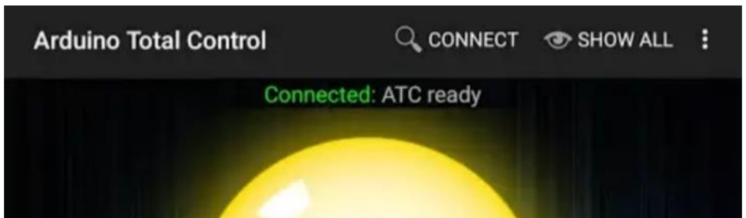


Figura 3.25. Envío del raw data “ATC ready” a la aplicación.

La implantación de raw data en ATC simplifica el uso de la aplicación para usuarios sin experiencia en programación, ya que no necesita más que caracteres y cadenas de texto para lograr el control remoto de un sistema simple.

Alive Command

El comando “Alive” o “latido” es enviado por la aplicación cada 2.5 segundos. La aplicación espera que el microcontrolador responda con cualquier información para asegurar que el microcontrolador está conectado y activo (independientemente si el adaptador de Bluetooth o el cliente TCP están conectados). El comando *alive* es de tipo raw data y es representado por el carácter '[', como se muestra en la Figura 3.26.

```
#define CMD_ALIVE   '['
```

Figura 3.26. Definición del comando Alive en el programa en el microcontrolador.

ATC Tags

Los “tags” en ATC son cadenas de texto que representan comandos específicos para la comunicación entre el dispositivo Android y el MCU. Todos los ATC tags empiezan con el símbolo “<”, lo cual le permite saber al microcontrolador y a la aplicación que la cadena que han recibido es un ATC tag.

User interface tags

Los tags de interfaz de usuario o UI tags le permiten al MCU controlar el estado de los elementos constructivos de la Aplicación:

Toggle Button tags, <ButnXX:Y\n.

Permite cambiar el estado de los botones de activación.
Dónde: “XX” es un número de 00 a 24 y es el número de botón en el Layout; “Y” es un número 0 o 1 y es el estado del botón (“0” apagado y “1” prendido).
Ejemplo: Serial.println("<Butn05:1"); Encenderá el botón de activación 5.

Text tags, <TextXX:YYYY\n

Le permite al microcontrolador cambiar la cadena de texto (YYYY…) mostrada en el texto XX.
“XX” es un número de 00 a 24 que representa el identificador del texto en el Layout. “YYYY…“ es una cadena de texto de hasta 1023 caracteres.
Ejemplo: Serial.println("<Text01:A1: 253"); mostrará el texto "A1: 253" en el texto 1.

Image Tags, <ImgsXX:Y\n

Un tag de imágenes es utilizado para cambiar el estado “Y” de la imagen número “XX”.
“XX” es un número de 00 a 24 y es el número de imagen. “Y” es el estado de la imagen (0 “Default”, 1 “Pressed”, o 2 “Extra”).
Ejemplo: server.println("<Imgs02:1"); cambiará la imagen 2 al estado *Pressed*.

Alarm tags, <Alrm00\n

Permite al usuario activar una alarma auditiva o “beep” en la aplicación.

Vibration Tags, <Vibr00:YYY\n

Le permite al microcontrolador controlar el vibrador del dispositivo móvil. YYY es un número de 000 a 999 que representa el tiempo de vibración en milisegundos.

Ejemplo: server.println(“<Vibr00:150"); hará vibrar el dispositivo móvil por 150ms.

Seekbar Tags, <SkbX:YYY\n

La aplicación le permite al MCU cambiar el estado a valor de las *seekbar* usando *Seekbar* tags. “X” es un número de 0 a 7 que representa el identificador de la barra de acción, y “YYY” es un número de 000 a 255 que representa el valor de la *seekbar*.

Ejemplo: server.println(“<Skb3:206"); Asignará el valor de 206 a la *seekbar* 3.

Analog Bar Tags, <AbarXX:YYY\n

Le permite al MCU controlar el nivel de llenado “YYY” de la barra analógica “XX”. “XX” es un número de 00 a 11 y es el número de barra analógica. “YYY” es un valor de 000 a 255 y es el nivel de la barra analógica.

Text to Speech Tags, <TtoS0X:YYYY\n

Los tags de texto a voz (“text to speech”) le permiten al usuario hacer uso del sintetizador de voz nativo del dispositivo móvil. Cuando el valor de “X” es 0, la aplicación “hablará” en inglés, cuando es 1, la aplicación hablará en el lenguaje por defecto del dispositivo Android. “YYYY…” es cualquier cadena de texto en el idioma seleccionado.

Ejemplo:
```
Serial.println("<TtoS00:Hello world");
Serial.println("<TtoS00:Hola mundo");
```

EL proceso para implementar la función de texto a voz en ATC se detalla el apéndice A.

Special Command tags

Un tag de comando especial es enviado por la aplicación al MCU para transferir información de paneles táctiles, seekbars y acelerómetro.

Touch Pad Tags

- "<PadXxx:YYY\n”. Dónde: “xx” es un número de 00 a 24 y es el número del touchpad, “YYY” es el valor en el eje X (de 0 a 255).
- "<PadYxx:YYY\n”. Dónde: “xx” es un número de 00 a 24 y es el número del touchpad, “YYY” es el valor del eje Y (de 0 a 255).

Seekbar (Special command) tags

- "<SkbX:SYYYYY\n". Dónde: “X” es un número de 0 a 7 y es el identificador de la *seekbar* fuente, “YYYYY” es el valor de la *seekbar* de 0 a 255. S es el valor del signo (+ o -). Para las *seekbar* éste siempre es “+”.

Accelerometer Tags

- <AccX:SYYYYY\n". Dónde: “X” puede ser X, Y o Z, y es el eje del acelerómetro, “YYYYY” es el valor del acelerómetro en m/s^2 multiplicado por 100, por ejemplo: Un valor recibido de 981 es igual a una aceleración de $9.81 m/s^2$. "S" es el valor del signo de la aceleración (+ o -).

Tag de registro

Un tag de registro es utilizado por el microcontrolador para enviar cadenas de texto que la aplicación guardará en un registro interno. El usuario puede hacer uso de los *<Logr* tags para depurar el sistema a base de microcontrolador. La sintaxis de un tag de registro se muestra a continuación:

<Logr00:YYYY\n. Donde “YYYY” es cualquier cadena de texto con un límite de 1023 caracteres.

Capítulo 4. El Sistema a base de microcontrolador.

Para lograr el control inalámbrico de sistemas embebidos compatibles con Bluetooth, Bluetooth LE y Wi-Fi; el sistema a base de microcontrolador debe contar con el hardware y el software (firmware) mínimo necesario para ser compatible con la aplicación Arduino Total Control.

Para que un sistema a base de microcontrolador sea compatible con ATC debe contar con algún módulo de comunicaciones inalámbricas de alguno de los estándares soportados (Bluetooth, Bluetooth LE o Wi-Fi). Además, su código debe contener las funciones de comunicación de Capa de Aplicación, que le permiten comunicarse con la aplicación bajo "el mismo idioma" o protocolo.

Además de este firmware mínimo, el sistema a base de microcontrolador cuenta con entradas y salidas, analógicas y digitales; para llevar a cabo las tareas de control y monitoreo específicas de cada sistema. La Figura 4.1 muestra el diagrama de bloques de un sistema a base de microcontrolador compatible con ATC.

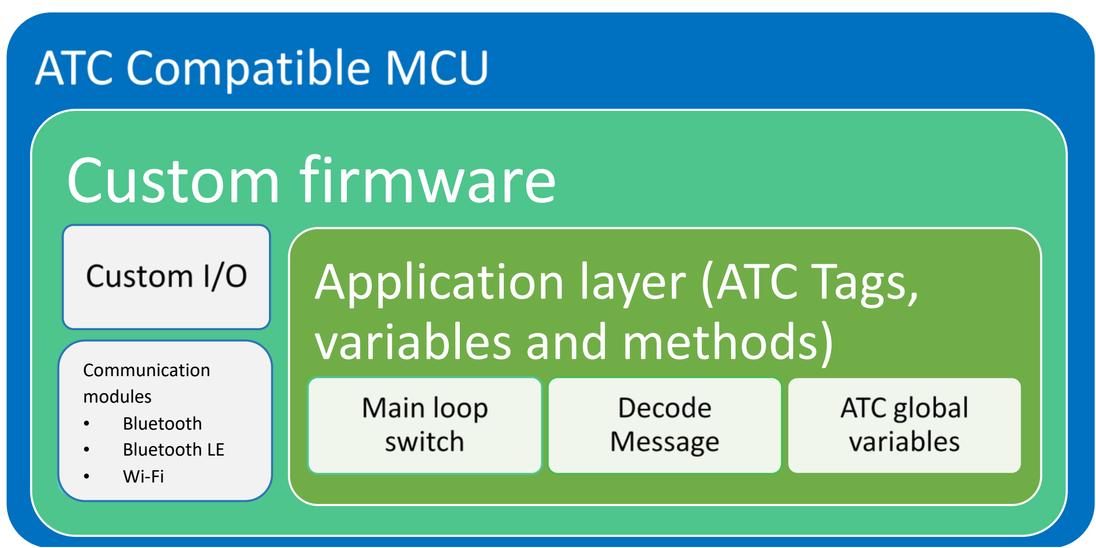


Figura 4.1. Diagrama de bloques del sistema a base de microcontrolador.

Como se menciona en el capítulo uno, Arduino es la plataforma de firmware que se utiliza en ATC; ya que está basada en hardware y software fácil de usar, además de ser la tarjeta de microcontrolador más usada en la creciente comunidad Maker. Sin embargo, ATC es compatible con cualquier sistema digital que pueda implementar el protocolo de Capa de Aplicación ATC sobre Bluetooth, Bluetooth LE o Wi-Fil; como se mostrará en el capítulo 5 de este trabajo.

Hardware mínimo

El hardware mínimo, es decir, el mínimo conjunto de elementos físicos que constituyen el sistema a base de microcontrolador en ATC; generalmente comprende el microcontrolador con sus periféricos para funcionar (oscilador, resistencias *pull-up* y fuente de energía); y el módulo de comunicaciones inalámbricas Bluetooth, Bluetooth LE o Wi-Fi.

La interfaz entre el microcontrolador y el módulo de comunicaciones inalámbricas depende del módulo en cuestión. En este trabajo, estas interfaces son comunicación SPI (para el *Ethernet Shield*) y UART (para ESP8266, HC-05, HC-06 y HM-10).

UART

Un Receptor-Transmisor Asíncrono Universal (UART) se usa en la transmisión de datos asíncronos entre el DTE (Equipo Terminal de Datos; en este caso el MCU) y el DCE (Equipo de Comunicación de Datos; e. g. el módulo Bluetooth). En una transmisión asíncrona no hay sincronización de la información transferida entre el DTE y el DCE. Las funciones principales del UART son [48]:

1. Hacer conversión de datos, de serie a paralelo y de paralelo a serie.
2. Detectar errores insertando y comprobando los bits de paridad.
3. Insertar y detectar los bits de arranque y de paro.

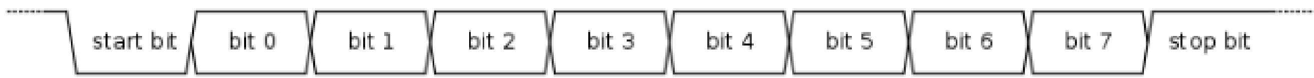


Figura 4.2. Diagrama de tiempos de una trama UART [49].

Cuando el UART está inactivo, la línea (el transmisor) está en estado alto. Cada carácter o byte es iniciado con un bit en estado bajo o bit de arranque (usualmente un carácter es de 8 bits, pero el usuario puede seleccionar tramas de 5 a 9 bits). Los siguientes 5 a 9 bits representan el carácter transmitido. Al final del carácter un bit de paridad es colocado si esta función está habilitada. Al final de la trama uno o dos bits de parada son colocados para señalizar el final de la transmisión. Si la línea es mantenida en estado bajo por más de la duración de un carácter, el UART lo detectará y señalizará la falla al microcontrolador.

La velocidad de transmisión recomendada entre el microcontrolador y el módulo de comunicaciones inalámbrica con UART en ATC es de 115,200 bits/s.

SPI

El *Serial Peripheral Interface* (SPI) permite la transmisión de datos síncronos de alta velocidad (hasta 4,000,000 bits/s para un Arduino con un cristal de 16MHz) entre el microcontrolador y dispositivos periféricos (en este caso el Ethernet Shield). SPI es un bus de comincaciones sincronas full-duplex, con modos de operación maestro y esclavo. La interfaz SPI tiene cuatro terminales: SCLK, MOSI, MISO, y SS (slave slect) como se muestra en la Figura 4.3.

- **SCLK** *(Reloj)*: Es el pulso que marca la sincronización. Con cada pulso de este reloj, se lee o se envía un bit. El reloj es proporcionado por el maestro.
- **SS/Select** *(Slave Select):* Un maestro SPI activa uno de sus puertos de salida para seleccionar a un esclavo en el bus. Un esclavo o Slave SPI usa su pin SS para determinar que ha sido seleccionado.
- **MOSI** *(Master Output Slave Input)*: Salida de datos del Master y entrada de datos al Slave.
- **MISO** *(Master Input Slave Output)*: Salida de datos del Slave y entrada al Master.

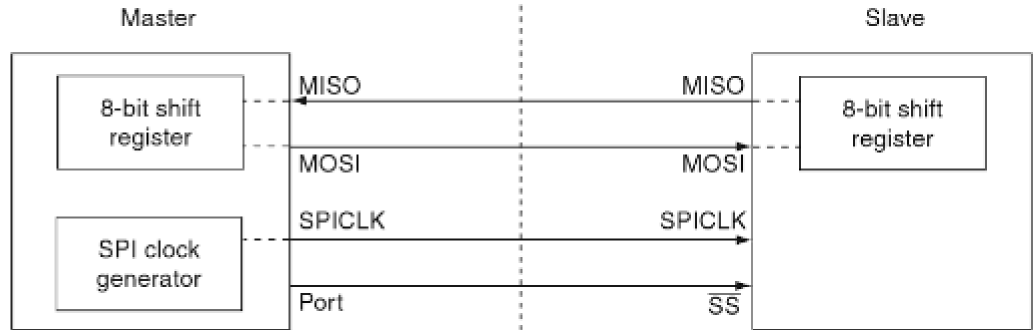


Figura 4.3. Conexión maestro-esclavo (*Master - Slave*) SPI simple [50].

Bluetooth Clásico (BR/EDR) / Bluetooth Low Energy

Como se menciona en el capítulo uno, los módulos de comunicación inalámbrica más utilizados en la comunidad de entusiastas son los módulos HC-05, HC-06 y HM-10. Debido a que estos usan la misma interface UART, para el microcontrolador es transparente si se utiliza un módulo HC-05 (Bluetooth BR/EDR maestro); HC-06 (Bluetooth BR/EDR esclavo); o HM-10 (Bluetooth LE). Estos módulos cuentan con las mismas terminales de conexión como se muestra en la Figura 4.4.

Figura 4.4. Comparación entre módulos HC-05 o HC-06 (arriba) y HM-10 (abajo).

Conexión

Para comunicarse con ATC, un módulo Bluetooth (como los descritos en esta sección) debe estar conectado como se muestra en la Figura 4.5 y se describe a continuación:

- Conectar Bluetooth Vcc a 5V o 3V.
- Conectar Bluetooth GND a Arduino GND ó 0V.
- Conectar Bluetooth Rx a Arduino Tx.
- Conectar Bluetooth Tx a Arduino Rx.

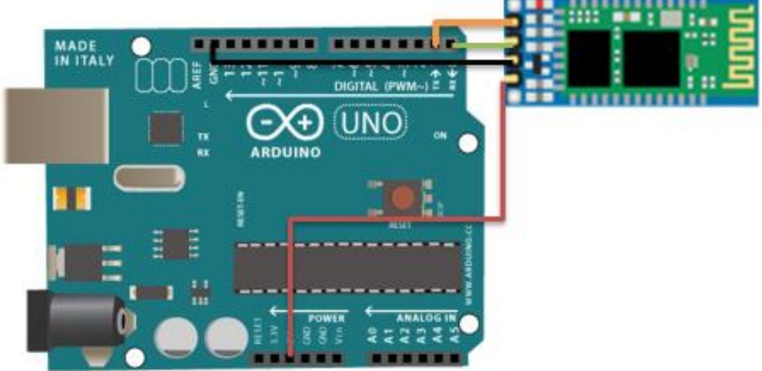


Figura 4.5. Conexión Bluetooth con Arduino.

Para realizar una correcta interfaz entre el microcontrolador y el módulo Bluetooth, el usuario debe tomar en cuenta los siguientes puntos:

- **Configurar el *baud rate* correcto**. El baud rate (tasa de transferencia o número de unidades de información por segundo) se configura en el sketch de Arduino modificando el valor de la constante #define BAUD_RATE y debe ser idéntico al baud rate del módulo Bluetooth. La mayoría de los módulos Bluetooth tienen una configuración por defecto de 9600 bits por segundo y es configurable a través de comandos AT (apéndice C).
- **Desconectar la terminal TX cuando se realice una descarga del programa**. Esto es aplicable cuando se utiliza un Arduino uno (ATmega328) ya que sólo cuenta con un puerto serial y es el mismo que utiliza para programar la memoria flash del microcontrolador.
- **No utilizar el pin 0 y 1 como salidas o entradas**. Esto es aplicable si se utiliza un Arduino UNO ya que estos son los pines que se utilizan para realizar la comunicación serial con el módulo Bluetooth.

- **Habilitar la resistencia de pull-up** en el pin Rx del UART del Arduino. Se ha observado un comportamiento más estable de la información entrante al MCU cuando la resistencia de pull up de la terminal receptora del UART está habilitada y esto se debe a que los módulos HC-05, HC-06 y HM-10 operan a 3.6V y el AVR a 5V.

Wi-Fi

Como se menciona en el capítulo 1, existen diferentes formas de lograr una conexión Wi-Fi con el MCU: a través de un Ethernet Shield, usando un Arduino YUN o utilizando un módulo ESP8266.

Ethernet Shield

Para realizar una conexión a través de Ethernet desde el MCU a la aplicación, se necesita (además de la tarjeta Ethernet Shield), un Reuter inalámbrico que sirva de punto de acceso para la aplicación. La conexión entre el *router* y el Ethernet Shield se realiza mediante un cable derecho UTP. La Figura 4.6 enlista el hardware mínimo para realizar una conexión utilizando el Ethernet Shield.

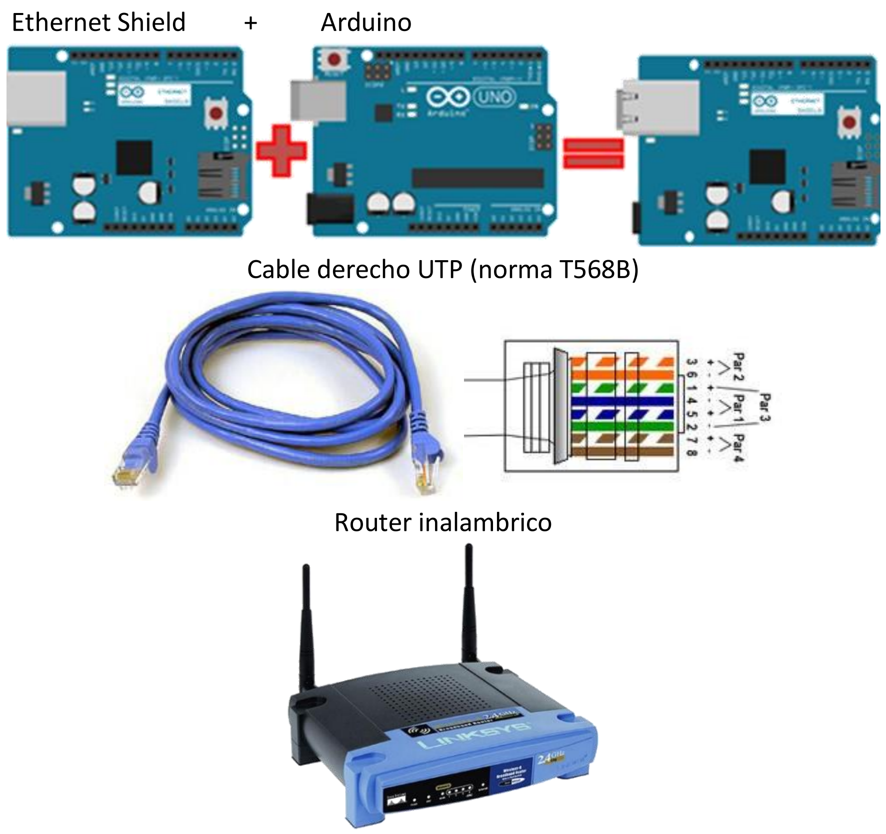


Figura 4.6. Hardware mínimo para conexión por Ethernet.

El conexionado en el Ethernet Shield, que se comunica con el MCU por SPI, es hecho de forma automática al montar el Shield sobre la placa Arduino. Sin embargo, el usuario deber tomar en cuenta los siguientes puntos:

- Para Arduino UNO los pines 10, 11, 12 y 13 son dedicados para la comunicación SPI y no se pueden usar como GPIO.
- Para Arduino MEGA los pines 10, 50, 51 y 52 son dedicados para la comunicación SPI y no se pueden usar como GPIO.
- La dirección IP y puerto deben estar configurados con el mismo valor tanto en la aplicación como en el MCU.

ESP8266

Como se mencionó en el capítulo 1, el módulo ESP8266 se comunica por medio de una UART con el MCU. El baud rate es configurable y 115200 bits/s por defecto. Un módulo ESP8266 se puede conectar directamente al dispositivo móvil

por medio *Wi-Fi Direct* (conexión directa entre dos dispositivos Wi-Fi sin un punto de acceso común) o por medio de un router inalámbrico como intermediario.

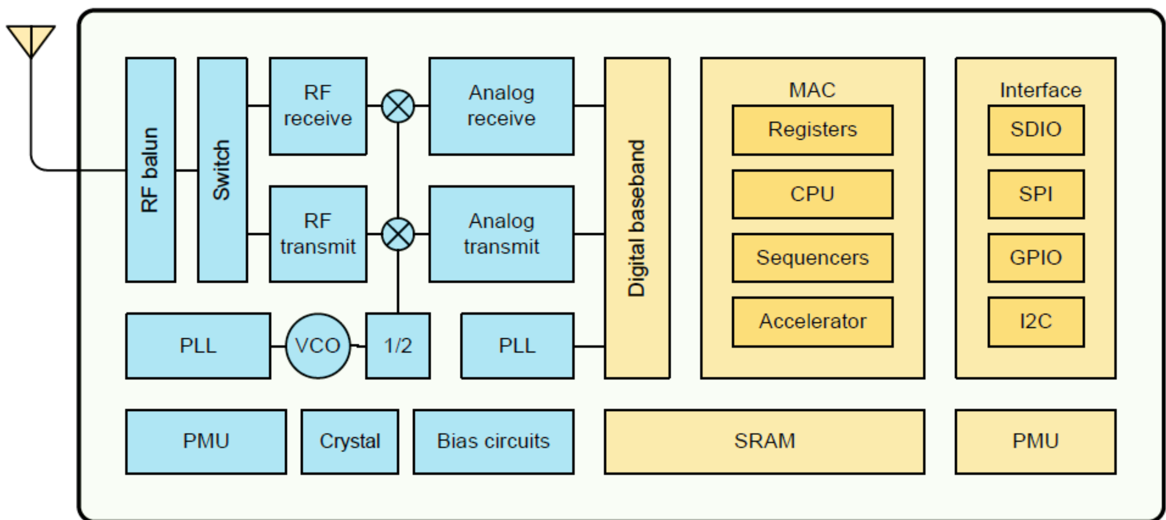


Figura 4.7. Diagrama de bloques ESP8266.

Conexionado

La conexión de un módulo ESP8266 no es tan clara como en los módulos anteriores, ya que las terminales no se encuentran impresas en la tarjeta. La Figura 4.8 muestra las terminales de un módulo ESP8266 y su conexión a un microcontrolador.

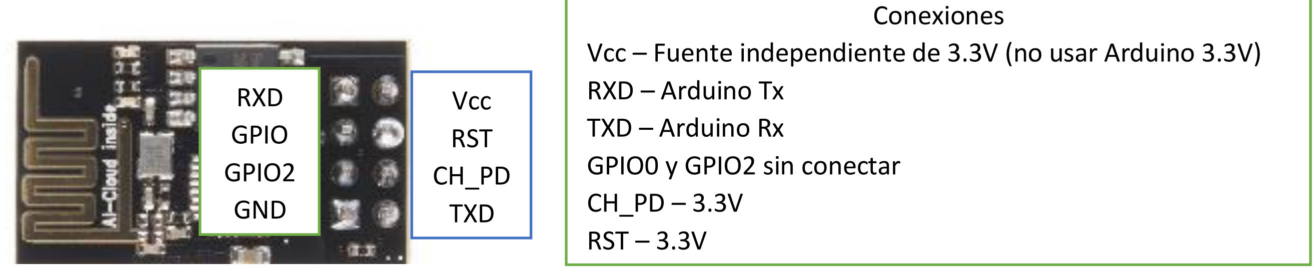


Figura 4.8. Terminales y conexión de un módulo ESP8266.

Como se muestra en la Figura 4.8, es conveniente conectar el módulo ESP8266 a una fuente independiente de 3.3V; ya que el regulador de tensión (LP2985-33DBVR) de 3.3V de un Arduino tiene una corriente de salida máxima de 150mA [51], y este módulo puede llegar a consumir hasta 300mA cuando transmite y recibe información a un ciclo de trabajo del 90% [26].

Para realizar una correcta interfaz entre el microcontrolador y el módulo ESP8266, el usuario debe configurar el módulo ESP8266 usando los comandos AT descritos en el apéndice E.

Arduino YUN

No se requiere hardware ni conexiones adicionales; ya que, como se vio en el capítulo uno, Arduino YUN contiene el módulo de comunicaciones inalámbricas integrado.

Estructura de un programa en Arduino

Un programa o *sketch* en Arduino tiene la terminación ".ino" y está formado por mínimo dos funciones; la función *setup()* y la función *loop().* Ambas funciones no retornan ningún valor como el prefijo *void* lo sugiere en la Figura 4.9.

```
void setup() {
  // put your setup code here, to run once:

}
void loop() {
  // put your main code here, to run repeatedly:

}
```

Figura 4.9. Funciones setup y loop en Arduino.

La función ***setup()*** es llamada cuando el sketch inicia, y es utilizada para inicializar variables, modos de pin (entrada o salida), uso de librerías, etc. La función setup solo correrá una vez, cada vez que la tarjeta Arduino es energizada o *reseteada* o restablecida.

La función ***loop()*** se repite consecutivamente, permitiendo que el programa cambie y responda. Esta función es usada para controlar la tarjeta Arduino.

La Figura 4.10 muestra la función *main*() que el Arduino IDE crea al momento de compilar el archivo ".ino". Como se puede observar, la función setup() solo será llamada una vez y la función loop() se ejecutará una y otra vez hasta que se presente un evento serial (e. g. carga de un nuevo programa) o el microcontrolador se apague o sea reseteado.

```
#include <Arduino.h>

// Declared weak in Arduino.h to allow user redefinitions.
int atexit(void (* /*func*/ )()) { return 0; }

// Weak empty variant initialization function.
// May be redefined by variant files.
void initVariant() __attribute__((weak));
void initVariant() { }

int main(void)
{
	init();

	initVariant();

#if defined(USBCON)
	USBDevice.attach();
#endif

	setup();

	for (;;) {
		loop();
		if (serialEventRun) serialEventRun();
	}

	return 0;
}
```

Figura 4.10. Loop Main para sketches Arduino.

Software mínimo

Además de contar con el hardware correcto, un sistema a base de MCU debe contener las variables y métodos o funciones de Capa de Aplicación ATC que le permitan comunicarse con la aplicación bajo el mismo protocolo. Cualquier sistema a base de microcontrolador compatible con ATC (ya sea que utiliza Bluetooth, o Wi-Fi) opera de acuerdo al diagrama de flujo de la Figura 4.13.

La Figura 4.13 muestra en **negrita** y recuadro **verde** las operaciones que forman parte del software mínimo que un MCU requiere para ser compatible con ATC (independientemente del módulo de comunicaciones inalámbricas utilizado). El primer paso es declarar e iniciar las variables y constantes de protocolo de Capa de Aplicación ATC en el *scope* global del programa del MCU como se muestra en la Figura 4.11.

```
// Special commands
#define CMD_SPECIAL '<'
#define CMD_ALIVE   '['

// Data and variables received from especial command
int Accel[3] = {0, 0, 0};
int SeekBarValue[8] = {0, 0, 0, 0, 0, 0, 0, 0};
int TouchPadData[24][2]; // 24 max touch pad objects, each one has 2 axis (x and Y)
String SpeechRecorder = "";
```

Figura 4.11. Variables y constantes del protocolo ATC en el MCU.

El segundo paso es inicializar el módulo de comunicaciones inalámbricas, el cual es dependiente del módulo en cuestión y se expondrá en la siguiente sección. El tercer paso consiste en verificar el estado del buffer de entrada y procesar la información entrante dependiendo si ésta es *raw data, special command, o alive command* (véase capítulo 3).

La discriminación de la información entrante se realiza mediante una sentencia *switch;* la cual puede tener como mínimo dos casos, *CMD_SPECIAL* y *CMD_ALIVE* como se muestra en la Figura 4.12. *appData* es una variable tipo *char,* local a la función *loop,* donde el primer carácter disponible del buffer de entrada se guarda temporalmente.

```
switch(appData){
case CMD_SPECIAL:
  // Special command received, seekbar value and accel value updates
  DecodeSpecialCommand();
  break;

case CMD_ALIVE:
  // Character '[' is received every 2.5s
  server.println("ATC ready");
  break;
}
```

Figura 4.12. Loop switch.

Si el comando recibido es de tipo "CMD_SPECIAL", la función *DecodeSpecialCommand()* es llamada para extraer la información de los *tags* de *seekbars, touchpads, speech recognizer* o acelerómetro enviados por la aplicación. Una vista preliminar de esta función es mostrada en la Figura 4.14.

Si la información recibida (*appData*) corresponde a un carácter de CMD_ALIVE, el MCU deberá enviar cualquier mensaje, carácter o cadena de texto a la aplicación en menos de 2.5 segundos. Como se mencionó en el capítulo 3, la aplicación envía este comando una vez cada 2.5s esperando que el MCU responda, si el último no envía ningún mensaje, la aplicación alarmará al usuario y terminará la conexión.

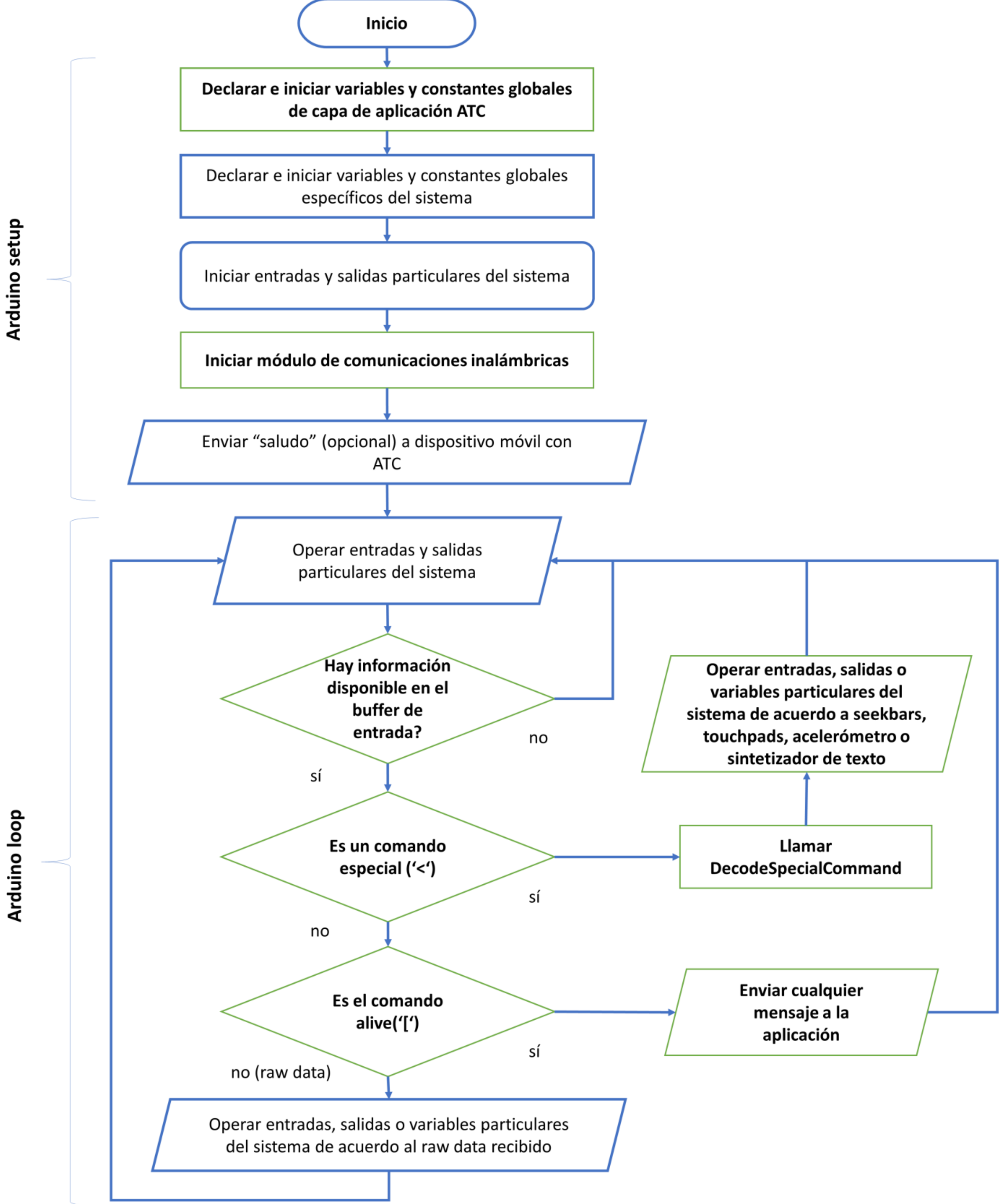


Figura 4.13. Diagrama de flujo del sistema a base de microcontrolador.

Cuando *appData* no es igual a CMD_ALIVE o CMD_SPECIAL, los datos recibidos son tratados como *raw data* y dependerá del sistema digital especifico el uso que se quiera dar a esta información.

```
void DecodeSpecialCommand() {
  // Read the whole command
  String thisCommand = Readln();

  // First 5 characters will tell us the command type
  String commandType = thisCommand.substring(0, 5);

  if (commandType.equals("AccX:")) {
    ...
  }
  if (commandType.equals("AccY:")) {
    ..
  }
  if (commandType.equals("AccZ:")) {
    ...
  }
  if (commandType.substring(0, 4).equals("PadX")) {
    ...
  }
  if (commandType.substring(0, 4).equals("PadY")) {
    ...
  }
  if (commandType.substring(0, 3).equals("Skb")) {
    ...
  }
  if (commandType.equals("StoT:")) {
    ...
  }
```

Figura 4.14. Vista preliminar de función *DecodeSpecialCommand().*

Consideraciones de la función loop en ATC

Como muestra el diagrama de flujo de la Figura 4.13, la función setup se encarga de iniciar el hardware y el software mínimo para correr una implementación compatible con ATC; mientras la función loop se encarga de operar las entradas y salidas, así como leer y enviar información activamente de y a la aplicación. Si operar las entradas y salidas del sistema, ya sea porque se ha recibido un dato de la aplicación, o si es parte del funcionamiento normal del sistema, genera algún retardo considerable; la velocidad de respuesta del microcontrolador a los comandos de entrada será afectada y se reflejará en una respuesta lenta o retrasada a los comandos enviados por el usuario al usar la aplicación.

Inicialización, envió y recepción de información Bluetooth EDR/BR y Bluetooth LE

Inicialización

Ya que tanto como los módulos Bluetooth EDR/BR (HC-05 y HC-06) y Bluetooth LE (HM-10) utilizan el puerto serial como una interfaz transparente entre la información recibida y enviada, desde y a la aplicación, lo único que se requiere para inicializar el módulo en cuestión es iniciar la UART. En Arduino, la UART o puerto *Serial* se inicia con la función *begin*, como se muestra en la Figura 4.15. El BAUD_RATE, que es la tasa de transferencia, es una constante global, y se recomienda un valor de 115200 bits/s.

```
// initialize BT Serial port
Serial.begin(BAUD_RATE);
```

Figura 4.15. Inicialización de la UART en Arduino.

Envío de información

Para enviar información a la aplicación utilizando un módulo Bluetooth, como los mencionados anteriormente, basta con llamar la función *println*, y colocar como argumento la cadena de texto a enviar. La función println inserta de forma

automática el carácter de fin de línea '\n' al final de la cadena a enviar. El ejemplo de la Figura 4.16 muestra como el MCU puede enviar tags de Interfaz de Usuario a la aplicación.

```
// Use <Text tags to display alphanumeric information in app
Serial.println("<Text" + AIAppId[i] + ":" + "An: " + String(sample));
// Use <Imgs tags to dinamically change pictures in app
Serial.println("<Imgs" + AIAppId[i] +  EvaluateAnalogRead(sample));
// Use <Abar tags to change analog bar levels from 0 to 255
Serial.println("<Abar" + RelayAppId[i] +  ":" + myIntToString(sample >> 2));
```

Figura 4.16. Ejemplo de envío de tags de Interfaz de Usuario a la aplicación.

Recepción de información

Para recibir información en un módulo Bluetooth, simplemente se llama la función *read* del puerto serial. *Serial.read*() lee un byte o carácter del buffer de entrada y lo almacena en la variable appData, como se muestra en la Figura 4.17. Si no hay información disponible en el buffer de entrada del puerto *Serial*, el valor retornado será **-1**.

```
appData = Serial.read();   // Get a byte from app, if available
switch (appData) {
  case CMD_SPECIAL:
```

Figura 4.17. Lectura de un byte en Bluetooth.

El firmware mínimo para una conexión Bluetooth EDR/R y Bluetooth LE se encuentra en el sketch *bt_firmware.ino* en el anexo 1 y el repositorio de GitHub:

https://github.com/JuanLuisGonzalez/ATC-Release-Codes/blob/master/bt_firmware/bt_firmware.ino.

Inicialización, envío y recepción de información Wi-Fi

Como se mencionó en el capítulo 3, la aplicación ATC actúa como un cliente TCP/IP y el sistema a base de MCU como un servidor. Los métodos para iniciar, enviar y recibir datos de una tarjeta Wi-Fi o Ethernet configurado como servidor TCP/IP cambian dependiendo del módulo a utilizar:

Ethernet Shield

Inicialización

A diferencia de la inicialización de un módulo Bluetooth, como los mencionados en la sección anterior; para iniciar un Shield Ethernet, el cual está basado en el circuito Wiznet W5100, es necesario incluir las librerías SPI y Ethernet, así como declarar e iniciar variables relacionadas a esta interfaz. Para utilizar un Shield Ethernet se requiere declarar un servidor, un cliente, una dirección IP y una dirección MAC como se muestra en la Figura 4.18.

```
#include <SPI.h>
#include <Ethernet.h>

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,1,60);

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(80);
EthernetClient client;
```

Figura 4.18. Variables y constantes globales para Ethernet Shield.

En la función setup, las funciones *begin* de la tarjeta Ethernet y del servidor TCP/IP son llamadas para iniciar un servidor TCP/IP, con la dirección IP y puerto previamente definidos, como se muestra en la Figura 4.19.

```
// start the Ethernet connection and the server:
Ethernet.begin(mac, ip);
server.begin();
```

Figura 4.19. Inicialización de un servidor TCP/IP en Ethernet Shield.

Envío de información

Homólogo al UART, para enviar información a la aplicación utilizando un Ethernet Shield se utiliza la función println, la cual toma como argumento la cadena de texto a enviar. Tanto el objeto server como *client* tienen la función println y ambos pueden ser usados para enviar información a la aplicación (el cliente). Sin embargo, *server.println*() enviará la cadena de texto a todos los dispositivos móviles conectados (broadcast); cuando *client.print*() solo envía la cadena de texto al cliente activo en ese ciclo de loop.

```
// Use <Text tags to display alphanumeric information in app
server.println("<Text" + AIAppId[i] + ":" + "An: " + String(sample));
// Use <Imgs tags to dinamically change pictures in app
server.println("<Imgs" + AIAppId[i] +  EvaluateAnalogRead(sample));
// Use <Abar tags to change analog bar levels from 0 to 255
server.println("<Abar" + RelayAppId[i] +  ":" + myIntToString(sample >> 2));
```

Figura 4.20. Envío de tags de interfaz de usuario en Ethernet Shield.

Recepción de información

Para recibir datos de un cliente TCP se inicializa el cliente en cada ciclo de loop, usando el método *available()* de server como se muestra en la Figura 4.21. Si hay un cliente disponible con datos listos para lectura, la condición *if(client)* será verdadera y se procederá a leer un byte del buffer de entrada.

```
client = server.available();
if (client){
  appData = client.read();
```

Figura 4.20. Lectura de un byte de un cliente TCP en Ethernet Shield.

El firmware mínimo para una conexión Ethernet (y un *router* inalámbrico) se encuentra en el sketch *eth_firmware.ino* en el anexo 2 y el repositorio de GitHub:

https://github.com/JuanLuisGonzalez/ATC-Release-Codes/blob/master/eth_firmware/eth_firmware.ino

ESP8266

Inicialización

Para iniciar este módulo como un servidor TCP/IP habilitado para manejar múltiples sockets o canales (un socket es la conexión entre el servidor y un cliente); los comandos AT correctos deben ser enviados al módulo ESP8266. Estos comandos se envían en la función *myEsp_Init(),* la cual es llamada en la función *setup* de Arduino antes de iniciar una conexión. La interfaz entre el microcontrolador y el ESP8266 se realiza mediante un UART. Los comandos AT de un módulo ESP8266 se describen en el apéndice E. La Figura 4.22 muestra los pasos para iniciar un módulo ESP8266.

Para conocer la dirección IPV4 asignada a un módulo ESP8266 se puede enviar el comando AT+CIFSR una vez que el módulo está conectado a la red local. CIFSR retornará la dirección IP del módulo si éste pudo conectarse exitosamente a la red local, de lo contrario será "0.0.0.0".

Envío de información

El envío de información usando un módulo ESP8266 configurado como un servidor TCP/IP multicanal en ATC se realiza mediante las funciones *myESP_Print* y *myESP_Println*. Para enviar una cadena de texto por medio del ESP8266 se utiliza el comando "AT+CIPSEND=X,n", donde; "X" es el canal o socket TCP/IP cliente; y "n" es el tamaño de la cadena de texto

a enviar. Después de enviar el comando AT+CIPSEND, el módulo ESP tomará los siguientes n bytes recibidos del microcontrolador como la cadena de texto a enviar al cliente TCP/IP conectado.

```
void myESP_Init() {
  // Initialize serial port with pull up in rx
  Serial.begin(BAUD_RATE);
  pinMode(RX_PIN, INPUT_PULLUP);

  // Reset module
  Serial.println("AT+RST"); // Reset the board
  delay(5000);
  // Uncomment lines below to set up network for the first time
  //Serial.println("AT+CWMODE=3"); // both wifi client and access
  //delay(100);
  //Serial.println("AT+CWJAP=\"linksys\",\"\""); // Join access p
  //delay(10000);
  Serial.println("AT+CIPMUX=1"); // Multichannel connection
  delay(500);
  Serial.println("AT+CIPSERVER=1," + sPort); // Open server socke
  delay(500);
}
```


Figura 4.22. Configuración inicial de un módulo ESP8266 como un servidor multicanal TCP/IP.

Como se puede observar en la Figura 4.23, el módulo ESP8266 requiere ciertos retardos para procesar la información entrante (proveniente del MCU), y enviarla por TCP/IP sobre 802.11 (Wi-Fi). Estos retardos están dados por *Serial.flush*() que pausa el programa hasta que el buffer de salida del puerto serial se vacía, y *delay*(x) la cual detiene la ejecución del programa por “x” milisegundos.

```
// Send string data to app
void myESP_Println(String data, int channel) {
  // Get data size, add 2 for nl and cr
  int dataSize = data.length() + 2;

  // submit cipsend command for channel
  Serial.println("AT+CIPSEND=" + String(channel) + "," + String(dataSize));
  Serial.flush(); // wait transmision complete
  delay(3);       // wait esp module process

  // Print actual data
  Serial.println(data);
  Serial.flush(); // wait transmision complete
  delay(6);       // wait esp module process
}

// Send string data to app
void myESP_Print(String data, int channel) {
  int dataSize = data.length();

  // submit cipsend command for channel
  Serial.println("AT+CIPSEND=" + String(channel) + "," + String(dataSize));
  Serial.flush(); // wait transmision complete
  delay(3);       // wait esp module process

  // Print actual data
  Serial.print(data);
  Serial.flush(); // wait transmision complete
  delay(8);       // wait esp module process, normally used for big data string, wait more
}
```

Figura 4.23. Funciones para envío de datos para el módulo ESP8266.

Debido al tiempo que requiere el módulo ESP8266 para enviar un paquete de datos, la función myESP_Println() es poco utilizada, en su lugar se usa la función myESP_Print(), cuyo argumento suele ser una cadena de texto o paquete mucho más grande que incluye múltiples comandos (en una sola cadena) para la aplicación, como se muestra en la Figura 4.24.

```
   // Concentrate all samples information in a single data packet
   // Use <Text tags to display alphanumeric information in app
   boardMessage = boardMessage + "<Text" + AIAppId[i] + ":" + "An: " + String(sample) + "\n";
   // Use <Imgs tags to dinamically change pictures in app
   boardMessage = boardMessage + "<Imgs" + AIAppId[i] + EvaluateAnalogRead(sample) + "\n";
   // Use <Abar tags to change analog bar levels from 0 to 255
   boardMessage = boardMessage + "<Abar" + RelayAppId[i] +  ":" + myIntToString(sample >> 2) + "\n";
 }
 // Send all info in a single print
 myESP_Print(boardMessage, 0);
```

Figura 4.24. Envío de múltiples comandos en una sola cadena (boardMessage).

Recepción de información

Tanto el envío como la recepción de información usando un módulo ESP8266 no son transparentes como en el caso de los módulos Bluetooth. Para recibir una cadena de texto de un módulo ESP8266 hay que detectar la cadena "+IPD,X,n:YYY…"; donde "X" es el canal o socket TCP/IP cliente fuente; "n" es el número de caracteres del mensaje; y "YYY…" es la cadena de texto de tamaño "n" recibida del canal "X". La Figura 4.25 muestra la función *myESP_Read*() que se utiliza para extraer la información útil de un *stream* proveniente de un módulo ESP8266.

```
// This read function is faster
// call this function when you have minimum 9 bytes available
// When data received: +IPD,0,2:[ (for this exaple 0 is the channel, and 2 is the data lenght)
String myESP_Read(int channel) {
  String message = "";
  int plusIndex = -1;
  String theIPD = "";
  String dataFromClient = "";

  // Read a complete line
  dataFromClient = Readln();

  // Find the '+'
  plusIndex = dataFromClient.indexOf('+');
  if (plusIndex == -1) return message;

  // If next 3 chars are IPD then this is a good command
  theIPD = dataFromClient.substring(plusIndex + 1, plusIndex + 4);
  if (theIPD.equals("IPD")) {
    // Extract message
    int twoPointsIndex = dataFromClient.indexOf(':');       // find the ':' on the command
    message = dataFromClient.substring(twoPointsIndex + 1); // set the message
  }

  return message;
```

Figura 4.25. Función myESP_Read.

Debido a que un mensaje proveniente del módulo ESP8266 al MCU como mínimo tendrá 10 caracteres (e. g. "+IPD,0,2:["), la función loop implementa la condición de que antes de llamar a la función myESP_Read, el buffer de entrada del puerto serial tiene que tener como mínimo 10 caracteres como se muestra en la Figura 4.26.

```
// ============================================================
// This is the point were you get data from the App
if (Serial.available() > 10) {
  appMessage = myESP_Read(0);    // Read channel 0
  appData = appMessage.charAt(0);

  switch (appData) {
    case CMD_SPECIAL:
      // Special command received
      DecodeSpecialCommand(appMessage.substring(1));
```

Figura 4.26. Condición de espera de 10 bytes mínimo en buffer de entrada.

El firmware mínimo para una conexión Wi-Fi utilizando el módulo ESP8266 se encuentra en el sketch *esp_firmware.ino* en el anexo 3 y el repositorio de GitHub:

https://github.com/JuanLuisGonzalez/ATC-Release-Codes/blob/master/esp_firmware/esp_firmware.ino

Arduino Yun

Inicialización

Homólogo al Ethernet Shield, Arduino Yun necesita declarar los objetos servidor y cliente, como variables globales antes de la función setup. En la función setup, Arduino Yun deberá iniciar la librería *Bridge*, la cual coordina la comunicación entre el microcontrolador y el módulo Wi-Fi integrado; además, el servidor TCP/IP deberá ser iniciado con la función begin(). Un servidor TCP/IP en Arduino Yun corre sobre el puerto 5555, y la dirección IP se puede conocer una vez que el módulo se conecta a la red local.

```
// Listen on default port 5555, the webserver on the Yun
// ip addres is asigned by your router
YunServer server;
YunClient client;
```

```
// Bridge startup, set up comm between arduino an wifi module
Bridge.begin();

// Listen for incoming connections
// Arduino acts as a TCP/IP server
server.begin();
```

Figura 4.27. Inicialización de Arduino Yun como Servidor TCP/IP. Declaración de variables globales (arriba) e inicialización de servidor en función setup (abajo).

Envío de información

El envío de información es homólogo al Shield Ethernet, usando la función server.println se logra enviar una cadena de texto a todos los clientes conectados al servidor.

```
case CMD_ALIVE:
  // Character '[' is received every 2.5s
  server.println("ATC ready");
  break;
```

Figura 4.28. Ejemplo de envío de información a aplicación en Arduino Yun.

Recepción de información

La recepción de datos en Arduino Yun es homóloga al Shield Ethernet como se muestra en la Figura 4.29. Si hay un cliente activo con información disponible, la condición *if(client)* será verdadera y un byte del puerto serial será

almacenado de forma temporal en la variable *appData* para su futura discriminación en *special command, alive command* o *raw data*.

```
// ===========================================================
// This is the point were you get data from the App
// Get clients coming from server
if(client) appData = client.read();   // Get a byte from app, if available
else client = server.accept();        // If client available accept connection
```

Figura 4.29. Lectura de información en Arduino Yun.

El firmware mínimo para una conexión Wi-Fi utilizando Arduino Yun se encuentra en el sketch *yun_firmware.ino* en el anexo 4 y el repositorio de GitHub:

https://github.com/JuanLuisGonzalez/ATC-Release-Codes/blob/master/yun_firmware/yun_firmware.ino

Capítulo 5. Pruebas funcionales y prototipos.

En los capítulos 3, 4 y 5 se expuso cómo se logra la interfaz gráfica personalizable en ATC; las clases que manejan el descubrimiento, conexión e intercambio de información entre el dispositivo Android y el módulo de comunicaciones inalámbricas o Equipo de Comunicación de Datos (DCE); y el software y hardware mínimo que requiere un microcontrolador para comunicarse con ATC respectivamente.

En la primera sección de este capítulo se mostrarán las diferentes pruebas que se realizaron para asegurar el correcto funcionamiento de la aplicación y el sistema a base de microcontrolador. En la segunda sección se presentan prototipos compatibles con la aplicación Arduino Total Control.

Pruebas funcionales

Las pruebas funcionales que se realizaron antes de construir los prototipos compatibles con la aplicación se dividen en dos grupos; las pruebas de comunicación entre el dispositivo móvil y los diferentes módulos de comunicación inalámbrica, las cuales se listan en la Tabla 5.1 y se realizaron para cada módulo de comunicación inalámbrica o DCE soportado[3]; y las pruebas de protocolo de Capa de Aplicación ATC, las cuales sólo se aplicaron en un Arduino equipado con una tarjeta Ethernet y se listan en la Tabla 5.2 (página 106), bajo la premisa de que las funciones de Capa de Aplicación son las mismas para todos los DCE soportados.

Pruebas de comunicaciones inalámbricas

El objetivo de las pruebas listadas en la Tabla 5.1 es asegurar que los módulos de comunicación inalámbrica se pueden conectar con la aplicación para enviar y recibir información al baud rate recomendado (115200 bit/s para los módulos comunicados vía UART). Estas pruebas no incluyen al microcontrolador ya que se llevan a cabo observando los datos enviados y recibidos del lado del DCE en el monitor serial de Arduino IDE usando un adaptador UART-USB; y *logcat*, el cual es una herramienta de visualización de mensajes para depurar aplicaciones en Android Studio. La Figura 5.1 muestra el flujo de información entre los distintos programas y módulos. En el caso de ethernet Shield, se usa un Arduino UNO para repetir los datos enviados y recibidos en el monitor serial.

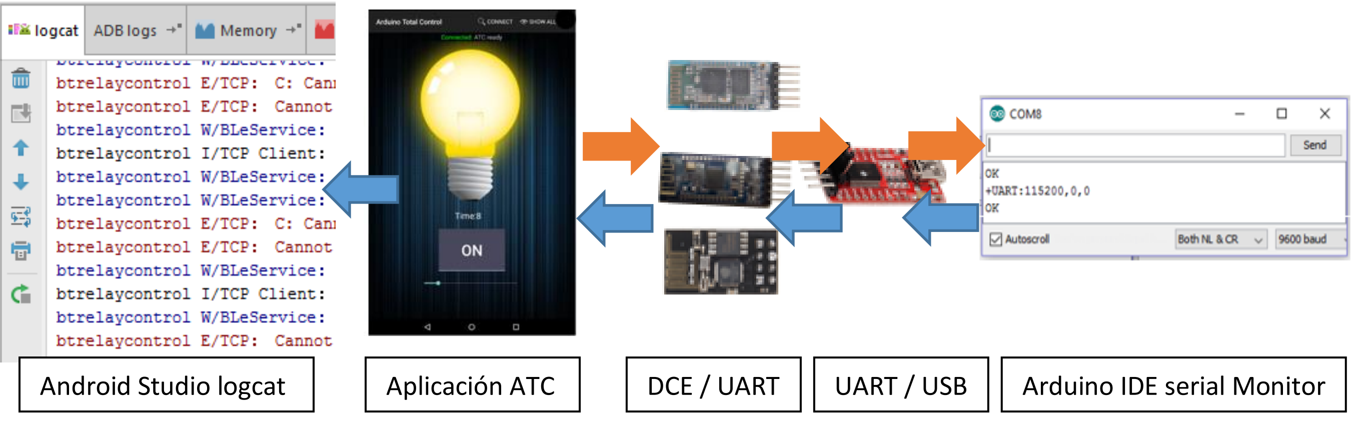


Figura 5.1. Esquema de circuito de pruebas de comunicaciones inalámbricas.

[3] A excepción de Arduino Yun, debido a que no se contaba con un circuito disponible.

Número	Nombre de la prueba	Descripción	Criterio de éxito	Resultado por tecnología				
				Bluetooth		BLE	Wi-Fi	
				HC05	HC06	HM10	Eth Shield	ESP8266
1	Configuración de tasa de transferencia	Utilizar los comandos AT para configurar el baud rate del DCE.	Se puede configurar el baud rate a 115200 bit/s	Exitoso	Exitoso	Exitoso	Exitoso	Exitoso
2	Visibilidad en red o scan	Utilizar la clase scan (Bluetooth o BLE) o la aplicación móvil de diagnóstico de red "fing" (en caso de variantes Wi-Fi) para identificar al DCE.	El DCE es visible; listado en el scan Bluetooth; o visible en red LAN.	Exitoso	Exitoso	Exitoso	Exitoso	Exitoso
3	Conexión	Ejecutar la conexión con cada módulo de comunicaciones inalámbricas.	El módulo DCE se conecta exitosamente.	Exitoso	Exitoso	Exitoso	Exitoso	Exitoso
4	Recepción de información en el DCE	Utilizar el monitor serial de Arduino IDE para mostrar, en la pantalla de una PC, los datos recibidos de la aplicación.	El DCE recibe los datos enviados por la aplicación	Exitoso	Exitoso	Exitoso	Exitoso	Exitoso
5	Recepción de información en aplicación ATC	Uso de la herramienta de presentación de mensajes de depuración, logcat (método Log.i), para mostrar la información recibida del DCE en Android Studio	La aplicación recibe los datos enviados por el DCE	Exitoso	Exitoso	Exitoso	Exitoso	Exitoso

Tabla 5.1. Pruebas de comunicaciones inalámbricas.

Configuración de tasa de transferencia

Bluetooth EDR/BR – HC05

Algunos módulos HC-05 tienen un *push button* integrado como se indica en rojo en la Figura 5.2; la función de éste es activar el modo de configuración, es decir, el modo en que el módulo puede recibir comandos AT. En módulos donde este botón no está presente (como en el HC06), el módulo está en modo de configuración siempre y cuando no esté conectado a un master Bluetooth. La Figura 5.3 muestra el resultado de enviar al HC05 el comando “AT” (después de presionar el *push button*). Los comandos AT para HC05 deben ser terminados por los caracteres de nueva línea y retorno de carro (‘\n’ y ‘\r’ respectivamente).

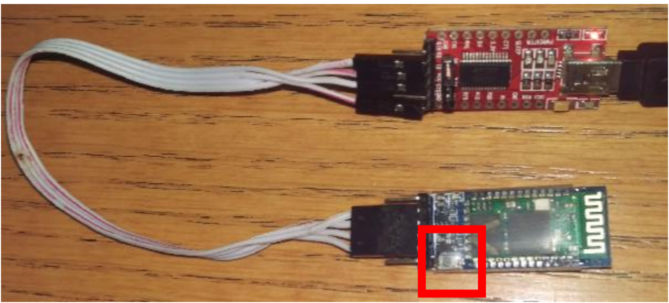


Figura 5.2. Conexión entre adaptador UART y HC05.

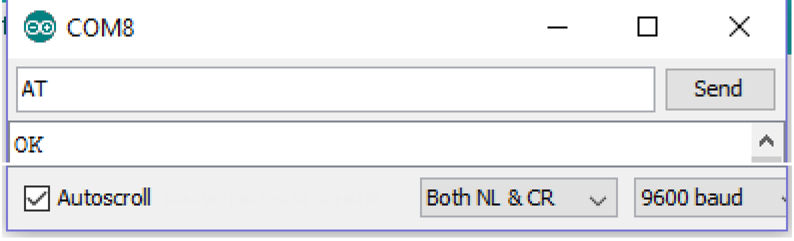


Figura 5.3. Respuesta a comando AT de módulo HC05.

Para configurar el baud rate de un módulo HC05 se utiliza el comando *AT+UART=<baud rate>, <parity bit>,<stop bits>.* Para asegurar la compatibilidad con ATC se recomienda un baud rate de 115200, sin bits de paridad (<parity bit>=0) y un bit de parada (<stop bits> = 0), como se muestra en la Figura 5.4a. Para terminar la configuración es necesario enviar un comando de reset (AT+RESET) como se muestra en la Figura 5.4b. La Figura 5.4c muestra el envío del comando "AT" a 115200 bits por segundo, es decir la configuración fue exitosa.

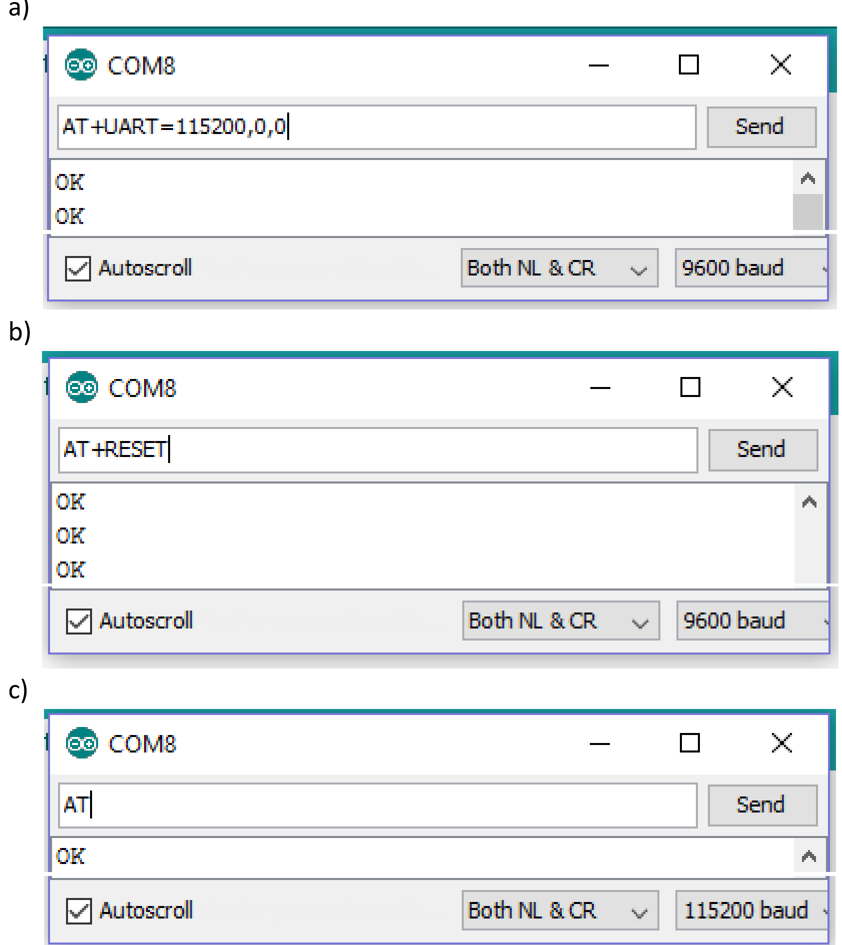


Figura 5.4. Configuración de baud rate de un módulo HC05.

Bluetooth EDR/BR – HC06

El uso de un módulo HC06 o HC05 es indistinto para la Aplicación y el MCU; sin embargo, a diferencia del módulo HC05, los comandos AT para el HC06 no requieren ser terminados por los caracteres '\n' y '\r'. Otra diferencia es el comando AT utilizado para configurar el baud rate; en HC06 se utiliza el comando AT+BAUDn, donde "n" es un número que representa un baud rate predefinido (ver apéndice C). El envío del comando AT+BAUD8 cambiará la configuración del baud rate a 115200 bit/s instantáneamente, es decir, sin necesidad de reiniciar el módulo como lo requiere el HC05. HC06 por defecto está configurado para tramas sin bits de paridad y con un bit de parada.

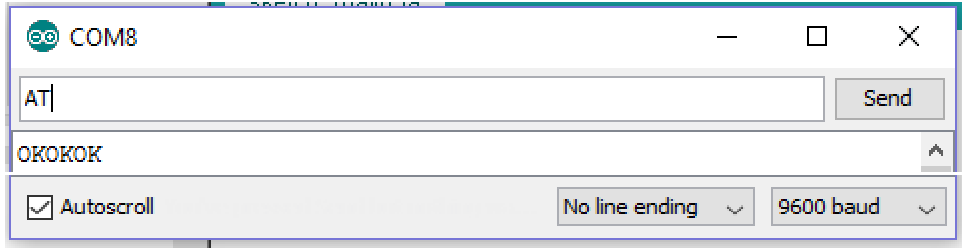


Figura 5.5. Envío de comando AT en HC06.

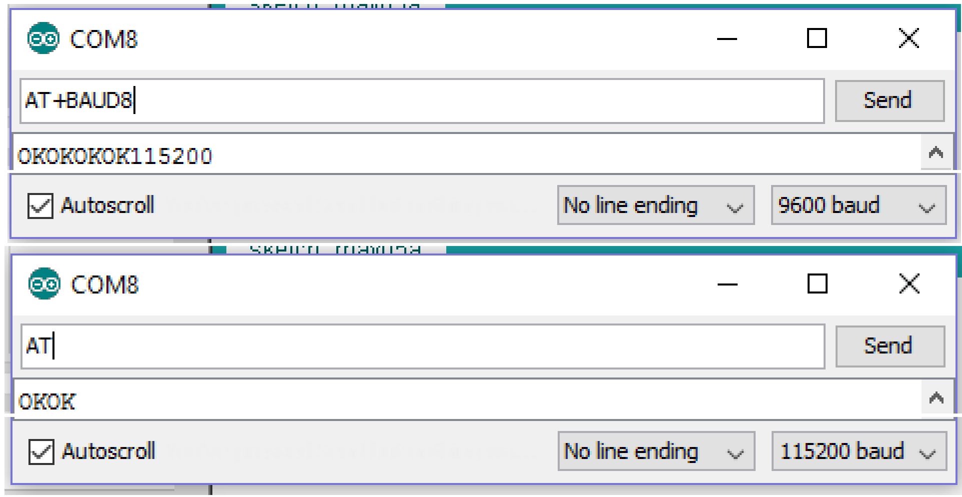


Figura 5.6. Configuración de baud rate de un módulo HC06.

Bluetooth LE – HM10

La configuración de un módulo HM10 es homóloga a la de un HC06, con la diferencia de que este módulo si requiere que el comando sea terminado por el carácter de nueva línea y retorno de carro. La configuración de un módulo HM10 se muestra en la Figura 5.7.

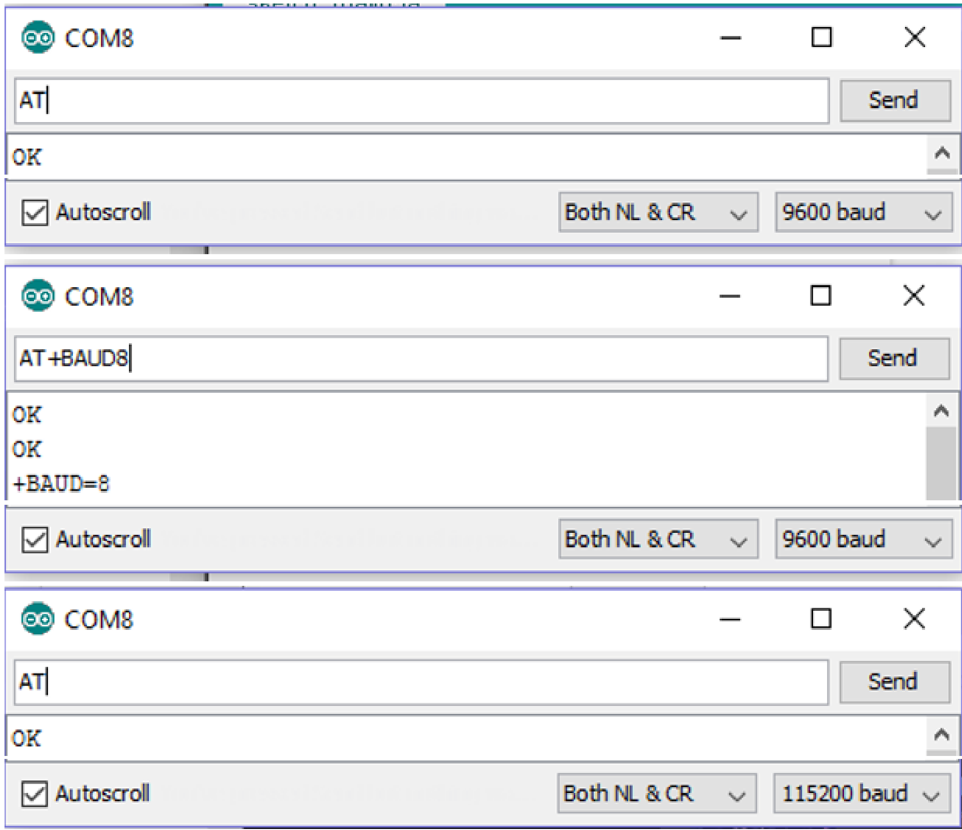


Figura 5.7. Configuración de baud rate de un módulo HM10.

Wi-Fi – ESP8266

Por defecto, el módulo ESP8266 está configurado a un baud rate de 115200 con cero bits de paridad y un bit de parada. Sin embargo, se puede usar el comando *AT+CIOBAUD=x*, donde x es cualquier baud rate (e. g. 115200, 9600, etc.) para aumentar o disminuir la tasa de transferencia como se muestra en la Figura 5.8.

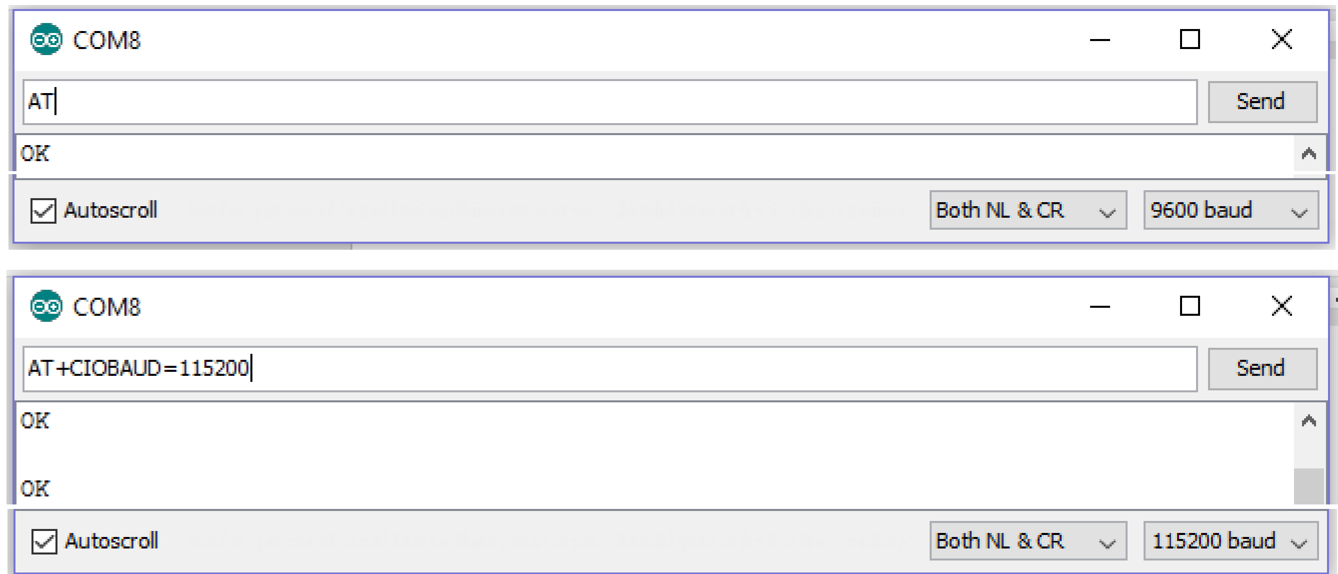


Figura 5.8. Configuración de baud rate de un módulo ESP8266.

Visibilidad y Conexión

Bluetooth EDR/BR (HC05 o HC06)

La Figura 5.9a muestra la actividad *BTScan*, exponiendo dispositivos emparejados previamente y descubriendo dispositivos disponibles en la red que cuenten con el UUID del perfil de puerto serial (ver capítulo 3). Las figuras 5.9b y 5.9c muestran la ejecución de la conexión cuando el usuario selecciona el HC-05 de la lista de la Figura 5.9a.

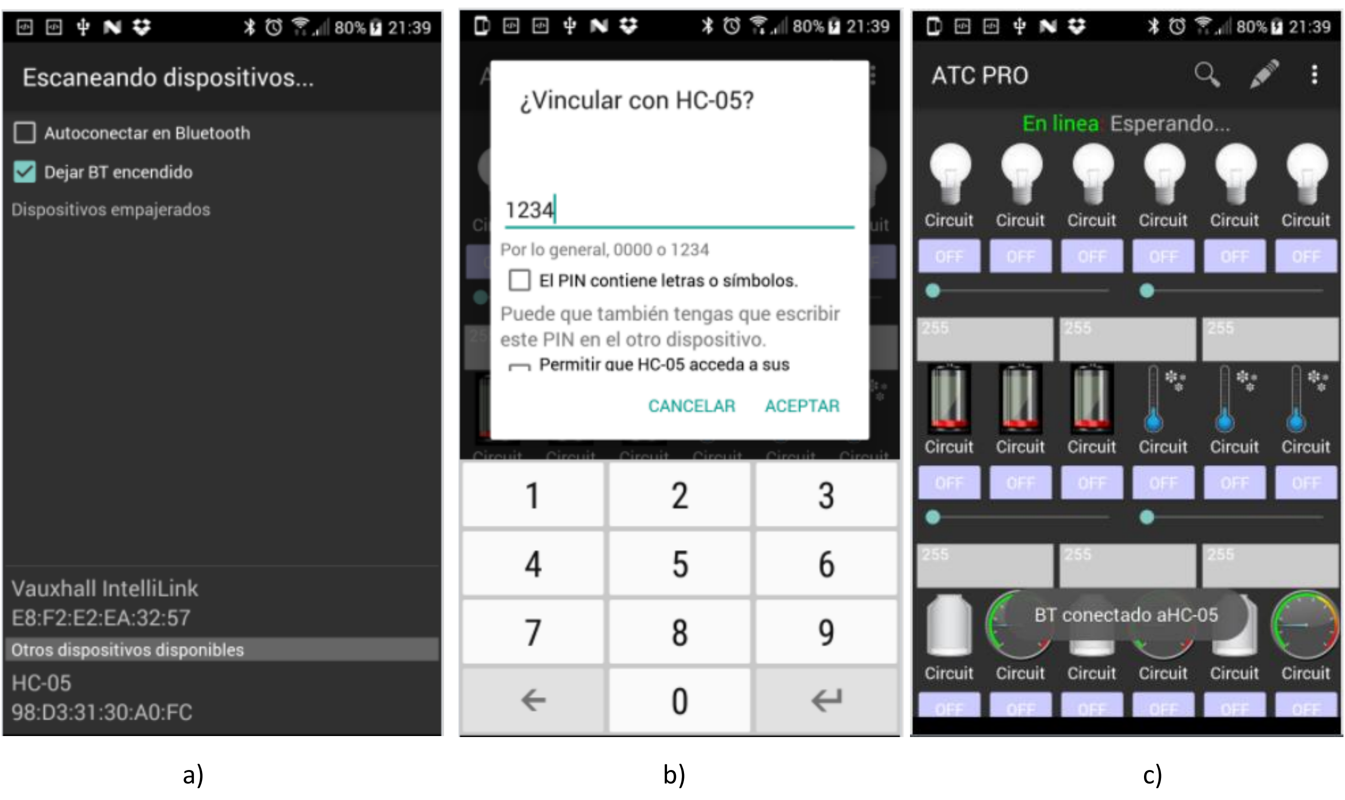


Figura 5.9. Proceso de conexión a módulo Bluetooth EDR/BR. a) Descubrimiento; b) Introducción de PIN de seguridad; c) Conexión.

Bluetooth LE (HM10)

La Figura 5.10a muestra la actividad BLEScan descubriendo dispositivos Bluetooth Low Energy cercanos. Al seleccionar el HM10 listado en (a), la aplicación muestra la lista en (b), donde el usuario debe elegir la característica donde se leerán y escribirán los datos para la comunicación entre el MCU y la aplicación (ver Capítulo 3). La figura 5.10c muestra la conexión exitosa con el HM10.

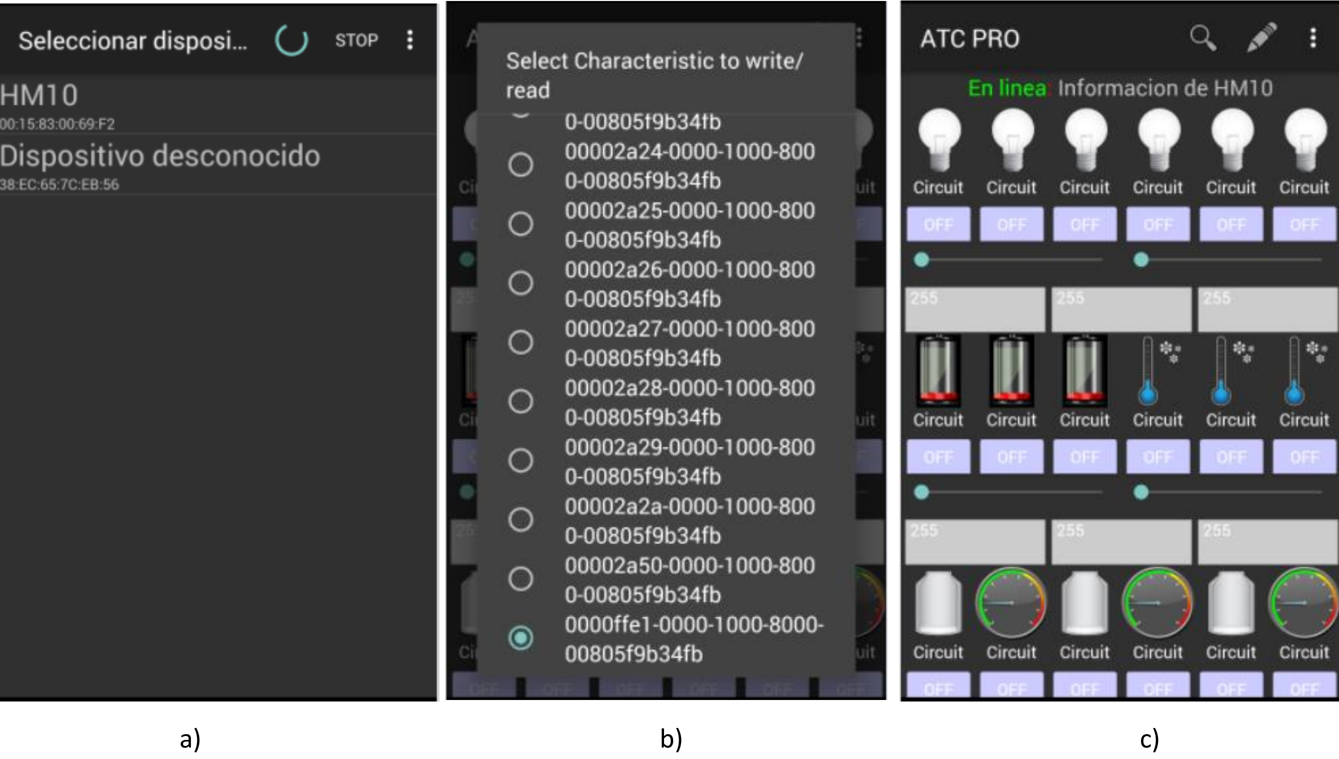


Figura 5.10. Proceso de conexión a módulo Bluetooth Low Energy. a) Descubrimiento, b) Selección de "característica", c) Conexión.

Wi-Fi (ESP8266 o Ethernet)

Para descubrir un dispositivo en una red 802.11 (Wi-Fi) se puede usar una aplicación de escáner de red. Para el desarrollo de la aplicación ATC se utilizó la herramienta de escáner de red, *Fing* [52]; la cual permite descubrir los dispositivos conectados a una red local usando el dispositivo móvil. Usar las funciones de *ipconfig* en la pantalla de comandos de una PC no es representativo, ya que el objetivo de esta prueba es verificar que el módulo Ethernet o Wi-Fi son visibles desde el dispositivo móvil que corre la aplicación ATC.

Figura 5.11. Logo de aplicación Fing [52].

Para la aplicación Arduino Total Control es indiferente si se utiliza un ESP8266 o un Ethernet Shield conectado a un router inalámbrico, ya que ambas opciones son compatibles con 802.11. La Figura 5.12a muestra la aplicación “Fing” escaneando la red; se descubre el Ethernet Shield (*Generic*) con la dirección IP 172.21.117.60; y el dispositivo móvil donde la aplicación está corriendo (HTC ONE M9), con la dirección IP 172.21.117.100. La Figura 5.12c muestra la conexión exitosa del Ethernet Shield con la aplicación.

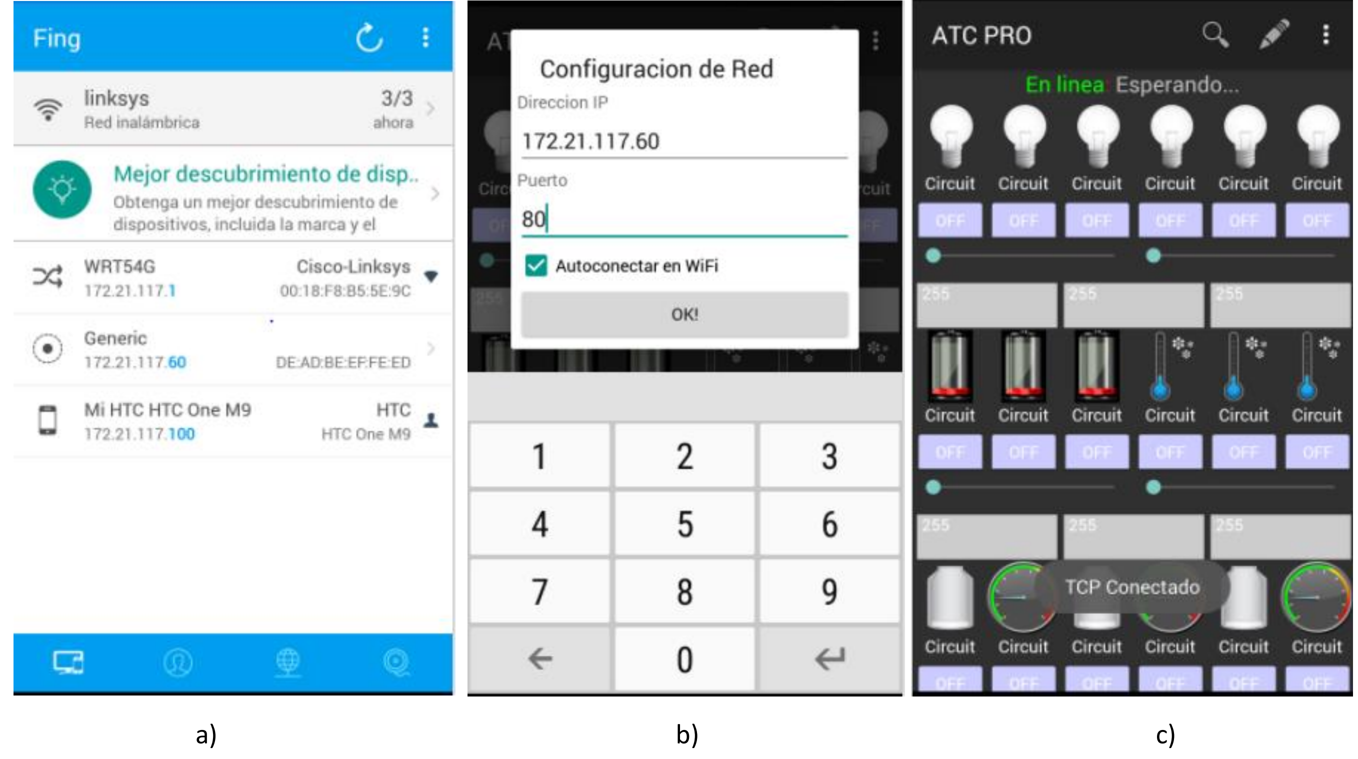


Figura 5.12. Proceso de conexión Wi-Fi. a) Descubrimiento usando aplicación *Fing*, b) Configuración de dirección IP y puerto, c) Conexión.

Envío y Recepción de Información

Bluetooth EDR/BR (HC05 y HC06)

Una vez iniciada la conexión, la aplicación empezará a enviar el comando ALIVE (‘[‘) cada 2.5s aproximadamente. La Figura 5.13a muestra cómo el DCE recibe información de la aplicación ATC. Las figuras 5.13b y 5.13c muestran cómo la aplicación recibe la información entrante del módulo Bluetooth en la interfaz de usuario y en *logcat* respectivamente.

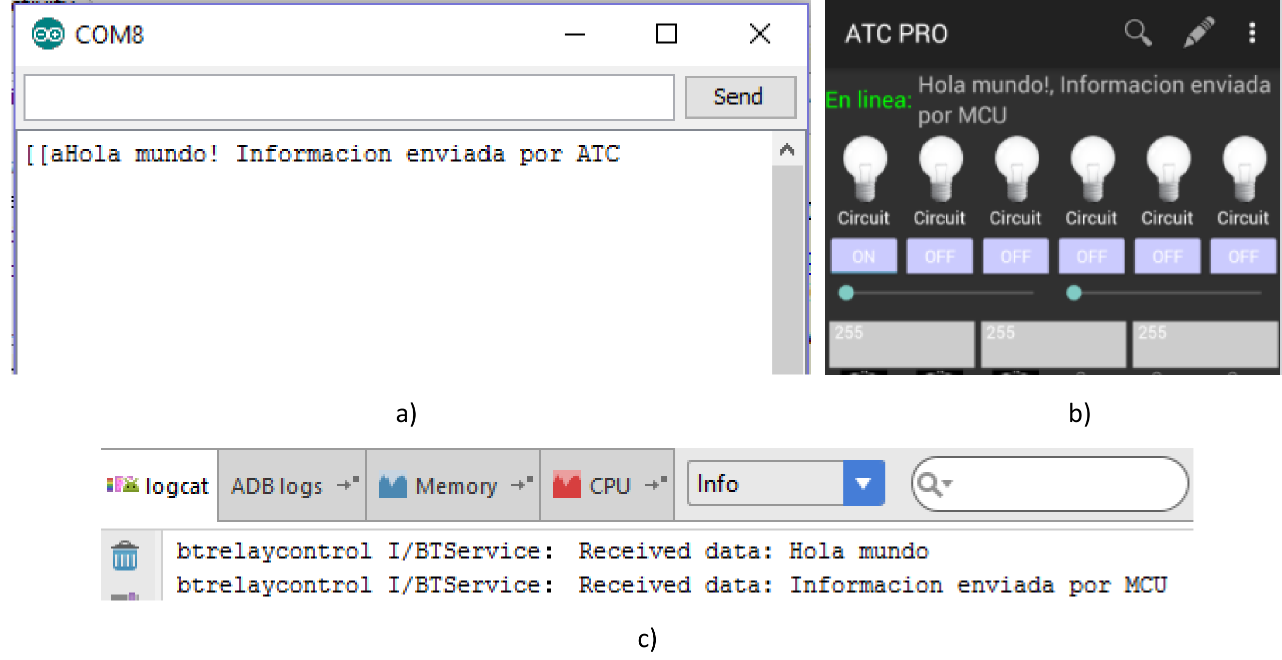


Figura 5.13. Comunicación bidireccional exitosa entre módulo HC05/HC06 y la aplicación.

Uso de Android logcat

Para usar la herramienta de línea de comandos *logcat* de Android Studio, basta con llamar la función Log.i (string, string). El primer argumento comúnmente es el nombre de la clase que genera el mensaje y el segundo es el mensaje a presentar en logcat. Para desplegar la información entrante a la aplicación en logcat, se llama la función Log.i antes de enviar la cadena recibida a Main, como se muestra en la Figura 5.14.

```
String st = new String(buffer, 0, index);
// Debug data
if (D) Log.i(TAG, "Received data: " + st);
// Send the obtained string to the UI Activity
mHandler.obtainMessage(Main.MESSAGE_READ, 5, -1, st).sendToTarget();
```

Figura 5.14. Ejemplo del uso de Android logcat para mostrar la información entrante en la aplicación.

Bluetooth LE (HM10)

La Figura 5.15 muestra la comunicación bidireccional entre el módulo HM10 y la aplicación. Sin embargo, durante las pruebas de funcionamiento se observa que la información recibida de la aplicación en el módulo HM10 está limitada a 20 caracteres y no está seccionada, como se muestra en la Figura 5.15a. En otras palabras, la parte faltante del mensaje "Hola mundo! Informac**ión enviada por ATC**" se ha perdido.

La información entrante a la aplicación también está limitada a 20 caracteres, pero sí está seccionada. Como se muestra en la Figura 5.15b, la parte faltante del mensaje es mostrada en el paquete siguiente.

Además, se observó durante las pruebas que el módulo HM10 inserta de forma automática el carácter de fin de línea tanto en la información de entrada como de salida.

Los puntos anteriores no representan un conflicto con el protocolo de Capa de Aplicación ATC, pero sí limitan la cantidad de información que puede ser enviada por un Tag. Los comandos más afectados por la limitante de 20 caracteres son Text to Speech Tag y Speech Recognizer Tag.

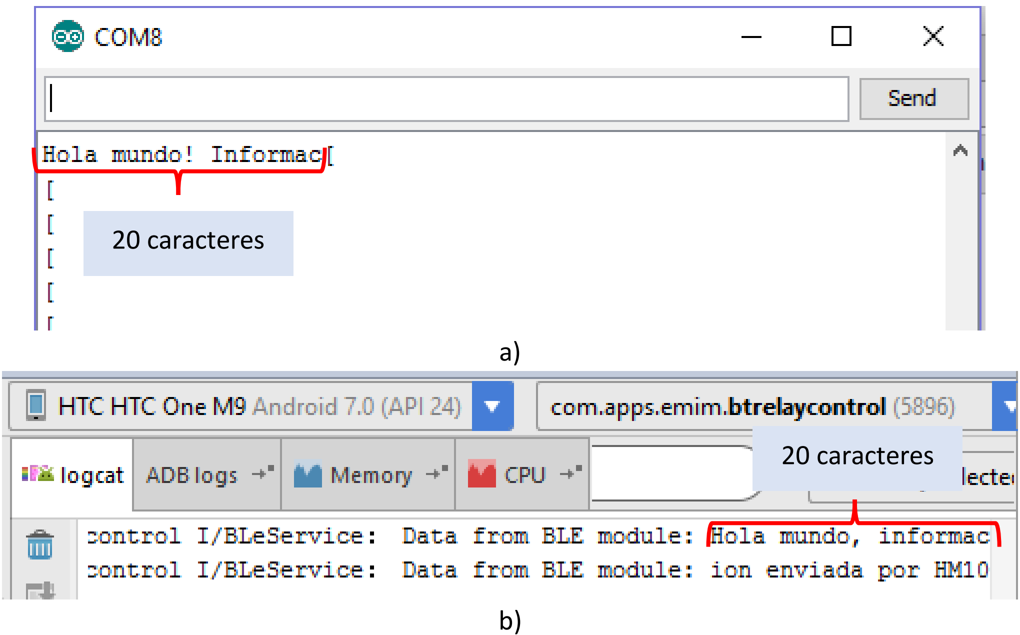


Figura 5.15. Comunicación bidireccional exitosa entre módulo HM10 y la aplicación.

Wi-Fi (Ethernet Shield)

Una vez establecida la conexión TC/IP, la comunicación entre el servidor (MCU + Ethernet Shield) y el cliente (la aplicación) se comporta como un puerto serial donde la información es transferida de forma directa (sin adicionar prefijos o sufijos a las cadenas de texto). La Figura 5.16 muestra la comunicación entre el Ethernet Shield y la aplicación.

Para mostrar los datos en el monitor serial de Arduino IDE se desarrolló un código de pruebas que repite la información de entrada y de salida (ver anexo 5, sketch ethernet_shield_test.ino).

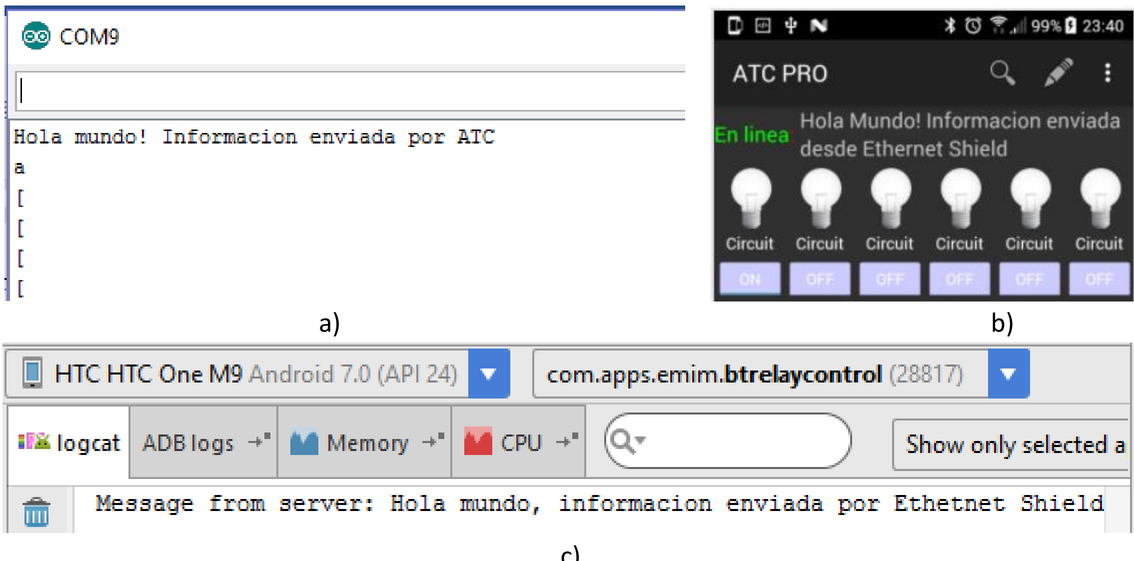


Figura 5.16. Comunicación bidireccional exitosa entre Ethernet Shield y la aplicación.

Wi-Fi (ESP8266)

Como se vio en el capítulo 4, el módulo ESP8266 agrega el sufijo +IPD,<canal>,<tamaño>):<cadena de texto> a la información proveniente de la aplicación antes de enviarla por el UART al MCU. La Figura 5.17 muestra la recepción de una cadena de texto y de un carácter (comando *alive*) proveniente de la aplicación.

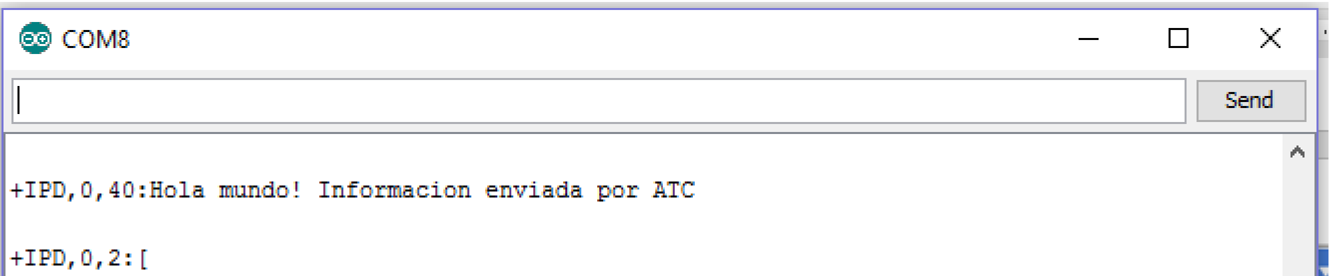
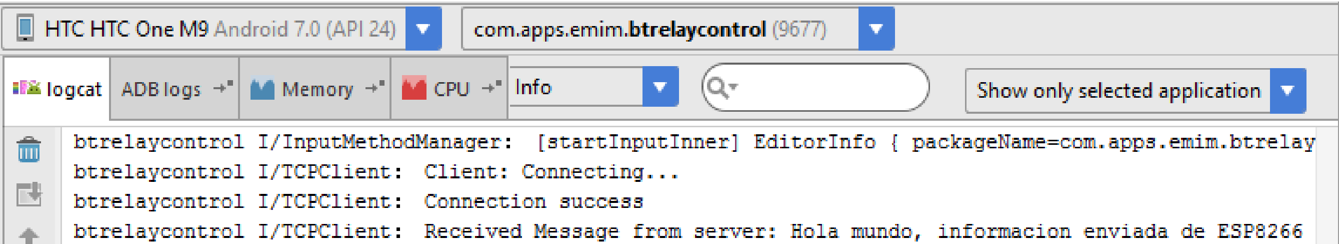


Figura 5.17. Recepción de datos de la aplicación usando ESP8266.

A diferencia de los módulos Bluetooth y Ethernet anteriormente mencionados, para enviar información a la aplicación desde un módulo ESP8266 es necesario notificar al módulo que se va a enviar información indicando el canal destino, así como el tamaño de la cadena. Como se vio en el capítulo 4, se utiliza el comando AT+CIPSEND=<canal>,<tamaño> para indicar que la siguiente cadena deberá ser transmitida a la aplicación (cliente TCP/IP).

Para lograr el resultado de la Figura 5.18a; el microcontrolador primero deberá enviar el comando CIPSEND como lo muestra la Figura 5.18b, seguido de la carga útil como se muestra en la Figura 5.18c.



a)

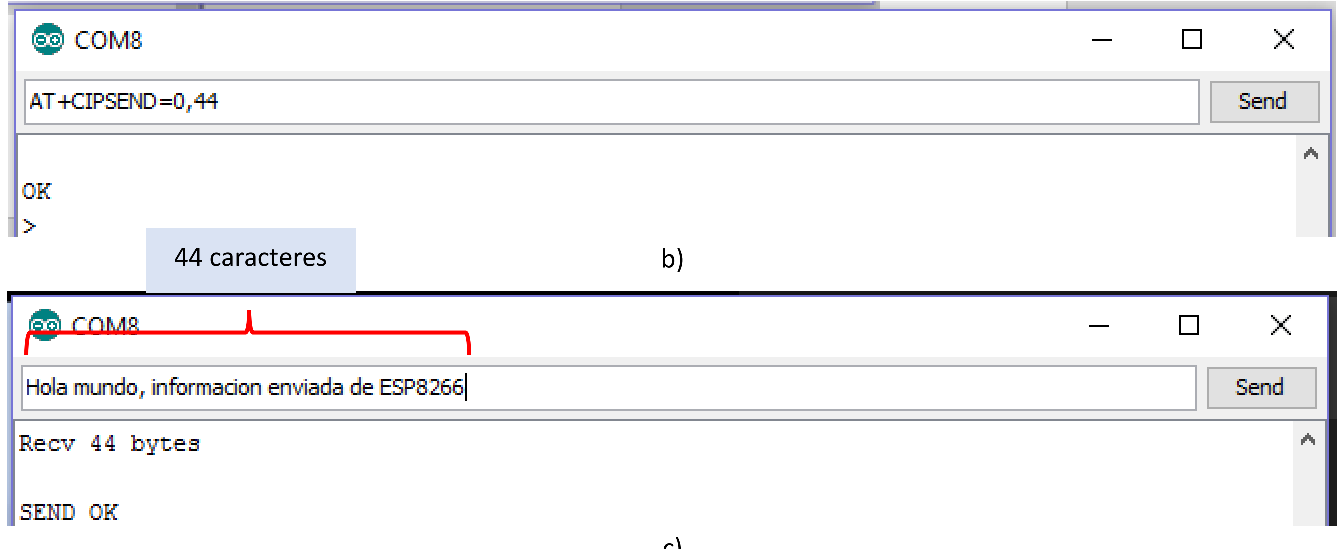


Figura 5.18. Envío de datos del módulo ESP8266 a la aplicación.

Pruebas de protocolo de Capa de Aplicación ATC

Las pruebas de protocolo de Capa de Aplicación ATC, solo se aplicarán en un Arduino equipado con una tarjeta Ethernet y se listan en la Tabla 5.2, bajo la premisa de que las funciones de Capa de Aplicación son las mismas para todos los DCE soportados.

Se eligió el Shield Ethernet ya que al comunicarse por SPI con la tarjeta Arduino se tiene disponible el puerto serial; dando la posibilidad de repetir la información recibida y enviada en el monitor serial de Arduino IDE, con motivos de depuración. El MCU de prueba será cargado con una versión modificada del programa eth_firmware.ino para el mismo propósito. El sketch usado para las siguientes pruebas puede ser encontrado en el anexo 6.

Comando Alive

Cuando el MCU recibe el comando alive (carácter '[', recibido cada 2.5s aproximadamente) y responde con cualquier cadena de texto ("ATC ready" para este ejemplo), la conexión se mantiene. Cuando la respuesta del MCU se deshabilita, la aplicación termina la conexión.

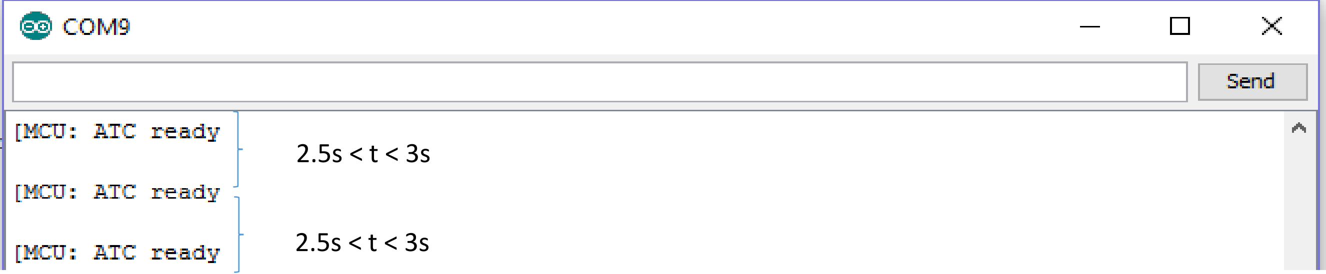


Figura 5.19. Recepción del comando "alive".

Raw data

Al enviar información tipo raw del MCU a la aplicación, ésta es presentada en la parte superior de la aplicación.

a) Envío de raw data desde el sketch b) Resultado de envió de raw data en aplicación

Figura 5.20. envío de la cadena raw data "Hola Mundo" a la aplicación.

Número	Nombre de la prueba	Descripción	Criterios de éxito	Resultado
Verificación del comando "alive"				
1	Comando Alive	El microcontrolador recibe el comando alive ('[') aproximadamente cada 2.5s. Si no hay respuesta del microcontrolador después de 3 intentos, la aplicación terminará la conexión.	El MCU recibe el comando ALIVE. La aplicación termina la conexión si no hay respuesta del MCU. La conexión continúa si el microcontrolador envía cualquier mensaje a la APP al recibir el comando.	Exitoso
Verificación de "raw data"				
2	Envió de raw data: MCU -> Aplicación	El MCU envía la cadena "Hola mundo" a la aplicación.	La cadena "Hola mundo" aparece en la parte superior de la APP.	Exitoso
3	Activación de Toggle Buttons	Al activar un Toggle Button en la aplicación, el MCU recibe el comando asignado, ya sea carácter o cadena de texto.	El MCU recibe el comando del Toggle Button.	Exitoso
4	Activación de Temporary Buttons	Al tocar una imagen habilitada como botón, la aplicación envía un comando (carácter o cadena de texto) "on"; al liberar la imagen, la aplicación envía un comando "off".	El MCU recibe el comando del Temporary Button.	Exitoso
5	Activación de Long Touch Button	Al tocar una imagen habilitada como Long Touch Button por aproximadamente 2 segundos, la aplicación envía un comando (carácter o cadena de texto) "on"; al liberar la imagen, la aplicación envía un comando "off".	El MCU recibe el comando del Long Touch Button.	Exitoso
Verificación de "User Interface Tags"				
6	Uso de Text Tags, Información fija	El microcontrolador usa Text Tags para mostrar la cadena "Hola mundo" en el texto 00.	El texto 00 cambia su contenido a "Hola mundo"	Exitoso
7	Uso de Text Tags, información variable	El microcontrolador usa Text Tags para mostrar el valor del Convertidor Analógico a Digital (ADC) A1 en el texto 01.	El valor del ADC A1 es mostrado en el texto 01 continuamente.	Exitoso
8	Uso Button Tags	El microcontrolador controla el estado de los botones ("on" o "off") usando button tags.	El estado de los Toggle Button en la App cambia de acuerdo a los comandos enviados por el MCU.	Exitoso
9	Uso Image Tags	El microcontrolador usa Image Tags para cambiar el estado de la imagen 01 (Default, Pressed y Extra) cada segundo.	El estado de la imagen cambia cada segundo de acuerdo a comandos enviados por el MCU.	Exitoso
10	Uso de Seekbar Tags	El microcontrolador controla el nivel o valor de una barra de acción enviando Seekbar Tags.	El valor de la barra de acción cambia de acuerdo a los comandos enviados por el MCU	Exitoso
11	Uso de Alarm Tags	Al enviar un Alarm Tag, el microcontrolador controla la reproducción de un sonido de alarma en la aplicación.	La aplicación emite un sonido de "beep" al recibir un Alarm Tag.	Exitoso
12	Uso de Vibrator Tags	El MCU controla la activación del vibrador del dispositivo así como la duración de la vibración usando Vibrator Tags.	El dispositivo móvil vibra por el tiempo designado por el microcontrolador al recibir un Vibrator tag.	Exitoso
13	Uso de Text To Speech Tags	Al recibir un Text to Speech tag, la aplicación "habla", es decir lee la cadena de texto enviada.	La aplicación pronuncia "hola mundo" cuando el tag "<TtoS01:Hola mundo" es recibido.	Exitoso
14	Uso de Analog Bar Tags	El microcontrolador controla el nivel o valor de una barra analógica	El valor de la barra analógica cambia de acuerdo a los comandos enviados por el microcontrolador	Exitoso
Verificación de Special Commands				
15	Activación de Seekbars	Al deslizar una seekbar la aplicación envía de forma continua el valor de la misma al MCU usando Seekbar Tags.	El microcontrolador extrae la información de la barra de acción correspondiente del Seekbar Tag.	Exitoso
16	Uso de Accelerometer Tags	Habilitar 3 textos como fuentes de datos de acelerómetro.	Cada texto muestra y envía tags de acelerómetro en "X", "Y" y "Z".	Exitoso
17		El microcontrolador recibe de forma continua información de acelerómetro en los tres ejes.	El microcontrolador extrae la información del Accelerometer Tag.	Exitoso
18	Uso de Touchpad Tags	Al tocar una imagen habilitada como touchpad, la aplicación envía la coordenada en el eje "X" y "Y" del toque, relativo al Touch Pad.	El microcontrolador extrae la información del Touchpad Tag.	Exitoso
19	Uso de Speech Recognizer Tags	Al tocar una imagen habilitada como Reconocimiento de Voz, la aplicación inicia la actividad de reconocimiento de voz, sintetiza el comando del usuario y lo envía en forma de texto al MCU.	El microcontrolador extrae la información del Speech Recognizer Tag.	Exitoso
Verificación de tags de registro				
20	Uso de tags de registro	El MCU envía tags de registro para mantener un historial de las entradas y salidas activadas durante las pruebas.	Al abrir la función ATC Logger en la aplicación, la información enviada con Log Tags esta registrada.	Exitoso

Tabla 5.2. Pruebas de protocolo de Capa de Aplicación ATC.

Activación de Toggle Buttons

Además de la herramienta del monitor serial, para verificar la funcionalidad de los botones de activación, se definió el Pin 6 como salida y se conectó un LED como indicador. Además, se programó en el sketch los casos necesarios para que el Pin 6 esté en estado alto cuando el MCU reciba un carácter ‘A’ y en estado bajo cuando éste reciba un carácter ‘a’, como se puede observar en la Figura 5.21.

```
switch(appData){
case 'A':
digitalWrite(6, HIGH);
server.println("<Logr00: Pin 6 HIGH");
break;

case 'a':
digitalWrite(6, LOW);
server.println("<Logr00: Pin 6 LOW");
break;
```

Figura 5.21. Activación de un LED usando raw data enviada por activación de botones.

Las figuras 5.22a y 5.22b muestran el resultado de esta prueba en el mundo físico cuando el Toggle Button está en estado “ON” y “OFF” respectivamente. La Figura 5.22c muestra el resultado en el monitor serial.

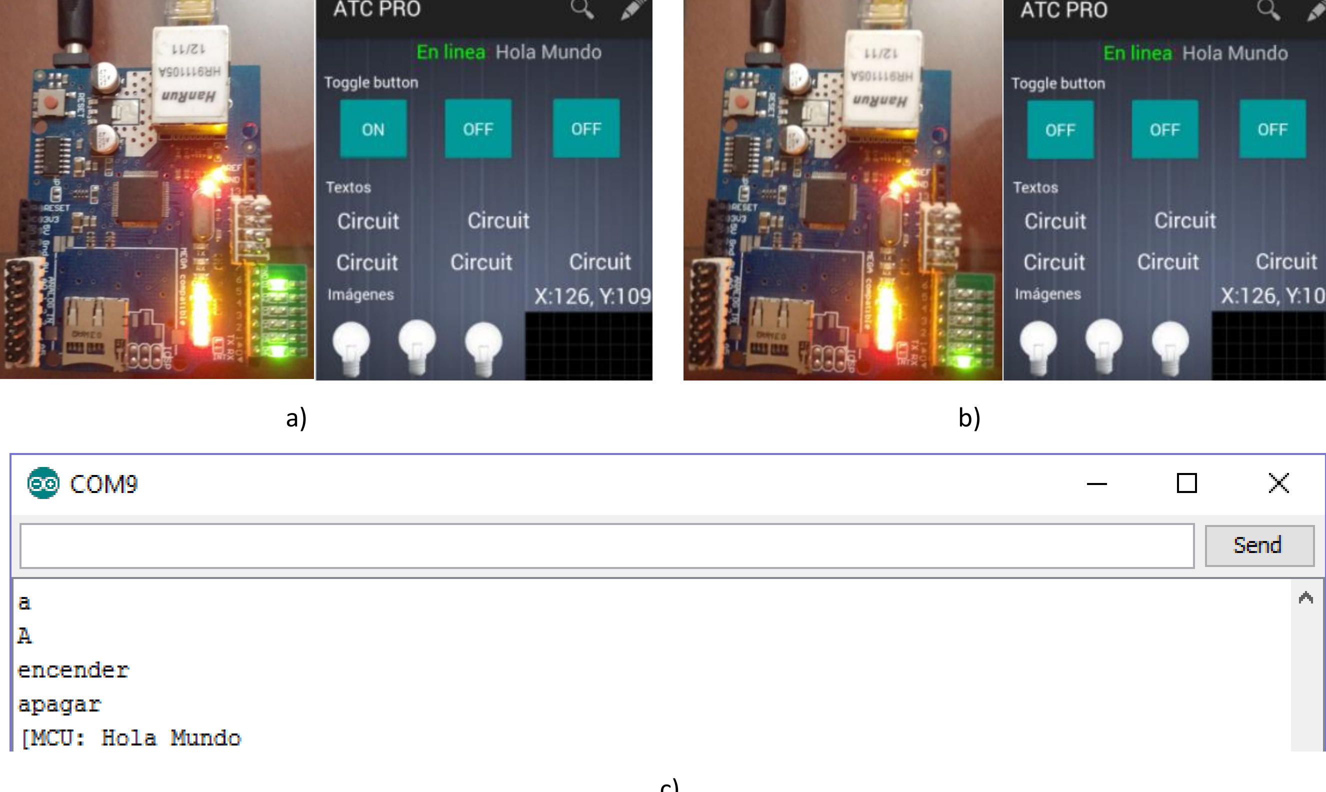


Figura 5.22. Activación de un Toggle Button; a) Encendido “ON”; b) apagado “OFF”; c) comandos recibidos en monitor serial.

Las cadenas de texto “encender” y “apagar” son enviadas por el botón 1, mientras que los caracteres ’a’ y ‘A’ son enviados por el botón 0. Los comandos a enviar en “ON” y “OFF” se definen en las propiedades de cada objeto como muestra la Figura 5.23.

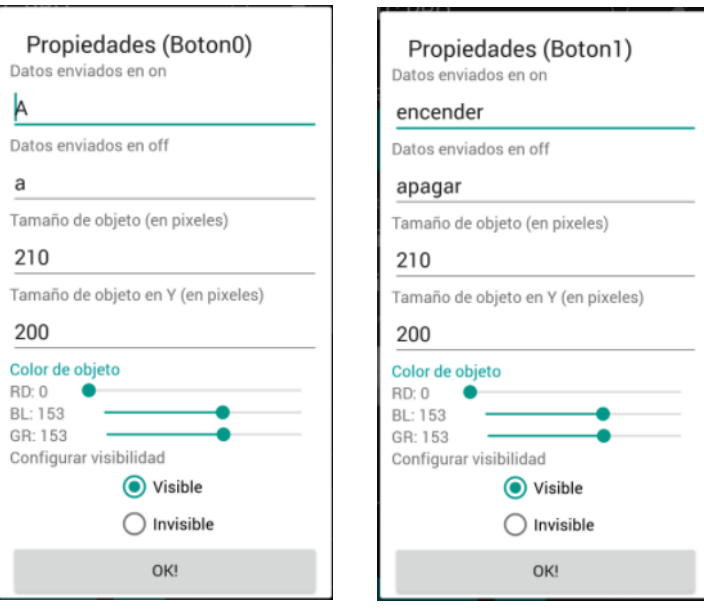
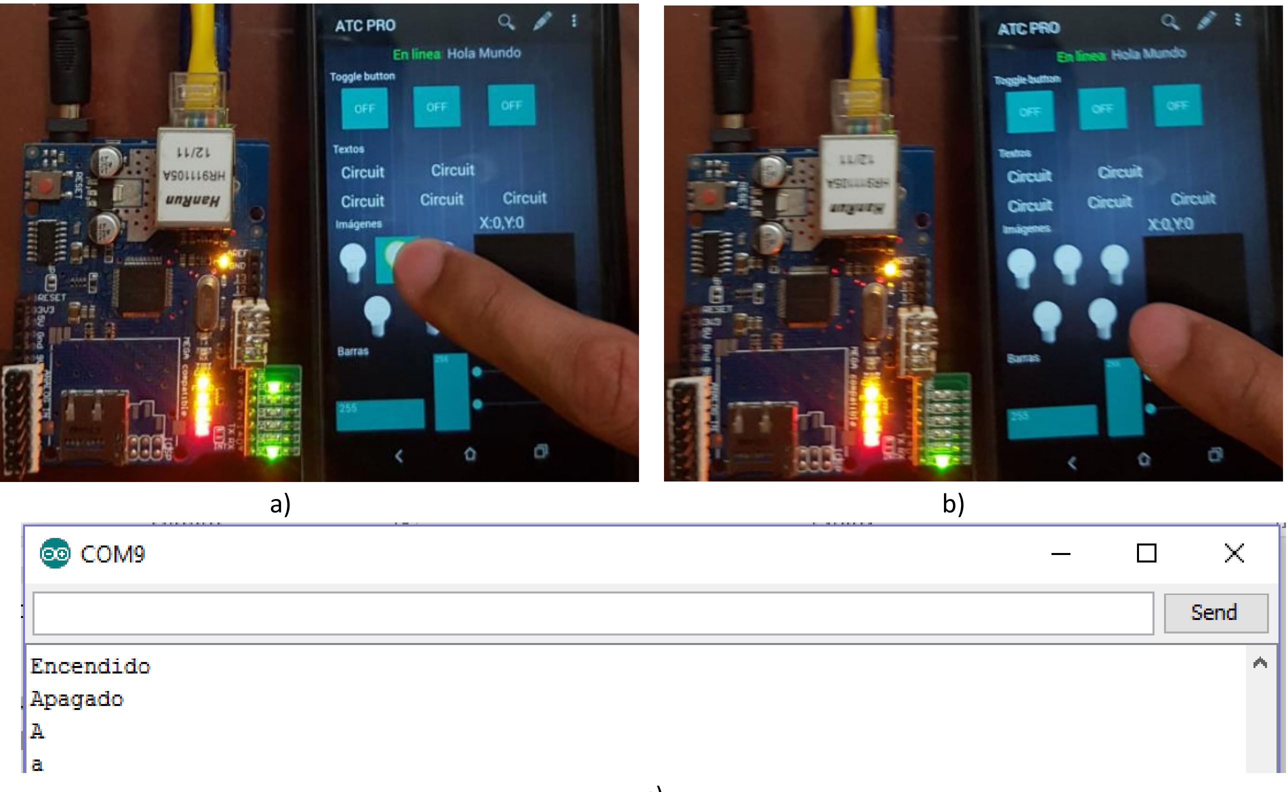


Figura 5.23. Actividad EProp (editar propiedades) para los botones 0 y 1.

Activación de Temporary Buttons

Una imagen habilitada como un Temporary button se comporta como un *push button*, es decir, se envía un comando "ON" al tocar el botón, y otro comando "OFF" al retirar el toque del botón. La Figura 5.25 muestra las propiedades de una imagen habilitada como Temporary Button.

Al igual que en los Toggle Button, se habilita el pin 6 como una salida digital para realizar esta prueba. Las figuras 5.24a y 5.24b muestran el resultado en el mundo físico cuando la imagen es "tocada"; y cuando se deja de tocar, respectivamente.



c)

Figura 5.24. Activación de un Temporary Button; a) Encendido "ON"; b) Apagado "OFF"; c) Comandos recibidos en monitor serial.

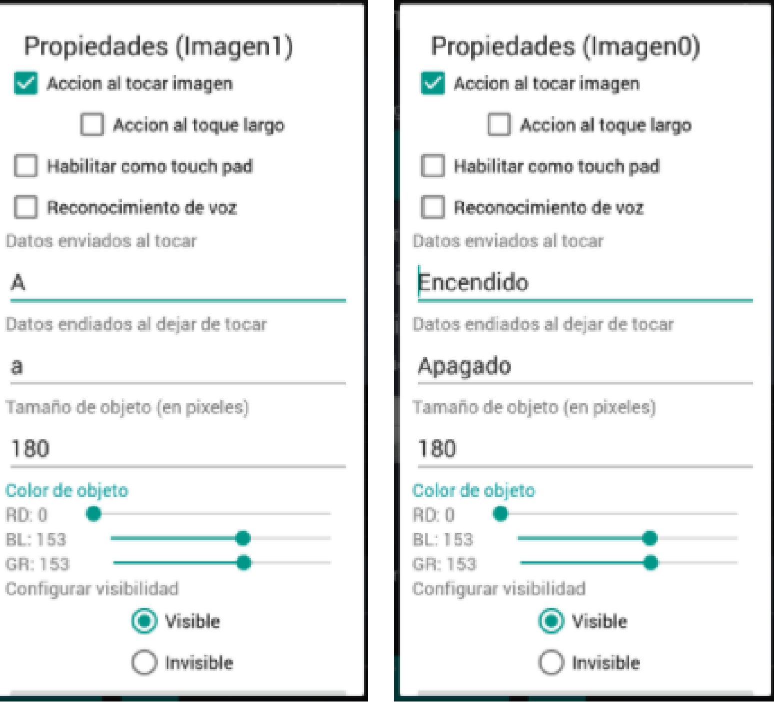


Figura 5.25. Propiedades de una imagen habilitada como botón temporal. Imagen que envía caracteres (derecha) e imagen que envía cadenas de texto (izquierda).

Activación de Long Touch Buttons

El uso de imágenes habilitadas como botones de activación retardada o *Long Touch Buttons* es similar al uso de botones temporales; la diferencia radica en que un Long Touch Button tiene seleccionada la opción “acción al toque largo” como se muestra en la Figura 5.25.

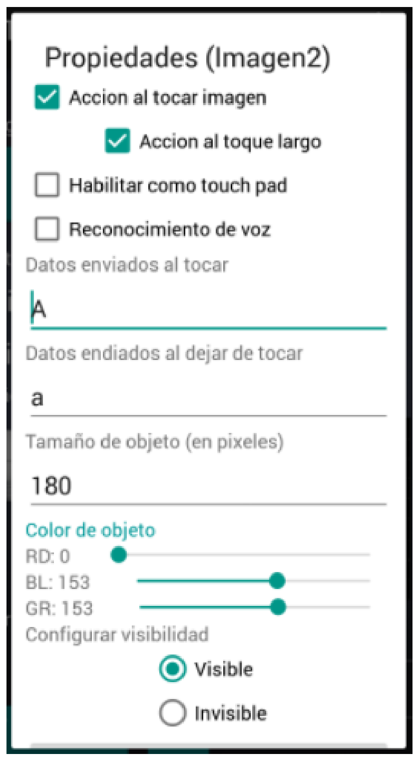


Figura 5.26. Configuración de una imagen como Long Touch Button.

La activación de un Long Touch Button se describe en la Figura 5.26.

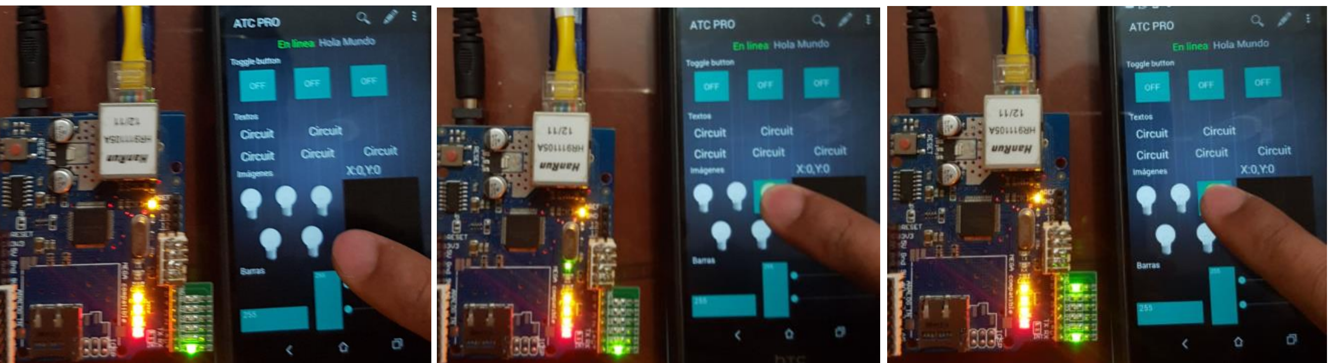


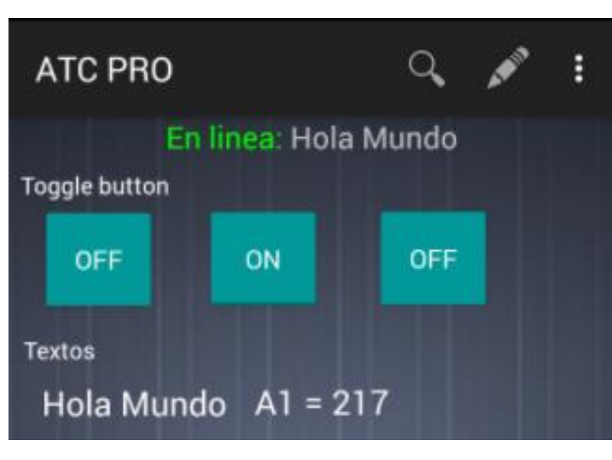
Figura 5.26. Activación de un Long Touch Button. a) Al liberar se envía un comando "OFF" ('a' para este ejemplo); b) No se envía ningún comando al toque; c) Después de 1s aproximadamente, se envía un comando "ON" ('A' en este ejemplo).

Uso de Text Tags

La Figura 5.27b muestra el resultado del uso de los Text Tags enviados por el MCU en (a).

```
server.println("<Text00:Hola Mundo");
server.println("<Text01: A1 = " + String(analogRead(A1)));
```

a)



b)

Figura 5.27. Uso de Text tags.

Uso de Button Tags

La Figura 5.28 muestra el control del estado de los Toggle Button usando Button Tags. Para probar esta característica, el Button tag se insertó en la sección de código que se ejecuta cuando los caracteres ‘A’ y ‘a’ son recibidos, para encender y apagar el botón 02 respectivamente.

Figura 5.28. Uso de Button Tags.

Uso de Image Tags

De la misma forma que se ejecutó en los Button Tags, la Figura 5.29 muestra el control del estado de las Imágenes usando *Image Tags*.

```
server.println("<Imgs03:1");          server.println("<Imgs03:0");
```

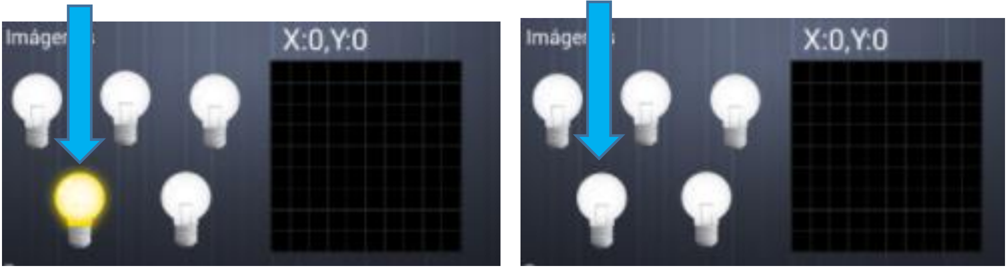


Figura 5.29. Uso de *Image Tags*.

Uso de Seekbar Tags

La Figura 5.30 muestra el control del estado de la barra de acción 0, usando Seekbar Tags. Para esta prueba se envía el valor del convertidor analógico digital (recorrido dos bits, ya que es un ADC de 10bits y la Seekbar es de 8bits) como el valor de la Seekbar 0.

```
server.println("<Skb0:" + String(myIntToString(analogRead(A1)>>2)));
```

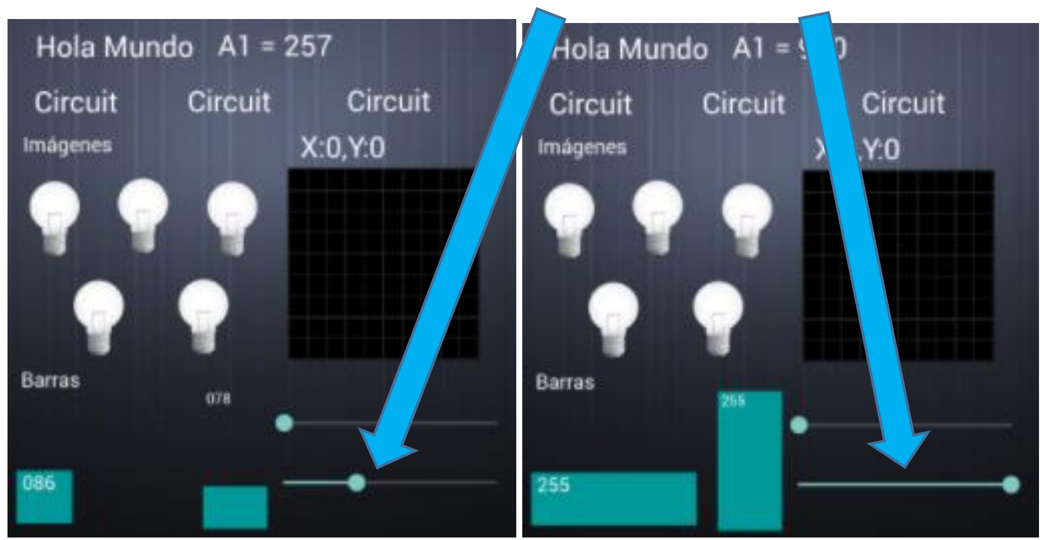


Figura 5.30. Uso de Seekbar Tags.

Uso de Alarm y Vibrator Tags

Para probar estas funciones de ATC, se envía un comando de vibración y un comando de alarma cada vez que el MCU recibe un comando *alive*. El uso de estos comandos se muestra en la Figura 5.31. El dispositivo móvil vibra y emite la alarma exitosamente al recibirlos.

```
server.println("<Alrm00");
server.println("<Vibr00:100");
```

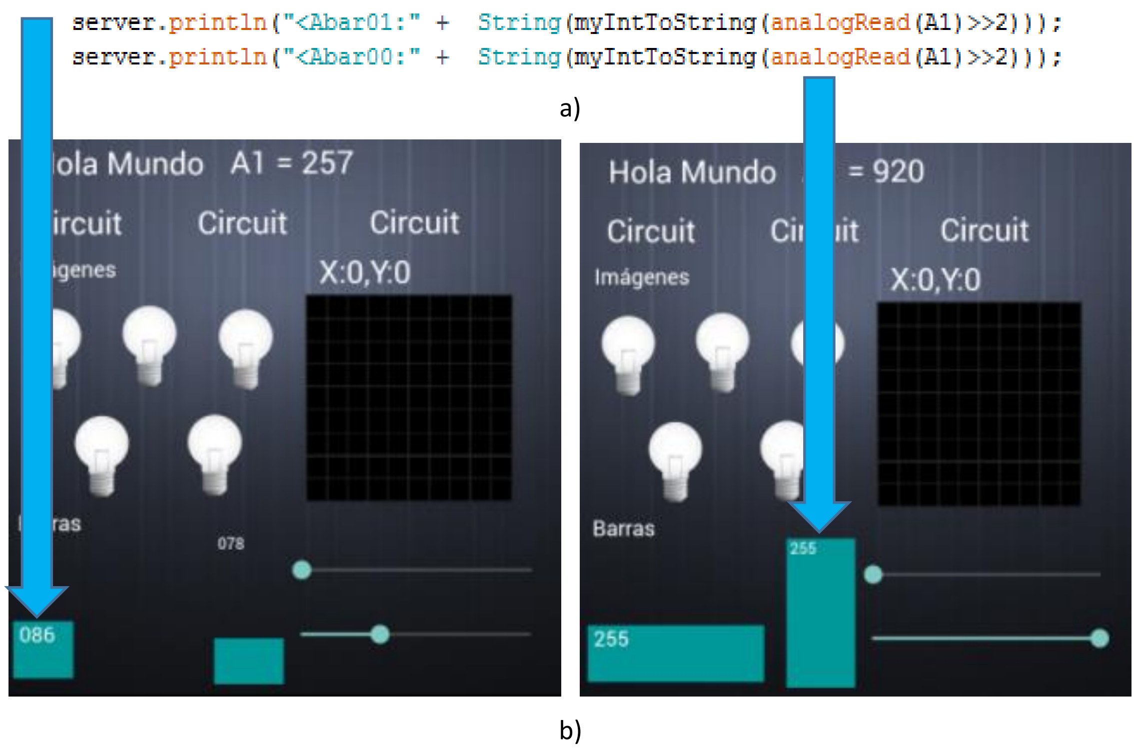
Figura 5.31. Uso de Alarm y Vibrator Tags.

Uso de Text to Speech Tags

Se utiliza el código *server.println("<TtoS01:Hola mundo");* para que la aplicación pronuncie el saludo "hola mundo". Para probar esta característica, esta línea es incluida en la sección de código que se ejecuta cuando el MCU recibe un comando alive.

Uso de Analog Bar Tags

La Figura 5.32b muestra dos tomas de pantalla de dos instantes diferentes en los que se usan Analog Bar Tags para controlar el estado de una barra analógica vertical y una horizontal. El valor de las barras analógicas es "atado" al convertidor analógico digital A1, como se muestra en la Figura 5.32a. El valor del ADC es recorrido 2 veces, debido a que éste es de 10 bits, y las barras analógicas operan en un rango de 0 a 255 (8 bits).



b)

Figura 5.32. Uso de Analog Bar Tags.

Uso de Seekbar Tags

Se utilizó el monitor serial para imprimir el valor extraído de cada Seekbar Tag. Este valor es obtenido del arreglo de variables enteras, *SeekBarValue*, la cual es actualizada por la función *DecodeSpecialCommand* y es directamente utilizable por el microcontrolador, como se muestra en la Figura 5.33a. La Figura 5.33b muestra el valor extraído de un Seekbar Tag entrante.

```
if(TagType == UPDATE_SKBAR)
// Print seekbar array
for(int i = 0; i < SKBAR_NO; i++){
  Serial.println("MCU: Seekvar value: " + String(i) + ": " + String(SeekBarValue[i]));
}
```

a)

```
<Skb1:+00103
MCU: Seekvar value: 0: 0
MCU: Seekvar value: 1: 103
MCU: Seekvar value: 2: 0
MCU: Seekvar value: 3: 0
MCU: Seekvar value: 4: 0
MCU: Seekvar value: 5: 0
```

```
<Skb0:+00167
MCU: Seekvar value: 0: 167
MCU: Seekvar value: 1: 103
MCU: Seekvar value: 2: 0
MCU: Seekvar value: 3: 0
MCU: Seekvar value: 4: 0
MCU: Seekvar value: 5: 0
```

b)

Figura 5.33. Impresión de carga útil de Seekbar Tags.

Accelerometer Tags

La configuración de un texto como acelerómetro se muestra en la Figura 5.34a y el resultado de configurar tres textos para que envíen datos de acelerómetro al MCU se muestra en la Figura 5.34b.

a) b)

Figura 5.34. Configuración de textos como fuente de acelerómetro.

Homólogo a las Seekbar Tags, se utilizó el monitor serial para imprimir el valor del acelerómetro de cada Accelerometer Tag entrante, como se muestra en las figuras 5.35a y 5.35b.

```
if(TagType == UPDATE_ACCEL)
// Print accel array
for(int i = 0; i < ACCEL_AXIS; i++){
  Serial.println("MCU: Accelerometer value " + String(i) + ": "  + String(Accel[i]));
}
```

a) Sección de código usada para imprimir el valor del acelerómetro en cada eje.

```
<AccY:+00801                          <AccY:+01217
MCU: Accelerometer value 0: -642      MCU: Accelerometer value 0: -22
MCU: Accelerometer value 1: 801       MCU: Accelerometer value 1: 1217
MCU: Accelerometer value 2: -391      MCU: Accelerometer value 2: -75
<AccZ:-00075                          <AccZ:+00074
MCU: Accelerometer value 0: -642      MCU: Accelerometer value 0: -22
MCU: Accelerometer value 1: 801       MCU: Accelerometer value 1: 1217
MCU: Accelerometer value 2: -75       MCU: Accelerometer value 2: 74
<AccX:-00022                          <AccX:+00286
MCU: Accelerometer value 0: -22       MCU: Accelerometer value 0: 286
MCU: Accelerometer value 1: 801       MCU: Accelerometer value 1: 1217
MCU: Accelerometer value 2: -75       MCU: Accelerometer value 2: 74
```

b) Resultado en monitor serial.

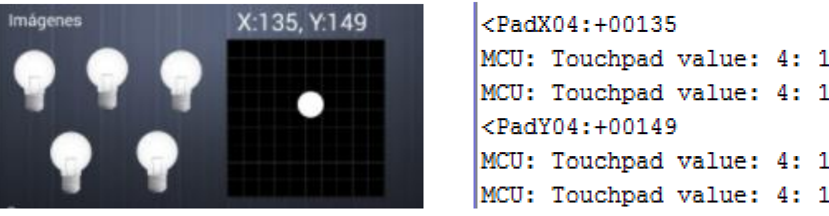
Figura 5.35. Impresión de carga útil de Accelerometer Tags.

Uso de Touchpad Tags

Homólogo a las Seekbar Tags, se utilizó el monitor serial para imprimir el valor de cada eje de cada Touchpad Tag entrante, como se muestra en las figuras 5.36a y 5.36b.

```
if(TagType == UPDATE_TPAD)
// Print touch pad array
for(int i = 0; i < TPAD_NO; i++){
  for(int j = 0; j < TPAD_AXIS; j++){
    if(TouchPadData[i][j] != 0) // don't display cero value pads
      Serial.println("MCU: Touchpad value: " + String(i) + ": " + String(TouchPadData[i][j]));
  }
}
```

a) Sección de código para imprimir el valor de cada de eje de un touchpad.




```
<PadX04:+00135
MCU: Touchpad value: 4: 135
MCU: Touchpad value: 4: 149
<PadY04:+00149
MCU: Touchpad value: 4: 135
MCU: Touchpad value: 4: 149
```

b) Resultado en monitor serial.

Figura 5.36. Impresión de carga útil de un Touch Pad Tag.

Uso de Speech Recognizer tags

Además de usar el puerto serial para verificar esta característica de la aplicación, se habilitó y programó el pin 5 de la tarjeta Arduino como salida; con la condición de que al recibir una cadena de texto del *speech recognizer* igual a "encender", este pin se maneje a estado alto; si una cadena de texto "apagar" es recibida, el pin 5 se coloca en estado bajo, como se muestra en la sección de código de la Figura 5.37a. Las figuras 5.37b y 5.37c muestran el resultado en el mundo físico y el monitor serial respectivamente.

```
if(TagType == UPDATE_SPCH)
// Print speech recognition
Serial.println("MCU: " + SpeechRecorder);
if(SpeechRecorder.equals("encender")){
  digitalWrite(5, HIGH);
}
else if(SpeechRecorder.equals("apagar")){
  digitalWrite(5, LOW);
}
break;
```

a)

```
<StoT:encender
MCU: encender

[MCU: Hola Mundo

[MCU: Hola Mundo

[MCU: Hola Mundo

<StoT:apagar
MCU: apagar
```

b) c)

Figura 5.37. Activación de una salida digital usando Speech Recognizer Tags.

Uso de tags de registro

Para verificar la funcionalidad de los tags de registro se insertó una línea de código en cada cambio de estado de las salidas digitales del MCU, de modo que, cada cambio de HIGH a LOW y de LOW a HIGH queda almacenado en el ATC Logger como se muestra en la Figura 5.38.

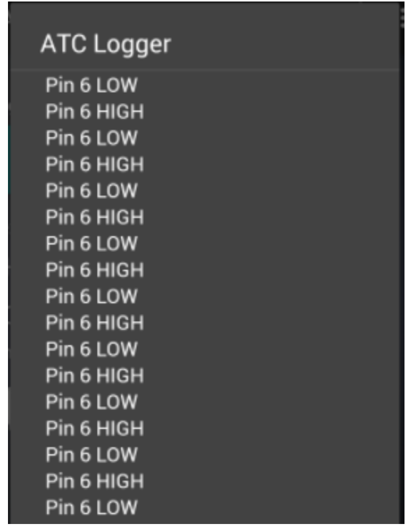


Figura 5.38. Prueba de ATC Logger.

Prototipos compatibles con ATC

ATC RC Car (Módulo de comunicaciones: HC05)

Descripción del sistema y justificación

Este prototipo es un auto a radio control, el cual utiliza Bluetooth para controlar su movimiento y los diferentes controles que ofrece la aplicación ATC. Además, la aplicación es usada para monitorear la corriente que consume el motor de tracción. El motivo para construir este prototipo es demostrar la capacidad de la aplicación para controlar un sistema simple en tiempo real; así como la ejecución de máquinas de estado al mismo tiempo que el programa en el microcontrolador recibe y envía datos desde y hacia la aplicación.

Componentes

a) 1 motor eléctrico de corriente directa de imán permanente con moto reductor (motor de tracción).
b) 1 motor eléctrico de corriente directa de imán permanente para controlar la dirección.
c) 1 circuito integrado puente H LM22N.
d) 1 módulo Bluetooth HC06.
e) 1 Arduino Nano.
f) 1 batería de litio de 7.5V.
g) 1 sensor de corriente ACS712.
h) 2 LED para luces traseras.
i) 2 LED para luces delanteras.

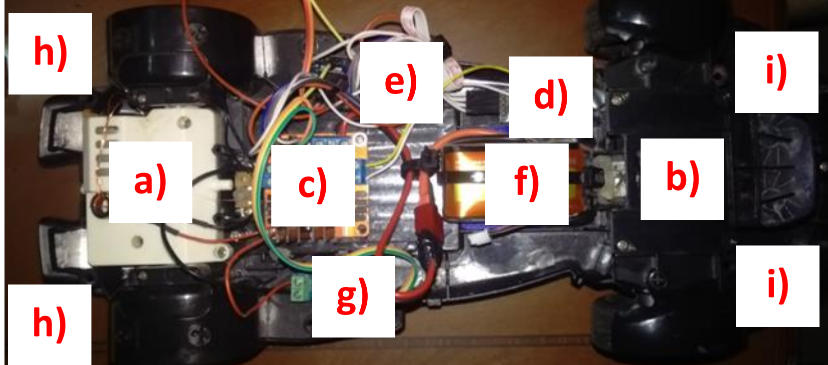


Figura 5.39. Componentes de ATC RC Car.

Diagrama de conexiones

Como se muestra en la Figura 5.40 los componentes del sistema se muestran como “cajas negras”, donde sólo las terminales de interés son presentadas.

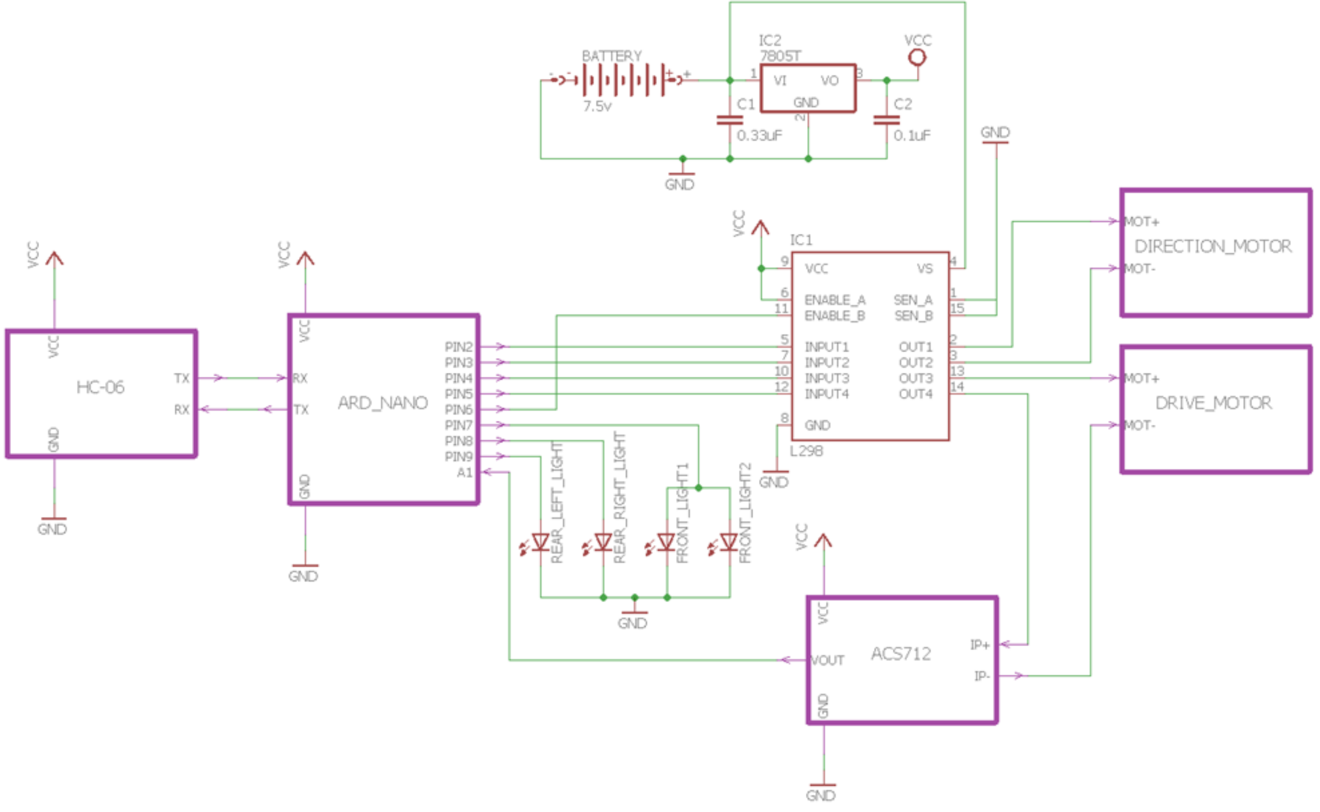
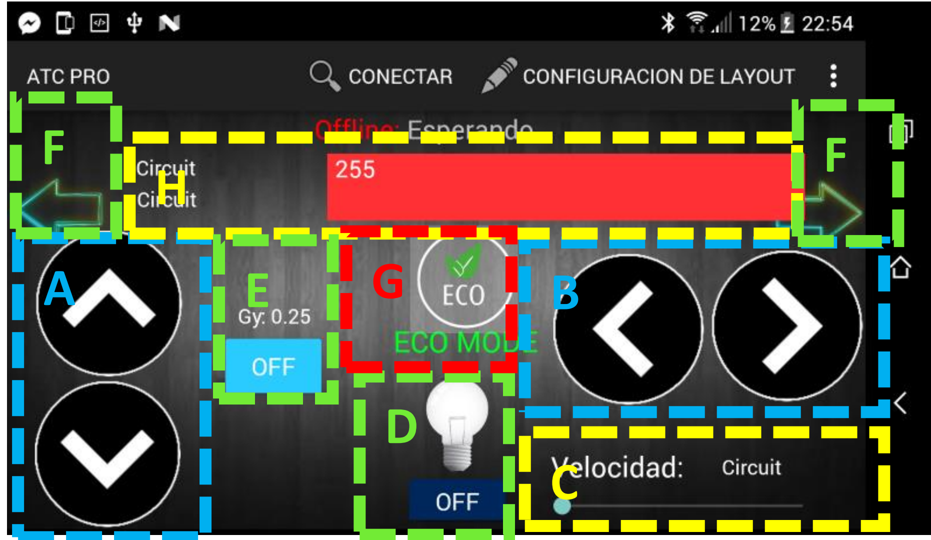


Figura 5.40. Diagrama electrico de ATC RC Car.

ATC Layout

El Layout en ATC para este prototipo cuenta con los elementos constructivos necesarios para controlar el vehículo inalámbricamente, como se describe en la Figura 5.41.



A) Control de motor de tracción (botones temp.).
B) Control de motor de dirección (botones temp.).
C) Control de velocidad de motor de tracción (seekbar).
D) Activación de luces frontales (toggle button y reconocimiento de voz).
E) Control de dirección usando acelerómetro.
F) Indicadores direccionales.
G) Activación de modo eco.
H) Consumo de corriente.

Figura 5.41. Layout para ATC RC Car.

Funcionamiento

El programa en Arduino para el ATC RC Car está basado en el bt_firmware. Al inicio del programa, se configuran las entradas y salidas particulares del sistema para controlar la dirección, la tracción y las “luces” del vehículo; además, se inicia el convertidor analógico digital para leer el voltaje que representa la corriente instantánea del motor de tracción usando un módulo basado en el circuito integrado ACS712. El sketch o programa completo en Arduino se puede encontrar en el anexo 7.

Para lograr una implementación responsiva de ATC, es decir, que el sistema digital responda en tiempo real, es necesario tomar en cuenta que se debe refrescar o ejecutar la función de lectura de información proveniente de la aplicación (*Serial.read*) lo más frecuentemente posible. Para lograr esto, se usan máquinas de estado o funciones de tiempo de ejecución corto que toman como referencia el retardo de 1ms en loop para realizar operaciones relacionadas con tiempos.

```
void loop() {
  int tagType;
  int appData;
  delay(1);

  CurrentSensing();
  FrontLightsControl(FrontLights);
  RearLightSM(DirLightStatus);
  DriveSM(DriveStatus);

  // ===============================================================
  // This is the point were you get data from the App
  appData = Serial.read();   // Get a byte from app, if available
  switch (appData) {
    case CMD_SPECIAL:
```

Funciones de tiempo de ejecución corto (< 1ms)

Figura 5.42. Ejecución de la función .read() en tiempo real.

Medición de corriente

Las propiedades del sensor de corriente de efecto hall ACS712 se muestran en la Tabla 5.3. La relación entre voltaje de salida y la corriente de entrada es de 66mV/A; además, como muestra la Figura 5.42, el valor del voltaje de salida cuando la corriente es cero es de 2.5V. Estos valores se toman en cuenta en el código para calcular la corriente medida.

Para medir y mostrar el valor de la corriente consumida por el motor de tracción en la aplicación se implementó la función *CurrentSensing*() (Figura 5.43). Esta función lee el valor de la corriente y envía la información a la aplicación del convertidor analógico digital (Text01) y su valor equivalente en Amperes (Text00); además de enviar el valor de la lectura a la barra analógica cero, proporcionando una visualización gráfica de la corriente.

x30A PERFORMANCE CHARACTERISTICS[1] T_A = –40°C to 85°C, C_F = 1 nF, and V_{CC} = 5 V, unless otherwise specified

Characteristic	Symbol	Test Conditions	Min.	Typ.	Max.	Units
Optimized Accuracy Range	I_P		–30	–	30	A
Sensitivity	Sens	Over full range of I_P, T_A = 25°C	63	66	69	mV/A
Noise	$V_{NOISE(PP)}$	Peak-to-peak, T_A = 25°C, 66 mV/A programmed Sensitivity, C_F = 47 nF, C_{OUT} = open, 2 kHz bandwidth	–	7	–	mV
Zero Current Output Slope	$\Delta V_{OUT(Q)}$	T_A = –40°C to 25°C	–	–0.35	–	mV/°C
		T_A = 25°C to 150°C	–	–0.08	–	mV/°C
Sensitivity Slope	ΔSens	T_A = –40°C to 25°C	–	0.007	–	mV/A/°C
		T_A = 25°C to 150°C	–	–0.002	–	mV/A/°C
Total Output Error[2]	E_{TOT}	I_P = ±30 A, T_A = 25°C	–	±1.5	–	%

Tabla 5.3. Características del circuito ACS712.

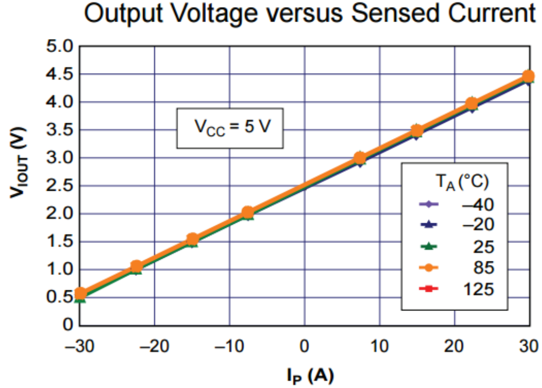


Figura 5.43. Voltaje de salida versus corriente de entrada.

```
void CurrentSensing(){
  static int prescaler = 0;

  if (prescaler++ > 100) {
    prescaler = 0;
    int analogValue = analogRead(I_CURRENT);
    DriveCurrent = ((((analogValue - 512) * 74) / 10) + DriveCurrent) >> 1; // Basic low pass
    Serial.println("<Text00: Corriente: " + String(DriveCurrent / 100) + "."
                  + myIntToString((abs(DriveCurrent) % 100) * 10) + "A");
    Serial.println("<Text01: A1: " + String(analogValue));
    Serial.println("<Abar00:" + myIntToString(abs(DriveCurrent)));
  }
```

Figura 5.44. Función para lectura de corriente.

La Figura 5.45a muestra el resultado de la función *CurrentSensing*() cuando el tren motriz del vehículo está en espera; la Figura 5.45b muestra el resultado de la medición al momento del arranque a rotor bloqueado. Como se muestra en la Figura 5.44, se envía la lectura de corriente a la aplicación aproximadamente cada 100ms, con el objetivo de proporcionar una frecuencia de muestreo adecuada para una interfaz visual.

a) b)

Figura 5.45. Resultado de la función *CurrentSensing*().

Control de luces frontales

Las luces frontales del vehículo son controladas por la función *FrontLightsControl*(), la cual es llamada constantemente en el loop principal y toma como argumento el estado de las luces. El estado de las luces frontales está definido por la variable global booleana *FrontLights*. El valor de esta variable puede cambiar por medio de un comando *raw data*, Figura 5.46c; o bien, por un comando especial de voz como se muestra en la Figura 5.46b.

```
void FrontLightsControl(boolean lightStatus){
  // Front light control
  if (lightStatus) {
    digitalWrite(O_FL, HIGH);
  }
  else {
    digitalWrite(O_FL, LOW);
  }
```

a)

```
case CMD_SPECIAL:
  // Special command received, seekbar value and a
  tagType = DecodeSpecialCommand();

  // Activate lights with voice
  if (tagType == UPDATE_SPCH) {
    if (SpeechRecorder.equals(CMD_SLON)) {
      FrontLights = true;
      Serial.println("<Imgs" + ID_LIGHT + ":1");
      Serial.println("<Butn" + ID_BLIGHT + ":1");
    }
    if (SpeechRecorder.equals(CMD_SLOFF)) {
      FrontLights = false;
      Serial.println("<Imgs" + ID_LIGHT + ":0");
      Serial.println("<Butn" + ID_BLIGHT + ":0");
    }
  }
```

b)

```
case CMD_LON:
  Serial.println("<TtoS01: encender luces");
  Serial.println("<Imgs" + ID_LIGHT + ":1");
  FrontLights = true;
  break;

case CMD_LOFF:
  Serial.println("<TtoS01: apagar luces");
  Serial.println("<Imgs" + ID_LIGHT + ":0");
  FrontLights = false;
  break;
```

c)

Donde:

```
String CMD_SLON  = "luces";
String CMD_SLOFF = "luces fuera";
```

```
#define CMD_LON    'E'
#define CMD_LOFF   'e'
```

Figura 5.46. Código de control de luces frontales.

La Figura 5.47 muestra el resultado del código de control de luces frontales en la aplicación y en el vehículo. El cambio de estado de la imagen que representa las luces frontales es dado por el microcontrolador como realimentación visual para el usuario.

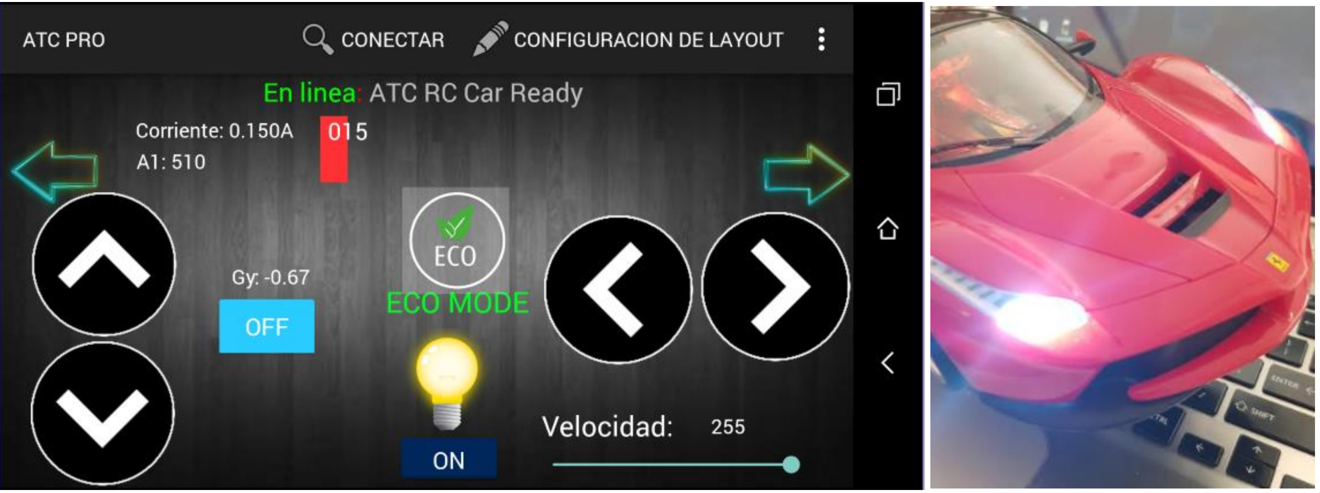


Figura 5.47. Resultado del código de control de luces frontales.

Control de luces traseras

Las luces traseras pueden funcionar como direccionales o luces de frenado. Una máquina de estados es usada para controlar su encendido y apagado dependiendo del estado del vehículo de acuerdo con la lista de estados de la Figura 5.48. La variable global entera *DirLightStatus* gobierna el estado de las luces traseras.

```
#define ST_NONE        0 // Apagadas
#define ST_BREAK       1 // Freno (ambas luces encendidas)
#define ST_TURN_LEFT   2 // Direccional izquierda (parpadeando)
#define ST_TURN_RIGHT  3 // Direccional derecha (parpadeando)
int DirLightStatus = ST_NONE;
```

Figura 5.48. Definición de estados de las luces traseras.

Usando la máquina de estados RearLightSM, se logra el parpadeo de ½ segundo de ciclo de trabajo de las luces direccionales sin crear trastornos en la operación del vehículo. Si en lugar de usar una máquina de estados, se usara la función *delay*() para lograr el tiempo de encendido y apagado, el vehículo perdería tiempo de respuesta a otros comandos entrantes.

```
void RearLightSM(int myStatus) {
  static int dir_prescaler = 0;
  static boolean onlyOnce = false;

  switch (myStatus) {
    case ST_NONE:
      digitalWrite(O_RL, LOW);
      digitalWrite(O_LL, LOW);
      if (onlyOnce) {
        Serial.println("<Imgs" + ID_RT + ":0");
        Serial.println("<Imgs" + ID_LT + ":0");
      }
      break;

    case ST_BREAK:
      digitalWrite(O_RL, HIGH);
      digitalWrite(O_LL, HIGH);
      break;

    case ST_TURN_LEFT:
      Serial.println("<Imgs" + ID_RT + ":0");
      digitalWrite(O_RL, LOW);
      dir_prescaler++;
      onlyOnce = true;
      if (dir_prescaler > 500) {
        dir_prescaler = 0;
        Serial.println("<Imgs" + ID_LT + ":" + String(digitalRead(O_LL) ? 0 : 1));
        digitalWrite(O_LL, digitalRead(O_LL) ? LOW : HIGH);
      }
      break;
```

Figura 5.49. Máquina de estados de control de luces traseras.

Además de ejecutar el encendido y apagado de la direccional correspondiente, el código de la Figura 5.49 también envía un comando a la aplicación para cambiar el estado de las imágenes que representan las direccionales en el layout, como se muestra en la Figura 5.50.

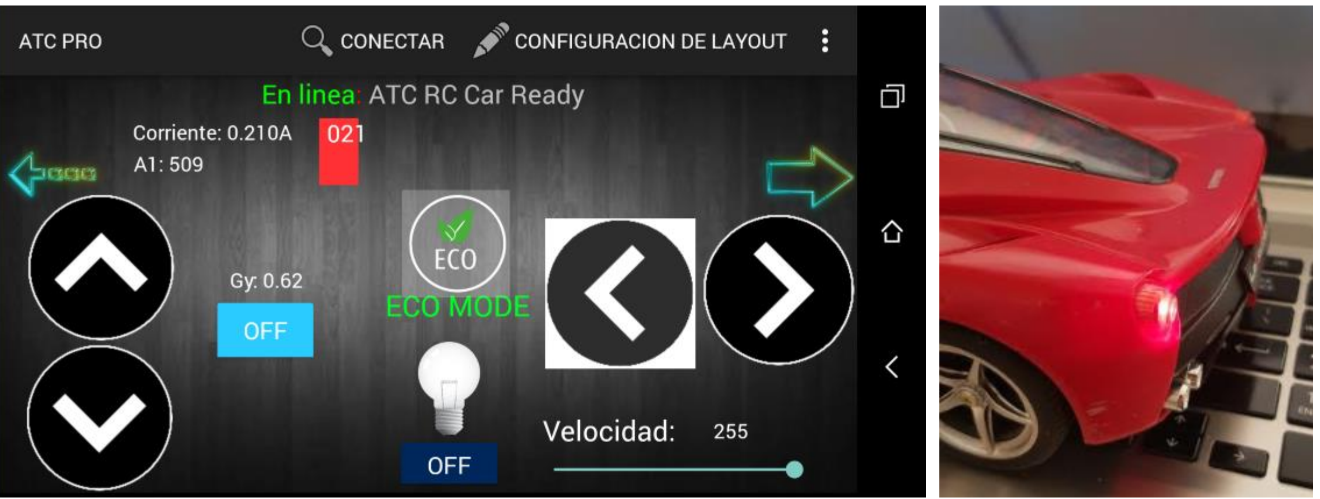


Figura 5.50. Activación de luces direccionales. Retroalimentación grafica en la aplicación (izquierda) y resultado en el mundo físico (derecha).

Control de dirección

Para controlar la dirección del vehículo, es posible utilizar los botones definidos en el Layout (inciso "B" de la Figura 5.41), o bien el acelerómetro. Para el primero simplemente se realiza la combinación correcta de las salidas que controlan el estado del motor de dirección dependiendo del comando recibido. Las combinaciones para la operación del motor se muestran en la Tabla 5.4, donde *Ven* es el voltaje de habilitación del circuito; C (o IN3) y D (o IN4) son las entradas del puente H. Los comandos *raw data* para el control de dirección y su uso dentro del programa se muestran en la Figura 5.51a y 5.51b respectivamente.

Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Tabla 5.4. Combinaciones para operación de puente H.

```
#define CMD_RIGHT   'C'
#define CMD_CENTER  'c'
#define CMD_LEFT    'D'
```

a)

```
case CMD_RIGHT:
  digitalWrite(O_IN3, HIGH);
  digitalWrite(O_IN4, LOW);
  DirLightStatus = ST_TURN_RIGHT;
  break;

case CMD_LEFT:
  digitalWrite(O_IN3, LOW);
  digitalWrite(O_IN4, HIGH);
  DirLightStatus = ST_TURN_LEFT;
  break;

case CMD_CENTER:
  digitalWrite(O_IN3, LOW);
  digitalWrite(O_IN4, LOW);
  DirLightStatus = ST_NONE;
  break;
```

b)

Figura 5.51. Definición de comandos para control de dirección.

Para controlar la dirección del vehículo utilizando el acelerómetro cuando éste está activo, se implementa la sección de código de la Figura 5.52; la cual compara el valor del acelerómetro (en m/s^2 * 100 como se vio en el capítulo 3) contra un valor establecido por el usuario (que representa una inclinación suficiente para activar el control de dirección).

Como se muestra en la Figura 5.50, para el dispositivo móvil utilizado y layout con orientación horizontal, se utiliza el eje "Y" del acelerómetro para controlar la dirección del vehículo. El valor de la aceleración en el eje "Y" corresponde al segundo elemento del arreglo de enteros *Accel[].*

```
// Enable accelerometer control
if (AccelMode) {
  if (Accel[1] > 200) {
    digitalWrite(O_IN3, LOW);
    digitalWrite(O_IN4, HIGH);
    DirLightStatus = ST_TURN_LEFT;
  }
  else if (Accel[1] < -200) {
    digitalWrite(O_IN3, HIGH);
    digitalWrite(O_IN4, LOW);
    DirLightStatus = ST_TURN_RIGHT;
  }
  else {
    digitalWrite(O_IN3, LOW);
    digitalWrite(O_IN4, LOW);
    DirLightStatus = ST_NONE;
  }
}
break;
```

Figura 5.52. Control de dirección por acelerómetro.

Tanto la Figura 5.51 como la Figura 5.52 muestran cómo se asigna el estado de la variable *DirLightStatus* para el control de las luces traseras cada vez que hay un cambio de dirección del vehículo.

Control de tracción

El control de tracción se realiza de forma similar al control de dirección cuando éste es por comandos raw data (listados en Figura 5.53a). Sin embargo, para controlar la tracción del vehículo se utiliza una máquina de estados, necesaria para controlar la aceleración gradual del auto cuando éste se encuentra en modo "eco". Los estados se configuran al recibir los comandos de control de la aplicación como se muestra en la Figura 5.53b.

a)

```
#define CMD_D       'A' // Adelante
#define CMD_N       'a' // Neutral
#define CMD_R       'B' // Reversa
```

b)

```
case CMD_D:
  DriveStatus = ST_D;
  break;

case CMD_N:
  DriveStatus    = ST_NONE;
  DirLightStatus = ST_NONE;
  break;

case CMD_R:
  DriveStatus    = ST_R;
  DirLightStatus = ST_BREAK;
  break;
```

Figura 5.53. Comandos de control de tracción.

Robot caminante ATC (Módulo de comunicaciones: HC05)

Descripción del sistema y justificación

El robot caminante es un prototipo de robot cuadrúpedo con dos grados de libertad en cada "pata". La intención de este robot no es estar programado con alguna secuencia de movimiento, sino estar habilitado para permitirle al usuario "programarlo" o "enseñarle posiciones" a través de la aplicación por medio de Bluetooth.

El objetivo de este prototipo es demostrar el uso de los touchpads para controlar las articulaciones del robot en tiempo real. Además, este prototipo demuestra que ATC es compatible con sistemas que necesitan administrar, guardar y leer información en medios de almacenamiento no volátiles, en este caso, memoria EEPROM.

Componentes

a) 1 Arduino UNO.
b) 8 servos Tower PRO (Torque: 15Kgcm).
c) 1 módulo Bluetooth HC05.
d) 1 batería 6v.
e) 1 juego de componentes mecánicos impresos en3D.

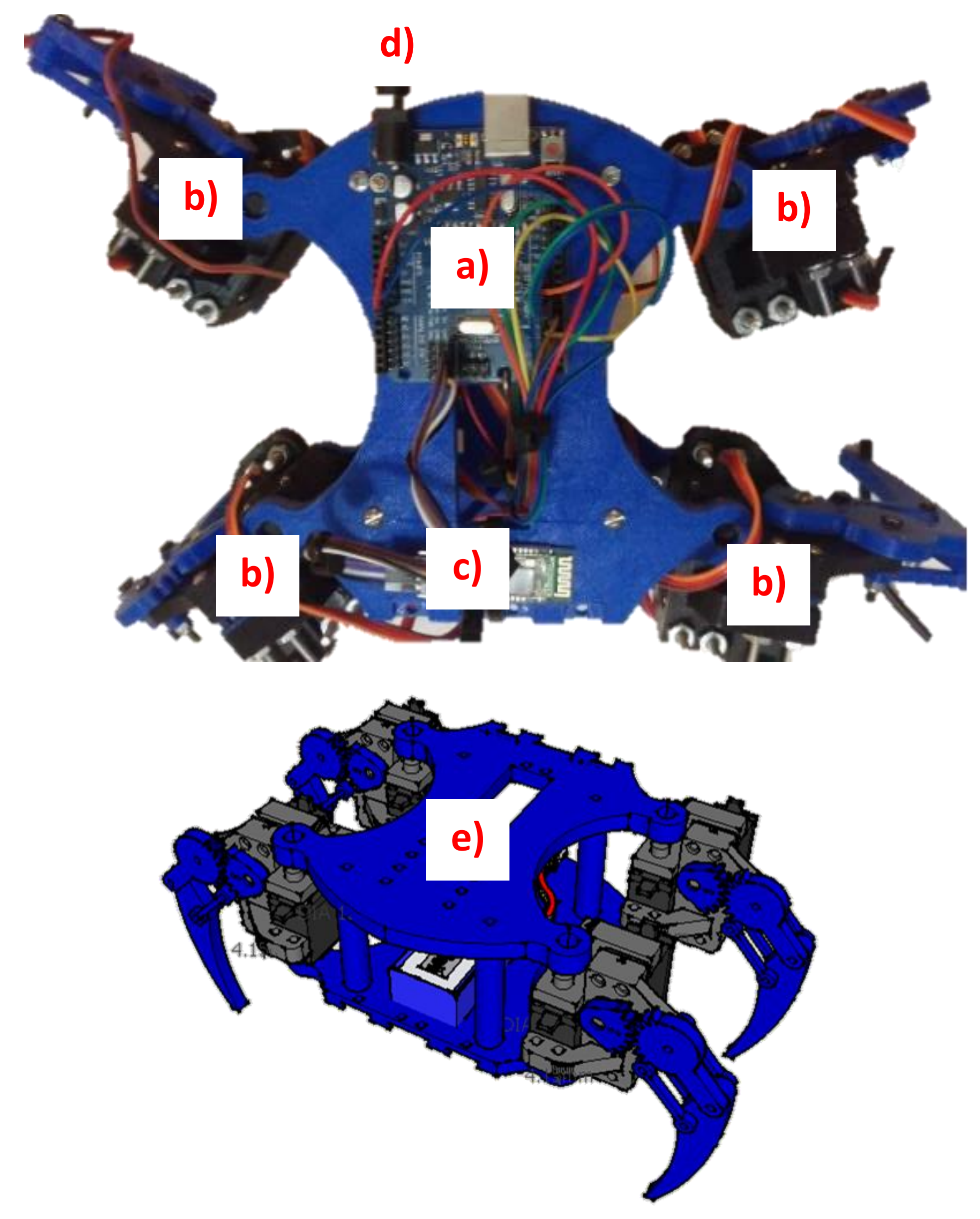


Figura 5.54. Componentes robot caminante (arriba) y diseño de robot en SketchUp (abajo).

Diagrama de conexiones

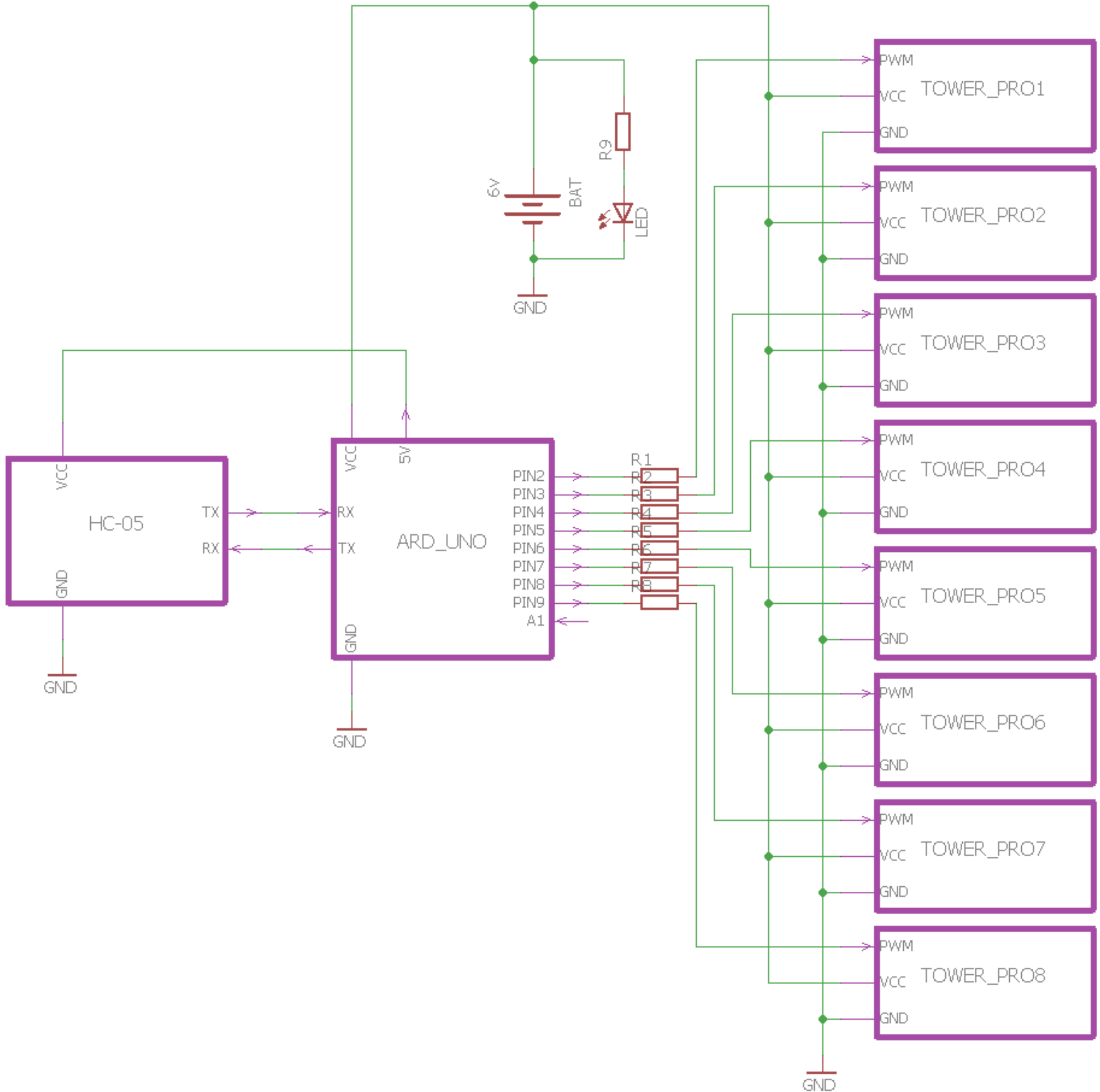


Figura 5.55. Diagrama eléctrico robot caminante.

ATC Layout

El layout en ATC está dividido en dos partes. La parte superior cuenta con 4 touchpads (uno para cada extremidad); cada touchpad controla dos servos simultáneamente en tiempo real, asignando el valor en X para un servo y el de Y para otro. La parte inferior cuenta con botones temporales para guardar, borrar y reproducir las secuencias de movimiento del robot. Como se muestra en la Figura 5.56, el layout permite interactuar con hasta 4 secuencias diferentes de movimiento (adelante, atrás, derecha e izquierda).

Funcionamiento

El código en Arduino para el robot ATC caminante puede ser encontrado en el anexo 8 y está basado en el siguiente seudocódigo:

1. Ejecutar función setup,
2. Iniciar comunicación serial, servos y variables de touchpad.

3. Restablecer punteros y arreglos de posiciones para cada secuencia de movimiento (adelante, atrás, derecha e izquierda).
4. Ejecutar función loop.
5. Imprimir el ángulo deposición de cada servo en la aplicación.
6. Verificar datos disponibles en puerto serial.
7. Si el dato disponible es un comando touchpad, guardar en arreglo *TouchPadData* y mover servo a posición.
8. Si el dato disponible es un comando raw data:
    a. Guardar la posición de los ocho servos en el arreglo correspondiente;
    b. Borrar el determinado arreglo de posiciones restableciendo a cero el valor del puntero correspondiente;
    c. Reproducir el arreglo de posiciones seleccionado.

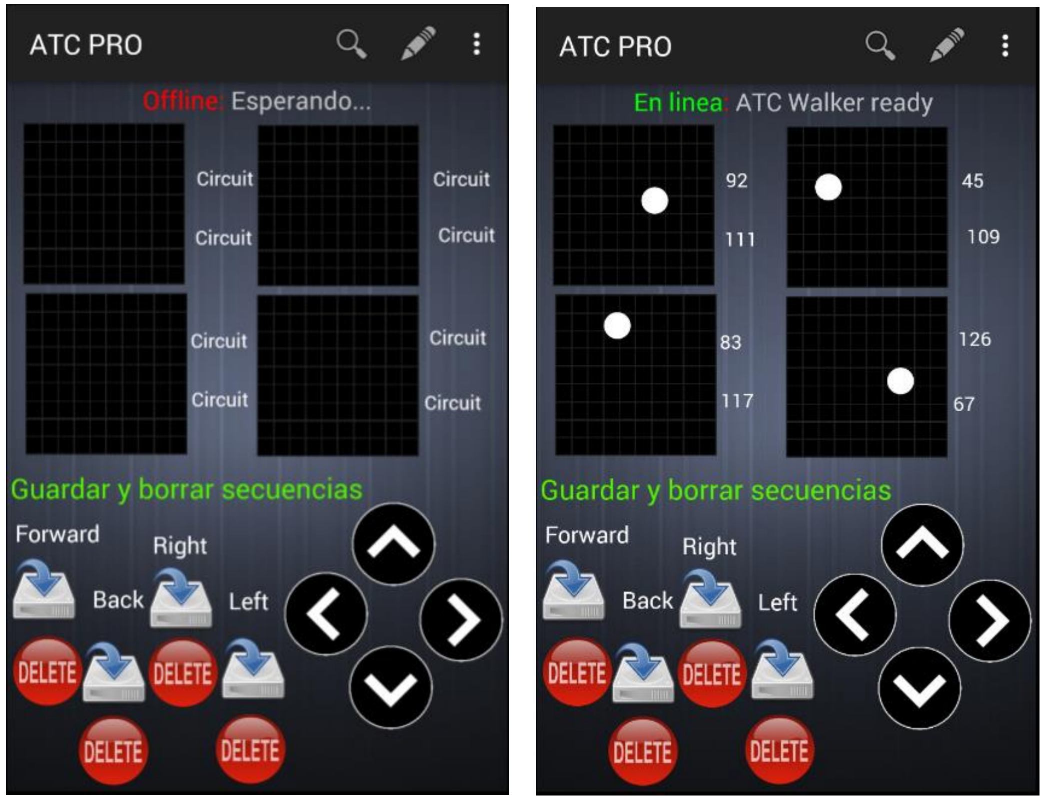


Figura 5.56. Layout ATC para robot caminante antes (izquierda) y después (derecha) de conectarse.

Tarjeta de entradas y salidas de propósito general (Módulo de comunicaciones: HM10)

Descripción del sistema y justificación

Este prototipo es un circuito capaz de monitorear y activar 6 salidas digitales, leer 6 entradas digitales, y leer 6 entradas analógicas usando la aplicación y un módulo de comunicaciones Bluetooth LE HM10. Este circuito de prueba es de propósito general, es decir, no está diseñado para interactuar con algún sistema físico en particular.

Este prototipo es construido para demostrar la capacidad de la aplicación para activar y monitorear múltiples entradas y salidas simultáneamente. Además, este prototipo demuestra la capacidad de la aplicación para interactuar con sistemas conectados vía Bluetooth LE, junto con las limitaciones respecto a la tasa de transferencia que estos conllevan (paquetes de datos pequeños y frecuencia de transferencia baja, véase capítulo 4) versus el Bluetooth convencional.

Componentes

a) 1 Arduino UNO.
b) 1 módulo Bluetooth LE HM10.
c) 1 tarjeta de relevadores.

d) 1 Fuente de poder 12V dc.

ATC Layout

El layout en ATC de este prototipo cuenta con los elementos necesarios para representar las entradas digitales y analógicas; así como las salidas digitales, como se muestra en la Figura 5.57.

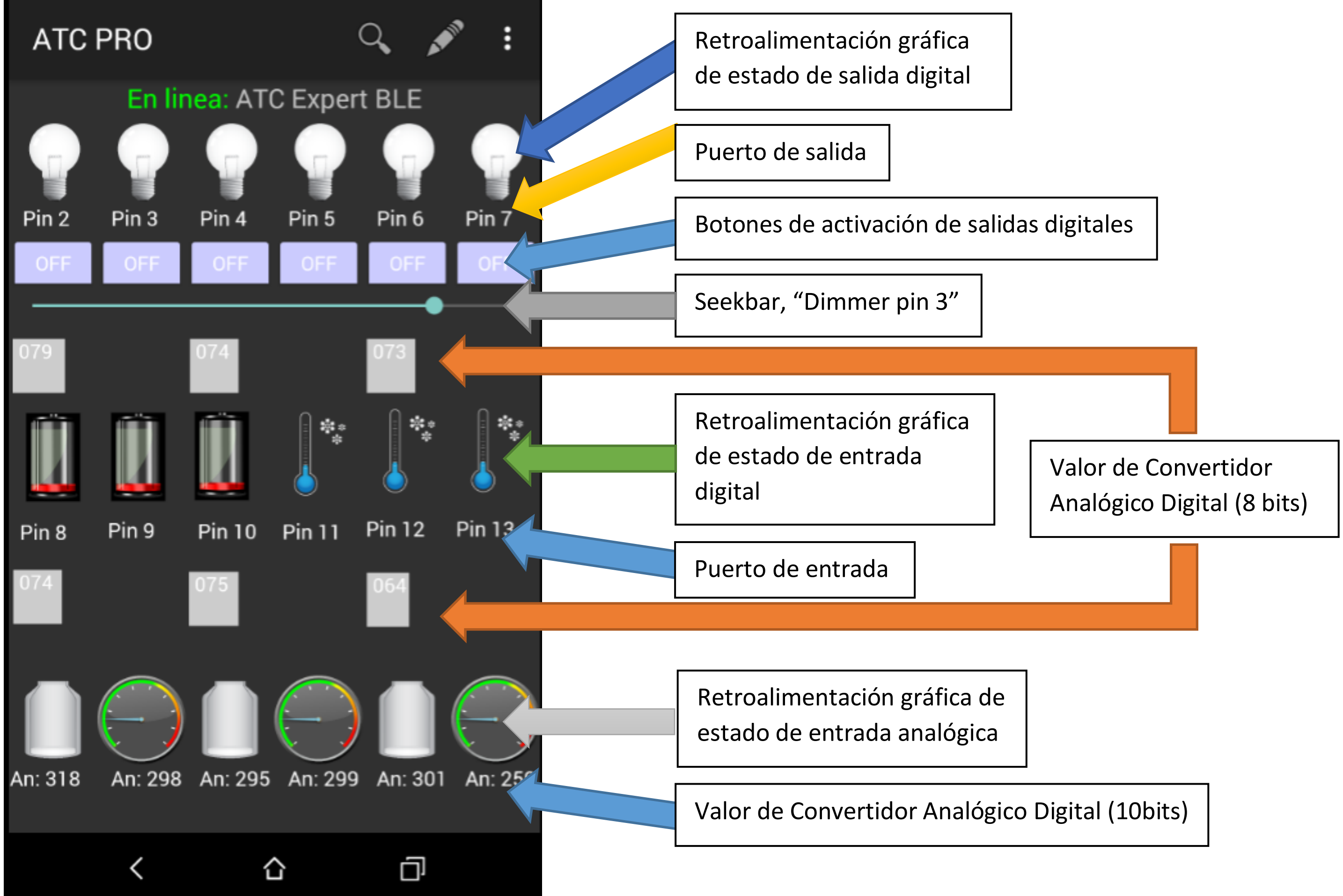


Figura 5.57. ATC Layout para prototipo "Tarjeta de entradas y salidas de propósito general".

Funcionamiento

El sketch de la tarjeta de entradas y salidas de propósito general BLE está basado en el *bt_firmware.ino*; con las rutinas respectivas de inicialización, control y monitoreo de entradas y salidas analógicas y digitales (ver anexo 9).

Salidas Digitales

Las salidas digitales son controladas por raw data provenientes de los botones de activación (ver Figura 5.57). Cuando el sistema embebido recibe un raw data determinado, envía un *Image Tag* a la aplicación para cambiar el estado de la imagen correspondiente al estado actual de la salida digital (bulbo encendido o apagado).

Entradas digitales

El microcontrolador monitorea el estado de las entradas digitales a una frecuencia de 10 Hz. El sistema enclava el estado de la entrada digital, de forma que el estado de la entrada no cambia hasta que el usuario deje de presionar y vuelva a presionar el botón. La baja frecuencia de muestreo hace innecesario una secuencia de *debounce*.

El estado enclavado de la entrada digital gobierna el estado de las salidas digitales del sistema, además de provocar un cambio en las imágenes de retroalimentación gráfica correspondientes (ver Figura 5.57).

Figura 5.58. Prototipo "Tarjeta de entradas y salidas de propósito general BLE".

Entradas analógicas

El sistema a base de microcontrolador monitorea el estado de las entradas analógicas aproximadamente cada 250ms; sin embargo, como se menciona en el capítulo 4, los módulos BLE HM10 tienen una limitante de tamaño de paquete de 20 bytes, además de que el tiempo de espera entre transmisiones debe ser entre 15 y 20ms. Para actualizar la visualización de cada entrada analógica se requieren al menos 3 ATC tags para actualizar el texto, la imagen y la barra analógica que representan cada ADC. Esto provoca un retardo en la ejecución del programa de 45ms por convertidor analógico digital, lo cual da un acumulado de 270ms, lo cual a su vez, provocaría una operación poco responsiva o "lenta" para él usuario.

Una forma de sobrepasar la limitante de tasa de transferencia de un módulo de BLE es leer y actualizar los valores del Convertidor Analógico Digital de forma secuencial, es decir, actualizar cada 250ms el valor de un y solo un ADC a la vez. Esto se logra con una variable contadora (analogPrescaler1) y la operación módulo, como se muestra en la Figura 5.59.

```
if (analogPrescaler++ > 250) {
  analogPrescaler = 0; // Reset prescaler

  // Use <Text tags to display alphanumeric information in app
  Serial.print("<Text" + AIAppId[analogPrescaler1] + ":" + "An: " + String(sample));
  delay(15);
  // Use <Imgs tags to dinamically change pictures in app
  Serial.print("<Imgs" + AIAppId[analogPrescaler1] +  EvaluateAnalogRead(sample));
  delay(15);
  // Use <Abar tags to change analog bar levels from 0 to 255
  Serial.print("<Abar" + RelayAppId[analogPrescaler1] +  ":" + myIntToString(sample >> 2));
  delay(15);
  analogPrescaler1 = ++analogPrescaler1 % MAX_A_INPUTS;
}
```

Figura 5.59. Lectura y actualización secuencial de ADC para módulos BLE.

Cerradura electrónica ATC (Ethernet Shield)

Descripción del sistema y justificación

La cerradura electrónica ATC es un prototipo simple que emula un sistema de acceso por contraseña. La contraseña es introducida en la aplicación; usando botones temporales que representan las "teclas" o dígitos de la contraseña; o bien, usando el reconocimiento de voz.

Este prototipo demuestra que el comando CMD_ALIVE puede ser usado para reconocer el estado de la conexión con el dispositivo móvil a nivel de Capa de Aplicación. Además, este prototipo demuestra la flexibilidad de la aplicación para ser utilizada en sistemas donde no necesariamente cada botón representa una acción en el mundo real.

Componentes

a) 1 Arduino UNO.
b) 1 Ethernet Shield.
c) 5 LED.
d) 1 cable ethernet.
e) 1 router inalámbrico.

ATC Layout

El Layout en ATC mostrado en la Figura 5.60 cuenta con los dígitos 1 a 9, los cuales son botones temporales cuyo comando raw data a enviar es el carácter que representa el mismo número. Los dos iconos de la parte inferior son para iniciar el reconocimiento de voz y para bloquear la cerradura respectivamente.

Figura 5.60. ATC Layout para cerradura electrónica.

Funcionamiento

Cuando el microcontrolador recibe una secuencia de 6 caracteres válidos a través de *raw data* (caracteres del '1' al '9') los compara con la contraseña definida en el programa; si ésta es idéntica "liberará" el cerrojo, representado por los LED conectados del pin 2 al 6, de lo contrario notifica al usuario. Cuando la contraseña es proporcionada por un comando de voz, la aplicación compara de igual forma la cadena de texto recibida con la contraseña definida en el programa. (Para el programa completo ver anexo 10).

Para reconocer el estado de la conexión con el dispositivo móvil a nivel de Capa de Aplicación se definen dos variables; una variable de estado *FirstTimeConnection*; y una variable de conteo *TimeOutCounter*. La primera será “true” cuando el microcontrolador reciba el primer CMD_ALIVE (Figura 5.61b) y se mantendrá en ese estado hasta que la segunda alcance el valor de TIME_OUT o tiempo fuera (20000 para este ejemplo) como se muestra en la Figura 5.61a. Cada loop tiene una duración aproximada de 1ms, por lo que el *TimeOutCounter* alcanzará la condición de tiempo fuera en 20 segundos, a menos que un comando “alive” sea recibido en ese intervalo.

Como se menciona en el Capítulo 4, la aplicación envía el CMD_ALIVE aproximadamente cada 2.5 segundos, por lo que, mientras la aplicación y el microcontrolador estén conectados a nivel de Capa de Aplicación, la variable *TimeOutCounter* nunca llegará a la condición de tiempo fuera.

```
void loop() {
  int appData;
  int commandType;
  delay(1);

  if (FirstTimeConnection) {
    digitalWrite(outs[0], HIGH);
    if (TimeOutCounter++ > TIME_OUT) {
      FirstTimeConnection = false;
      TimeOutCounter = 0;
    }
  }
  else {
    digitalWrite(outs[0], LOW);
    LockStatus = ST_CLOSED;
  }
```

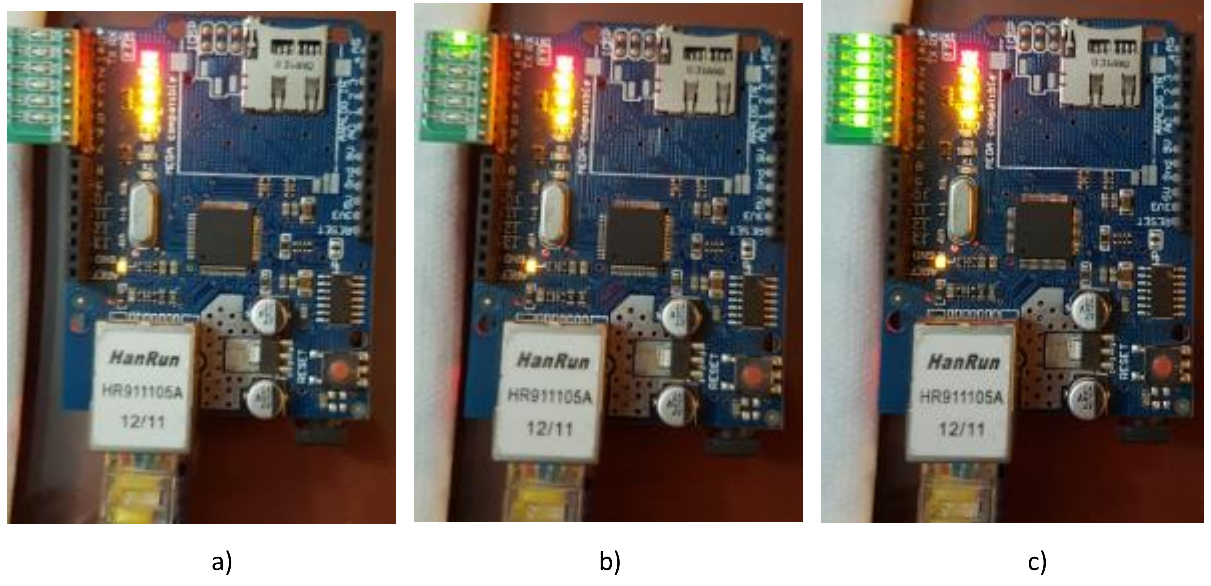
a)

```
case CMD_ALIVE:
  // Character '[' is received every 2.5s
  server.println("ATC ready");
  // Reset timeout
  TimeOutCounter = 0;
  // Greet first connection
  if (!FirstTimeConnection) {
    FirstTimeConnection = true;
    server.println("<TtoS01: Bienvenido a casa,
  }
  break;
```

b)

Figura 5.61. Código para detectar estado de conexión a nivel de Capa de Aplicación.

Al validar la conexión con el dispositivo móvil, el MCU coloca en estado alto la salida 1 (Figura 5.62b). Cuando la contraseña introducida es correcta, las salidas 2 a 6 son puestas en estado alto para representar la liberación de la cerradura (Figura 5.62c). Si el botón de bloqueo de cerradura es accionado por el usuario, las salidas 2 a 6 son puestas en estado bajo como muestra la Figura 5.62b. Si el microcontrolador pierde conexión con la aplicación, todas las salidas (de la 1 a la 6) son puestas en estado bajo como muestra la Figura 5.62a.



a) b) c)

Figura 5.62. Estados de la cerradura electrónica ATC.

Discusión de resultados

Como muestran los resultados de las pruebas de comunicaciones inalámbricas (Tabla 5.1) y las pruebas del protocolo de Capa de Aplicación (Tabla 5.2), la aplicación cumple las funciones para las que fue diseñada. Además, en conjunto con los prototipos compatibles construidos en el capítulo 5, se demuestra la funcionalidad y flexibilidad de la aplicación para controlar y monitorear sistemas físicos a base de microcontrolador.

Durante el proceso de desarrollo y obtención de resultados de este trabajo se presentaron algunas complicaciones y limitaciones que se detallan a continuación:

Cambios de permisos requeridos para establecer una conexión Bluetooth.

A partir de Android 6.0 Marshmallow, se agrega el permiso ACCESS_COARSE_LOCATION como requerimiento para escanear un dispositivo Bluetooth EDR/BR o Bluetooth LE, lo que provoca que los dispositivos móviles que cuenten con esta versión no puedan encontrar dispositivos Bluetooth cercanos. Agregar el nuevo permiso en el manifiesto, como se muestra en la Figura 6.1, resuelve el problema, pero ha concientizado al autor de la importancia de mantener el código probado y actualizado con cada nueva versión de Android lanzada al mercado.

```
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
```

Figura 6.1. Permiso ACCESS_COARSE_LOCATION.

Variaciones en envió de datos continuos

Las funciones que pueden consumir el mayor ancho de banda de salida en ATC son el acelerómetro y los touchpads, debido a que pueden enviar hasta 3 y 2 ATC tags “al mismo tiempo” respectivamente. El envío de estos datos funcionaba correctamente para dispositivos menores a Android 6.0; cuando empezó a fallar, al enviar valores de touchpads y de acelerómetro incompletos. Estos datos incompletos consistían en enviar solo algunos de los tags de aceleración o de posición de touchpad; es decir, se enviaba el dato del eje “X” y “Y”, pero no el de “Z”, en el caso del acelerómetro; y se enviaba sólo el valor de “X” en el caso de los touchpads.

La solución definitiva a este problema fue enviar todos los tags involucrados en el mismo paquete de la Capa de Aplicación a la Capa de Transporte; es decir, en la misma cadena de texto, como se muestra en la Figura 6.2 para los touchpad.

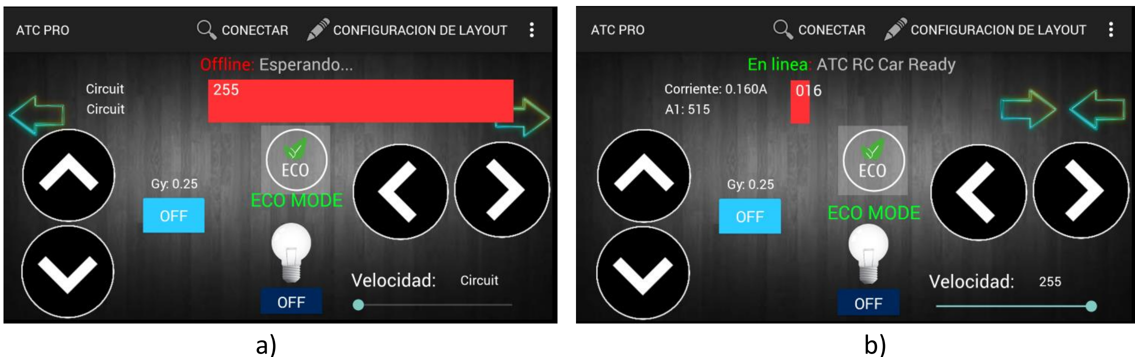
```
// Send touchpad data in correct format to corresponding channels
void TouchPadSendCommand(int selector, int x, int y) {
    String withFormatX = "<PadX" + sIndexList[selector]
            + ":" + get16BitStringNumber(x) + "\n";
    String withFormatY = "<PadY" + sIndexList[selector]
            + ":" + get16BitStringNumber(y) + "\n";

    MainSend(EProp.TYPE_RELAY_IMG, selector, withFormatX + withFormatY);
    //MainSend(EProp.TYPE_RELAY_IMG, selector, withFormatY);
}
```

Figura 6.2. Envió de datos continuos compatible con dispositivos Android 6.0+.

Errores no identificados en el posicionamiento de los objetos en el layout.

La Figura 6.3a muestra la posición correcta de las flechas de dirección en el layout del vehículo a radio control ATC, mientras que la Figura 6.3b muestra el resultado obtenido después de cerrar y reiniciar la aplicación. Éste es un error que no se ha podido identificar en el programa ya que dejó de ocurrir después de que el proceso de relocalizar los elementos, cerrar y abrir la aplicación se repitiera un par de veces.

Se sospecha que tiene que ver con colocar el punto cero de las imágenes fuera del layout, es decir, muy cercano a las bordes de la pantalla.



a) b)

Figura 6.3. Error de posicionamiento de objetos en layout.

Variaciones de tiempo de ejecución entre dispositivos Android

Para una comunicación TCP/IP sobre Wi-Fi, se observó que en algunos dispositivos Android la recepción de información proveniente del microcontrolador se veía retrasada cada vez que el dispositivo móvil enviaba información al MCU por el mismo medio. Ya que ambas secuencias (envío y recepción de datos) se encontraban en el mismo thread (TCPClient), se optó primero por correr la secuencia de envío de datos en el Main thread. Lo anterior funcionó para algunos dispositivos, pero ocasionó "application not responding error" en otros.

La solución definitiva consistió en ejecutar un *asynch task* para enviar los datos del dispositivo Android al MCU. Con esto, el envío de datos se ejecuta en un thread independiente a la recepción y al *main* thread.

Este error anómalo demuestra la importancia de realizar pruebas de una aplicación en Android en distintos dispositivos móviles de distintos fabricantes antes de liberar la aplicación en la tienda de Google.

Memoria de datos del microcontrolador

Generalmente, un programa de un sistema a base de microcontrolador compatible con ATC no tendrá dificultades relacionadas a la limitada capacidad de procesamiento, memoria de datos y de programa de un microcontrolador convencional. Sin embargo, es importante tener en cuenta estos parámetros cuando se diseña cualquier sistema a base de MCU en particular. En el caso del robot caminante ATC, la baja capacidad de memoria RAM de un atmega328 genera la advertencia "poca memoria disponible, problemas de estabilidad pueden ocurrir" como se muestra en la Figura 6.4b. Lo anterior limita el número de posiciones por secuencia de movimiento del caminante a 10 poses. Como referencia, la memoria de programa y de datos utilizada en el sketch esp_firmware.ino es mostrada en la Figura 6.4a.

```
Sketch uses 6,924 bytes (21%) of program storage space. Maximum is 32,256 bytes.

Global variables use 482 bytes (23%) of dynamic memory, leaving 1,566 bytes for local variables. Maximum is 2,048 bytes.
```

a)

```
Sketch uses 9,758 bytes (30%) of program storage space. Maximum is 32,256 bytes.

Global variables use 1,888 bytes (92%) of dynamic memory, leaving 160 bytes for local variables. Maximum is 2,048 bytes.

Low memory available, stability problems may occur.
```

b)

Figura 6.4. Resultado de compilación del sketch esp_firmware (a), y del bt_ATCWalker (b).

ESP8266 como Access Point

De acuerdo con el manual del ESP8266, el circuito se puede configurar para que funcione como un Access Point (Punto de Acceso) y servidor TCP/IP al mismo tiempo, lo que eliminaría la necesidad de un router inalámbrico para conectarse con la aplicación. A pesar de que se logra una conexión, durante las pruebas de funcionamiento realizadas al módulo en este modo, no se logra una conexión estable. Como se muestra en la Figura 6.5, el módulo ESP se conecta y desconecta constantemente.

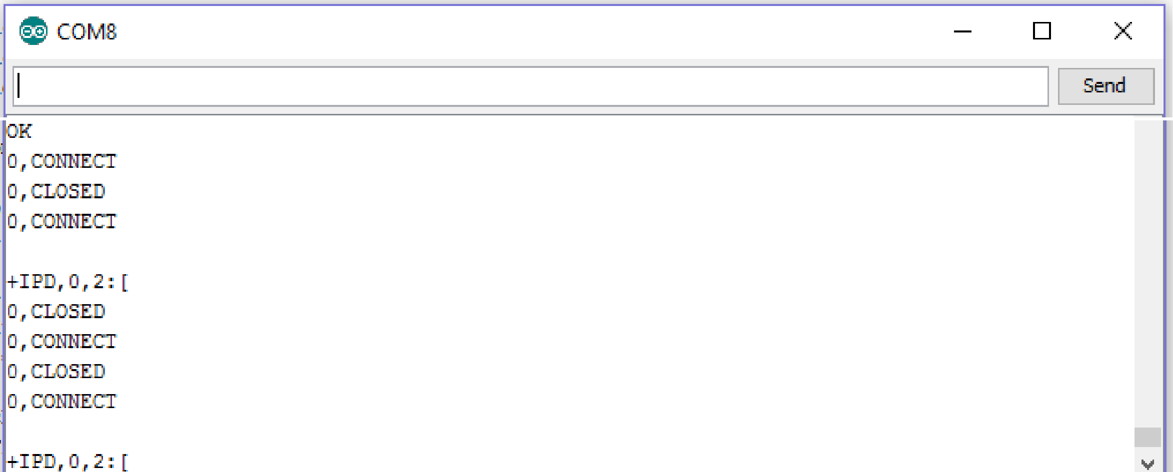


Figura 6.5. Conexión inestable de ESP8266 en modo AP.

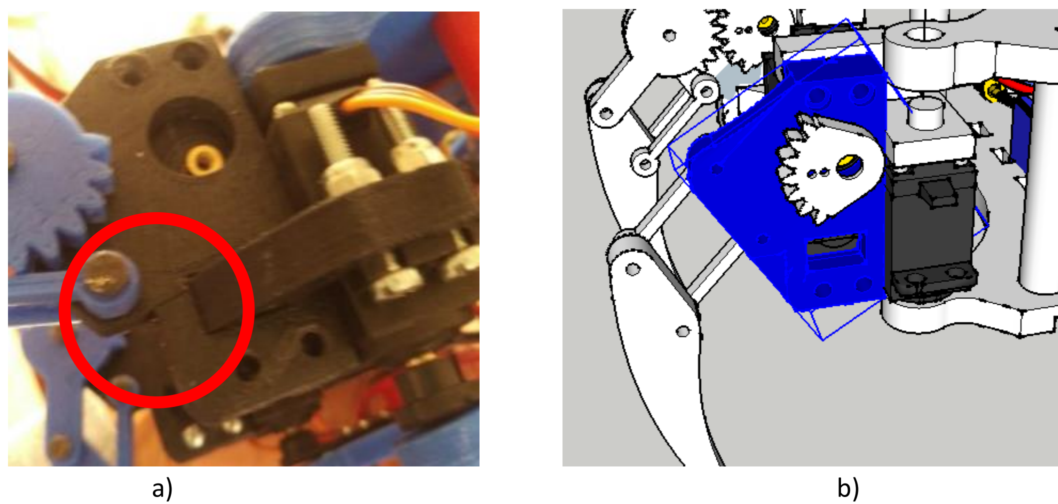
El mismo módulo, configurado en modo de servidor TCP/IP solamente funciona de forma normal, y pasa todas las pruebas de comunicación como se demostró en el capítulo 5.

Conexión fallida a módulo kc-4114

Para la realización de Bluetooth LE se realizaron pruebas también con el módulo BLE kc-4114, las cuales se detallan en el apéndice B. La falta de información e incongruencia de los manuales y hojas de datos con el comportamiento real del módulo fueron las principales causas de no haber logrado comunicar este módulo exitosamente.

Fallas mecánicas y eléctricas en ATC Walker

Durante la programación y pruebas de funcionamiento se presentaron diferentes complicaciones relacionadas con la construcción física de los prototipos. En particular, el ATC Walker sufrió una avería en una de sus piezas de montaje, como se muestra en la Figura 6.6a. La hipótesis es que esta falla es el resultado de un alto esfuerzo y un diseño mecánicamente débil en esta zona de la pieza plástica. La Figura 6.6b muestra la solución al problema.



a) b)

Figura 6.6. Avería mecánica y solución propuesta de robot caminante.

Además de la pieza rota, el caminante ATC sufrió también de averías eléctricas, al dañarse o "quemarse" dos servomotores. La causa fue una alimentación incorrecta 7.2V versus 6V recomendados.

Comparación con aplicaciones similares

Como se mencionó en el capítulo uno, otras aplicaciones para controlar sistemas compatibles con Bluetooth y Wi-fi están disponibles en la tienda oficial de aplicaciones de Android (Google Play). La Tabla 6.1 compara las características de Blynk, Arduino Commander y Arduino Total Control. A diferencia de Arduino Commander, que proporciona una interfaz gráfica fija que representa una tarjeta Arduino; la aplicación desarrollada en esta tesis le permite al usuario configurar libremente el layout, en cuanto tamaños, posiciones, colores, imágenes y contenidos se trata. En contraste, Blynk solo le permite al usuario modificar el número, tipo, y color de los controles mostrados en la interfaz gráfica. Por último, el ajuste de posición de los elementos es relativo a los elementos existentes en el layout, es decir, no es una posición libre en el plano de la pantalla, como lo es en el caso de ATC.

Una de las desventajas de ATC respecto a Blynk, es que al manejar un layout completamente configurable, el orden y la estética del mismo dependerán del operador humano; mientras que en Blynk, el diseño gráfico de los controles proveídos y el layout estilo matriz proporcionan una vista mucho más atractiva para el usuario.

A diferencia de Blynk y Arduino Commander, ATC soporta los tres principales medios de comunicación inalámbrica de área local de un dispositivo móvil; Bluetooth, Bluetooth LE y Wi-Fi. Además, ATC es compatible con cualquier dispositivo que implemente el protocolo de Capa de Aplicación ATC sobre Bluetooth, Bluetooth LE o Wi-Fi, a diferencia de Blynk y Arduino Commander que están acotados a determinados circuitos. Otra característica exclusiva de ATC, es la multi conexión de dispositivos Bluetooth BR/EDR, donde se pueden conectar hasta 7 dispositivos Bluetooth esclavos.

ATC implementa la función de texto a voz, lo que le permite al usuario darle voz a su sistema a base de microcontrolador, lo cual no se encuentra disponible en Blynk ni en Arduino Commander.

Característica	Blynk	Arduino Commander	ATC
Interfaz Gráfica	Estética y Personalizable con elementos predefinidos	La Interfaz asemeja una tarjeta Arduino	Completamente Personalizable
Microcontroladores compatibles	*Arduino Uno, Nano, Mini, Pro Mini, Pro Micro, Mega, YÚN (Bridge), Due, 101, Raspberry Pi, Particle Core, Particle Photon, ESP8266, TinyDuino (CC3000), Wicked WildFire (CC3000)*	*Diecimila, Duemilanove, Uno r1/r2/r3, Mega, Leonardo, Nano*	*Cualquier Microcontrolador equipado con un módulo Bluetooth, Bluetooth LE o WiFi*
Bluetooth	No	Sí	Sí

Bluetooth LE	Sí	No	Sí
Wifi	Sí	Sí	Sí
Multiconexión	No	No	Sí (Solo Bluetooth BR/EDR)
USB	Sí	Sí	No
Facilidad de uso	Código autogenerado dependiendo de los objetos habilitados en la interfaz gráfica.	Código Autogenerado y cargable directamente desde el dispositivo Android a la tarjeta Arduino.	Código autogenerado dependiendo de los objetos habilitados en la interfaz gráfica (beta)
Soporte en línea	Sí	Sí	Sí
Compartir proyectos	Sí	No	Sí
Servidor propio para control sobre internet	Sí	No	No
Sensores (Acelerómetro, barómetro, etc.)	Sí	Sí (costo adicional)	Sí (acelerómetro)
Reconocimiento de voz	No	Sí	Sí
Graficador de datos	Sí	Sí	Sí (barras)
Texto a voz	No	No	Sí
Comercialización	Aplicación Gratis con opciones de compra dentro de la aplicación	Aplicación Gratis con opciones de compra dentro de la aplicación	Versiones gratis y de paga.

Tabla 6.1. Comparativa Blink, Arduino Commander y ATC.

Publicación de la aplicación en la tienda Google Play

Desde su publicación en la tienda de Google Play, la aplicación a reunido más de 100mil descargas entre la aplicación gratis y de paga, como se muestra en el recorte de pantalla de la consola de desarrollador de la Figura 6.7.

Nombre de la app	Precio	Instalaciones activas/totales	Calificación promedio / Total	Última actualización	Estado
Arduino Total Control free 7.8.7	Gratis	8,830 / 98,280	★ 3.98 / 528	10/1/2016	**Publicada**
ArduinoTC -Arduino/BT/WiFi/BLE 8.1.9	MXN 35.00	636 / 1,736	★ 4.34 / 74	18/12/2016	**Publicada**

Figura 6.7. Desempeño de la aplicación en la tienda de Google Play.

Revisiones de este trabajo

A partir de la versión ATC 7.6.0, que fue la primera versión en publicarse en la tienda de Google play, esta aplicación ha tenido 45 revisiones. Cada actualización incrementó las capacidades de edición de layout y de conexión de la app. La Tabla 6.2 muestra las ultimas 4 versiones de ATC y sus respectivas fechas de lanzamiento.

		Release	Started
		8.1.9	Dec 18, 2016, 6:44 AM
		8.1.8	Dec 13, 2016, 3:54 AM
		8.1.7	Oct 26, 2016, 6:05 AM
		8.1.6	Sep 5, 2016, 3:16 AM

Tabla 6.2. Versiones de la aplicación ATC.

Por parte del trabajo escrito, además de las revisiones por capítulo, se tuvo 3 ciclos de revisión completa.

Siguientes pasos

Para mejorar la experiencia del usuario e incrementar las capacidades de la aplicación los siguientes puntos podrían ser implementados:

1. **Adicionar más sensores**. Además del acelerómetro, las API's de Android soportan sensores de temperatura ambiental, giroscopio, iluminación, campo magnético, presión, proximidad y humedad relativa [53] que podrían ser incluidos en el protocolo de Capa de Aplicación de ATC.
2. **GPS**. Además de enviarle al MCU información referente a sensores, el dispositivo móvil también podría enviar al MCU información del sistema de posicionamiento global.
3. **Multi conexión Wi-Fi usando UDP/IP**. UDP/IP podría ser más eficiente en implementaciones que no requieran una comunicación en modo de conexión y que necesiten conectarse a múltiples dispositivos remotos al mismo tiempo.
4. **Multi conexión Bluetooth LE**. Al ser de baja energía, la tecnología Bluetooth LE es ideal para establecer múltiples conexiones para monitorear y controlar distintos sistemas al mismo tiempo.
5. **Graficador de datos**. Mostrar información graficada en tiempo real en un plano cartesiano con el objetivo de mostrarle al usuario la evolución de alguna variable o señal a través del tiempo.

Conclusiones

La importancia de este trabajo radica, por un lado; en reducir el nivel de conocimientos, el tiempo y el costo necesario para realizar una interfaz hombre máquina; y por el otro, en incrementar el uso de dispositivos móviles para el control de los sistemas digitales, o bien el Internet de las Cosas (IoT). Los objetivos planteados al inicio de esta tesis se cumplieron exitosamente y se listan a continuación:

Tal como se demuestra en el capítulo 5, este trabajo cumple con el objetivo de **desarrollar una aplicación programada en java para el control y monitoreo inalámbrico de sistemas digitales**; compatible con Bluetooth BR/EDR, Bluetooth LE y Wi-Fi.

Como lo demuestran las pruebas de protocolo de comunicaciones, se cumplió el objetivo de **desarrollar un protocolo de Capa de Aplicación robusto y sencillo**, que permite el intercambio de información entre el dispositivo móvil y el sistema a base de MCU. Al diseñar el protocolo de Capa de Aplicación ATC se tomó en cuenta los diferentes niveles de conocimiento de los posibles usuarios de la aplicación, desde aquel estudiante sin conocimientos de electrónica; hasta una persona con estudios en sistemas digitales. Usando este protocolo, el usuario puede empezar a comunicarse con la aplicación usando mensajes tan simples como un solo carácter; o bien, enviando información tan compleja como una cadena de reconocimiento de voz mediante ATC tags. También es importante resaltar que este protocolo está abierto a la declaración de más “tags”, conforme las nuevas funcionalidades de la aplicación lo requiera.

El capítulo 5 pone de manifiesto la flexibilidad de la aplicación; al utilizar los elementos constructivos definidos en el capítulo 2, para controlar los distintos prototipos compatibles con ATC, cumpliendo con el objetivo de desarrollar una **aplicación personalizable**. Después de las funciones de comunicaciones inalámbricas, se puede considerar la sección de edición de layout como la parte más importante de la aplicación, ya que es la que le permite al usuario de la aplicación (aquel que edita el layout y programa el sistema digital en cuestión) y al usuario final (aquel que opera el sistema digital en cuestión) establecer el vínculo entre lo que se observa en el dispositivo móvil y el mundo real.

El hardware y software mínimo de un sistema a base de MCU, para cada tecnología de comunicaciones inalámbrica soportada, se expone en el capítulo 4. Esto cumple con el objetivo de proporcionarle al usuario un marco inicial de trabajo con un **circuito a base de microcontrolador** capaz de codificar y decodificar la información proveniente de y hacia la aplicación; con la función de operar entradas y salidas binarias y analógicas.

Por último, todos los “sketches” o programas en Arduino, citados en esta tesis, se encuentran publicados en el **repositorio de GitHub** “ATC release codes” (ver GitHub JuanLuisGonzalez/ATC-Release-Codes), cumpliendo con el último objetivo secundario de este trabajo. Es importante contar con un repositorio oficial para el almacenamiento y distribución de los sketches compatibles con la aplicación, ya que permite tener control de las revisiones de los sketches relativas a las actualizaciones de la aplicación.

Apéndices

Apéndice A. Objeto Text to Speech en ATC

La clase *TextToSpeech* en Android sintetiza voz a partir de texto para reproducirlo inmediatamente o para crear un archivo de sonido. Una instancia de *TextToSpeech* puede usarse para sintetizar voz una vez que su proceso de inicialización ha sido completado. El *callback “*TextToSpeech.OnInitListener” debe ser implementado en Main para que la actividad principal sea notificada de que la inicialización se completó.

Antes de intentar inicializar la instancia del sintetizador de voz, la aplicación solicita al sistema operativo información relativa al estado de instalación de un sintetizador de voz, con la acción ACTION_CHECK_TTS_DATA, como se muestra en la Figura A.1.

```
// start up speech engine
try {
    Intent checkIntent = new Intent();
    checkIntent.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
    startActivityForResult(checkIntent, MY_DATA_CHECK_CODE);
} catch (Exception e) {
    e.printStackTrace();
    DisplayToast("Please install any Text to Speech Engine");
}
```

Figura A.1. Preparación preliminar del sintetizador de voz.

Si el resultado de la actividad es CHECK_VOICE_DATA_PASS, se inicializará la instancia *TextToSpeech*, de lo contrario se iniciará una actividad para que el usuario instale un sintetizador de voz compatible, usando la acción ACTION_INSTALL_TTS_DATA como se muestra en la Figura A.2.

```
case MY_DATA_CHECK_CODE:
    if (resultCode == TextToSpeech.Engine.CHECK_VOICE_DATA_PASS) {
        // success, create the TTS instance
        mTts = new TextToSpeech(getApplicationContext(), this);
    } else {
        // missing data, install it
        Intent installIntent = new Intent();
        installIntent.setAction(
                TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
        startActivity(installIntent);
    }
    break;
```

Figura A.2. Solicitud de instalación de sintetizador de voz.

La Figura A.3 muestra el callback “*onInit*”, el cual es llamado tras inicializar la instancia TextToSpeech, *mTts*. Si mTts se inicializo correctamente, el programa hará un chequeo de la disponibilidad del idioma inglés, ya que el protocolo ATC está diseñado para sintetizar el texto ya sea que el usuario lo haya enviado en inglés o en el idioma por defecto del teléfono.

Cuando la aplicación es terminada, es importante liberar los recursos usados por la instancia TextToSpeech, llamando la función *shutdown()* en el método *onDestroy()*.

```java
@Override
public void onInit(int status) {
    if (mTts != null) {
        if (mTts.isLanguageAvailable(Locale.US) == TextToSpeech.LANG_MISSING_DATA) {
            // missing data
            isEnglishAvailable = false;
            DisplayToast("English not available for Text to Speech");
        } else
            isEnglishAvailable = true;
    } else {
        isEnglishAvailable = false;
        DisplayToast("Please install any Text to Speech Engine");
    }
}
```

Figura A.3. Verificación de inicialización de mTts.

Cuando la aplicación recibe un ATC tag, y éste es equivalente a un TTS tag, se ejecutará la función “speak” de la instancia de TextToSpeech en función del lenguaje seleccionado, 0 para inglés y 1 para el lenguaje por defecto del dispositivo móvil como se muestra en la Figura A.4.

```java
// If text to speech to be displayed
else if (tag.equals(TTS_TAG)) {
    if (mTts != null) {
        // Use english if tag number eq 0
        if (tagNumber == 0)
            mTts.setLanguage(Locale.US);
        else
            mTts.setLanguage(Locale.getDefault());
        mTts.speak(message.substring(8), TextToSpeech.QUEUE_FLUSH, null);
    }
}
```

Figura A.4. Ejecución de la función "speak" del sintetizador de voz.

Apéndice B. Conexión fallida a módulo kc-4114

Para realizar el desarrollo de la conectividad Bluetooth LE, además de usar el circuito HM10, se realizaron pruebas con el módulo kc-4114, que está basado en el chipset CSR1010. Las pruebas de conexión demostraron ser fallidas al no lograr una conexión estable con el módulo.

Figura B.1. Módulo KcEnergy Kc-4114.

Características del hardware

- Dimensiones: 21.0mm x 15.0mm x 2.3mm.
- *CSR1010 µEnergy Chipset.*
- Rango 200m.
- *+7.5dBm Transmitter.*
- *-92.5dBm Receiver Sensitivity.*
- *12 Digital Programmable I/O Pins.*
- *3 Analog Programmable I/O Pins.*
- *UART Interface.*
- *Onboard Antenna Port.*
- *PWM Drivers.*

Características del firmware

- *Bluetooth v4.0 Low Energy (Single Mode).*
- *Wireless Data Communications System.*
- *Wireless Firmware Updating.*
- *Easy to Use AT Command Interface Using UART.*
- *Up to 80Kbps data transfer.*
- *Cuztom Firmware Available.*

Al igual que los módulos Bluetooth HC-05 y Hc-06, esta tarjeta se configura utilizando comandos “AT”. La lista de comandos AT, según el manual, se muestra en la Tabla B.1:

Addr	Dbug	Link	Pio
Aio	Dtim	Mem	Role
Bat	Hid	Meas	Rset
Bcon	I2c	Mode	Rssi
Coms	Idle	Name	Uart
Conn	Info	Pair	Pio

Tabla B.1. Comandos AT módulo Kc Energy.

El envío de un comando debe ser seguido por el carácter de nueva línea para indicar al módulo que se ha enviado un comando AT. El detalle de cada uno de los comandos se puede encontrar en el *KcEnergy User Manual V0.14*.

Para enviar y recibir información continua entre el módulo Bluetooth LE y el dispositivo Android se utiliza el servicio COMS y la característica Data Transfer como se muestra en la Tabla B.2.

Service			UUID
COMS			0000F100-0000-AAAA-BBBB-CCCCDDDDEEEE
Characteristic	Bytes	Type	UUID
Data Transfer	20	String	0000F101-0000-AAAA-BBBB-CCCCDDDDEEEE

Tabla B.2. UUIDs de características y servicios de un módulo kc4114.

Interfaz a microcontrolador

Al igual que los módulos HC-05, HC-06, y ESP8266, el módulo KcEnergy se comunica con el microcontrolador en cuestión por transmisión serial de información. La configuración por defecto de la interfaz serial es:

Baud Rate: 115200 bits / segundo

Número de Bits: 8 bits

Número de Bits de parada: 1 Bit

Bit de Paridad: Deshabilitado

Nivel de voltaje: 0 – 3.3v (Se utilizó un divisor de voltaje resistivo para convertir la señal del transmisor del Arduino al receptor del módulo).

Para que el módulo KcEnergy funcione de forma similar al HC-05 y HC-06, es decir como un simple Gateway por donde se envía y recibe la información proveniente del dispositivo Inteligente, es necesario enviar el comando **"AT Coms 0"**, por medio del puerto serial una vez que se ha iniciado la comunicación entre el sistema operado por Android y el Módulo BLE.

Pruebas de funcionamiento

Antes de conectar el módulo BLE a una tarjeta de microcontrolador, se realizaron pruebas utilizando un convertidor de USB a Serial TTL como el mostrado en la Figura B.2.

Figura B.2. Convertidor Serial a TTL basado en circuito FTDI.

Las pruebas realizadas utilizando el módulo convertidor serial y módulo BLE se detallan en la Tabla B.3.

Prueba	Resultado
Uso del comando ***AT Addr Get*** para obtener la Dirección MAC del dispositivo	Addr 646E6CC482D6
Uso del comando ***AT Info Get***	cmd info kcEnergy v0.18 Std Date May 3 2016 BLE v4.1 GATT Hw:1.0 Sw:0.17
Uso del comando ***AT Reset Set***	cmd rset kcEnergy v0.18 by KC Wirefree 646E6CC482D6 Server Role Command Mode [fast] [bonded]

Uso del comando ***AT Coms Set 0*** Para poner el módulo en envío y recepción de información de forma continua	cmd coms Después de mandar este comando el módulo no responde a comandos AT posteriores.
Salir de modo de datos continuos, envío de **"~~~"**	ConnDn [fast] [bonded] cmd info kcEnergy v0.18 Std Date May 3 2016 BLE v4.1 GATT Hw:1.0 Sw:0.17 El módulo vuelve a reconocer comandos AT posteriores
Conexión al módulo Bluetooth usando Aplicación Demo *BluetoothLEGatt*	El módulo aparece listado en la aplicación y muestra cada uno de los servicios disponibles (ver Figura B.3). Sin embargo, a pesar de que se logra la conexión con el módulo, esta es inestable, es decir, se conecta y se desconecta de forma continua.

Tabla B.3. Pruebas preliminares al desarrollo de los programas en Arduino y Android.

Figura B.3. Módulo Bluetooth BLE mostrado en el servicio de "escaneo" en la aplicación.

Apéndice C. Comandos AT para módulos HC-05 y HC-06

Comandos AT para HC-06

Parámetros por defecto. Baud rate: 9600N81, ID: linvor, Password:1234

Comando	Función	Respuesta	Parámetros
AT	Prueba de comunicaciones	OK	NA
AT+BAUD1	Cambiar baud rate	OK1200	1– 1200 2– 2400 3– 4800 4– 9600 (Default) 5– 19200 6– 38400 7– 57600 8– 115200 9– 230400 A– 460800 B— 921600 C— 1382400
AT+NAMEname	Cambiar nombre	OKname	"name" o el nuevo nombre del dispositivo (limitado a 20 caracteres)
AT+PINxxxx	Cambiar contraseña de emparejamiento	OKsetpin	"xxxx" la nueva contraseña, un número de 4 dígitos.
AT+PN	Deshabilitar bit de paridad	OK NONE	NA
AT+PO	Configurar bit de paridad impar	OK ODD	NA
AT+PE	Configurar bit de paridad par	OK EVEN	NA

Tabla C.1. Comandos AT en un módulo Bluetooth esclavo HC-06.

Los comandos AT en el módulo HC06 no requieren ningún terminador (sin carácter de nueva línea '\n' ni de retorno de carro '\r').

Comandos AT para HC05

La Tabla C.2 muestra los comandos AT necesarios para el uso del módulo HC05 en modo esclavo. Los parámetros por defecto de un HC05 son:

- Device type: 0.
- Inquire code: 0x009e8b33.
- Module work mode: Slave Mode.
- Connection mode: Connect to the Bluetooth device specified.
- Serial parameter: Baud rate: 38400 bits/s; Stop bit: 1 bit; Parity bit: None.
- Passkey: "1234".
- Device name: "HC-05".

Comando	Función	Respuesta	Parámetros
AT	Prueba	OK	NA
AT+RESET	Reset	OK	NA
AT+VERSION?	Obtener la versión del software	+VERSION: <Param> OK	Param: Número de versión
AT+ORGL	Restablecer estado de fabrica	OK	NA
AT+ADDR?	Obtener la dirección del módulo Bluetooth	+ADDR: <Param> OK	Param: Nombre del dispositivo Bluetooth

AT+NAME=<Param>	Cambiar nombre del dispositivo	OK	Param: Nombre del dispositivo Bluetooth
AT+NAME?	Solicitar nombre del dispositivo	+NAME:<Param>	Param: Nombre del dispositivo Bluetooth
AT+PSWD=<Param>	Cambiar contraseña de emparejamiento	OK	Param: Contraseña Contraseña por defecto: "1234"
AT+UART=<Param>,<Param2>,<Param3>		OK	Param1: baud rate (bits/s). El valor deberá ser uno de los siguientes: 4800, 9600, 19200, 38400, 57600, 115200, 23400, 460800, 921600, 1382400. Param2: stop bit: 0->1 bit, 1->2 bits Param3: parity bit 0->None, 1->Odd parity, 2->Even parity. Valores por defecto: 9600, 0, 0

Tabla C.2. Comandos AT para un módulo Bluetooth maestro esclavo HC05.

Los comandos AT en el módulo HC05 van seguidos por el carácter de nueva línea '\n' y de retorno de carro '\r'.

Apéndice D. Comandos AT para HM10

Los comandos necesarios para configurar un módulo HM10 como Bluetooth LE esclavo se listan en la Tabla D.1. Para el conjunto completo de comandos AT consultar la hoja de datos del módulo Bluetooth HM10 [54]. Los parámetros por defecto del módulo HM10 son:

- Baud rate: 0 (9600 bit/s).
- Bits de paridad: 0 (ninguno).
- Un bit de parada: 0 (un bit de parada).
- Nombre: HMSoft.
- Contraseña: 000000.

Comando	Función	Respuesta	Parámetros
AT	Comando de prueba	OK	NA
AT+ADDR?	Solicitar dirección del dispositivo	OK+ADDR:MAC Address	NA
AT+BAUD?	Cambiar baud rate del UART	OK+Get:[para1]	Para1: Baud rate en bit/s 0 - 9600 1 - 19200 2 - 38400 3 - 57600 4 -115200 5 - 4800 6 - 2400 7 - 1200 8 - 230400
AT+NAME[para1]	Cambiar nombre del dispositivo	OK+Set[para1]	Para1: Nombre del módulo (máximo 12 caracteres)
AT+PARI[para1]	Cambiar bit de paridad	OK+Set:[para1]	Para1: 0,1,2 0:None, 1:EVEN, 2:ODD
AT+PIN[para1]	Cambiar contraseña de emparejamiento	OK+Set:[para1]	Para1 es la contraseña de 000000 a 999999
AT+STOP[para1]	Cambiar bits de parada	OK+Set:[para1]	Para1: 0, 1 0: One stop bit, 1: Two stop bit

Tabla D.1. Comandos AT de un módulo Bluetooth LE HM10.

Los comandos AT en el módulo HM10 van seguidos por el carácter de nueva línea '\n' y de retorno de carro '\r'.

Apéndice E. Comandos AT para ESP8266

La Tabla E.1 lista los comandos AT necesarios para configurar un módulo ESP8266 como un servidor TCP/IP y comunicarse con la aplicación. Para el conjunto completo de comandos AT consultar la guía ESP8266 AT Instrucción Set [55]. Un módulo ESP8266 está configurado por defecto a un baud rate de 115200 bits/s, sin bits de paridad y un bit de parada.

Comando	Función	Respuesta	Parámetros
AT	Prueba de funcionamiento	OK	NA
AT+RST	Reinicia la tarjeta	OK	NA
AT+CWLAP	Enlista todas las redes disponibles	+CWLAP: (3, "INFINITUMtcyv", -92, "f8:35:dd:20:4d:e4", 1)	Información de las redes disponibles
AT+GMR	Retorna la versión de firmware del módulo	Firmware XX	NA
AT+CWMODE=Para1	Configura el modo de comunicación del módulo	OK	Para1: 1, 2, 3 1 - Client 2 - Access Point 3 - Cliente y Access Point
AT+CWJAP=Para1, Para2	Conecta el módulo a la red "Para1" usando la contraseña "Para2"	Connected	Para1 - Nombre de la red Para2 - Contraseña de la red
AT+CIOBAUD=Para1	Cambiar baud rate del UART	OK	Para1 = Baud rate de 110 a 115200 bits / seg
AT+CIFSR	Solicitar dirección IP asignada	Dirección IP	NA
AT+CIPMUX=<mode>	Configurar conexión	OK	Mode: 0, 1 0 - Single Connection 1 - Multi-Channel Connection
AT+CIPSERVER=<mode>, <port>	Abrir y cerrar server socket	OK	Mode: 0, 1 0 - Cerrar el Socket Server 1 - Abrir el Socket Server Port: 0…65535
+IPD,Para1, Para2: Para3	Recepción de un número de "Para2" de datos del canal "Para1". La carga útil está contenida en "Para3".	NA	Para1: Canal del socket TCP/IP Para2: Tamaño de la carga útil Para3: Carga útil.
AT+CIPSEND=Para1, Para2	Envió de un número de datos "Para2" sobre el canal TCP/IP definido por "Para1".	Sent OK	Para1: Canal del socket TCP/IP Para2: Tamaño de la carga útil a enviar.

Tabla E.1. Comandos AT de un módulo Wi-Fi ESP8266.

Los comandos AT en el módulo ESP8266 van seguidos por el carácter de nueva línea '\n' y de retorno de carro '\r'.

Bibliografía

[1] I. O. f. Standarization, «ISO 9241-110:2006». 2006.

[2] Wikipedia, «User Interface,» 2016. [En línea]. Available: https://en.wikipedia.org/wiki/User_interface.

[3] Copadata, «Interfaz Hombre Máquina,» 2016. [En línea]. Available: https://www.copadata.com/es-mx/soluciones-hmi-scada/interfaz-hombre-maquina-hmi/.

[4] Rockwell Automation, «Human Machine Interface,» 2016. [En línea]. Available: http://www.rockwellautomation.com/rockwellsoftware/hmi.page.

[5] Google, «Android - About,» 2016. [En línea]. Available: https://developer.android.com/about/android.html.

[6] B. Elgin, «Google Buys Android for Its Mobile Arsenal,» Tech-Insider, 2005. [En línea]. Available: http://tech-insider.org/mobile/research/2005/0817.html.

[7] Business Insider UK, «Android Market Share,» 2016. [En línea]. Available: http://uk.businessinsider.com/apple-ios-v-android-market-share-2016-1.

[8] Google, «Touch Devices | Android Open Source,» [En línea]. Available: http://source.android.com/devices/input/touch-devices.html. [Último acceso: 2017].

[9] Google, «The Android Source Code,» 2016. [En línea]. Available: http://source.android.com/source/index.html.

[10] statista.com, «Number of apps available in leading app stores as of March 2017,» 03 2017. [En línea]. Available: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

[11] Oracle, «JAR File specification,» 2010. [En línea]. Available: http://docs.oracle.com/javase/1.5.0/docs/guide/jar/jar.html.

[12] Google, «Android History,» 2016. [En línea]. Available: https://www.android.com/intl/es-419_mx/history/.

[13] Androidpolice.com, «"CyanogenMod Has Now Been Installed On Over 2 Million Devices, Doubles Install Numbers Since January,» 2012. [En línea]. Available: http://www.androidpolice.com/2012/05/28/cyanogenmod-has-been-installed-over-2-million-times-doubles-install-numbers-since-january/.

[14] F. E. Valdés, Microcontroladores: fundamentos y aplicaciones con PIC., Marcombo, 2007.

[15] Wikipedia, «Microcontroller,» 2017. [En línea]. Available: https://en.wikipedia.org/wiki/Microcontroller.

[16] Semico, «Momentum Carries MCUs Into 2011,» 2011. [En línea]. Available: http://semico.com/content/momentum-carries-mcus-2011.

[17] ATMEL, ATMEGA328 Datasheet, 2015.

[18] Arduino.cc, «What is Arduino?,» 2016. [En línea]. Available: https://www.arduino.cc/en/Guide/Introduction# .

[19] Arduino AG, «Arduino Products,» 2017. [En línea]. Available: https://www.arduino.cc/en/Main/Products.

[20] Blync Inc., «Blync Details,» 2017. [En línea]. Available: https://play.google.com/store/apps/details?id=cc.blynk.

[21] A. Smirnov, «Arduino Commander Details,» 2016. [En línea]. Available: https://play.google.com/store/apps/details?id=name.antonsmirnov.android.arduinocommander.

[22] Qualcomm, «BlueCore4-Ext,» 2015. [En línea]. Available: http://www.csr.com/products/29/bluecore4-ext.

[23] Texas Instruments, «CC2540 SimpleLink Bluetooth low energy wireless MCU with USB,» 2016. [En línea]. Available: http://www.ti.com/product/CC2540.

[24] Arduino CC, «Arduino Yun,» 2016. [En línea]. Available: https://www.arduino.cc/en/Main/ArduinoBoardYun.

[25] Arduino CC, «Ethernet Shield,» 2016. [En línea]. Available: https://www.arduino.cc/en/Main/ArduinoEthernetShieldV1.

[26] Espressif Systems, *ESPRESSIF SMART CONNECTIVITY PLATFORM: ESP8266,* 2013.

[27] I. O. f. Standarization, «ISO/IEC 7489-1:1994». 1994.

[28] Wikipedia, «Bluetooth,» [En línea]. Available: http://es.wikipedia.org/wiki/Bluetooth. [Último acceso: 16 3 2017].

[29] Bluetoot SIG, «Bluetooth-How it works,» 2017. [En línea]. Available: https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works. [Último acceso: 16 03 2017].

[30] Bluetooth SIG, «Bluetooth 5,» 2017. [En línea]. Available: https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/bluetooth5.

[31] Bluetooth SIG, «Bluetooth Core Specification,» 2017. [En línea]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification.

[32] Wikipedia, «Bluetooth Low Energy,» 2017. [En línea]. Available: https://en.wikipedia.org/wiki/Bluetooth_low_energy.

[33] Bluetooth SIG, «Bluetooth LE,» 2017. [En línea]. Available: https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/low-energy.

[34] Wikipedia, «Wi-Fi,» 2017. [En línea]. Available: https://en.wikipedia.org/wiki/Wi-Fi.

[35] Wi-Fi Alliance, «Who we are,» 2017. [En línea]. Available: http://www.wi-fi.org/who-we-are.

[36] IEEE, «802 IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture,» 2014.

[37] O. A. Sanchez Soto, Comparación de la eficiencia volumétrica entre redes inalámbricas Wi-Fi y WiMAX, Universidad Nacional Autonoma de Mexico, 2011.

[38] Wikipedia, «IEEE 802.11,» 2017. [En línea]. Available: https://en.wikipedia.org/wiki/IEEE_802.11.

[39] Google, «Aspectos fundamentales de la Aplicacion en Android,» 2017. [En línea]. Available: https://developer.android.com/guide/components/fundamentals.html?hl=es.

[40] Oracle, «What is a class?,» 2017. [En línea]. Available: https://docs.oracle.com/javase/tutorial/java/concepts/class.html.

[41] Google, «Public Class Activity,» [En línea]. Available: https://developer.android.com/reference/android/app/Activity.html. [Último acceso: 2017].

[42] Google, «ViewGroup,» [En línea]. Available: https://developer.android.com/reference/android/view/ViewGroup.html. [Último acceso: 2017].

[43] Oracle, «Class Thread,» 2016. [En línea]. Available: https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html. [Último acceso: 2017].

[44] Google, «AsyncTask,» [En línea]. Available: https://developer.android.com/reference/android/os/AsyncTask.html. [Último acceso: 2017].

[45] Google, «Bluetooth Api Guide,» [En línea]. Available: https://developer.android.com/guide/topics/connectivity/bluetooth.html?hl=es-419#TheBasics. [Último acceso: 2017].

[46] IETF, «Requirements for Internet Hosts,» 1989. [En línea]. Available: https://tools.ietf.org/html/rfc1122.

[47] IBM, «Protocolos TCP/IP,» [En línea]. Available: https://www.ibm.com/support/knowledgecenter/es/ssw_aix_72/com.ibm.aix.networkcomm/tcpip_protocols.htm. [Último acceso: 2017].

[48] W. Tomasi, Sistemas de comunicaciones electrónicas, Pearson Eductation, 2003.

[49] Wlikipedia (English), «UART,» 9 03 2017. [En línea]. Available: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter.

[50] F. C. H. P. David Calcutt, 8051 Microcontroller: An Applications Based Introduction, Newnes, 2003.

[51] Mouser Electronics, «LP2985-33DBVR,» [En línea]. Available: http://www.mouser.mx/Search/ProductDetail.aspx?qs=paYhMW8qfiswyUdMjD%2FxlQ%3D%3D. [Último acceso: 2017].

[52] Domotz Ltd, «Fing - Escáner de red,» 14 04 2017. [En línea]. Available: https://play.google.com/store/apps/details?id=com.overlook.android.fing&hl=es_419.

[53] Google, «Location and Sensors,» [En línea]. Available: https://developer.android.com/guide/topics/sensors/sensors_overview.html. [Último acceso: 06 2017].

[54] JNHuaMao Technology Company, «Bluetooth 4.0 BLE module datasheet,» Shandong, China, 2014.

[55] Espressif Systems IOT Team , «ESP8266 AT Instruction Set,» 2016.

[56] Google, «Android Studio,» 2016. [En línea]. Available: https://developer.android.com/studio/features.html.

[57] J. T. Girones, El gran libro de Android, Editorial Alfaomega, 2015.

[58] Arduino AG, «What is Arduino?,» 2017. [En línea]. Available: https://www.arduino.cc/en/Guide/Introduction#.

[59] Bluetooth SIG, Bluetooth Core Specification v5.0, 2016.

[60] Bluetooth SIG, «Service Discovery,» 2017. [En línea]. Available: https://www.bluetooth.com/specifications/assigned-numbers/service-discovery.

Anexos

Anexo 1. Sketch bt_firmware.ino

```
/*
Author: Juan Luis Gonzalez Bello
Date: June 2017
Get the app: https://play.google.com/store/apps/details?id=com.apps.emim.btrelaycontrol
** After copy-paste of this code, use Tools -> Atomatic Format
*/

// Baud rate for bluetooth module
// (Default 9600 for most modules)
#define BAUD_RATE 9600 // Tip: Configure your bluetooth device for 115200 for better performance

// Special commands
#define CMD_SPECIAL '<'
#define CMD_ALIVE   '['

// Return values for special command
#define UPDATE_FAIL  0
#define UPDATE_ACCEL 1
#define UPDATE_SKBAR 2
#define UPDATE_TPAD  3
#define UPDATE_SPCH  4

// Data and variables received from especial command
int Accel[3] = {0, 0, 0};
int SeekBarValue[8] = {0,0,0,0,0,0,0,0};
int TouchPadData[24][2]; // 24 max touch pad objects, each one has 2 axis (x and Y)
String SpeechRecorder = "";

void setup() {
  // initialize BT Serial port
  Serial.begin(BAUD_RATE);
}

void loop() {
  int appData;

  // =================================================================
  // This is the point were you get data from the App
  appData = Serial.read();   // Get a byte from app, if available
  switch(appData){
  case CMD_SPECIAL:
    // Special command received, seekbar value and accel value updates
    DecodeSpecialCommand();
    break;
```

```
   case CMD_ALIVE:
     // Character '[' is received every 2.5s
     Serial.println("ATC ready");
     break;
   }
   // ==================================================================
 }
/**
 * DecodeSpecialCommand
 * A '<' flags a special command comming from App. Use this function
 * to get Accelerometer data (and other sensors in the future)
 * Input:     None
 * Output:    None
 */
int DecodeSpecialCommand() {
  int tagType = UPDATE_FAIL;
  int isAccelData = -1;
  int isPadData = -1;

  // Read the whole command
  String thisCommand = Readln();

  // First 5 characters will tell us the command type
  String commandType = thisCommand.substring(0, 5);

  if (commandType.equals("AccX:"))
    isAccelData = 0;
  if (commandType.equals("AccY:"))
    isAccelData = 1;
  if (commandType.equals("AccZ:"))
    isAccelData = 2;

  if(isAccelData!= -1){
    // Next 6 characters will tell us the command data
    String commandData = thisCommand.substring(5, 11);
    if (commandData.charAt(0) == '-') // Negative acceleration
      Accel[isAccelData] = -commandData.substring(1, 6).toInt();
    else
      Accel[isAccelData] = commandData.substring(1, 6).toInt();
    tagType = UPDATE_ACCEL;
  }

  if (commandType.substring(0, 4).equals("PadX"))
    isPadData = 0;
  if (commandType.substring(0, 4).equals("PadY"))
    isPadData = 1;

  if(isPadData != -1){
    // Next 2 characters will tell us the touch pad number
    int padNumber = thisCommand.substring(4, 6).toInt();
    // Next 3 characters are the X axis data in the message
    String commandData = thisCommand.substring(8, 13);
    TouchPadData[padNumber][isPadData] = commandData.toInt();
    tagType = UPDATE_TPAD;
  }
```

```
  if (commandType.substring(0, 3).equals("Skb")) {
    // Next 6 characters will tell us the command data
    String commandData = thisCommand.substring(5, 11);
    int sbNumber = commandType.charAt(3) & ~0x30;
    SeekBarValue[sbNumber] = commandData.substring(1, 6).toInt();
    tagType = UPDATE_SKBAR;
  }

  if (commandType.equals("StoT:")) {
    // Next characters are the converted speech
    SpeechRecorder = thisCommand.substring(5, thisCommand.length() - 1);
    tagType = UPDATE_SPCH;
  }
  return tagType;
}

/**
 * Convert integer integer into 3 digit string
 */
String myIntToString(int number) {
  if (number < 10) {
    return "00" + String(number);
  }
  else if (number < 100) {
    return "0" + String(number);
  }
  else
    return String(number);
}
```

Anexo 2. Sketch eth_firmware.ino

```
/*
Author: Juan Luis Gonzalez Bello
Date: May 2017
Get the app: https://play.google.com/store/apps/details?id=com.apps.emim.btrelaycontrol
** After copy-paste of this code, use Tools -> Atomatic Format
*/

#include <SPI.h>
#include <Ethernet.h>

// Special commands
#define CMD_ALIVE   '['
#define CMD_SPECIAL '<'

// Signal to update special commands
#define UPDATE_FAIL  0
#define UPDATE_ACCEL 1
#define UPDATE_SKBAR 2
#define UPDATE_TPAD  3
#define UPDATE_SPCH  4

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,1,60);

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(80);
EthernetClient client;

// Data and variables received from especial command
int Accel[3] = {
  0, 0, 0};
int SeekBarValue[8] = {
  0,0,0,0,0,0,0,0};
int TouchPadData[24][2]; // 24 max touch pad objects, each one has 2 axis (x and Y)
String SpeechRecorder = "";

void setup() {
  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();

  // Initialize touch pad data,
  // this is to avoid having random numbers in them
  for (int i = 0; i < 24; i++) {
    TouchPadData[i][0] = 0; //X
    TouchPadData[i][1] = 0; //Y
  }
}
```

```
void loop() {
  int appData;

  // ============================================================
  // This is the point were you get data from the App
  client = server.available();
  if (client){
    appData = client.read();
  }

  switch(appData){
  case CMD_SPECIAL:
    // Special command received
    // After this function accel and seek bar values are updated
    DecodeSpecialCommand();
    break;

  case CMD_ALIVE:
    // Character '[' is received every 2.5s
    server.println("ATC ready");
    break;
  }
  // ============================================================
}

int DecodeSpecialCommand(){
  int tagType = UPDATE_FAIL;

  // Read the whole command
  String thisCommand = Readln();

  // First 5 characters will tell us the command type
  String commandType = thisCommand.substring(0, 5);

  if(commandType.equals("AccX:")){
    // Next 6 characters will tell us the command data
    String commandData = thisCommand.substring(5, 11);
    if(commandData.charAt(0) == '-') // Negative acceleration
      Accel[0] = -commandData.substring(1, 6).toInt();
    else
      Accel[0] = commandData.substring(1, 6).toInt();
    tagType = UPDATE_ACCEL;
  }

 if(commandType.equals("AccY:")){
   // Next 6 characters will tell us the command data
   String commandData = thisCommand.substring(5, 11);
   if(commandData.charAt(0) == '-') // Negative acceleration
     Accel[1] = -commandData.substring(1, 6).toInt();
   else
     Accel[1] = commandData.substring(1, 6).toInt();
   tagType = UPDATE_ACCEL;
 }
```

```
  if(commandType.equals("AccZ:")){
    // Next 6 characters will tell us the command data
    String commandData = thisCommand.substring(5, 11);
    if(commandData.charAt(0) == '-') // Negative acceleration
      Accel[2] = -commandData.substring(1, 6).toInt();
    else
      Accel[2] = commandData.substring(1, 6).toInt();
    tagType = UPDATE_ACCEL;
  }

  if(commandType.substring(0, 4).equals("PadX")){
    // Next 2 characters will tell us the touch pad number
    int padNumber = thisCommand.substring(4, 6).toInt();
    // Next 3 characters are the X axis data in the message
    String commandData = thisCommand.substring(8, 13);
    TouchPadData[padNumber][0] = commandData.toInt();
    tagType = UPDATE_TPAD;
  }

  if(commandType.substring(0, 4).equals("PadY")){
    // Next 2 characters will tell us the touch pad number
    int padNumber = thisCommand.substring(4, 6).toInt();
    // Next 3 characters are the Y axis data in the message
    String commandData = thisCommand.substring(8, 13);
    TouchPadData[padNumber][1] = commandData.toInt();
    tagType = UPDATE_TPAD;
  }

  if(commandType.substring(0, 3).equals("Skb")){
    // Next 6 characters will tell us the command data
    String commandData = thisCommand.substring(5, 11);
    int sbNumber = commandType.charAt(3) & ~0x30;
    SeekBarValue[sbNumber] = commandData.substring(1, 6).toInt();
    tagType = UPDATE_SKBAR;
  }

  if (commandType.equals("StoT:")) {
    // Next characters are the converted speech
    SpeechRecorder = thisCommand.substring(5, thisCommand.length() - 1);
    tagType = UPDATE_SPCH;
  }
  return tagType;
}
```

```
// Readln
// Use this function to read a String line from Bluetooth
// returns: String message, note that this function will pause the program
//          until a hole line has been read.
String Readln(){
  char inByte = -1;
  String message = "";

  while(inByte != '\n'){
    inByte = -1;

    client = server.available();
    if (client)
      inByte = client.read();

    if(inByte != -1)
      message.concat(String(inByte));
  }

  return message;
}
```

Anexo 3. Sketch esp_firmware.ino

```
/*
Author: Juan Luis Gonzalez Bello
Date: May 2017
Get the app: https://play.google.com/store/apps/details?id=com.apps.emim.btrelaycontrol
** After copy-paste of this code, use Tools -> Atomatic Format
*/

#include <EEPROM.h>

// Baud rate for bluetooth module
// (Default 9600 for most modules)
#define BAUD_RATE 115200 // TIP> Set your bluetooth baud rate to 115200 for better performance
#define RX_PIN       0 // Check this out! this is to enable arduino pull up on rx pin, critical!
String sPort = "80";

// Special commands
#define CMD_ALIVE   '['
#define CMD_SPECIAL '<'

// Signal to update special commands
#define UPDATE_FAIL  0
#define UPDATE_ACCEL 1
#define UPDATE_SKBAR 2
#define UPDATE_TPAD  3
#define UPDATE_SPCH  4

// Data and variables received from especial command
int Accel[3] = {0, 0, 0};
int SeekBarValue[8] = {0, 0, 0, 0, 0, 0, 0, 0};
int TouchPadData[24][2]; // 24 max touch pad objects, each one has 2 axis (x and Y)
String SpeechRecorder = "";

void setup() {
  // Initialize touch pad data,
  // this is to avoid having random numbers in them
  for (int i = 0; i < 24; i++) {
    TouchPadData[i][0] = 0; //X
    TouchPadData[i][1] = 0; //Y
  }

  // initialize WiFI module;
  myESP_Init();

  // Greets on top of the app
  myESP_Println("ATC Expert ESP8266", 0);

  // Make the app talk in english (lang number 00, use 01 to talk in your default language)
  myESP_Println("<TtoS00: welcome to ATC expert", 0);
}
```

```
void loop() {
  int appData = 0;
  String appMessage = "";
  delay(1);

  // ==========================================================
  // This is the point were you get data from the App
  if (Serial.available() > 10) {
    appMessage = myESP_Read(0);    // Read channel 0
    appData = appMessage.charAt(0);

    switch (appData) {
      case CMD_SPECIAL:
        // Special command received
        DecodeSpecialCommand(appMessage.substring(1));
        break;

      case CMD_ALIVE:
        // Character '[' is received every 2.5s, use
        myESP_Println("ATC Expert ESP8266", 0);
        break;
    }
  }
}

int DecodeSpecialCommand(String thisCommand) {
  int tagType = UPDATE_FAIL;

  // First 5 characters will tell us the command type
  String commandType = thisCommand.substring(0, 5);

  if (commandType.equals("AccX:")) {
    // Next 6 characters will tell us the command data
    String commandData = thisCommand.substring(5, 11);
    if (commandData.charAt(0) == '-') // Negative acceleration
      Accel[0] = -commandData.substring(1, 6).toInt();
    else
      Accel[0] = commandData.substring(1, 6).toInt();
    tagType = UPDATE_ACCEL;
  }

  if (commandType.equals("AccY:")) {
    // Next 6 characters will tell us the command data
    String commandData = thisCommand.substring(5, 11);
    if (commandData.charAt(0) == '-') // Negative acceleration
      Accel[1] = -commandData.substring(1, 6).toInt();
    else
      Accel[1] = commandData.substring(1, 6).toInt();
    tagType = UPDATE_ACCEL;
  }
```

```
  if (commandType.equals("AccZ:")) {
    // Next 6 characters will tell us the command data
    String commandData = thisCommand.substring(5, 11);
    if (commandData.charAt(0) == '-') // Negative acceleration
      Accel[2] = -commandData.substring(1, 6).toInt();
    else
      Accel[2] = commandData.substring(1, 6).toInt();
    tagType = UPDATE_ACCEL;
  }

  if (commandType.substring(0, 4).equals("PadX")) {
    // Next 2 characters will tell us the touch pad number
    int padNumber = thisCommand.substring(4, 6).toInt();
    // Next 3 characters are the X axis data in the message
    String commandData = thisCommand.substring(8, 13);
    TouchPadData[padNumber][0] = commandData.toInt();
    tagType = UPDATE_TPAD;
  }

  if (commandType.substring(0, 4).equals("PadY")) {
    // Next 2 characters will tell us the touch pad number
    int padNumber = thisCommand.substring(4, 6).toInt();
    // Next 3 characters are the Y axis data in the message
    String commandData = thisCommand.substring(8, 13);
    TouchPadData[padNumber][1] = commandData.toInt();
    tagType = UPDATE_TPAD;
  }
   if (commandType.substring(0, 3).equals("Skb")) {
     // Next 6 characters will tell us the command data
     String commandData = thisCommand.substring(5, 11);
     int sbNumber = commandType.charAt(3) & ~0x30;
     SeekBarValue[sbNumber] = commandData.substring(1, 6).toInt();
     tagType = UPDATE_SKBAR;
   }

   if (commandType.equals("StoT:")) {
     // Next characters are the converted speech
     SpeechRecorder = thisCommand.substring(5, thisCommand.length() - 1);
     tagType = UPDATE_SPCH;
   }
   return tagType;
}
String ReadIn() {
  char inByte = -1;
  String message = "";
  int timeoutTimer = 0;

  while (inByte != '\n') {
    inByte = -1;

    if (Serial.available() > 0)
      inByte = Serial.read();
```

```
    if (inByte != -1)
      message.concat(String(inByte));

    // If we dont find a complete line after x time then abort
    if (timeoutTimer++ > 10000) {
      message = "";
      break;
    }
  }
/*
 Set up esp 8266 wifi module on Serial
 AT+CIPMUX=1
 AT+CIPSERVER=1,80
 */
void myESP_Init() {
  // Initialize serial port with pull up in rx
  Serial.begin(BAUD_RATE);
  pinMode(RX_PIN, INPUT_PULLUP);

  // Reset module
  Serial.println("AT+RST"); // Reset the board
  delay(5000);
  // Uncomment lines below to set up network for the first time
  //Serial.println("AT+CWMODE=3"); // both wifi client and access point
  //delay(100);
  //Serial.println("AT+CWJAP=\"linksys\",\"\""); // Join access point linksys, no password
  //delay(10000);
  Serial.println("AT+CIPMUX=1"); // Multichannel connection
  delay(500);
  Serial.println("AT+CIPSERVER=1," + sPort); // Open server socket at sPort
  delay(500);
}

// Send string data to app
// To send data AT+CIPSEND=0,X (0: channel, X data size)
void myESP_Println(String data, int channel) {
  // Get data size, add 2 for nl and cr
  int dataSize = data.length() + 2;

  // submit cipsend command for channel
  Serial.println("AT+CIPSEND=" + String(channel) + "," + String(dataSize));
  Serial.flush(); // wait transmision complete
  delay(3);       // wait esp module process

  // Print actual data
  Serial.println(data);
  Serial.flush(); // wait transmision complete
  delay(6);       // wait esp module process
}
```

```
// Send string data to app
// To send data AT+CIPSEND=0,X (0: channel, X data size)
void myESP_Print(String data, int channel) {
  int dataSize = data.length();

  // submit cipsend command for channel
  Serial.println("AT+CIPSEND=" + String(channel) + "," + String(dataSize));
  Serial.flush(); // wait transmision complete
  delay(3);       // wait esp module process

  // Print actual data
  Serial.print(data);
  Serial.flush(); // wait transmision complete
  delay(8);       // wait esp module process, normally used for big data string, wait more
}

// This read function is faster
// call this function when you have minimum 9 bytes available
// When data received: +IPD,0,2:[ (for this exaple 0 is the channel, and 2 is the data lenght)
String myESP_Read(int channel) {
  String message = "";
  int plusIndex = -1;
  String theIPD = "";
  String dataFromClient = "";

  // Read a complete line
  dataFromClient = Readln();

  // Find the '+'
  plusIndex = dataFromClient.indexOf('+');
  if (plusIndex == -1) return message;

  // If next 3 chars are IPD then this is a good command
  theIPD = dataFromClient.substring(plusIndex + 1, plusIndex + 4);
  if (theIPD.equals("IPD")) {
    // Extract message
    int twoPointsIndex = dataFromClient.indexOf(':');       // find the ':' on the command
    message = dataFromClient.substring(twoPointsIndex + 1); // set the message
  }

  return message;
}
```

Anexo 4. Sketch yun_firmware.ino

```
/*
Author: Juan Luis Gonzalez Bello
Date: June 2017
Get the app: https://play.google.com/store/apps/details?id=com.apps.emim.btrelaycontrol
** After copy-paste of this code, use Tools -> Atomatic Format
*/

#include <Bridge.h>
#include <YunServer.h>
#include <YunClient.h>

// Special commands
#define CMD_SPECIAL '<'
#define CMD_ALIVE   '['

// Signal to update special commands
#define UPDATE_FAIL  0
#define UPDATE_ACCEL 1
#define UPDATE_SKBAR 2
#define UPDATE_TPAD  3
#define UPDATE_SPCH  4

// Listen on default port 5555, the webserver on the Yun
// ip addres is asigned by your router
YunServer server;
YunClient client;

// Data and variables received from especial command
int Accel[3] = {0, 0, 0};
int SeekBarValue[8] = {0, 0, 0, 0, 0, 0, 0, 0};
int TouchPadData[24][2]; // 24 max touch pad objects, each one has 2 axis (x and Y)
String SpeechRecorder = "";

void setup() {
  // Bridge startup, set up comm between arduino an wifi module
  Bridge.begin();

  // Listen for incoming connections
  // Arduino acts as a TCP/IP server
  server.begin();
}

void loop() {
  int appData = -1;

  // ==============================================================
  // This is the point were you get data from the App
  // Get clients coming from server
  if (client) appData = client.read();  // Get a byte from app, if available
  else client = server.accept();        // If client available accept connection
```

```
  switch (appData) {
    case CMD_SPECIAL:
      // Special command received, seekbar value and accel value updates
      DecodeSpecialCommand();
      break;

    case CMD_ALIVE:
      // Character '[' is received every 2.5s
      server.println("ATC ready");
      break;
  }
  // ==========================================================
}

int DecodeSpecialCommand() {
  int tagType = UPDATE_FAIL;

  // Read the whole command
  String thisCommand = Readln();

  // First 5 characters will tell us the command type
  String commandType = thisCommand.substring(0, 5);

  if (commandType.equals("AccX:")) {
    // Next 6 characters will tell us the command data
    String commandData = thisCommand.substring(5, 11);
    if (commandData.charAt(0) == '-') // Negative acceleration
      Accel[0] = -commandData.substring(1, 6).toInt();
    else
      Accel[0] = commandData.substring(1, 6).toInt();
    tagType = UPDATE_ACCEL;
  }

  if (commandType.equals("AccY:")) {
    // Next 6 characters will tell us the command data
    String commandData = thisCommand.substring(5, 11);
    if (commandData.charAt(0) == '-') // Negative acceleration
      Accel[1] = -commandData.substring(1, 6).toInt();
    else
      Accel[1] = commandData.substring(1, 6).toInt();
    tagType = UPDATE_ACCEL;
  }

  if (commandType.equals("AccZ:")) {
    // Next 6 characters will tell us the command data
    String commandData = thisCommand.substring(5, 11);
    if (commandData.charAt(0) == '-') // Negative acceleration
      Accel[2] = -commandData.substring(1, 6).toInt();
    else
      Accel[2] = commandData.substring(1, 6).toInt();
    tagType = UPDATE_ACCEL;
  }
```

```
  if (commandType.substring(0, 4).equals("PadX")) {
    // Next 2 characters will tell us the touch pad number
    int padNumber = thisCommand.substring(4, 6).toInt();
    // Next 3 characters are the X axis data in the message
    String commandData = thisCommand.substring(8, 13);
    TouchPadData[padNumber][0] = commandData.toInt();
    tagType = UPDATE_TPAD;
  }

  if (commandType.substring(0, 4).equals("PadY")) {
    // Next 2 characters will tell us the touch pad number
    int padNumber = thisCommand.substring(4, 6).toInt();
    // Next 3 characters are the Y axis data in the message
    String commandData = thisCommand.substring(8, 13);
    TouchPadData[padNumber][1] = commandData.toInt();
    tagType = UPDATE_TPAD;
  }

  if (commandType.substring(0, 3).equals("Skb")) {
    // Next 6 characters will tell us the command data
    String commandData = thisCommand.substring(5, 11);
    int sbNumber = commandType.charAt(3) & ~0x30;
    SeekBarValue[sbNumber] = commandData.substring(1, 6).toInt();
    tagType = UPDATE_SKBAR;
  }

  if (commandType.equals("StoT:")) {
    // Next characters are the converted speech
    SpeechRecorder = thisCommand.substring(5, thisCommand.length() - 1);
    tagType = UPDATE_SPCH;
  }
  return tagType;
}

// ReadIn
// Use this function to read a String line from Bluetooth
// returns: String message, note that this function will pause the program
//           until a hole line has been read.
String ReadIn() {
  char inByte = -1;
  String message = "";

  while (inByte != '\n') {
    inByte = -1;

    if (client) inByte = client.read();
    else client = server.accept();

    if (inByte != -1)
      message.concat(String(inByte));
  }

  return message;
}
```

Anexo 5. Sketch ethernet_shield_test.ino

```
/*
 * Test for ethetnet shield
 * This program will repeat the information received into the serial monitor,
 * any info taped into the serial monitor will be sent to the ethetnet shield.
 */

#include <SPI.h>
#include <Ethernet.h>

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(172, 21, 117, 60);

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(80);
EthernetClient client;

void setup() {
  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.begin(115200);
}

void loop() {
  int appData;

  // ============================================================
  // This is the point were you get data from the App
  client = server.available();
  if (client) {
    appData = client.read();
    Serial.print((char)appData);
  }

  if (Serial.available()) {
    appData = Serial.read();
    server.print((char)appData);
  }
}
```

Anexo 6. Sketch eth_firmware_ap_layer_test.ino

Las funciones “DecodeSpecialCommand”, “ReadIn” y “myIntToString” se omiten por simplicidad en este anexo. Para el código de estas funciones ver el anexo 2.

```
/*
Author: Juan Luis Gonzalez Bello
Date: June 2017
Get the app: https://play.google.com/store/apps/details?id=com.apps.emim.btrelaycontrol
** After copy-paste of this code, use Tools -> Atomatic Format
*/

#include <SPI.h>
#include <Ethernet.h>

// Special commands
#define CMD_SPECIAL '<'
#define CMD_ALIVE   '['

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(172,21,117,60);

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(80);
EthernetClient client;

// Data and variables received from especial command
#define ACCEL_AXIS 3
#define SKBAR_NO   8
#define TPAD_NO    24
#define TPAD_AXIS  2

#define UPDATE_FAIL  0
#define UPDATE_ACCEL 1
#define UPDATE_SKBAR 2
#define UPDATE_TPAD  3
#define UPDATE_SPCH  4

int Accel[ACCEL_AXIS] = {
  0, 0, 0};
int SeekBarValue[SKBAR_NO] = {
  0,0,0,0,0,0,0,0};
int TouchPadData[TPAD_NO][TPAD_AXIS]; // 24 max touch pad objects, each one has 2 axis (x and Y)
String SpeechRecorder = "";
```

```
void setup() {
  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.begin(115200);

  // Initiate touch pad array
  for(int i = 0; i < TPAD_NO; i++){
    for(int j = 0; j < TPAD_AXIS; j++){
      TouchPadData[i][j] = 0;
    }
  }

  // Set up pins for LED
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
}

void loop() {
  int appData;
  int TagType = 0;

  // ============================================================
  // This is the point were you get data from the App
  client = server.available();
  if (client){
    appData = client.read();
    Serial.print((char)appData);
  }

  switch(appData){
  case 'A':
  digitalWrite(6, HIGH);
  server.println("<Butn02:1");
  server.println("<Imgs03:1");
  server.println("<Logr00: Pin 6 HIGH");
  break;

  case 'a':
  digitalWrite(6, LOW);
  server.println("<Butn02:0");
  server.println("<Imgs03:0");
  server.println("<Logr00: Pin 6 LOW");
  break;
```

```
  case CMD_SPECIAL:
    // Special command received
    // After this function accel and seek bar values are updated
    TagType = DecodeSpecialCommand();

    if(TagType == UPDATE_TPAD)
    // Print touch pad array
    for(int i = 0; i < TPAD_NO; i++){
      for(int j = 0; j < TPAD_AXIS; j++){
        if(TouchPadData[i][j] != 0) // don't display cero value pads
          Serial.println("MCU: Touchpad value: " + String(i) + ": " + String(TouchPadData[i][j]));
      }
    }

    if(TagType == UPDATE_SKBAR)
    // Print seekbar array
    for(int i = 0; i < SKBAR_NO; i++){
      Serial.println("MCU: Seekvar value: " + String(i) + ": " + String(SeekBarValue[i]));
    }

    if(TagType == UPDATE_ACCEL)
    // Print accel array
    for(int i = 0; i < ACCEL_AXIS; i++){
      Serial.println("MCU: Accelerometer value " + String(i) + ": "  + String(Accel[i]));
    }

    if(TagType == UPDATE_SPCH)
    // Print speech recognition
    Serial.println("MCU: " + SpeechRecorder);
    if(SpeechRecorder.equals("encender")){
      digitalWrite(5, HIGH);
      server.println("<Logr00: Pin 5 LOW");
    }
    else if(SpeechRecorder.equals("apagar")){
      digitalWrite(5, LOW);
      server.println("<Logr00: Pin 6 LOW");
    }
    break;

  case CMD_ALIVE:
    // Character '[' is received every 2.5s
    server.println("Hola Mundo");
    server.println("<Text00:Hola Mundo");
    server.println("<Text01: A1 = " + String(analogRead(A1)));
    //server.println("<Skb0:" + String(myIntToString(analogRead(A1)>>2)));
    server.println("<Abar01:" +  String(myIntToString(analogRead(A1)>>2)));
    server.println("<Abar00:" +  String(myIntToString(analogRead(A1)>>2)));
    server.println("<Alrm00");
    server.println("<Vibr00:100");
   // server.println("<TtoS01:Hola mundo");
    Serial.println("MCU: Hola Mundo");
    break;
  }
  // ============================================================
}
```

Anexo 7. Sketch bt_ATC_RC_CAR.ino

Las funciones “DecodeSpecialCommand”, “ReadIn” y “myIntToString” se omiten por simplicidad en este anexo. Para el código de estas funciones ver el anexo 1.

```
/*Bluetooth controller RC Car
 Bluetooth, Accelerometer, and voice controlled car.

 Author: Juan Luis Gonzalez Bello
 Date: June 2017
 */

// Baud rate for bluetooth module
// (Default 9600 for most modules)
#define BAUD_RATE 115200 // Tip: Configure your bluetooth device for 115200

// Special commands
#define CMD_SPECIAL '<'
#define CMD_ALIVE   '['

#define UPDATE_FAIL  0
#define UPDATE_ACCEL 1
#define UPDATE_SKBAR 2
#define UPDATE_TPAD  3
#define UPDATE_SPCH  4

// Data and variables received from especial command
int Accel[3] = {0, 0, 0};
int SeekBarValue[8] = {0, 0, 0, 0, 0, 0, 0, 0};
int TouchPadData[24][2]; // 24 max touch pad objects, each one has 2 axis (x and Y)
String SpeechRecorder = "";

// System specific outputs
#define O_IN1 2
#define O_IN2 3
#define O_IN3 4
#define O_IN4 5
#define O_ENB 6
#define O_FL  7
#define O_RL  8
#define O_LL  9

// System specific inputs
#define I_CURRENT A1
#define I_RX      0

// System specific commands
#define CMD_D       'A' // Adelante
#define CMD_N       'a' // Neutral
#define CMD_R       'B' // Reversa
#define CMD_RIGHT   'C'
#define CMD_CENTER  'c'
#define CMD_LEFT    'D'
#define CMD_LON     'E'
#define CMD_LOFF    'e'
#define CMD_ECO_ON  'F'
#define CMD_ACC_ON  'G'
#define CMD_ACC_OFF 'g'
```

```
String CMD_SLON  = "luces";
String CMD_SLOFF = "luces fuera";
String ID_RT     = "07";
String ID_LT     = "06";
String ID_ECO    = "05";
String ID_LIGHT  = "04";
String ID_BLIGHT = "00";

// System specific variables
boolean EcoMode     = false;
boolean AccelMode   = false;
boolean FrontLights = false;
int DriveCurrent = 0;
int DriveSpeed = 0;
#define ST_N  0
#define ST_D  1
#define ST_R  2
int DriveStatus = ST_N;

#define ST_NONE        0 // Apagadas
#define ST_BREAK       1 // Freno (ambas luces encendidas)
#define ST_TURN_LEFT   2 // Direccional izquierda (parpadeando)
#define ST_TURN_RIGHT  3 // Direccional derecha (parpadeando)
int DirLightStatus = ST_NONE;

void setup() {
  // initialize BT Serial port
  Serial.begin(BAUD_RATE);

  // Initialize inputs and outputs
  pinMode(O_IN1, OUTPUT);
  pinMode(O_IN2, OUTPUT);
  pinMode(O_IN3, OUTPUT);
  pinMode(O_IN4, OUTPUT);
  pinMode(O_ENB, OUTPUT);
  pinMode(O_RL, OUTPUT);
  pinMode(O_LL, OUTPUT);
  pinMode(O_FL, OUTPUT);
  pinMode(I_RX, INPUT_PULLUP);
  pinMode(I_CURRENT, INPUT);

  digitalWrite(O_IN1, LOW);
  digitalWrite(O_IN2, LOW);
  digitalWrite(O_IN3, LOW);
  digitalWrite(O_IN4, LOW);
  digitalWrite(O_LL, LOW);
  digitalWrite(O_RL, LOW);
  digitalWrite(O_FL, LOW);
  digitalWrite(O_ENB, HIGH);

  // Greet the app
  Serial.println("<Imgs" + ID_LIGHT + ":0");
  Serial.println("<TtoS01:Vehiculo ATC en linea");
}
```

```
void loop() {
  int tagType;
  int appData;
  delay(1);

  CurrentSensing();
  FrontLightsControl(FrontLights);
  RearLightSM(DirLightStatus);
  DriveSM(DriveStatus);

  // =================================================================
  // This is the point were you get data from the App
  appData = Serial.read();   // Get a byte from app, if available
  switch (appData) {
    case CMD_SPECIAL:
      // Special command received, seekbar value and accel value updates
      tagType = DecodeSpecialCommand();

      // Activate lights with voice
      if (tagType == UPDATE_SPCH) {
        if (SpeechRecorder.equals(CMD_SLON)) {
          FrontLights = true;
          Serial.println("<Imgs" + ID_LIGHT + ":1");
          Serial.println("<Butn" + ID_BLIGHT + ":1");
        }
        if (SpeechRecorder.equals(CMD_SLOFF)) {
          FrontLights = false;
          Serial.println("<Imgs" + ID_LIGHT + ":0");
          Serial.println("<Butn" + ID_BLIGHT + ":0");
        }
      }

      // Update drive speed value
      DriveSpeed = SeekBarValue[0];
      Serial.println("<Text03:" + myIntToString(DriveSpeed));

     // Enable accelerometer control
     if (AccelMode) {
       if (Accel[1] > 200) {
         digitalWrite(O_IN3, LOW);
         digitalWrite(O_IN4, HIGH);
         DirLightStatus = ST_TURN_LEFT;
       }
       else if (Accel[1] < -200) {
         digitalWrite(O_IN3, HIGH);
         digitalWrite(O_IN4, LOW);
         DirLightStatus = ST_TURN_RIGHT;
       }
       else {
         digitalWrite(O_IN3, LOW);
         digitalWrite(O_IN4, LOW);
         DirLightStatus = ST_NONE;
       }
     }
     break;
```

```
case CMD_ALIVE:
  // Character '[' is received every 2.5s
  Serial.println("ATC RC Car Ready");
  break;

case CMD_D:
  DriveStatus = ST_D;
  break;

case CMD_N:
  DriveStatus     = ST_NONE;
  DirLightStatus = ST_NONE;
  break;

case CMD_R:
  DriveStatus     = ST_R;
  DirLightStatus = ST_BREAK;
  break;

case CMD_RIGHT:
  digitalWrite(O_IN3, HIGH);
  digitalWrite(O_IN4, LOW);
  DirLightStatus = ST_TURN_RIGHT;
  break;

case CMD_LEFT:
  digitalWrite(O_IN3, LOW);
  digitalWrite(O_IN4, HIGH);
  DirLightStatus = ST_TURN_LEFT;
  break;

case CMD_CENTER:
  digitalWrite(O_IN3, LOW);
  digitalWrite(O_IN4, LOW);
  DirLightStatus = ST_NONE;
  break;

case CMD_LON:
  Serial.println("<TtoS01: encender luces");
  Serial.println("<Imgs" + ID_LIGHT + ":1");
  FrontLights = true;
  break;

case CMD_LOFF:
  Serial.println("<TtoS01: apagar luces");
  Serial.println("<Imgs" + ID_LIGHT + ":0");
  FrontLights = false;
  break;
```

```
    case CMD_ECO_ON:
      Serial.println("<Skb0:" + myIntToString(DriveSpeed));
      if (EcoMode) {
        Serial.println("<TtoS01: modo eco apagado");
        Serial.println("<Imgs" + ID_ECO + ":0");
        EcoMode = false;
      }
      else {
        Serial.println("<TtoS01: modo eco activado");
        Serial.println("<Imgs" + ID_ECO + ":1");
        EcoMode = true;
      }
      break;

    case CMD_ACC_ON:
      Serial.println("<TtoS01: modo acelerometro activado");
      AccelMode = true;
      break;

    case CMD_ACC_OFF:
      Serial.println("<TtoS01: modo acelerometro desactivado");
      AccelMode = false;
      break;
  }
  // =================================================================
}

/**
 * Current sensing
 */
void CurrentSensing(){
  static int prescaler = 0;

  if (prescaler++ > 100) {
    prescaler = 0;
    int analogValue = analogRead(I_CURRENT);
    DriveCurrent = ((((analogValue - 512) * 74) / 10) + DriveCurrent) >> 1; // Basic low pass (average) filter
    Serial.println("<Text00: Corriente: " + String(DriveCurrent / 100) + "."
                  + myIntToString((abs(DriveCurrent) % 100) * 10) + "A");
    Serial.println("<Text01: A1: " + String(analogValue));
    Serial.println("<Abar00:" + myIntToString(abs(DriveCurrent)));
  }
}

/**
 * Front lights control
 */
void FrontLightsControl(boolean lightStatus){
  // Front light control
  if (lightStatus) {
    digitalWrite(O_FL, HIGH);
  }
  else {
    digitalWrite(O_FL, LOW);
  }
}
```

```
/**
 * State machine for controlling drive power
 */
void DriveSM(int myStatus) {
  static int theSpeed = 0;
  static bool skipThis = false;

  switch (myStatus) {
    case ST_N:
      digitalWrite(O_IN1, LOW);
      digitalWrite(O_IN2, LOW);
      theSpeed = 0;
      break;

    case ST_D:
      digitalWrite(O_IN1, HIGH);
      digitalWrite(O_IN2, LOW);
      if (EcoMode) {
        if (skipThis) {
          if (theSpeed < DriveSpeed) theSpeed++;
          analogWrite(O_ENB, theSpeed);
        }
        skipThis = !skipThis;
      }
      else {
        analogWrite(O_ENB, DriveSpeed);
      }
      break;

    case ST_R:
      digitalWrite(O_IN1, LOW);
      digitalWrite(O_IN2, HIGH);
      theSpeed = 0;
      break;
  }
}

/**
 * State machine for rear light control (break or turn)
 */
void RearLightSM(int myStatus) {
  static int dir_prescaler = 0;
  static boolean onlyOnce = false;

  switch (myStatus) {
    case ST_NONE:
      digitalWrite(O_RL, LOW);
      digitalWrite(O_LL, LOW);
      if (onlyOnce) {
        Serial.println("<Imgs" + ID_RT + ":0");
        Serial.println("<Imgs" + ID_LT + ":0");
      }
      break;

    case ST_BREAK:
      digitalWrite(O_RL, HIGH);
      digitalWrite(O_LL, HIGH);
      break;
```

```
    case ST_TURN_LEFT:
      Serial.println("<Imgs" + ID_RT + ":0");
      digitalWrite(O_RL, LOW);
      dir_prescaler++;
      onlyOnce = true;
      if (dir_prescaler > 500) {
        dir_prescaler = 0;
        Serial.println("<Imgs" + ID_LT + ":" + String(digitalRead(O_LL) ? 0 : 1));
        digitalWrite(O_LL, digitalRead(O_LL) ? LOW : HIGH);
      }
      break;

    case ST_TURN_RIGHT:
      Serial.println("<Imgs" + ID_LT + ":0");
      digitalWrite(O_LL, LOW);
      dir_prescaler++;
      onlyOnce = true;
      if (dir_prescaler > 500) {
        dir_prescaler = 0;
        Serial.println("<Imgs" + ID_RT + ":" + String(digitalRead(O_RL) ? 0 : 1));
        digitalWrite(O_RL, digitalRead(O_RL) ? LOW : HIGH);
      }
      break;
  }
}
```

Anexo 8. Sketch bt_ATC_Walker.ino

Las funciones “DecodeSpecialCommand”, “Readln” y “myIntToString” se omiten por simplicidad en este anexo. Para el código de estas funciones ver el anexo 1.

```
/*ATC Walker
 An user programable 4 leg walker robot

 Connect BT module to Rx and Tx
 Connect Servomotors at ServoPINS[]

 Author: Juan Luis Gonzalez Bello
 Date: June 2017
 */

#include<Servo.h>
#include<EEPROM.h>

// Baud rate for ble module
#define BAUD_RATE 115200

// Special commands
#define CMD_SPECIAL '<'
#define CMD_ALIVE   '['

// Return values for special command
#define UPDATE_FAIL  0
#define UPDATE_ACCEL 1
#define UPDATE_SKBAR 2
#define UPDATE_TPAD  3
#define UPDATE_SPCH  4

// Data and variables received from especial command
int Accel[3] = {0, 0, 0};
int SeekBarValue[8] = {90,90,90,90,90,90,90,90};
int TouchPadData[24][2]; // 24 max touch pad objects, each one has 2 axis (x and Y)
String SpeechRecorder = "";

// Sistem specific variables
#define SERVO_NO 8
#define INIT_POS 90

// Atmega328p has 2k ram and 1k EEPROM.
// Constants for eeprom addresses
#define F_EE_ADD    0
#define F_EE_PADD 170
#define B_EE_ADD  200
#define B_EE_PADD 370
#define R_EE_ADD  400
#define R_EE_PADD 570
#define L_EE_ADD  600
#define L_EE_PADD 770
```

```
// Commands to save (s), play (p) and delete (d) positions
#define CMD_S_F  'A'
#define CMD_P_F  'a'
#define CMD_D_F  '1'
#define CMD_S_B  'B'
#define CMD_P_B  'b'
#define CMD_D_B  '2'
#define CMD_S_R  'C'
#define CMD_P_R  'c'
#define CMD_D_R  '3'
#define CMD_S_L  'D'
#define CMD_P_L  'd'
#define CMD_D_L  '4'
#define CMD_STOP 'S'
#define MAX_POSITIONS 10

// Robot walking status
#define ST_NONE 0
#define ST_F    1
#define ST_B    2
#define ST_R    3
#define ST_L    4

// Servo position limits
int MAX_POS[SERVO_NO] = {180, 144, 116, 144, 117, 157, 132, 126};
int MIN_POS[SERVO_NO] = {60,   54,   0,  57,  51,  50,  63,  31};

Servo mServo[SERVO_NO];
int ServoAngle[SERVO_NO];
int ServoPINS[SERVO_NO] = {2,3,4,5,6,7,8,9};
int ids[] = {16, 17, 18, 23}; // touch pad IDS
int RobotStatus = ST_NONE;

int FSpointer = 0;
int FPositions[MAX_POSITIONS][SERVO_NO];
int BSpointer = 0;
int BPositions[MAX_POSITIONS][SERVO_NO];
int LSpointer = 0;
int LPositions[MAX_POSITIONS][SERVO_NO];
int RSpointer = 0;
int RPositions[MAX_POSITIONS][SERVO_NO];

void mServoInit(){
  // Initialize servos
  for(int i = 0; i < SERVO_NO;i++){
    mServo[i].attach(ServoPINS[i]);
    ServoAngle[i] = INIT_POS;
    mServoSetPos(i, INIT_POS);
  }

  // Initialize touch pad value to 90 in order to avoid weird moves
  for(int i = 0; i < 24; i++){
    TouchPadData[i][0] = 90; //X
    TouchPadData[i][1] = 90; //Y
  }
```

```
  // Restore pointers from memory
  FSpointer = EEPROM.read(F_EE_PADD);
  if(FSpointer == 0xFF) FSpointer = 0;
  RSpointer = EEPROM.read(R_EE_PADD);
  if(RSpointer == 0xFF) FSpointer = 0;
  LSpointer = EEPROM.read(L_EE_PADD);
  if(LSpointer == 0xFF) FSpointer = 0;
  BSpointer = EEPROM.read(B_EE_PADD);
  if(BSpointer == 0xFF) BSpointer = 0;

  // Restore forward positions
  for(int i = 0; i < FSpointer; i++){
    for(int j = 0; j < SERVO_NO; j++){
      FPositions[i][j] = EEPROM.read(F_EE_ADD + (i * SERVO_NO) + j);
    }
  }

  // Restore backward positions
  for(int i = 0; i < BSpointer; i++){
    for(int j = 0; j < SERVO_NO; j++){
      BPositions[i][j] = EEPROM.read(B_EE_ADD + (i * SERVO_NO) + j);
    }
  }

  // Restore left positions
  for(int i = 0; i < LSpointer; i++){
    for(int j = 0; j < SERVO_NO; j++){
      LPositions[i][j] = EEPROM.read(L_EE_ADD + (i * SERVO_NO) + j);
    }
  }

  // Restore right positions
  for(int i = 0; i < RSpointer; i++){
    for(int j = 0; j < SERVO_NO; j++){
      RPositions[i][j] = EEPROM.read(R_EE_ADD + (i * SERVO_NO) + j);
    }
  }
}

void mServoSetPos(int servoNo, int value){
  if(value > MAX_POS[servoNo]) value = MAX_POS[servoNo];
  if(value < MIN_POS[servoNo]) value = MIN_POS[servoNo];
  mServo[servoNo].write(value);
  ServoAngle[servoNo] = value;
}

void FSavePos(){
  if(FSpointer >= MAX_POSITIONS){
    Serial.println("<TtoS01:Maximas posiciones alcanzadas");
    return;
  }
```

```
  for(int i = 0; i < SERVO_NO; i++){
    FPositions[FSpointer][i] = ServoAngle[i];
    EEPROM.write(F_EE_ADD + FSpointer * SERVO_NO + i, ServoAngle[i]);
  }
  EEPROM.write(F_EE_PADD, FSpointer++);
}

void BSavePos(){
  if(BSpointer >= MAX_POSITIONS){
    Serial.println("<TtoS01:Maximas posiciones alcanzadas");
    return;
  }

  for(int i = 0; i < SERVO_NO; i++){
    BPositions[BSpointer][i] = ServoAngle[i];
    EEPROM.write(B_EE_ADD + BSpointer * SERVO_NO + i, ServoAngle[i]);
  }
  EEPROM.write(B_EE_PADD, BSpointer++);
}

void RSavePos(){
  if(RSpointer >= MAX_POSITIONS){
    Serial.println("<TtoS01:Maximas posiciones alcanzadas");
    return;
  }
  for(int i = 0; i < SERVO_NO; i++){
    RPositions[RSpointer][i] = ServoAngle[i];
    EEPROM.write(R_EE_ADD + RSpointer * SERVO_NO + i, ServoAngle[i]);
  }
  EEPROM.write(R_EE_PADD, RSpointer++);
}

void LSavePos(){
  if(LSpointer >= MAX_POSITIONS){
    Serial.println("<TtoS01:Maximas posiciones alcanzadas");
    return;
  }

  for(int i = 0; i < SERVO_NO; i++){
    LPositions[LSpointer][i] = ServoAngle[i];
    EEPROM.write(L_EE_ADD + LSpointer * SERVO_NO + i, ServoAngle[i]);
  }
  EEPROM.write(L_EE_PADD, LSpointer++);
}

void Play(int aPointer, int aPosArray[][8]){
  if(aPointer == 0) return;
  for(int i = 0; i < aPointer; i++){
    for(int j = 0; j < SERVO_NO; j++){
      mServoSetPos(j, aPosArray[i][j]);
    }
    delay(500);
  }
}
```

```
void setup() {
  // initialize BT Serial port
  Serial.begin(BAUD_RATE);
  mServoInit();
  Serial.println("<TtoS01:caminante en linea");
  Serial.println("Caminante en linea");
}

void loop() {
  int appData;
  static int prescaler = 0;
  static int selector = 0;
  delay(1);

  // Print angles in app
  if(prescaler++ > 50){
    prescaler = 0;
    selector = (++selector) % 8;
    Serial.println("<Text0" + String(selector) + ":" + String(ServoAngle[selector]));
  }

  // Play the position arrays according to robot status
  switch(RobotStatus){
    case ST_F:
      Play(FSpointer, FPositions);
    break;

    case ST_B:
      Play(BSpointer, BPositions);
    break;
    case ST_R:
      Play(RSpointer, RPositions);
    break;

    case ST_L:
      Play(LSpointer, LPositions);
    break;
  }

  // ================================================================
  // This is the point were you get data from the App
  appData = Serial.read();   // Get a byte from app, if available
  switch(appData){
  case CMD_SPECIAL:
    // Special command received, seekbar value and accel value updates
    DecodeSpecialCommand();
    for(int i = 0; i < (SERVO_NO / 2); i++){
      mServoSetPos(i * 2, (TouchPadData[ids[i]][0]*9)/13);
      mServoSetPos((i * 2) + 1, (TouchPadData[ids[i]][1]*9)/13);
    }
    Serial.println("Touch pad received");
    break;
  case CMD_ALIVE:
    // Character '[' is received every 2.5s
    Serial.println("ATC Walker ready");
    break;
```

```
  case CMD_S_F:
    FSavePos();
  break;

  case CMD_P_F:
    RobotStatus = ST_F;
  break;

  case CMD_D_F:
    FSpointer = 0;
    EEPROM.write(F_EE_PADD, FSpointer);
  break;

  case CMD_S_B:
    BSavePos();
  break;

  case CMD_P_B:
    RobotStatus = ST_B;
  break;

  case CMD_D_B:
    BSpointer = 0;
    EEPROM.write(B_EE_PADD, BSpointer);
  break;
  case CMD_S_R:
    RSavePos();
  break;

  case CMD_P_R:
    RobotStatus = ST_R;
  break;

  case CMD_D_R:
    RSpointer = 0;
    EEPROM.write(R_EE_PADD, RSpointer);
  break;

  case CMD_S_L:
    LSavePos();
  break;

  case CMD_P_L:
    RobotStatus = ST_L;
  break;

  case CMD_D_L:
    LSpointer = 0;
    EEPROM.write(L_EE_PADD, LSpointer);
  break;

  case CMD_STOP:
    RobotStatus = ST_NONE;
  break;
  }
  // =================================================================
}
```

Anexo 9. Sketch ble_generalIO.ino

Las funciones “DecodeSpecialCommand”, “Readln” y “myIntToString” se omiten por simplicidad en este anexo. Para el código de estas funciones ver el anexo 1.

```
/* General purpose input / ouotput BLE board
 Bluetooth Module attached to Serial port
 Controls 6 relays connected to RelayPins[MAX_RELAYS]
 Take analog samples from sensors connected from AnalogInputs[MAX_A_INPUTS]
 Buttons from GND to DigitalInputs[MAX_D_INPUTS] explain how to manualy turn on/off relays

 Author: Juan Luis Gonzalez Bello
 Date: june 2017
 */

#include <EEPROM.h>

// Baud rate for bluetooth module
// (Default 9600 for most modules)
#define BAUD_RATE 115200 // TIP> Set your bluetooth baud rate to 115200 for better performance

// Special commands
#define CMD_SPECIAL '<'
#define CMD_ALIVE   '['

#define UPDATE_FAIL  0
#define UPDATE_ACCEL 1
#define UPDATE_SKBAR 2
#define UPDATE_TPAD  3
#define UPDATE_SPCH  4

/*
 * Relay outputs config
 */
#define MAX_RELAYS 6
// Relay 1 is at pin 8, relay 2 is at pin 9 and so on.
int RelayPins[MAX_RELAYS]  = {2, 3, 4, 5, 6, 7};
// Relay 1 will report status to toggle button and image 00, relay 2 to button 01 and so on.
String RelayAppId[MAX_RELAYS] = {"00", "01", "02", "03", "04", "05"};
// Command list (turn on - off for eachr relay)
const char CMD_ON[MAX_RELAYS] = {'A', 'B', 'C', 'D', 'E', 'F'};
const char CMD_OFF[MAX_RELAYS] = {'a', 'b', 'c', 'd', 'e', 'f'};
// Used to keep track of the relay status in eeprom
int RelayStatus = 0;
int STATUS_EEADR = 20;

/*
 * Digital input config
 */
#define MAX_D_INPUTS 6
boolean DigitalLatch[MAX_D_INPUTS] = {false, false, false, false, false, false};
boolean DIStatus[MAX_D_INPUTS] = {false, false, false, false, false, false};
int DigitalInputs[MAX_D_INPUTS] = {8, 9, 10, 11, 12, 13};
String DIAppId[MAX_D_INPUTS] = {"06", "07", "08", "09", "10", "11"};
```

```
/*
 * Analog input config
 */
#define MAX_A_INPUTS 6
int AnalogInputs[MAX_A_INPUTS] = {A0, A1, A2, A3, A4, A5};
String AIAppId[MAX_A_INPUTS] = {"12", "13", "14", "15", "16", "17"};
int TriggerRelayEnable[MAX_A_INPUTS] = {false, false, false, false, true, true};
int TriggerThreshold[MAX_A_INPUTS] = {512, 512, 512, 512, 512, 512};

// Data and variables received from especial command
int Accel[3] = {0, 0, 0};
int SeekBarValue[8] = {0, 0, 0, 0, 0, 0, 0, 0};
int TouchPadData[24][2]; // 24 max touch pad objects, each one has 2 axis (x and Y)
String SpeechRecorder = "";

void setup() {
  // Initialize touch pad data,
  // this is to avoid having random numbers in them
  for (int i = 0; i < 24; i++) {
    TouchPadData[i][0] = 0; //X
    TouchPadData[i][1] = 0; //Y
  }

  // initialize BT Serial port
  Serial.begin(BAUD_RATE);
  // Initialize digital input ports with input pullup
  for (int i = 0; i < MAX_D_INPUTS; i++) {
    pinMode(DigitalInputs[i], INPUT_PULLUP);
    Serial.print("<Text" + DIAppId[i] + ":Pin " + String(DigitalInputs[i]));
    delay(20);
  }

  // Initialize analog input ports
  for (int i = 0; i < MAX_A_INPUTS; i++) {
    pinMode(AnalogInputs[i], INPUT);
  }

  // Initialize relay output ports
  for (int i = 0; i < MAX_RELAYS; i++) {
    pinMode(RelayPins[i], OUTPUT);
    digitalWrite(RelayPins[i], HIGH); // active low relays
    Serial.print("<Text" + RelayAppId[i] + ":Pin " + String(RelayPins[i]));
    delay(20);
  }

  // Load last known status from eeprom
  RelayStatus = EEPROM.read(STATUS_EEADR);
  for (int i = 0; i < MAX_RELAYS; i++) {
    // Turn on and off according to relay status
    if ((RelayStatus & (1 << i)) == 0) {
      digitalWrite(RelayPins[i], HIGH);
      Serial.print("<Butn" + RelayAppId[i] + ":0");
      delay(20);
      Serial.print("<Imgs" + RelayAppId[i] + ":0");
      delay(20);
    }
```

```
    else {
      digitalWrite(RelayPins[i], LOW);
      Serial.print("<Butn" + RelayAppId[i] + ":1");
      delay(20);
      Serial.print("<Imgs" + RelayAppId[i] + ":1");
      delay(20);
    }
  }

  // Greets on top of the app
  Serial.print("ATC Expert BT");
  delay(20);

  // Make the app talk in english (lang number 00, use 01 to talk in your default language)
  Serial.print("<TtoS00:Welcome ATC");
  delay(20);
}

void loop() {
  int appData;
  int testRelay;
  int tagType;
  static int analogPrescaler = 0;
  static int analogPrescaler1 = 0;
  static int digitalPrescaler = 0;
  static int aliveRefreshPrescaler = 0;
  delay(1);

  // ============================================================
  // This is true each 1/4 second approx.
  if (analogPrescaler++ > 250) {
    analogPrescaler = 0; // Reset prescaler

    // Take analog samples and send to app once at a time
    int sample = analogRead(AnalogInputs[analogPrescaler1]);
    // Trigger relay output if feature enabled
    if (TriggerRelayEnable[analogPrescaler1]) {
      if (sample > TriggerThreshold[analogPrescaler1]) {
        // Example of how to make beep alarm sound
        Serial.print("<Alrm00");
        delay(15);
        setRelayState(analogPrescaler1, 1);
      }
      else
        setRelayState(analogPrescaler1, 0);
    }
    // Use <Text tags to display alphanumeric information in app
    Serial.print("<Text" + AIAppId[analogPrescaler1] + ":" + "An: " + String(sample));

  delay(15);
  // Use <Imgs tags to dinamically change pictures in app
  Serial.print("<Imgs" + AIAppId[analogPrescaler1] +  EvaluateAnalogRead(sample));
  delay(15);
  // Use <Abar tags to change analog bar levels from 0 to 255
  Serial.print("<Abar" + RelayAppId[analogPrescaler1] +  ":" + myIntToString(sample >> 2));
  delay(15);
```

```
  analogPrescaler1 = ++analogPrescaler1 % MAX_A_INPUTS;
}

// ============================================================
// This is the point were you get data from the App
appData = Serial.read();   // Get a byte from app, if available
switch (appData) {
  case CMD_SPECIAL:
    // Special command received
    tagType = DecodeSpecialCommand();

    if (tagType == UPDATE_SKBAR) {
      // Example of how to use seek bar data to dim a LED connected in pin 13
      analogWrite(3, SeekBarValue[0]);
    }

    if (tagType == UPDATE_SPCH) {
      // Example of how to use speech recorder
      testRelay = 0; // this is the relay number which will turn off or on with voice
      Serial.print("<Text" + RelayAppId[testRelay] + ":" + SpeechRecorder); // display received voice command
      delay(20);

      if (SpeechRecorder.equals("Apagar")) {
        setRelayState(testRelay, 0);
      }
      if (SpeechRecorder.equals("encender")) {
        setRelayState(testRelay, 1);
      }
    }
    break;

  case CMD_ALIVE:
    // Character '[' is received every 2.5s, use
    // this event to refresh the android all relay states
    // Refresh button states to app (<BtnXX:Y\n)
    if (!digitalRead(RelayPins[aliveRefreshPrescaler])) {
      Serial.print("<Butn" + RelayAppId[aliveRefreshPrescaler] + ":1");
      delay(15);
      Serial.print("<Imgs" + RelayAppId[aliveRefreshPrescaler] + ":1");
      delay(15);
    }
    else {
      Serial.print("<Butn" + RelayAppId[aliveRefreshPrescaler] + ":0");
      delay(15);
      Serial.print("<Imgs" + RelayAppId[aliveRefreshPrescaler] + ":0");
      delay(15);
    }
    // Update one relay at a time
    aliveRefreshPrescaler = (++aliveRefreshPrescaler) % MAX_RELAYS;
    // Greets on top of the app

    // Greets on top of the app
    Serial.print("ATC Expert BLE");
    delay(15);
    break;
```

```
    default:
      // If not '<' or '[' then appData may be for turning on or off relays
      for (int i = 0; i < MAX_RELAYS; i++) {
        if (appData == CMD_ON[i]) {
          // Example of how to make phone vibrate
          Serial.print("<TtoS00:light on"); // 16 char
          delay(20);
          //Serial.print("<Vibr00:100");
          // Example of how to make beep alarm sound
          Serial.print("<Alrm00"); // 7 char
          delay(20);
          setRelayState(i, 1);
          //Serial.print("<Logr00: Command: " + String(CMD_ON[i]) + " received");
          //Serial.print("<Logr00: Relay: " + String(i) + " turned on");
        }
        else if (appData == CMD_OFF[i]) {
          Serial.print("<TtoS00:light off"); // 17 char
          delay(20);
          setRelayState(i, 0);
          //Serial.print("<Logr00: Command: " + String(CMD_OFF[i]) + " received");
          //Serial.print("<Logr00: Relay: " + String(i) + " turned off");
        }
      }
  }

  // ===========================================================================

  // Digital inputs are used to toggle relay outputs in board
  // this condition is true each 1/10 of a second approx
  if (digitalPrescaler++ > 100) {
    digitalPrescaler = 0;
    for (int i = 0; i < MAX_D_INPUTS; i++) {
      if (!digitalRead(DigitalInputs[i])) { // If button pressed
        // don't change status until button has been released and pressed again
        if (DigitalLatch[i]) {
          DIStatus[i] = !DIStatus[i];
          if (DIStatus[i]) {
            Serial.print("<Imgs" + DIAppId[i] + ":1"); // Set image to pressed state
            delay(20);
          }
          else {
            Serial.print("<Imgs" + DIAppId[i] + ":0"); // Set image to default state
            delay(20);
          }
          // Uncomment line below if you want to turn on and off relays using physical buttons
          setRelayState(i, !digitalRead(RelayPins[i])); // toggle relay 0 state
          DigitalLatch[i] = false;
        }
      }
      else {
        // button released, enable next push
        DigitalLatch[i] = true;
      }
    }
  }
}
```

```
// Sets the relay state for this example
// relay: 0 to 5 relay number
// state: 0 is off, 1 is on
void setRelayState(int relay, int state) {
  if (state == 1) {
    digitalWrite(RelayPins[relay], LOW);           // Write ouput port
    ///Serial.print("<Butn" + RelayAppId[relay] + ":1"); // Feedback button state to app
    Serial.print("<Imgs" + RelayAppId[relay] + ":1"); // Set image to pressed state
    delay(15);
    RelayStatus |= (0x01 << relay);                // Set relay status
    EEPROM.write(STATUS_EEADR, RelayStatus);       // Save new relay status
  }
  else {
    digitalWrite(RelayPins[relay], HIGH);           // Write ouput port
    //Serial.print("<Butn" + RelayAppId[relay] + ":0"); // Feedback button state to app
    Serial.print("<Imgs" + RelayAppId[relay] + ":0"); // Set image to default state
    delay(15);
    RelayStatus &= ~(0x01 << relay);               // Clear relay status
    EEPROM.write(STATUS_EEADR, RelayStatus);       // Save new relay status
  }
}

// Evaluate analog read
// Use this function to tell the app if the analog sample is good to display picture in state 0, 1, or 2
String EvaluateAnalogRead(int analogSample) {
  if (analogSample < 330) {
    return ":0"; // Default  state
  }
  else if (analogSample < 660) {
    return ":1"; // Pressed state
  }
  else {
    return ":2"; // Extra state
  }
}
```

Anexo 10. Sketch eth_doorlock.ino

Las funciones “DecodeSpecialCommand”, “Readln” y “myIntToString” se omiten por simplicidad en este anexo. Para el código de estas funciones ver el anexo 2.

```
/* Ethernet controlled electronic doorlock
 Dool lick sistem using pattern or voice fom ATC app.

 Author: Juan Luis Gonzalez Bello
 Date: May 2017
 */

#include <SPI.h>
#include <Ethernet.h>

// Special commands
#define CMD_ALIVE   '['
#define CMD_SPECIAL '<'

// Signal to update special commands
#define UPDATE_FAIL  0
#define UPDATE_ACCEL 1
#define UPDATE_SKBAR 2
#define UPDATE_TPAD  3
#define UPDATE_SPCH  4

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(172, 21, 117, 60);

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(8080);
EthernetClient client;

// Data and variables received from especial command
int Accel[3] = {
  0, 0, 0
};
int SeekBarValue[8] = {
  0, 0, 0, 0, 0, 0, 0, 0
};
int TouchPadData[24][2]; // 24 max touch pad objects, each one has 2 axis (x and Y)
String SpeechRecorder = "";

// Sistem specific ports and variables
#define TIME_OUT 20000 // 20 seconds
#define OUT_NO       6
#define ST_CLOSED    0
#define ST_OPEN      1
#define PASS_NO     10
#define PASS_SIZE    6
#define CMD_ARM     'A'
```

```
int outs[OUT_NO] = {1, 2, 3, 4, 5, 6};
boolean FirstTimeConnection = false;
int TimeOutCounter = 0;
String Password = "1 3 2 5 4 6";
String PasswordTest = "";
int LockStatus = ST_CLOSED;
int LockSequence = 0;
int PassNumbers[PASS_NO] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};

void setup() {
  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();

  // Used to ground output leds
  pinMode(7, OUTPUT);
  digitalWrite(7, LOW);

  // Initialize outputs
  for (int i = 0; i < OUT_NO; i++) {
    pinMode(outs[i], OUTPUT);
    digitalWrite(outs[i], LOW);
  }
}

void loop() {
  int appData;
  int commandType;
  delay(1);

  if (FirstTimeConnection) {
    digitalWrite(outs[0], HIGH);
    if (TimeOutCounter++ > TIME_OUT) {
      FirstTimeConnection = false;
      TimeOutCounter = 0;
    }
  }
  else {
    digitalWrite(outs[0], LOW);
    LockStatus = ST_CLOSED;
  }

  // lock or release according to the status
  if (LockStatus == ST_OPEN) {
    for (int i = 1; i < OUT_NO; i++) {
      digitalWrite(outs[i], HIGH);
    }
  }
  else {
    for (int i = 1; i < OUT_NO; i++) {
      digitalWrite(outs[i], LOW);
    }
  }
```

```
// Check password
if (LockSequence == PASS_SIZE) {
  server.println("Recivido:" + PasswordTest);
  if (PasswordTest.equals(Password)) {
    server.println("<Vibr00:050");
    server.println("<TtoS01: contraseña correcta, adelante");
    LockStatus = ST_OPEN;
  }
  else {
    server.println("<TtoS01: contraseña incorrecta");
    server.println("<Alrm00");
    server.println("<Vibr00:999");
    LockStatus = ST_CLOSED;
  }
  LockSequence = 0;
  PasswordTest = "";
}
// Should not happen
if (LockSequence > PASS_SIZE) {
  LockStatus = ST_CLOSED;
  LockSequence = 0;
  PasswordTest = "";
}

// ==============================================================
// This is the point were you get data from the App
client = server.available();
if (client) {
  appData = client.read();
}

switch (appData) {
  case CMD_SPECIAL:
    // Special command received
    // After this function accel and seek bar values are updated
    commandType = DecodeSpecialCommand();
    if (commandType == UPDATE_SPCH) {
      server.println("Recivido:" + SpeechRecorder);
      if (SpeechRecorder.equals(Password)) {
        server.println("<Vibr00:050");
        server.println("<TtoS01: contraseña correcta, adelante");
        LockStatus = ST_OPEN;
      }
      else {
        server.println("<TtoS01: contraseña incorrecta");
        server.println("<Alrm00");
        server.println("<Vibr00:999");
        LockStatus = ST_CLOSED;
      }
    }
    break;

  case CMD_ALIVE:
    // Character '[' is received every 2.5s
    server.println("ATC ready");
    // Reset timeout
    TimeOutCounter = 0;
```

```
      // Greet first connection
      if (!FirstTimeConnection) {
        FirstTimeConnection = true;
        server.println("<TtoS01: Bienvenido a casa, juan luis");

      }
      break;

    case CMD_ARM:
      server.println("<TtoS01: Bloqueado");
      LockStatus = ST_CLOSED;
    break;

    default:
      for (int i = 0; i < PASS_NO; i++) {
        if (appData == PassNumbers[i]) {
          if(LockSequence == 0){
            PasswordTest = String(char(PassNumbers[i]));
          }
          else{
            PasswordTest = PasswordTest + " " + char(PassNumbers[i]);
          }
          LockSequence++;
        }
      }
  }
  // ==========================================================
}
```

Glosario

1	HMI	Interfaz Hombre Maquina, o Human Machine Interface por sus siglas en inglés.
2	Acelerómetro	Instrumento para medir aceleraciones
3	ADC	Convertidor Analógico Digital, o Analog to Digital Converter por sus siglas en inglés
4	AES (encriptado)	Estándar Avanzado de Encriptado, o Advanced Encryption Standard por sus siglas en inglés
5	API	Interfaz de programación de aplicaciones, o Aplication Programing Interface por sus siglas en inglés
6	AsyncTask	Tarea asíncrona, herramienta de manejo de threads en Android.
7	ATC	Arduino Total Control
8	BLE	Bluetooth de Baja Energía, o Bluetooth Low Energy por sus silgas en inglés
9	Bluetooth SIG	Grupo de Interés Especial Bluetooth, o Bluetooth Single Interest Group por sus siglas en inglés
10	Broadcast	Difusión amplia, en telecomunicaciones se refiere a enviar información a todos los elementos conectados a la red.
11	BR/EDR	Tasa de transferencia básica / Tasa de transferencia mejorada, o Base Rate / Enhanced Data Rate por sus siglas en inglés.
12	Callback	Retro llamada (en inglés: callback) es una función "A" que se usa como argumento de otra función "B".
13	CPU	Unidad Central de Procesamiento (en inglés: Central Processing Unit)
14	DAC	Convertidor Digital a Analógico, o Digital to Analog Converter por sus siglas en ingles.
15	DCE	Equipo de Comunicación de Datos (en inglés: Data Communication Equipment)
16	DPSK	Esquema de modulación digital Differential Phase-Shift Keying por sus siglas en inglés.
17	DQPSK	Esquema de modulación digital Differential Quadrature Phase-Shift Keying
18	DSSS	Esquema de modulación digital Direct Sequence Spread Spectrum
19	DTE	Equipo Terminal de Datos, (en inglés: Data Terminal Equipment)
20	Footprint	(En inglés) área.
21	Framework	Conjunto de librerías para un sistema de software
22	GitHub	Repositorio de control de versión basado en la Web.
23	GPIO	Puertos de Entrada y Salida de Propósito Genera, o General Purpose Input and Output por sus siglas en inglés.
24	hardware	Componentes físicos que componen un sistema de computo

25	Heartbeath	Latido, en comunicaciones puede referirse a una señal o dato que se envía para hacerle saber a los sistemas involucrados el estado de la conexión.
26	IDE	Entorno de Desarrollo Integrado, o Integrated Development Environment por sus siglas en inglés.
27	IP	Internet Protocol
28	ISO	International Organization for Standardization
29	LAN	Local Área Network (Red de Area Local)
30	Layout	En ATC, conjunto de elementos constructivos que forman una "pantalla" interactiva para el usuario.
31	LED	Light Emitting Diode
32	MCU	Microcontrolador, o Microcontroller Unit por su abreviación en inglés.
33	OFDM	Esquema de modulación digital Orthogonal frequency-division multiplexing
34	OSI	(Modelo de) Interconexión de Sistemas Abiertos
35	PDU	Protocol Data Unit
36	Piconet	Red de área personal Bluetooth.
37	SDK	Software Development Kit (Kit de Desarrollo de Software)
38	Shield	(Circuito) Tarjeta de expansión compatible con Arduino.
39	Software	Parte de un sistema de cómputo que consiste en datos o instrucciones para el computador.
40	SPP	Serial Port Profile (Perfil de Puerto Serial)
41	TCP	Transfer Control Protocol
42	TCP Client	Cliente TCP
43	TCP Server	Servidor TCP
44	Top-down	Diseño de arriba a abajo. Se formula un resumen del sistema, sin especificar detalles. Cada parte del sistema se refina diseñando con mayor detalle. hasta que la especificación completa es lo suficientemente detallada para validar el modelo
45	UART	Receptor-Transmisor Asíncrono Universal, o Universal Asynchronous Receiver-Transmitter por sus siglas en ingles
46	UTP	Unshielded twisted pair (UTP) o par trenzado sin blindaje
47	UUID	Identificador Único Universal o Universally Unique Identifier es un número de 16 bytes (128 bits) usado para identificar información en un sistema de computo
48	WLAN	Wireless LAN, o Red de área local inalámbrica

Índice de tablas y figuras

Tabla 1. Metodología del diseño y puesta en marcha para aplicación ATC. 10
Tabla 1.1. Características y comparativa de aplicaciones HMI para Android 20
Tabla 1.2. Clases de Bluetooth de acuerdo a potencia de salida. [31] 30
Tabla 1.3. Versiones de la especificación de Bluetooth. 31
Tabla 1.4. Variantes del IEEE 802.11 [37]. 34
Tabla 2.1. Las clases de java definidas en ATC. 43
Tabla 2.2. Librerías de compatibilidad usadas en ATC. 43
Tabla 2.3. Las clases de java definidas en ATC para comunicaciones. 66
Tabla 3.1. GATT, characteristic y service. 72
Tabla 3.2. Clases de la aplicación demo Bluetooth LEGATT. 73
Tabla 3.3. Característica objetivo de un módulo HM10. 74
Tabla 5.1. Pruebas de comunicaciones inalámbricas. 99
Tabla 5.2. Pruebas de protocolo de Capa de Aplicación ATC. 108
Tabla 5.3. Características del circuito ACS712. 119
Tabla 5.4. Combinaciones para operación de puente H. 123
Tabla 6.1. Comparativa Blink, Arduino Commander y ATC. 136
Tabla 6.2. Versiones de la aplicación ATC. 137
Tabla B.1. Comandos AT módulo Kc Energy 141
Tabla B.2. UUIDs de características y servicios de un módulo kc4114. 142
Tabla B.3. Pruebas preliminares al desarrollo de los programas en Arduino y Android 143
Tabla C.1. Comandos AT en un módulo Bluetooth esclavo HC-06. 144
Tabla C.2. Comandos AT para un módulo Bluetooth maestro esclavo HC05 145
Tabla D.1. Comandos AT de un módulo Bluetooth LE HM10. 146
Tabla E.1. Comandos AT de un módulo Wi-Fi ESP8266. 147

Figura 1. Esquema general de funcionamiento de la aplicación. 7
Figura 2. Diagrama de bloques de ATC (arriba), ícono de aplicación ATC (abajo). 9
Figura 3. Diagrama de bloques de sistema a base de microcontrolador compatible con ATC. 10
Figura 1.1. Una interfaz de usuario está conformada por software y hardware [2]. 11
Figura 1.2. HMI industrial moderno [4]. 11
Figura 1.3. Línea de tiempo de las interfaces de usuario. 12
Figura 1.4. IOS vs Android OS Market Share [7]. 12
Figura 1.5. Interfaz “Home” en Android. 13
Figura 1.6. Logo de Android Studio. IDE oficial para desarrollar aplicaciones para cualquier dispositivo Android [9]. 13
Figura 1.7. Ícono del formato “.APK”. 13
Figura 1.8. Diagrama de Arquitectura de Android [9]. 15
Figura 1.9. Esquema básico general de un microcomputador [14]. 16
Figura 1.10. Diagrama de bloques del microcontrolador ATMEGA328 [17]. 17
Figura 1.11. Logo oficial de Arduino y Placas Arduino [19]. 18
Figura 1.12. Primeros pasos de la aplicación, BT Relay Control. 19
Figura 1.13. Módulo HC-05. 21
Figura 1.14. Módulo HM10 basado en el TI CC2540. 22
Figura 1.15. Arduino Yún. 22
Figura 1.16. Ethernet Shield. 23
Figura 1.17. Módulo ESP8266. 23

Figura 1.18. Elementos básicos de OSI [27]. 24
Figura 1.19. Capas en un sistema abierto [27]. 25
Figura 1.20. Capas del modelo OSI [27]. 26
Figura 1.21. Sistema abierto actuando como relevador [27]. 27
Figura 1.22. Logo Bluetooth. 30
Figura 1.23. Stack de la especificación Bluetooth [33]. 31
Figura 1.24. Logo de la Wi-Fi Alliance. 33
Figura 1.25. Modelo básico de referencia IEEE 802 [36]. 33
Figura 1.26. Pila de protocolos IEEE 802.11 [37]. 34
Figura 1.27. Técnicas de transmisión de la Capa Física en Wi-Fi [37]. 35
Figura 1.28. Representación gráfica de los canales de la banda 2.4GHz [38]. 35
Figura 1.29. Ejemplo de archivo de manifiesto. 38
Figura 1.30. Ejemplo de filtros intent. 39
Figura 1.31. Manifiesto que requiere una cámara y las API introducidas en Android 2.1 (nivel 7) como mínimo. 39
Figura 2.1. Diseño Top-Down de la Interfaz Gráfica en ATC. 42
Figura 2.2. Declaración de la aplicación principal en el Manifiesto. 43
Figura 2.3. Diagrama de flujo de la actividad Main. 44
Figura 2.4. Estados de un ToggleButton. 45
Figura 2.5. Registro de Callback “onClick” en la clase principal (this = Main.class). 46
Figura 2.6. Implementación del método onClick en la actividad Main. 46
Figura 2.7. Función para filtrar el ID de un supuesto Togglebutton. 46
Figura 2.8. Filtrado del ID utilizando la función DecodeButtonID. 46
Figura 2.9. Envío de una cadena de texto a MainSend. 47
Figura 2.10. Propiedades editables en ATC para un ToggleButton. 47
Figura 2.11.En rojo, ejemplos de TextView. 47
Figura 2.12. Propiedades editables en ATC para un Texto. 48
Figura 2.13. Declaración de variables tipo sensor. 48
Figura 2.14. Inicialización del servicio SensorManager y objeto sensor. 48
Figura 2.15. Registro del callback de sensor en Main (this = Main). SENSOR_DELAY_UI es la frecuencia de muestreo del sensor adecuada para una interfaz de usuario. 48
Figura 2.16. Implementación del método onSensorChanged en la actividad Main. 48
Figura 2.17. Extracción de los valores del acelerómetro. Si el texto “i” del “CurrentDisplay” está habilitado como acelerómetro, aplicar formato y enviar a la función MainBSend para procesamiento adicional. 49
Figura 2.18. Ejemplos de SeekBar con diferentes niveles de progreso. 49
Figura 2.19. Configuración del valor máximo y registro de evento de SeekBar. 49
Figura 2.20. Implementación del método onProgressChanged en la actividad principal. 50
Figura 2.21. Lectura del nuevo valor de la Barra deslizante y envío a función MainSend. 50
Figura 2.22. Propiedades editables en ATC para una SeekBar. 50
Figura 2.23. Uso de ImageView como botones y panel de toque. 51
Figura 2.24. AsyncTask loadPictures() ejecutada en callback “onStart( )”. 51
Figura 2.25. Propiedades editables de una Imagen. 52
Figura 2.26. Registro del evento onTouch en actividad Main. 52
Figura 2.27. Implementación del método onTouchListener en la actividad principal. 52
Figura 2.28. Switch de decisión para el evento de toque para un botón temporal. 53
Figura 2.29. Cambio de estado de la imagen de acuerdo al comando a enviar. 53
Figura 2.30. Cuadrícula de paneles táctiles. 54
Figura 2.31. Inicializando un touchpad. 54
Figura 2.32. Escala y traslación de ejes en un touchpad. 54
Figura 2.33. Switch de decisión para el evento de toque para un touchpad. 55

Figura 2.34. Lanzamiento de intent de reconocimiento de voz. 55
Figura 2.35. La aplicación recupera la cadena de texto del reconocimiento de voz con el código de solicitud REQUEST_SR y utilizando el método data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS). 56
Figura 2.36. Display analógico simple con variación vertical (izquierda) y con variación horizontal (izquierda). 56
Figura 2.37. Creación de una nueva clase. 56
Figura 2.38. Clase AnalogBarView heredando los métodos de la clase View. 57
Figura 2.39. Configuración del objeto paint en el método analogBarPaintInit(). 57
Figura 2.40. Sobrecarga de onSizeChanged. 57
Figura 2.41. Variables locales para control de estado de la barra. 58
Figura 2.42. Sobrecarga de la función onDraw. 58
Figura 2.43. Manejo de la barra vertical. 58
Figura 2.44. Método para control de orientación de la barra analógica. 59
Figura 2.45. Asignación del valor de la barra. 59
Figura 2.46. Métodos para cambiar la apariencia de la barra. 60
Figura 2.47. Propiedades de una barra analógica. 60
Figura 2.48. Herramientas de edición primaria. 61
Figura 2.49. Interfaces de edición de propiedades de los elementos constructivos. 61
Figura 2.50. Despliegue de herramientas de edición secundarias. 62
Figura 2.51. Carpeta del layout exportado (izquierda) y archivo de configuración (derecha). 62
Figura 2.52.Tags para exportar propiedades de un Imagen. 63
Figura 2.53. La actividad Main, llama los métodos de la clase ExProp para generar el contenido del archivo de configuración ".txt" 63
Figura 2.54. Lectura de contenido de un layout exportado. 64
Figura 3.1. Diseño Top-Down de conectividad en ATC. 65
Figura 3.2.Permisos necesarios para implementar Bluetooth en ATC. 66
Figura 3.3. Ejemplo de nombres de dispositivos Bluetooth con su dirección física. 66
Figura 3.4. Diagrama de flujo de una conexión Bluetooth en ATC. 67
Figura 3.5. Captura de pantalla de actividad BTScanActivity. 68
Figura 3.6. Obtención de dispositivos Bluetooth previamente conectados. 68
Figura 3.4. Registro e inicialización del descubrimiento de nuevos dispositivos Bluetooth. 68
Figura 3.5. Receptor de mensajes para escaneo de dispositivos nuevos. 69
Figura 3.6. Fin de BTScanActivity. 69
Figura 3.7. Inicio de una conexión Bluetooth 70
Figura 3.8. Conexión del BluetoothSocket. 70
Figura 3.9. Asignación de los flujos de entrada y salida de un BlkuetoothSocket. 71
Figura 3.10. Envío de un buffer de datos a un dispositivo Bluetooth. 71
Figura 3.11. Recepción de información Bluetooth. 72
Figura 3.12. Conexión de un dispositivo Bluetooth LE. 74
Figura 3.13. BroadcastReceiver mGattUpdateReceiver. 74
Figura 3.14. Envío de información desde ATC al dispositivo BLE. 75
Figura 3.15. Pila de protocolos TCP/IP [47] (izquierda) y su relación con ATC (derecha). 75
Figura 3.16. Movimiento de información desde la aplicación remitente hasta el sistema destinatario. [47] 76
Figura 3.17. Permisos en el manifiesto para uso de Internet. 76
Figura 3.18. Actividad IPSet con valores por defecto para dirección IP y puerto. 77
Figura 3.19. Inicializador de TCPClient. 77
Figura 3.20. Proceso de conexión a un servidor TCP. 78
Figura 3.21. Recepción de información del servidor TCP/IP. 78
Figura 3.22. Envió de datos por TCP/IP. 79
Figura 3.23. Función Main Broadcast Send (MainBSend). 80

Figura 3.24. Todos los datos entrantes se envían a Main.decodeMessage()........80
Figura 3.25. Envío del raw data “ATC ready” a la aplicación. 81
Figura 3.26. Definición del comando Alive en el programa en el microcontrolador. 81
Figura 4.1. Diagrama de bloques del sistema a base de microcontrolador. 83
Figura 4.2. Diagrama de tiempos de una trama UART [49]. 84
Figura 4.3. Conexión maestro-esclavo (Master - Slave) SPI simple [50]........84
Figura 4.4 Comparación entre módulos HC-05 o HC-06 (arriba) y HM-10 (abajo). 85
Figura 4.5. Conexión Bluetooth con Arduino. 85
Figura 4.6. Hardware mínimo para conexión por Ethernet. 86
Figura 4.7. Diagrama de bloques ESP8266. 87
Figura 4.8. Terminales y conexión de un módulo ESP8266. 87
Figura 4.9. Funciones setup y loop en Arduino. 88
Figura 4.10. Loop Main para sketches Arduino........88
Figura 4.11. Variables y constantes del protocolo ATC en el MCU........89
Figura 4.12. Loop switch. 89
Figura 4.13. Diagrama de flujo del sistema a base de microcontrolador. 90
Figura 4.14. Vista preliminar de función DecodeSpecialCommand(). 91
Figura 4.15. Inicialización de la UART en Arduino. 91
Figura 4.16. Ejemplo de envío de tags de Interfaz de Usuario a la aplicación. 92
Figura 4.17. Lectura de un byte en Bluetooth........92
Figura 4.18. Variables y constantes globales para Ethernet Shield. 92
Figura 4.19. Inicialización de un servidor TCP/IP en Ethernet Shield. 93
Figura 4.20. Envío de tags de interfaz de usuario en Ethernet Shield. 93
Figura 4.21. Lectura de un byte de un cliente TCP en Ethernet Shield. 93
Figura 4.22. Configuración inicial de un módulo ESP8266 como un servidor multicanal TCP/IP. 94
Figura 4.23. Funciones para envío de datos para el módulo ESP8266. 94
Figura 4.24. Envío de múltiples comandos en una sola cadena (boardMessage)........95
Figura 4.25. Función myESP_Read. 95
Figura 4.26. Condición de espera de 10 bytes mínimo en buffer de entrada. 96
Figura 4.27. Inicialización de Arduino Yun como Servidor TCP/IP. Declaración de variables globales (arriba) e inicialización de servidor en función setup (abajo). 96
Figura 4.28. Ejemplo de envío de información a aplicación en Arduino Yun. 96
Figura 4.29. Lectura de información en Arduino Yun. 97
Figura 5.1. Esquema de circuito de pruebas de comunicaciones inalámbricas. 98
Figura 5.2. Conexión entre adaptador UART y HC05. 99
Figura 5.3. Respuesta a comando AT de módulo HC05. 99
Figura 5.4. Configuración de baud rate de un módulo HC05. 100
Figura 5.5. Envío de comando AT en HC06. 100
Figura 5.6. Configuración de baud rate de un módulo HC06. 101
Figura 5.7. Configuración de baud rate de un módulo HM10. 101
Figura 5.8. Configuración de baud rate de un módulo ESP8266. 102
Figura 5.9. Proceso de conexión a módulo Bluetooth EDR/BR. a) Descubrimiento; b) Introducción de PIN de seguridad; c) Conexión. 102
Figura 5.10. Proceso de conexión a módulo Bluetooth Low Energy. a) Descubrimiento, b) Selección de "característica", c) Conexión. 103
Figura 5.11. Logo de aplicación Fing [52]. 103
Figura 5.12. Proceso de conexión Wi-Fi. a) Descubrimiento usando aplicación Fing, b) Configuración de dirección IP y puerto, c) Conexión. 104
Figura 5.13. Comunicación bidireccional exitosa entre módulo HC05/HC06 y la aplicación. 104

Figura 5.14. Ejemplo del uso de Android logcat para mostrar la información entrante en la aplicación. 105
Figura 5.15. Comunicación bidireccional exitosa entre módulo HM10 y la aplicación..105
Figura 5.16. Comunicación bidireccional exitosa entre Ethernet Shield y la aplicación. 106
Figura 5.17. Recepción de datos de la aplicación usando ESP8266. 106
Figura 5.18. Envío de datos del módulo ESP8266 a la aplicación. 107
Figura 5.19. Recepción del comando "alive"...107
Figura 5.20. envío de la cadena raw data "Hola Mundo" a la aplicación. 107
Figura 5.21. Activación de un LED usando raw data enviada por activación de botones. 109
Figura 5.22. Activación de un Toggle Button; a) Encendido “ON”; b) apagado “OFF”; c) comandos recibidos en monitor serial. 109
Figura 5.23. Actividad EProp (editar propiedades) para los botones 0 y 1. 110
Figura 5.24. Activación de un Temporary Button; a) Encendido “ON”; b) Apagado “OFF”; c) Comandos recibidos en monitor serial..110
Figura 5.25. Propiedades de una imagen habilitada como botón temporal. Imagen que envía caracteres (derecha) e imagen que envía cadenas de texto (izquierda). 111
Figura 5.25. Configuración de una imagen como Long Touch Button..111
Figura 5.26. Activación de un Long Touch Button. a) Al liberar se envía un comando "OFF" ('a' para este ejemplo); b) No se envía ningún comando al toque; c) Después de 1s aproximadamente, se envía un comando "ON" ('A' en este ejemplo). 112
Figura 5.27.Uso de Text tags. 112
Figura 5.28. Uso de Button Tags..112
Figura 5.29. Uso de Image Tags. 113
Figura 5.30. Uso de Seekbat Tags..113
Figura 5.31. Uso de Alarm y Vibrator Tags. 113
Figura 5.32. Uso de Analog Bar Tags. 114
Figura 5.33. Impresión de carga útil de Seekbar Tags. 114
Figura 5.34. Configuración de textos como fuente de acelerómetro. 115
Figura 5.35. Impresión de carga útil de Accelerometer Tags. 115
Figura 5.36. Impresión de carga útil de un Touch Pad Tag. 116
Figura 5.37. Activación de una salida digital usando Speech Recognizer Tags. 116
Figura 5.38. Prueba de ATC Logger. 117
Figura 5.39. Componentes de ATC RC Car. 117
Figura 5.40. Diagrama electrico de ATC RC Car..118
Figura 5.41. Layout para ATC RC Car. 118
Figura 5.42. Ejecución de la función .read() en tiempo real. 119
Figura 5.43. Voltaje de salida versus corriente de entrada. 120
Figura 5.44. Función para lectura de corriente...120
Figura 5.45. Resultado de la función CurrentSensing()...120
Figura 5.46. Código de control de luces frontales. 121
Figura 5.47. Resultado del código de control de luces frontales..121
Figura 5.48. Definición de estados de las luces traseras...122
Figura 5.49. Máquina de estados de control de luces traseras. 122
Figura 5.50. Activación de luces direccionales. Retroalimentación grafica en la aplicación (izquierda) y resultado en el mundo físico (derecha)..123
Figura 5.51. Definición de comandos para control de dirección. 123
Figura 5.52. Control de dirección por acelerómetro...124
Figura 5.53. Comandos de control de tracción. 124
Figura 5.54. Componentes robot caminante (arriba) y diseño de robot en SketchUp (abajo)...125
Figura 5.55. Diagrama eléctrico robot caminante. 126

Figura 5.56. Layout ATC para robot caminante antes (izquierda) y después (derecha) de conectarse. 127
Figura 5.57. ATC Layout para prototipo "Tarjeta de entradas y salidas de propósito general". 128
Figura 5.58. Prototipo “Tarjeta de entradas y salidas de propósito general BLE”. 129
Figura 5.59. Lectura y actualización secuencial de ADC para módulos BLE. 129
Figura 5.60. ATC Layout para cerradura electrónica...130
Figura 5.61. Código para detectar estado de conexión a nivel de Capa de Aplicación..131
Figura 5.62. Estados de la cerradura electrónica ATC...131
Figura 6.1.Permiso ACCESS_COARSE_LOCATION...132
Figura 6.2. Envió de datos continuos compatible con dispositivos Android 6.0+...132
Figura 6.3. Error de posicionamiento de objetos en layout. 133
Figura 6.4. Resultado de compilación del sketch esp_firmware (a), y del bt_ATCWalker (b). 133
Figura 6.5. Conexión inestable de ESP8266 en modo AP. 134
Figura 6.6. Avería mecánica y solución propuesta de robot caminante. 134
Figura 6.7. Desempeño de la aplicación en la tienda de Google Play. 136
Figura A.1. Preparación preliminar del sintetizador de voz...139
Figura A.2. Solicitud de instalación de sintetizador de voz..139
Figura A.3. Verificación de inicialización de mTts. 140
Figura A.4. Ejecución de la función "speak" del sintetizador de voz. 140
Figura B.1. Módulo KcEnergy Kc-4114. 141
Figura B.2. Convertidor Serial a TTL basado en circuito FTDI. 142
Figura B.3. Módulo Bluetooth BLE mostrado en el servicio de “escaneo” en la aplicación. 143