UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO E IMPLEMENTACIÓN DE APLICACIONES DOMÓTICAS CONTROLADAS CON DISPOSITIVO ANDROID

INFORME DE SUFICIENCIA

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

DAVID DANIEL CHENG ZÁRATE

PROMOCIÓN
2010-II
LIMA – PERÚ
2014

DESARROLLO E IMPLEMENTACION DE APLICACIONES DOMOTICAS CONTROLADAS CON DISPOSITIVO ANDROID

Agradecimientos

En primer lugar agradezco a Dios quien hace todas las cosas posibles. En segundo lugar agradezco a mi madre quien fue la persona que me dio el apoyo total y los ánimos que necesitaba para continuar y terminar lo que empecé. También a mi hermano que estuvo ahí dándome aliento. A mis profesores en pregrado y a mi asesor que me dieron la base para poder culminar esta carrera. Finalmente, doy las gracias a Claudia y a sus padres que también estuvieron apoyándome a lo largo de este tiempo dedicado a la obtención del título.

SUMARIO

En este trabajo se realiza el diseño de un sistema de control de la luminaria en el hogar tanto la activación y desactivación de las bombillas de iluminación así como también la posibilidad de regular la intensidad de la luz.

Para implementar el sistema, se ha desarrollado una aplicación móvil bajo el sistema operativo *Android*, dicha aplicación se programa con una interfaz sencilla para el usuario y con las librerías más antiguas para obtener la mayor compatibilidad posible con dispositivos *Android* antiguos.

El dispositivo móvil debe enviar las instrucciones necesarias a una tarjeta basada en la plataforma Arduino, en la cual está incorporado un microprocesador y puertos de entrada y salida digitales, para activar con los pulsos requeridos al circuito de potencia donde se encuentra la luminaria a ser controlada.

La comunicación entre el dispositivo *Android* y la placa Arduino se realiza mediante la norma de comunicación inalámbrica más usado entre dispositivos *Bluetooth*, para ello se debe utilizar un módulo *Bluetooth* que esté conectado al Arduino.

El circuito de potencia se conforma por un relevador (*relay*) simple que debe actuar como un interruptor para el activado y desactivado de la luz mientras que para el control de intensidad de luz, el circuito presenta un detector de cruce por cero que permite sincronizar la red eléctrica con el disparo del tiristor conectado en serie a las bombillas.

ÍNDICE

INTRODUCCIÓN

Debido al auge de los dispositivos inteligentes como *smartphones* y/o *tablets*, la mayoría de personas cuentan con un equipo móvil. Por ende, la posibilidad de hacer uso de dichos dispositivos aprovechando sus características para realizar sistemas de automatización en un hogar se hace más viable y económica con el paso del tiempo. Además, los sistemas automatizados en casa no solo dan la posibilidad de brindar un servicio a las personas dispuestas a adquirir dicho sistema domótico sino que también, dado que se trata de un control a distancia, sería de gran ayuda a personas con alguna discapacidad física, haciendo posible mejorar la calidad de vida de ellos y también de personas de avanzada edad.

Dada las razones previamente mencionadas, el presente trabajo tiene como propósito implementar una aplicación domótica la cual consiste en controlar la luminaria en un determinado ambiente cerrado de forma remota, ya sea una habitación, oficina o casa. Sin embargo, cabe mencionar que existen ciertas limitaciones en el sistema empleado como por ejemplo el tipo de arquitectura centralizada teniendo como único nodo la tarjeta Arduino. Este inconveniente en la topología puede provocar en caso que la posición del celular no tenga el alcance suficiente para conectarse al módulo *Bluetooth*, no habría posibilidad de establecer la comunicación y por ende, la transmisión de datos no se realizaría y no se podría controlar la luminaria.

Dicho sistema implementado pretende ser simple, sencilla, de bajo costo y compatible con la mayoría de dispositivos *Android* que existen en el mercado; además de fomentar la utilización de tecnología y dispositivos recientes. Para lograr la compatibilidad con la mayoría de dispositivos *Android*, la interfaz de la aplicación se limita a realizar con efectividad la comunicación por *Bluetooth* sin importar librerías avanzadas gráficas al código de la aplicación que solicitan versiones de dispositivos *Android* mínimas requeridas, con ello se lleva a cabo la simplicidad y compatibilidad con los equipos celulares. Haciendo uso de dispositivos actuales y además de ajustarse apropiadamente a la necesidad requerida en este trabajo, la tarjeta de *hardware* libre Arduino es adquirida para realizar la tarea del microprocesador que activará los puertos de salidas digitales

conectados a los focos para encenderlos o atenuarlos, dependiendo el caso. Además esta placa cuenta con puertos físicos seriales que son necesarios para la conexión con el módulo *Bluetooth*. Para la etapa de potencia, se trató de minimizar el número de dispositivos electrónicos y potencia de consumo de dichos elementos. Esto implica un menor tamaño de la tarjeta PCB que estará conectada a la placa Arduino.

Debido a que este trabajo presenta elementos y tecnologías actuales como *Android* y la plataforma Arduino, la mayor cantidad de contenido bibliográfico proviene de páginas web oficiales y confiables. Sin embargo también fue posible obtener referencias de algunos libros disponibles de manera gratuita en internet, especialmente de *Android* que ya cuenta con ciertas editoriales que publican libros de programación en esta plataforma. Es también necesario aclarar que la mayor cantidad de estas fuentes se encuentran en el idioma inglés por lo que las referencias se encuentran en dicho idioma, no existe mucha información de ese tipo en español. Para temas relacionados a la etapa de potencia, sí fue posible referenciar libros de ediciones antiguas ya que la teoría se ha mantenido en el tiempo y los circuitos siguen siendo los mismos. Igualmente, cabe mencionar que se obtuvo referencia de trabajos de tesis de grado de otras universidades. Gracias a la naturaleza práctica del proyecto, fue posible comprobar experimentalmente lo señalado en el fundamento teórico y diseño de los elementos de los circuitos, además del código de los programas.

CAPÍTULO I

PLANTEAMIENTO DE INGENIERÍA DEL PROBLEMA

1.1 Descripción del problema

La domótica es el área de la ingeniería encargada del diseño de un conjunto de sistemas que automatizan las diferentes instalaciones de una vivienda. A medida que la tecnología ha avanzado, los elementos que controlan las distintas variables en un hogar han ido reduciendo el costo de implementación, el tamaño de los dispositivos y las posibilidades de usar dispositivos cotidianos como instrumentos de control remoto. Bajo esta temática, los teléfonos inteligentes juegan un papel crucial en la actualidad dado que cuentan con sistemas operativos, sensores y tecnologías de comunicación inalámbrica disponibles para el uso y desarrollo de un usuario. Bajo esta perspectiva, este trabajo está enfocado en lograr el control de una variable común del hogar como lo es la iluminación analizando la factibilidad de la implementación de un sistema capaz de controlar la luminaria, tanto el encendido y apagado, así como también la regulación de la intensidad luminosa.

1.2 Objetivos del Trabajo

Los objetivos de este trabajo se dividen en dos tipos los cuales se detallan a continuación.

1.2.1 Objetivo General

El objetivo es diseñar e implementar un sistema que sea capaz de controlar las luces del hogar y en la cual el usuario pueda manejarlo fácilmente mediante una aplicación móvil.

1.2.2 Objetivos Específicos

- Desarrollar la aplicación bajo la plataforma *Android*, debido a su mayor consumo a nivel mundial y nacional.
- Diseñar la aplicación con el menor número de herramientas gráficas avanzadas para lograr la mayor compatibilidad con dispositivos *Android* antiguos.

- Utilizar una tarjeta de desarrollo popular, multiplataforma y de bajo costo como el Arduino.
- Simplificar el circuito de potencia para su menor tamaño en un PCB y de acuerdo a la carga utilizada la cual es en este caso una carga resistiva.

1.3 Limitaciones del Trabajo

- La aplicación móvil sólo es implementada en dispositivos Android sin posibilidad de instalarlo en otros teléfonos inteligentes como *Iphone* o *Blackberry*.
- La cantidad de puertos de salida digital libres del Arduino UNO se limita a 11, por lo cual en caso de requerir mayor cantidad de luces, se necesitaría otra tarjeta Arduino con mayor cantidad de puertos, por ende incrementándose el costo.
- El sistema de control de intensidad de la luz es posible solo para focos incandescentes, limitando la posibilidad de usar luminaria fluorescente comúnmente usada.

1.4 Metodología

Para implementar este sistema, se llevó a cabo el diseño por partes o etapas, dividiéndose de la siguiente forma:

- Diseño de la aplicación móvil
- Diseño del subsistema del Arduino
- Comunicación *Bluetooth* de los dispositivos *Android* y Arduino
- Diseño de la etapa de potencia

Para el desarrollo de la aplicación móvil, se usará el Entorno de Desarrollo Integrado (*IDE*) con el Kit de Desarrollo de *Software* (*SDK*) de *Android*, el cual contiene las librerías necesarias para crear la aplicación. El lenguaje de programación a utilizar en *Android* es *Java*.

Arduino presenta su propio entorno de desarrollo basado en *Processing* al igual que su lenguaje de programación propio basado en *Wiring*, con la ventaja de ser multiplataforma. Para el presente trabajo, todo el *software* utilizado fue utilizado en el sistema operativo Linux.

El estándar de comunicación *Bluetooth* es usada para la conectividad y transmisión de datos de los dispositivos. Para ello se utiliza un módulo *Bluetooth* JY-MCU de bajo costo y simple uso. La velocidad de transmisión está configurada por defecto a 9600 baudios, por lo que no habría inconvenientes en cuanto a configuración del dispositivo.

Para finalizar se diseñará el circuito de potencia en un *protoboard* con los integrados necesarios para la elaboración del circuito detector de cruce por cero y regulador de intensidad también llamado *dimmer*. Los elementos a utilizar son principalmente 2 optoacopladores, un TRIAC y resistencias, las cuales serán diseñadas de acuerdo a las especificaciones de las hojas de datos de los integrados.

CAPÍTULO II

MARCO TEÓRICO CONCEPTUAL

2.1 Antecedentes del Problema

La necesidad de integrar la tecnología en un recinto para facilitar o automatizar tareas siempre ha estado presente en el tiempo. En tiempos antiguos, la adquisición de sistemas para controlar diversas variables en un hogar (luz, temperatura o equipos tecnológicos) se consideraba casi un lujo. Sin embargo, gracias a los grandes avances en la electrónica e informática, la automatización de una vivienda se puede llevar a cabo de manera económica y definiendo los propios dispositivos del usuario como los elementos que controlarán dichas variables. A todo el conjunto de sistemas que automatizan las diferentes instalaciones de una vivienda se le denominó domótica.

El automatismo se inició durante el siglo XIX con el desarrollo industrial. Con el paso del tiempo, los sistemas se han ido perfeccionando hasta llegar al punto en donde las industrias basan gran parte de sus fases de producción en tareas automatizadas. Mucho tiempo después llegó noción de un edificio o inmueble inteligente siendo Estados Unidos y Japón los países pioneros. Comercialmente, el origen de la domótica se remonta a los años setenta, cuando en Estados Unidos aparecieron los primeros dispositivos de automatización de edificios basados en la tecnología X-10, el cual permite el uso del cableado de corriente alterna existente para transmitir señales y controlar remotamente la luminaria y electrodomésticos del hogar, creando una red de dispositivos. En 1978, luego de varios años refinando la tecnología, los productos X10 comenzaron a aparecer. Se formó una sociedad con BSR, conocida como X10 Ltd, y el sistema X10 BSR nació. Dicho sistema consistía de una consola de comando de 16 canales, un módulo Lámpara, y un módulo Aparato. Pronto apareció el módulo Interruptor de pared y el primer temporizador X10 [1]. Aunque este sistema es 100% escalable, su mayor inconveniente es que depende directamente de la calidad con la que llega la señal eléctrica, lo que no lo hace un sistema muy fiable.

Posteriormente a fines de los 80, comenzaron a aparecer sistemas con cableado estructurado repartido por todo el recinto para facilitar la conexión de distintos tipos de

periféricos y terminales. Los sistemas con electrodomésticos avanzados y otros dispositivos automáticos llegaron a comienzos de los años noventa, junto con el desarrollo de las computadoras personales, las cuales llevaron al nacimiento de aplicaciones de control, seguridad, comunicaciones.

Finalmente, gracias al veloz desarrollo de la tecnología, surgen los sistemas domóticos inalámbricos usando estándares de comunicación como *Wi-Fi*, *Bluetooth*, *Zigbee* o GSM, siendo aunque con diferentes propósitos. Por ejemplo *Wi-Fi* está destinado a la conexión a Internet o a una red de computadoras locales, mientras que la red GSM conecta dispositivos a través de la red celular. *Bluetooth* está destinada a la conexión de periféricos con mucho menor consumo de potencia que las anteriores [2]. *Zigbee* está enfocado a la conectividad con topología de malla teniendo distintos nodos distribuidos a lo largo del recinto y de esta manera ampliando el rango de conectividad, además en caso que algún nodo falle, la comunicación entre los demás nodos no se vea afectada de ninguna forma[3].

Paralelamente, los dispositivos móviles evolucionaron hasta ser equipos con muchas características implementadas dentro del equipo tanto en *hardware* como en *software*. A estos dispositivos se les denominaron teléfonos inteligentes, dado que tenían un sistema operativo que controla el *hardware*, haciendo también posible además que estos dispositivos sean programables de acuerdo a la necesidad del usuario y publicando otras aplicaciones de distintos desarrolladores. Todo este avance tecnológico ha hecho posible que aquellos sistemas domóticos que eran difíciles y/o costosos para implementar en un hogar, sean posibles implementarlas el día de hoy.

2.2 Estado del arte

Actualmente los dos principales sistemas operativos que presentan los teléfonos inteligentes y *tablets* son los populares sistemas *iOS* y *Android* las cuales pertenecen a las reconocidas empresas *Apple* y *Google* respectivamente. Sin embargo, mundialmente la cantidad de teléfonos móviles con sistema operativo *Android*, supera de manera categórica a los demás como muestra la cuota de mercado realizada por la *International Data Corporation* (*IDC*). Igualmente a nivel nacional también se muestra una preferencia a este sistema debida en gran parte a modelos económicos de *Android* y múltiples empresas que las producen a diferencia de iOS. La tabla 2.1 muestra el incremento que hubo del tercer trimestre del año 2012 al tercer trimestre para el año 2013.

TABLA N° 2.1 Cuota de mercado del tercer trimestre 2012 y 2013 [4]

Sistema Operativo	3Q13 Volúmenes de Envío	3Q13 Cuota de mercado	3Q12 Volúmenes de Envío	3Q12 Cuota de mercado	Cambio Año tras Año
Android	211.6	81.0%	139.9	74.9%	51.3%
iOS	33.8	12.9%	26.9	14.4%	25.6%
Windows Phone	9.5	3.6%	3.7	2.0%	156.0%
BlackBerry	4.5	1.7%	7.7	4.1%	-41.6%
Others	1.7	0.6%	8.4	4.5%	-80.1%
Total	**261.1**	**100.0%**	**186.7**	**100.0%**	**39.9%**

Como se mencionó, la popularidad de esta plataforma también se observa en Perú y Latinoamérica ya que según el Mapa de Colonización *Mobile* 2013, los dispositivos *Android* lideran la preferencia ampliamente sobre los demás sistemas operativos tal y como lo muestra la tabla 2.2.

TABLA N° 2.2 Cuota de mercado a nivel de Perú [5]

Sistema Operativo	*Smartphone*
Android	66.59%
iOS	23.20%
Windows Phone	5.35%
BlackBerry	2.42%

2.3 *Android*

Android es un sistema operativo basado en el *kernel* de Linux para dispositivos móviles desarrollado inicialmente por la empresa *Android Inc.* En el año 2005, fue adquirida por *Google* y lanzada oficialmente en el año 2007 bajo la *Open Handset Alliance* (*OHA*), el cual es un consorcio de compañías de *hardware* y *software*. En el 2008 se lanzó el primer teléfono inteligente con el sistema operativo de *Android*, la línea de tiempo se muestra en la figura 2.1.

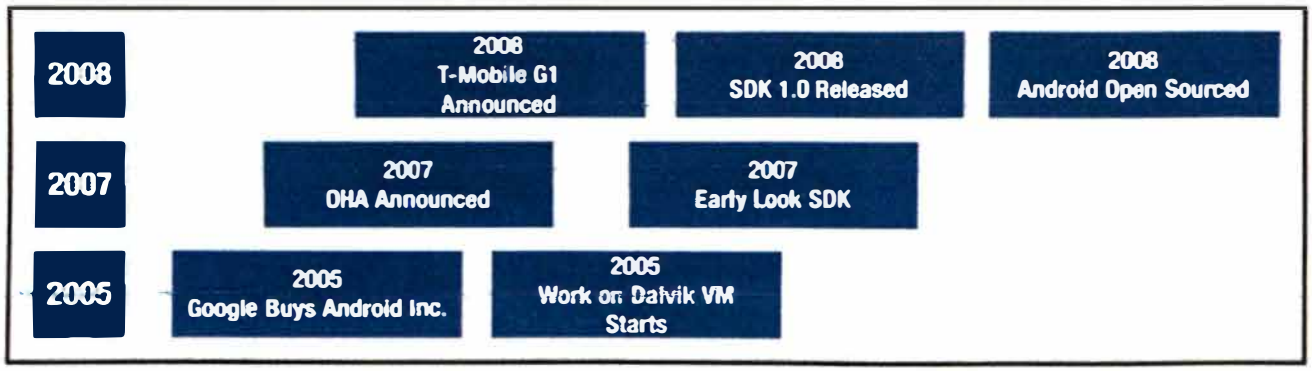


Figura 2.1 Línea de tiempo de *Android* [6]

Una de sus principales características es la posibilidad de desarrollar aplicaciones en un ambiente de desarrollo abierto ya que las aplicaciones bajo esta plataforma no se

construyen bajo sistemas propietarios que priorizan las aplicaciones nativas sobre las creadas por terceros, restringiendo la comunicación entre la información nativa del dispositivo y las aplicaciones. Todas las aplicaciones creadas tienen acceso al *hardware* a través de una serie de librerías o *APIs* provenientes del kit de desarrollo de *software* (*SDK*) de *Android*.

Figura 2.2 Interfaz gráfica de dispositivos *Android*

2.3.1 Arquitectura

La plataforma de *Android* es una pila de *software* que va desde los servicios del sistema operativo de bajo nivel que manejan el dispositivo hasta las aplicaciones del móvil. La arquitectura de *Android* está compuesta por diferentes capas como se muestra en la figura 2.3.

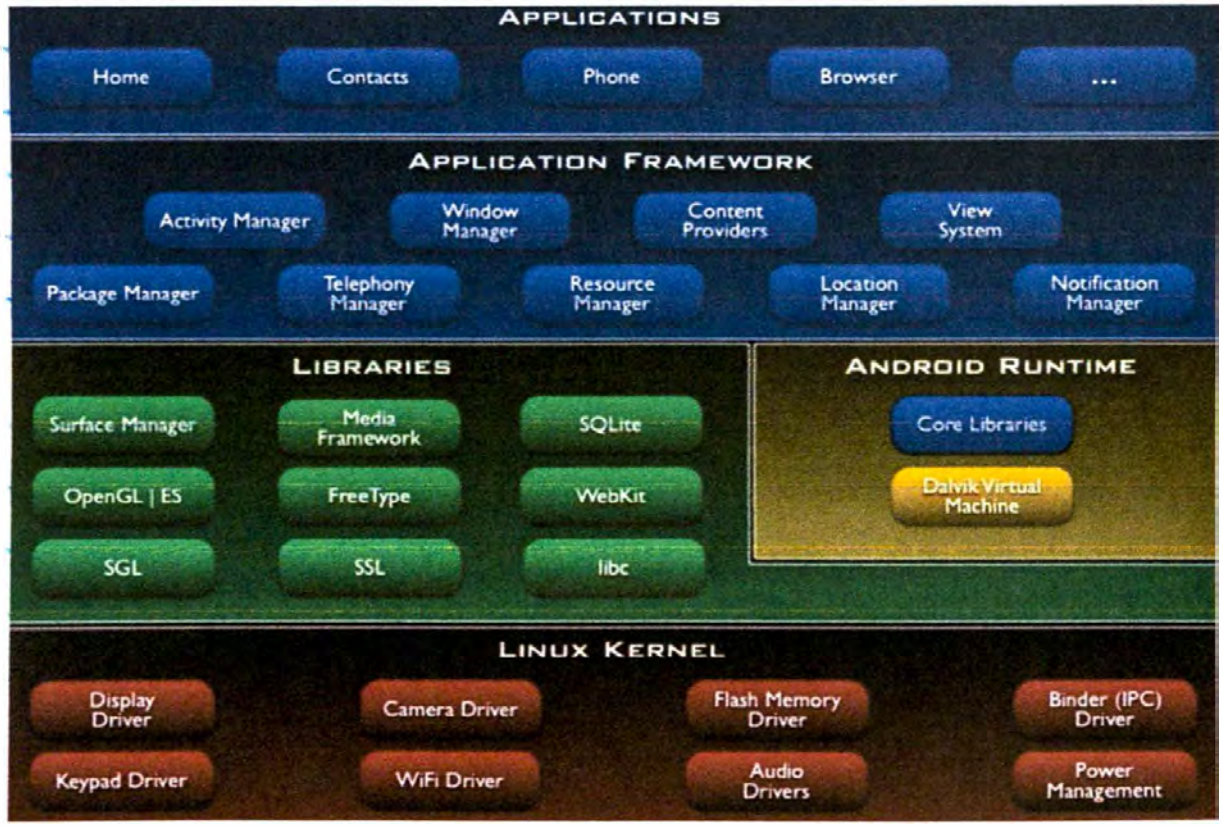


Figura 2.3 Arquitectura de *Android* [7]

- ***Kernel* de Linux:** Es la capa más baja de la pila, la cual provee los principales servicios genéricos del sistema operativo. Por ejemplo, provee los permisos y restricciones a las aplicaciones para acceder a las distintas características del *hardware*. Además se encarga de la administración de la memoria junto al manejo de procesos I/O de archivos y de la red. Los controladores de dispositivos añadidos son también administrados por esta capa. En resumen, el *Kernel* de Linux controla y realiza la comunicación del *hardware* con el *software*.

- **Librerías:** La siguiente capa contiene una serie de librerías del sistema también referidas como librerías nativas las cuales están escritas en el lenguaje C o C++. Por ejemplo, Bionic Libc que es una implementación del sistema C estándar (libc) que se encarga de los procesos, creación de hilos, computo matemático o asignación de memoria. También cuenta con la librería multimedia (*Media Framework*) para la reproducción de audio y video, además de contener la base de datos relacional *SQLite*.

  Además en esta capa se encuentra el sistema en tiempo de ejecución (*Android Runtime*) compuesta por las librerías principales de *Java* y la máquina virtual Dalvik. Dado que las aplicaciones se encuentran escritas en lenguaje *Java*, las librerías principales también se encuentran escritas en dicho lenguaje y contiene a las principales clases de *Java* como *java* y *javax*. Mientras que quien ejecutará las aplicaciones será la máquina virtual de *Java* diseñada para ambientes con recursos limitados como lo son los dispositivos móviles que en comparación a las computadoras presentan menor memoria RAM, menor CPU y batería limitada. Esta es la máquina virtual Dalvik [6].

- **Infraestructura de Aplicación:** Esta capa contiene el *software* reutilizable que un dispositivo puede necesitar. Está compuesto mayoritariamente por *Managers*, los cuales son necesarios al programar una aplicación para acceder a ciertos servicios del sistema como por ejemplo el *Window Manager* que maneja las distintas ventanas que contienen las aplicaciones, el *Location Manager* que se encarga de brindar distintos valores de posicionamiento del dispositivo o el *View System* que provee los elementos o iconos en la interfaz del sistema. Además existe el *Activity Manager* que maneja el ciclo de vida de una aplicación el cual será explicado más adelante.

- **Aplicaciones:** En esta última capa se encuentran las aplicaciones que ya presenta el dispositivo por defecto como el lector de correo, la lista de contactos, marcador de número telefónico, la pantalla principal, etc.

2.3.2 Componentes de una aplicación

En *Android*, una aplicación presenta los siguientes componentes:

- **Actividades (*Activities*):** Una actividad presenta la interfaz gráfica de la aplicación al usuario, es decir, crea los componentes de la interfaz que se muestra en la pantalla. Además, captura la interacción del usuario con los elementos de la aplicación. Cada actividad está enfocado a cumplir una sola tarea para el usuario, esto se puede deber al hecho que los dispositivos que contienen el sistema operativo *Android* son limitados en tamaño, por ello que cada actividad presente una interfaz gráfica diferente que se superponga a la actividad anterior. Las actividades en una aplicación son independientes pero pueden interactuar entre sí. Por ejemplo, la aplicación encargada del manejo de los mensajes de texto contiene diferentes actividades, uno de ellos es la visualización del total de mensajes recibidos o enviados divididos por contacto. Al presionar uno de ellos, se crea otra actividad que muestra el contenido de todos los mensajes con dicho contacto, mientras que al presionar el ícono de responder mensaje, lanza otra actividad encargada de escribir el texto de respuesta. De esta forma cada actividad realiza una tarea diferente pero todas o algunas conectadas entre sí.

  La figura 2.4 muestra el ciclo de vida de una actividad. Como se puede observar, al momento de abrir la aplicación, el tiempo total de vida empieza con la función *onCreate*() de la clase *Activity* principal, usualmente en este método se inicializa los componentes principales como la interfaz gráfica, mientras que en el método *onDestroy*() la vida de la aplicación se termina, por ello se debe ejecutar todo el código que libere los recursos usados por la actividad durante la aplicación.

  El tiempo de vida visible de una actividad ocurre entre la llamada a la función *onStart*() hasta que se llame a la función *onStop*(). Se dice solo visible debido a que a pesar de ya haberse creado los elementos de la interfaz y mostrándose en la pantalla, aún el usuario no puede interactuar con dichos elementos.

  El tiempo de vida en primer plano de una actividad sucede entre la llamada de la función *onResume*() hasta la llamada a la función *onPause*(). Durante este tiempo, el usuario puede interactuar con los elementos de la actividad, la cual está en frente de todas las actividades que se hayan abierto. Una actividad puede variar frecuentemente

entre estos estados como por ejemplo al ir el dispositivo al modo *sleep*, por ello se recomienda no sobrecargar de código a estos métodos para evitar una transición lenta entre actividades.

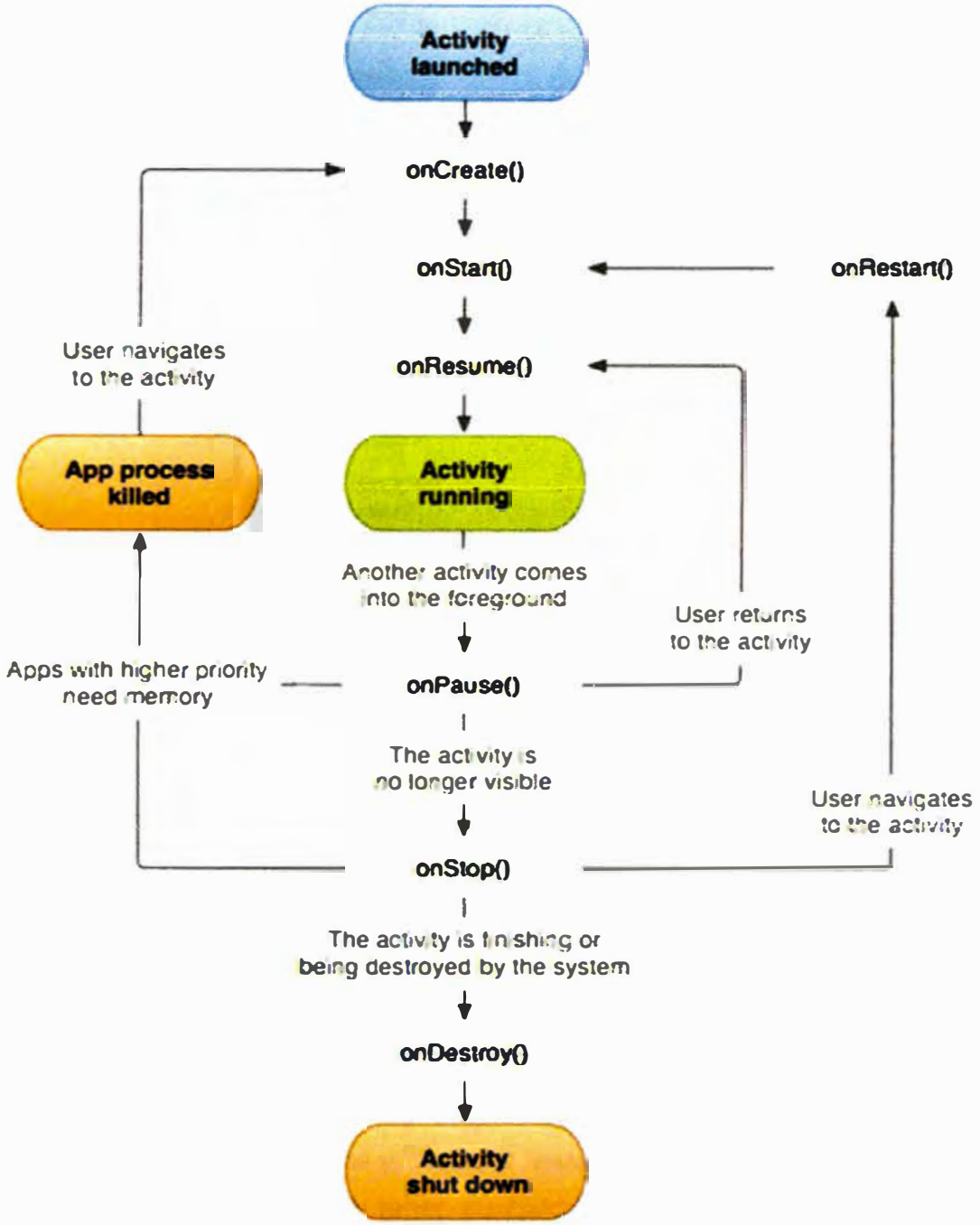


Figura 2.4 Ciclo de vida de una actividad [8]

- **Servicios (*Services*):** Los servicios son componentes que se ejecutan en segundo plano, normalmente son operaciones de larga duración o ejecución de procesos remotos. Por ello, un servicio no presenta una interfaz gráfica, como por ejemplo un

servicio que reproduce música en segundo plano o realiza una descarga a través de la red sin bloquear la interacción del usuario con otra actividad. De esta forma el usuario puede utilizar otra actividad mientras se ejecutan los servicios en otro plano.

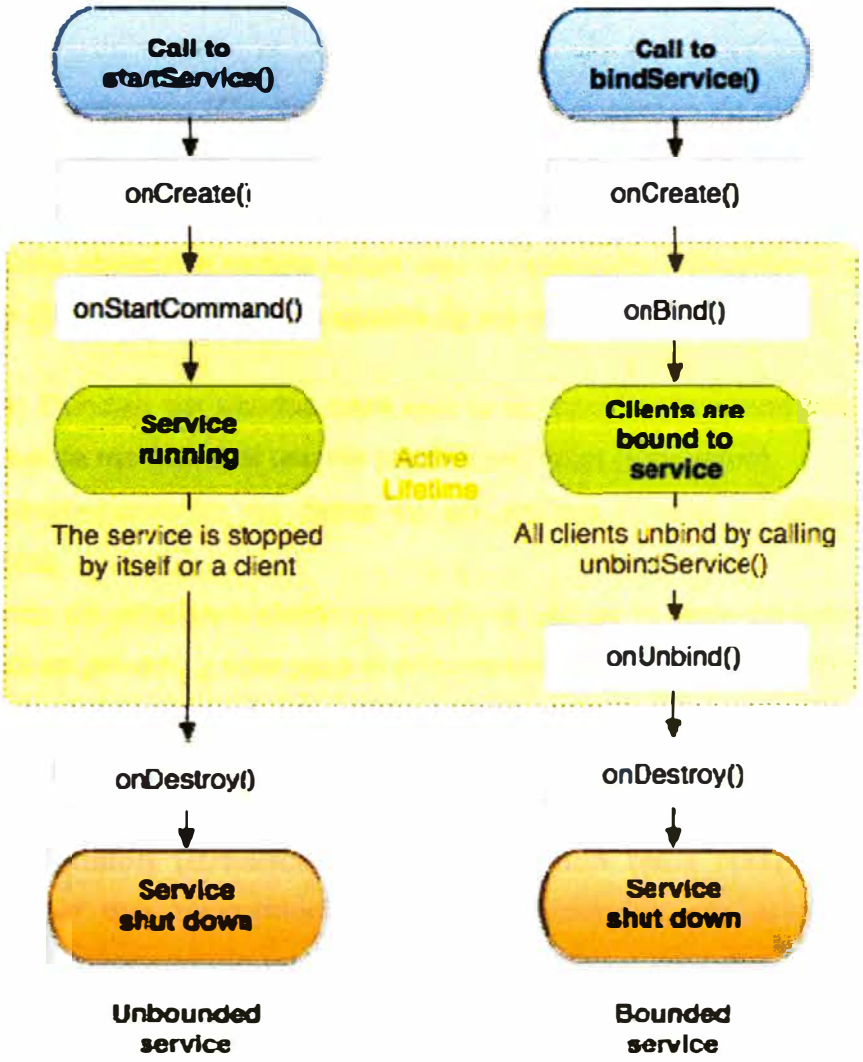


Figura 2.5 Ciclo de vida de un servicio [9]

La figura 2.5 muestra el ciclo de vida de un servicio. A diferencia de una actividad, un servicio presenta un ciclo de vida simple de un solo sentido. Esto se debe a la ausencia de una interfaz gráfica y a que el usuario no interactúa directamente con el servicio, por lo que se debe tener cuidado al crearlos, ya que pueden seguir ejecutándose sin que el usuario se percate.

Similar al tiempo total de vida de la actividad, el tiempo de un servicio se inicia con el método *onCreate*() y finaliza con el método *onDestroy*(), asignando las variables a usar y liberando los recursos utilizados respectivamente. Luego en el tiempo de vida activo se inician con el método *onStartCommand*() o el método *onBind*(), dependiendo

del tipo de implementación del servicio. En el primer caso, un componente como una actividad inicia el servicio para realizar una tarea específica como la descarga de un archivo. En el segundo caso, un componente necesita unirse a un servicio para interactuar mediante una conexión de larga duración, para cerrar el enlace se llama al método *onUnbind*().

- **Proveedores de Contenido (*Content Providers*):** Se encargan de manejar el acceso a un conjunto estructurado de datos de las aplicaciones, con ello otras aplicaciones pueden solicitar estos datos y modificarlos en caso de tener permiso. Para acceso a datos internos, una aplicación puede hacer uso de cualquier mecanismo de acceso o almacenamiento [6]. Puede ser de cualquiera de las siguientes formas:

  - **Preferencias:** Pueden ser usadas para que la aplicación almacene información de forma permanente mediante el uso de pares llave/valor (*key/value*).
  - **Archivos:** Almacenamiento de datos en un archivo interno en algún medio de almacenamiento.
  - ***SQLite*:** Método de almacenamiento mediante el uso de la base de datos relacional *SQLite*, el cual es privado y solo para el almacenamiento en la aplicación.
  - **Red:** Un mecanismo que permite recuperar o guardar los datos externamente mediante Internet vía HTTP.

- **Receptores de difusión (*Broadcast Receivers*):** Son otros componentes de un proceso en *Android* que responden a mensajes enviados por el sistema. Dicho mensaje puede ser invocado y/o respondido por más de un receptor. Un componente como un servicio o una actividad puede usar el método *sendBroadcast*() para enviar un evento a otras actividades o servicios para que al momento de recibir el evento, ejecute la tarea programada en el código [6].

- **Intenciones (*Intents*):** Son mensajes que describen una acción a ser ejecutada, proveyendo posteriores vínculos entre los códigos de diferentes actividades, servicios e incluso otras aplicaciones del dispositivo. Aunque las intenciones facilitan la comunicación de diferentes maneras, existen tres casos de uso fundamentales:

  - Iniciar una actividad
  - Iniciar un servicio
  - Lanzar un mensaje *broadcast*

2.4 Arduino

De acuerdo a la propia página oficial de Arduino, éste se define como "una plataforma de electrónica abierta para la creación de prototipos basada en *software* y *hardware* flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos" [10]. De esta forma, se tiene una placa que contiene un microcontrolador que puede detectar diversas variables del ambiente mediante la recepción de la señal de una variedad de sensores en sus puertos de entradas analógicas y así mediante sus puertos de salidas digitales controlar distintos actuadores como luces o motores.

En la actualidad existen muchas tarjetas de desarrollo de diferentes universidades o entidades y con diferentes arquitecturas. Sin embargo, una de las más reconocidas y pioneras es esta plataforma con muchas ventajas sobre otras. Dentro de sus ventajas están [11]:

- **Asequible:** Las placas Arduino son más asequibles comparadas con otras plataformas de microcontroladores. Los precios van desde 20 € hasta 60 €, dependiendo del modelo de la tarjeta. A nivel nacional, las distribuidoras oficiales ofrecen el precio de S/.85 para la tarjeta Arduino UNO Rev3 la cual es la más común y económica.
- **Multi-Plataforma:** El *software* de Arduino funciona en los sistemas operativos *Windows*, *Macintosh OSX* y Linux. La mayoría de los entornos para microcontroladores están limitados a *Windows*.
- **Entorno de programación simple y directo:** El entorno de programación de Arduino es fácil de usar para principiantes y lo suficientemente flexible para los usuarios avanzados. Arduino está basado en el entorno de programación de *Processing* con lo que el estudiante que aprenda a programar en este entorno se sentirá familiarizado con el entorno de desarrollo Arduino.
- ***Software* ampliable y de código abierto:** El *software* Arduino esta publicado bajo una licencia libre y preparado para ser ampliado por programadores experimentados. El lenguaje puede ampliarse a través de librerías de C++, y si se está interesado en profundizar en los detalles técnicos, se puede dar el salto a la programación en el lenguaje AVR C en el que está basado. De igual modo se puede añadir directamente código en AVR C en los programas.
- ***Hardware* ampliable y de Código abierto:** Arduino está basado en los microcontroladores ATMEGA168, ATMEGA328 y ATMEGA1280. Los planos de los módulos están publicados bajo la licencia *Creative Commons*, por lo que diseñadores

de circuitos con experiencia pueden hacer su propia versión del módulo, ampliándolo u optimizándolo. Incluso usuarios relativamente inexpertos pueden construir la versión para placa de desarrollo para entender cómo funciona y ahorrar algo de dinero.

2.4.1 Entorno de programación

Respecto a la programación del microcontrolador de la tarjeta, ésta se realiza utilizando el entorno de desarrollo de Arduino con su lenguaje de programación propio está basado en el lenguaje *Wiring*. Dicho entorno contiene un editor de texto, un área de mensaje, una consola de texto, una barra de herramientas con botones de funciones comunes y una barra de menú como se muestra en la figura 2.6.

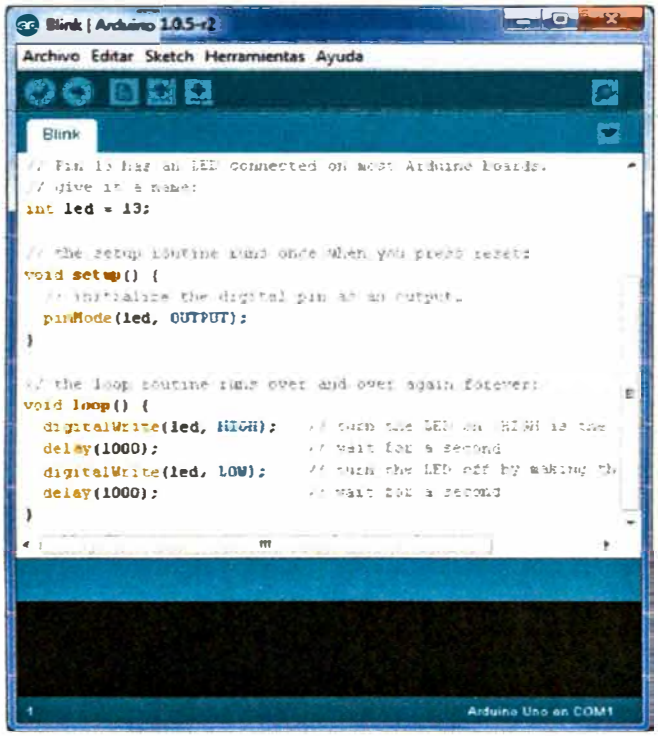


Figura 2.6 Interfaz del Arduino IDE

Los programas escritos en Arduino se les denominan *sketches* los cuales se guardan con la extensión ino. Luego de escribir el código de un *sketch* se procede a compilarlo para verificar que no exista ningún error en tiempo de compilación, es decir errores de sintaxis o de semántica y finalmente se carga el programa a la tarjeta. Los *sketches* presentan en su estructura dos funciones principales, las cuales no devuelven ningún valor ni se les asigna ningún parámetro:

- **Función *setup*():** esta función es llamada por única vez cuando se inicia el *sketch*, al encender o reiniciar la placa. Es usualmente utilizado para inicializar variables y definir el modo, sea de entrada o salida, de los pines analógicos o digitales a usar. Dentro de

esta función también se puede definir las rutinas de interrupción y la asignación de velocidad de transferencia de datos de un puerto serial de la tarjeta.

- **Función *loop*():** luego de crear la función *setup* que inicializa y asigna los valores iniciales, la función *loop* será la función que se ejecutará consecutivamente y de manera continua durante todo el tiempo que esté encendida la tarjeta. De esta forma, esta función controla activamente la placa Arduino durante su funcionamiento.

Para este trabajo es importante ver la definición de la función de interrupción, la cual se define dentro de la función *setup*(). Su definición e inicialización está dada por el comando *attachInterrupt*(interrupción, función, modo). A continuación se explica los valores que se asigna a dichos parámetros [12]:

- **Interrupción:** Es el primer parámetro del comando, en donde se define el puerto que recibirá la señal externa de interrupción. La mayoría de modelos presentan dos pines de interrupciones externas, el número 0 para el pin de interrupción 2 y el número 1 para el pin 3. La tabla 2.3 muestra los puertos de interrupción disponibles en distintos modelos.

TABLA N° 2.3 Puertos de interrupción disponibles [12]

Modelo Arduino	int.0	int.1	int.2	int.3	int.4	int.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	

- **Función:** Previamente se debe definir la función que se ejecutará en el momento que llegue la señal de interrupción. Sin embargo, no se puede definir cualquier función, deben cumplirse 2 condiciones para esta función. En primer lugar, la función no debe devolver ningún tipo de valor, es decir debe estar definida como *void*. En segundo lugar, la función no puede admitir parámetros.

- **Modo:** Este parámetro define el momento en el cual se ejecutará la función creada. Existen cuatro valores predefinidos para señalar el momento en que actuará la función [12]:

  - ***LOW*:** Este valor dispara la interrupción en el momento en que el valor del puerto es bajo.
  - ***CHANGE*:** Este valor dispara la interrupción cuando el puerto cambia de valor, sea de valor bajo a alto o de alto a bajo.

- ***RISING***: Este valor dispara la interrupción cuando el puerto cambia de valor bajo a nivel alto.
- ***FALLING***: Este valor dispara la interrupción cuando el puerto cambia de valor alto a nivel bajo.

2.4.2 Productos de Arduino

Debido a que Arduino es una plataforma de código abierto tanto en *software* como en *hardware*, ha sido posible tener una gran variedad de modelos de placas, dependiendo del objetivo que se tenga del proyecto a realizar. De esta forma existen tarjetas con diferentes microprocesadores, mayor o menor cantidad de entradas y salidas digitales, módulos montados en la misma placa como el modelo Arduino BT que contiene un módulo *Bluetooth* integrado en la tarjeta. Entre sus productos más reconocidos se tienen [13]:

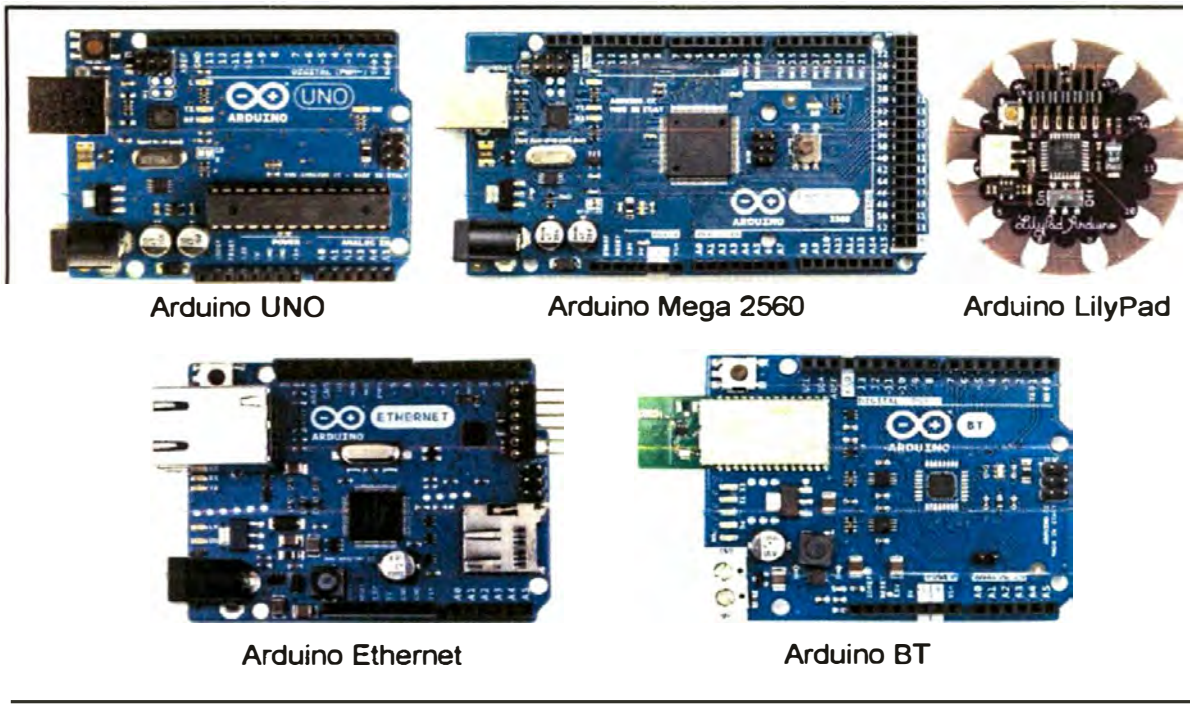


Figura 2.7 Variantes de una placa Arduino

- **Arduino UNO**

Este modelo es el más usado para proyectos electrónicos y para propósitos educativos debido a que es el más económico y con suficientes puertos para controlar diversos periféricos. Éste presenta un microcontrolador ATmega328 con 14 puertos de entradas/salidas digitales de las cuales 6 pueden ser usadas como salidas PWM, 6 entradas analógicas, conexión USB, un reloj de 16MHz, entre otros. Sus especificaciones se muestran en la tabla 2.4.

TABLA N° 2.4 Especificaciones de Arduino UNO

Microcontrolador	ATmega328
Voltaje de Operación	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de Entrada (limites)	6-20V
Puertos I/O digitales	14 (6 proveen salida PWM)
Puertos de Entrada Analógica	6
Corriente DC por puerto I/O	40 mA
Corriente DC para puerto 3.3V	50 mA
Memoria Flash	32 KB (ATmega328)
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad de Reloj	16 MHz

- **Arduino Mega 2560**

Este modelo es otro de los más usados, ya que presenta características similares al Arduino UNO pero cuenta con muchos más puertos tanto digitales como analógicos, además de poseer cuatro puertos seriales implementados en *hardware*. Sus especificaciones se muestran en la tabla 2.5.

TABLA N° 2.5 Especificaciones de Arduino Mega 2560

Microcontrolador	ATmega2560
Voltaje de Operación	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de Entrada (limites)	6-20V
Puertos I/O digitales	54 (15 proveen salida PWM)
Puertos de Entrada Analógica	16
Corriente DC por puerto I/O	40 mA
Corriente DC para puerto 3.3V	50 mA
Memoria Flash	256 KB
SRAM	8 KB
EEPROM	4 KB
Velocidad de Reloj	16 MHz

- **Arduino Leonardo**

Este modelo tiene características similares a las anteriores, sin embargo se diferencia en que esta tarjeta tiene comunicación USB incluida, así eliminando la necesidad de un procesador secundario. Esto permite al Arduino Leonardo aparecer conectada a una computadora como un mouse o teclado, además de un puerto COM. Sus especificaciones se muestran en la tabla 2.6.

TABLA N° 2.6 Especificaciones de Arduino Leonardo

Microcontrolador	ATmega32u4
Voltaje de Operación	5V

Voltaje de entrada (recomendado)	7-12V
Voltaje de Entrada (limites)	6-20V
Puertos I/O digitales	20
Puertos de Entrada Analógica	12
Corriente DC por puerto I/O	40 mA
Corriente DC para puerto 3.3V	50 mA
Memoria Flash	32 KB
SRAM	2.5 KB
EEPROM	1 KB
Velocidad de Reloj	16 MHz

2.5 *Bluetooth*

Bluetooth es un estándar de comunicación inalámbrica desarrollado originalmente por Jaap Haartsen y Sven Mattison, ingenieros de *Ericsson* en Suecia en el año 1994. Empezó con el objetivo de reemplazar los cables de accesorios que se conectan a los teléfonos móviles y a computadoras por un enlace de radio de corto alcance. Sin embargo, es en el año 1998 cuando se adopta oficialmente el nombre *Bluetooth* con la fundación del *Special Interest Group* (SIG), siendo sus socios fundadores *Ericsson*, *IBM*, *Intel*, *Toshiba* y *Nokia* [14]. Es así que al año siguiente, en 1999, la especificación de la primera versión es lanzada (*Bluetooth* 1.0). De manera sucesiva, se ha ido incrementando la tecnología hasta la especificación actual 4.0. Esta tecnología está diseñada especialmente para dispositivos de bajo consumo energético con un rango de cobertura limitado. Actualmente, las principales aplicaciones son la transferencia de archivos, conectividad de periféricos y la sincronización de dispositivos. Existen cientos de dispositivos que cuentan con esta tecnología como teléfonos móviles, consolas, teclados, impresoras, computadoras, audífonos, etc.

2.5.1 Pila de Protocolos *Bluetooth*

La pila de protocolos fue desarrollada por la fundación *Bluetooth* SIG y está dividido en dos zonas principales. Como se muestra en la figura 2.8, la primera zona es la compuesta por el *hardware* donde se encuentran la interfaz de radio y la banda base, el cual está encargado de las tareas relacionadas con el envío de información a través de la interfaz de radiofrecuencia. La segunda zona de *software* está encargada de la parte relacionada con las capas superiores de enlace y aplicación donde se encuentran protocolos como L2CAP y RFCOMM los cuales se detallan más adelante. Ambas zonas se comunican a través de la interfaz de controlador de *host* (*HCI*). Cabe destacar que la comunicación para audio es más directa partiendo desde la banda base.

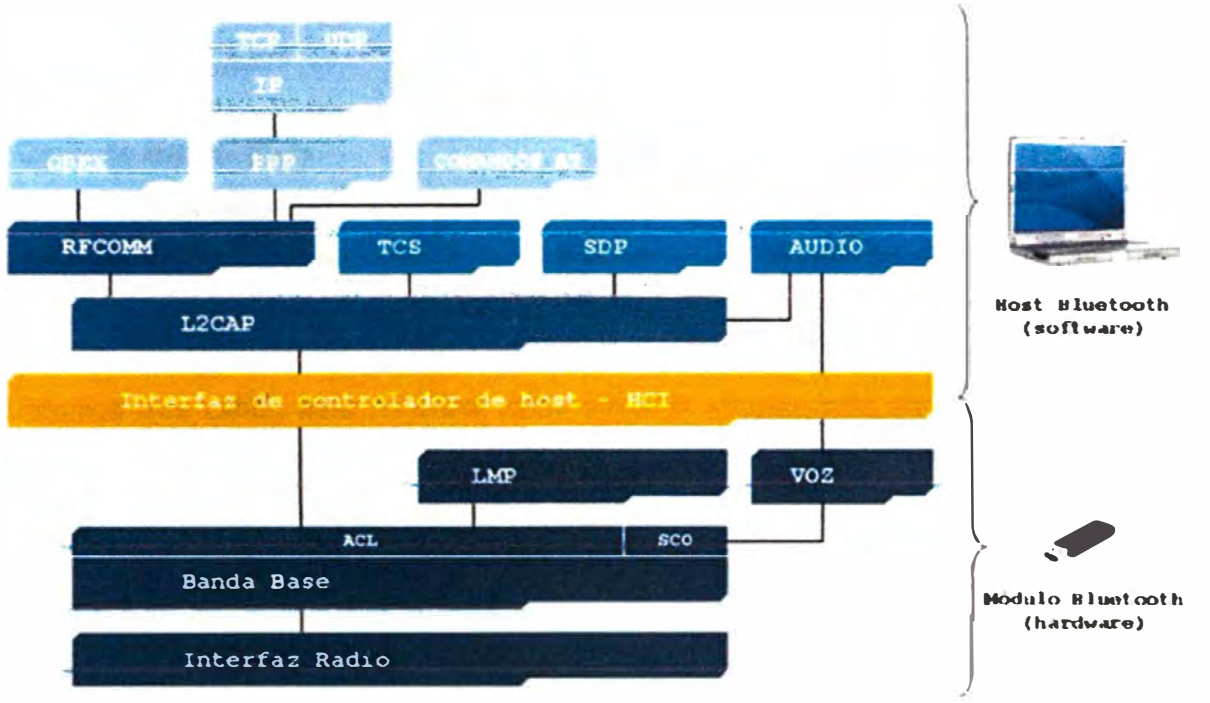


Figura 2.8 Pila de Protocolos *Bluetooth*

La pila de protocolos está compuesta por [14, 15]:

a) Interfaz de Radio

La especificación de Radio es un documento corto donde solo se define la frecuencia de portadora y la potencia de salida. Existen muchas limitaciones que deben ser tomadas en cuenta respecto al diseño de la interfaz de radio debido a que los dispositivos *Bluetooth* típicamente son dispositivos portátiles que funcionan con batería. Por ello se definieron 3 tipos de clases dependiendo del consumo de potencia y en consecuencia el rango de cobertura mostrados en la tabla 2.7.

TABLA N° 2.7 Clases de dispositivos *Bluetooth*

Clase	Potencia máxima permitida	Alcance
Clase 1	100 mW (20 dBm)	100 metros
Clase 2	2,5 mW (4 dBm)	10 metros
Clase 3	1 mW (0 dBm)	1 metro

Bluetooth opera en la banda *ISM* (*Industrial*, *Scientific* and *Medical*) a la frecuencia de 2.4 GHz usando 79 canales entre 2.402 GHz y 2.480 GHz. Un esquema doble de salto de frecuencia y división de tiempo es usado para la transmisión con una tasa de 1600 saltos por segundo. El tiempo entre dos saltos se le denomina *slot*, el cual tiene un intervalo de

625 µs y cada *slot* usa una frecuencia diferente. Este estándar usa 79 saltos o canales igualmente espaciados con 1 MHz.

b) Capa de Banda Base

Las funciones de esta capa son ligeramente complejas dado que no solo está encargado del acceso al medio y el salto de frecuencia para mitigar la interferencia sino que también define los enlaces físicos y muchos formatos de los paquetes. Los paquetes son alternados por TDD (*Time Division Duplex*) entre la transmisión y la recepción.

La técnica que utiliza este estándar para evitar la interferencia es FHSS (*Frequency-Hopping Spread Spectrum*) el cual consiste en dividir la banda de frecuencia en 79 canales de 1 MHz y realizar 1600 saltos por segundo de manera pseudoaleatoria. Esta técnica intercambia eficiencia de ancho de banda por confiabilidad, integridad y seguridad.

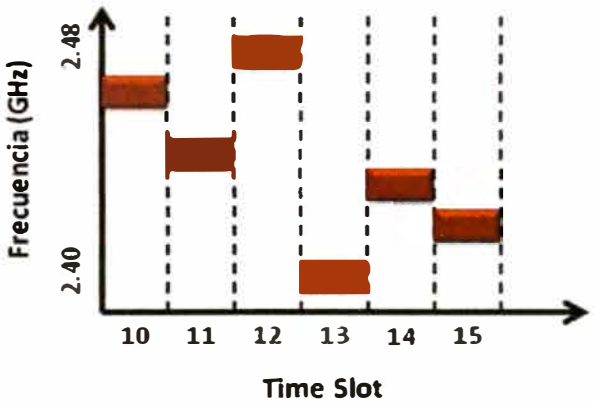


Figura 2.9 Técnica FHSS

La capa de banda base ofrece dos tipos de enlace dependiendo del tipo de aplicación y entorno de operación. El primero es un enlace síncrono orientado a la conexión (*SCO*, *Synchronous Connection-Oriented*) y el segundo es un enlace asíncrono no orientado a la conexión (*ACL*, *Asynchronous Connectionless*) [14]. Este enlace puede tener una tasa de transferencia en subida y bajada asimétrica o simétrica.

- **Enlace Síncrono Orientado a la Conexión (*SCO*):** Este tipo de enlace permite establecer conexiones punto-punto entre un dispositivo maestro y un dispositivo esclavo. Con este tipo de enlace se garantiza que los paquetes lleguen de forma ordenada del dispositivo maestro al esclavo haciéndolo el adecuado para la transmisión de voz. Cada enlace transmite, usualmente voz, a 64 kb/s. En este enlace cada dispositivo maestro reserva 2 slots consecutivos a intervalos fijos, pudiendo soportar hasta 3 enlaces con el mismo o distinto dispositivo esclavo.

- **Enlace Asíncrono sin Conexión (*ACL*):** Permite establecer conexiones punto-multipunto entre un dispositivo que actúa como maestro y otros dispositivos como esclavos. Este tipo de enlace no es el adecuado para la transmisión de voz, sino que es utilizado para la transmisión de datos ya que al tratarse de un enlace de tipo asíncrono, no se garantiza unos retardos de transmisión constantes entre dispositivos. Tampoco se garantiza la entrega de paquetes por lo que, en caso de que un paquete no llegue correctamente a su destino, se retransmitirá de nuevo ese paquete. Puede soportarse una tasa de tráfico asimétrica máxima de 723,2 kb/s y hasta 57,6 kb/s en el sentido de comunicación opuesto, o de 433,9 kb/s por cada uno de los sentidos de comunicación.

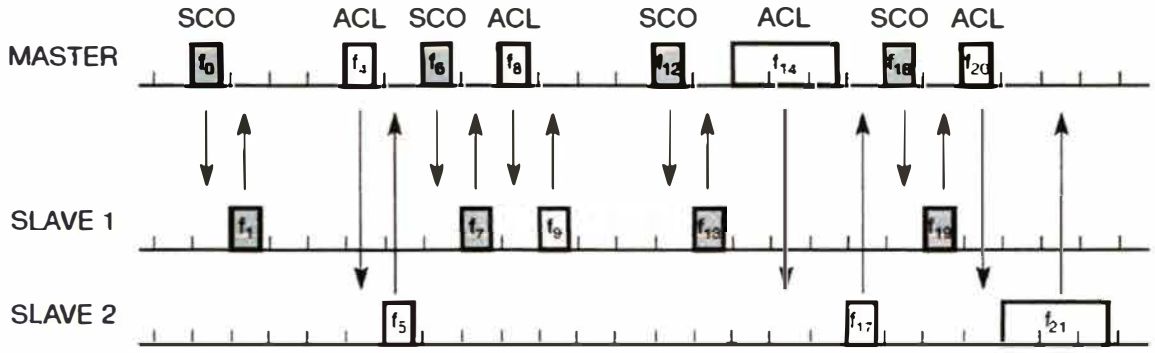


Figura 2.10 Ejemplo de transmisión de datos [14]

c) Protocolo de Gestión de Enlace (*LMP*)

Este protocolo maneja varios aspectos del enlace de radio entre un dispositivo maestro y esclavo. *LMP* mejora la funcionalidad de banda base pero capas superiores pueden aún acceder directamente a la banda base. Este protocolo realiza las siguientes funciones:

- **Autenticación, Emparejamiento y Encriptación:** Aunque la autenticación básica es manejada en la banda base, *LMP* tiene el control de intercambiar números aleatorios y respuestas. El servicio de emparejamiento se necesita para establecer la relación inicial de confianza entre dos dispositivos que nunca se han comunicado antes. El resultado del emparejamiento es una llave de enlace que puede ser cambiada, aceptada o rechazada. Este protocolo no está directamente implicado en el proceso de encriptación pero sí asigna el modo (sin encriptación, punto a punto o *broadcast*), el tamaño de la llave y la velocidad.

- **Sincronización:** En una red *Bluetooth*, la sincronización precisa es de mayor importancia. El valor del reloj es actualizado cada vez que se recibe un paquete del dispositivo maestro. Adicionalmente, paquetes especiales de sincronización pueden ser recibidos
- **Negociación de Capacidad:** No solo la versión de *LMP* puede ser intercambiada, también la información sobre características soportadas. No todos los dispositivos *Bluetooth* soportan todas las características que son descritas en el estándar, así que los dispositivos deben acordar el uso de ciertas características como los paquetes *multi-slot*, encriptación, enlaces *SCO*, etc.
- **Negociación de Calidad de Servicio (*QoS*):** Diferentes parámetros controlan la calidad de servicio de un dispositivo *Bluetooth* en capas inferiores. Dependiendo de la calidad del canal, paquetes en modo directo pueden ser usados, el número de paquetes *broadcast* pueden ser controlados. Un dispositivo maestro puede también limitar el número de *slots* disponibles para esclavos para incrementar su propio ancho de banda.
- **Control de Potencia:** Un dispositivo *Bluetooth* puede medir la potencia de la señal recibida de los demás dispositivos. Dependiendo de su nivel de señal, el dispositivo puede incrementar o disminuir la potencia de transmisión para mejorar la comunicación.
- **Supervisión del Enlace:** *LMP* controla la actividad de un enlace, puede asignar nuevos enlaces sincronizados orientados a la conexión o puede también declarar la falla de un enlace.

d) Interfaz de Controlador de *Host* (*HCI*)

La capa *HCI* (*Host Controller Interface*) es la frontera entre las capas de protocolo de *hardware* y *software*. Éste provee una interfaz de comando al controlador de banda base y el protocolo de gestión de enlace, además brinda acceso al estado del *hardware* y a los registros de control. La especificación de esta interfaz de comunicación define:

- Los comandos *HCI* que son generados por el *Host* para consultar registros *hardware* o iniciar una tarea, como por ejemplo, la búsqueda de dispositivos, el establecimiento de una conexión, o la configuración de parámetros de una conexión.
- Los eventos *HCI* que son generados por el controlador *Bluetooth* en respuesta a los comandos *HCI* o para notificar la ocurrencia de un evento.
- El formato de los paquetes utilizados para la transferencia de datos *ACL* y *SCO* entre el *Host* y el controlador *Bluetooth*.

- El mecanismo de control de flujo usado en la comunicación entre el *Host* y el controlador *Bluetooth*.

e) Protocolo de Adaptación y Control del Enlace Lógico (*L2CAP*)

L2CAP (*Logical Link Control and Adaptation Protocol*) es un protocolo de control de enlace de datos que ofrece canales lógicos entre dispositivos *Bluetooth* con propiedades de calidad de servicio. *L2CAP* está disponible solamente para el tipo de enlace *ACL*. Aplicaciones de audio usando enlaces *SCO* usan la capa de banda base directamente. Este protocolo provee tres tipos diferentes de canales lógicos:

- **Sin conexión:** Estos canales unidireccionales son usados típicamente para mensajes broadcasts de un dispositivo maestro a todos los demás dispositivos esclavos.
- **Orientado a la conexión:** Cada canal de este tipo es bidireccional y soporta especificaciones del flujo de calidad de servicio para cada dirección. Estas especificaciones define la tasa de datos promedio, tasa de datos promedio y latencia.
- **Señalizado:** Este tipo de canal lógico es usado para intercambiar mensajes de señalización entre entidades L2CAP.

Cada canal puede ser identificado por su identificador de canal (*CID*). Canales de tipo señalizado tienen siempre el valor de 1, un *CID* de valor 2 es reservado para canales de tipo sin conexión. Para canales de tipo orientado a conexión, se asigna dinámicamente un *CID* único de valor superior a 64 para identificar la conexión (*CID* de 3 a 63 están reservados).

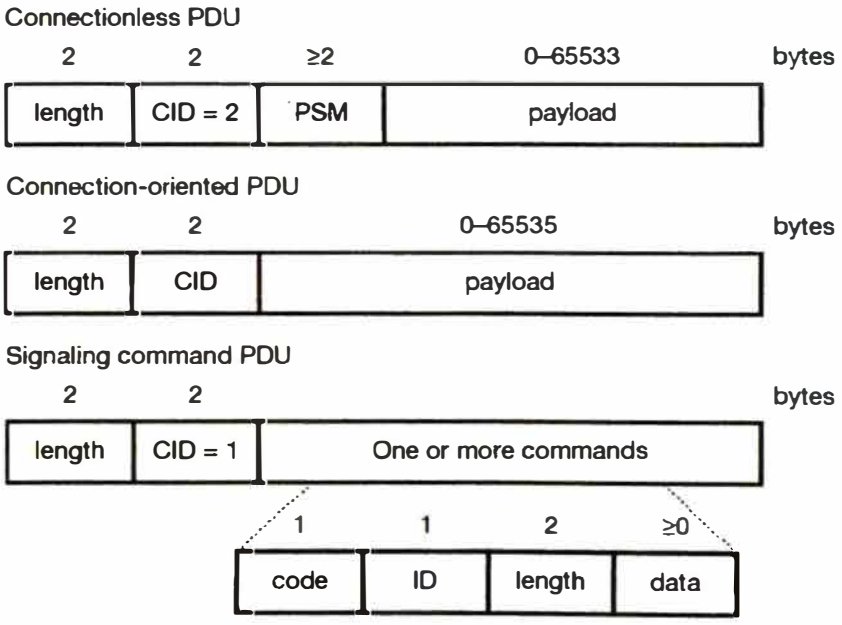


Figura 2.11 Formato de paquetes L2CAP

La figura 2.11 muestra los tres tipos de paquetes que corresponden a los tres tipos de canales lógicos, detallando los *bytes* y valores preestablecidos de las cabeceras del paquete.

f) Protocolo RFCOMM

RFCOMM (*Radio Frequency Communication*) es un protocolo de transporte que opera sobre L2CAP y se usa para emular los puertos serie RS-232. Es este protocolo que posibilita aplicaciones como la conexión de impresoras y escáneres vía *Bluetooth*.

RFCOMM es un protocolo que soporta hasta nueve puertos serie sobre un solo canal físico y permite hasta 60 conexiones simultáneas (canales RFCOMM) entre dos dispositivos *Bluetooth*. Sin embargo cabe aclarar que solo debe existir una sesión RFCOMM entre dos dispositivos que es identificada por las direcciones BD_ADDR (*Bluetooth Address*) de ambos dispositivos.

Finalmente, cabe mencionar que RFCOMM es un protocolo que depende de la transmisión confiable de las capas anteriores por lo que cuando se agota el tiempo de espera de una respuesta del extremo remoto, se cierra la sesión.

g) Protocolo de Descubrimiento de Servicios (*SDP*)

Los dispositivos *Bluetooth* deben ser capaces de trabajar conjuntamente en ambientes desconocidos, por lo que es esencial saber que dispositivos y que servicios están disponibles en su entorno. Para ello, se definió el *SDP* (*Service Discovery Protocol*). Este protocolo permite a una aplicación cliente obtener información sobre servidores *SDP* disponibles en otros dispositivos *Bluetooth* cercanos. Después de haber localizado los servicios disponibles en un dispositivo, el usuario puede elegir aquel de ellos que resulte más apropiado para el tipo de comunicación que desea establecer. Toda la información que un servidor *SDP* tiene sobre un servicio está contenida en un registro de servicio, el cual consiste en una lista de atributos identificados por un registro de 32 bits. *SDP* no informa a los clientes de algún servicio añadido o removido.

2.5.3 Topología de la red *Bluetooth*

Los dispositivos con *Bluetooth* no necesitan equipos adicionales como *switches*, *routers* para establecer comunicación entre ellos, a diferencia de las redes *LAN* o *WLAN*. La red que forman estos dispositivos se denomina una red de área personal (*PAN*). Cuando un dispositivo está en el rango de cobertura de otro, puede establecer una conexión de hasta máximo 8 dispositivos. Esta red que se forma se llama *Piconet*, mientras que varias *Piconets* pueden establecer conexión formando una red llamada *Scatternet* las cuales se explican a continuación.

- ***Piconet***: Una *Piconet* es una colección de dispositivos *Bluetooth* en donde existe solo un dispositivo maestro y el resto son esclavos, los cuales se encuentran sincronizados con la misma secuencia de saltos. El dispositivo maestro es quien define el patrón de salto en la *Piconet* mientras que los esclavos tienen que sincronizar a ese patrón. Existen otros dos estados denominados estacionados (*Parked*) los cuales no participan activamente en la red pero son conocidos y pueden ser reactivados en milisegundos, y en espera (*Standby*) quienes no participan en absoluto. En una *Piconet*, se asigna una dirección para cada dispositivo que participa directa o indirectamente en la red. Dado que existe un máximo de 8 dispositivos en la red, cada uno posee una dirección llamada *AMA* (*Active Member Address*) de 3 *bits*, mientras que todos los dispositivos estacionados poseen una dirección de 8 bits denominada *PMA* (*Parked Member Address*), los dispositivos en espera no poseen dirección. La figura 2.12 muestra la estructura de la red *Piconet*.

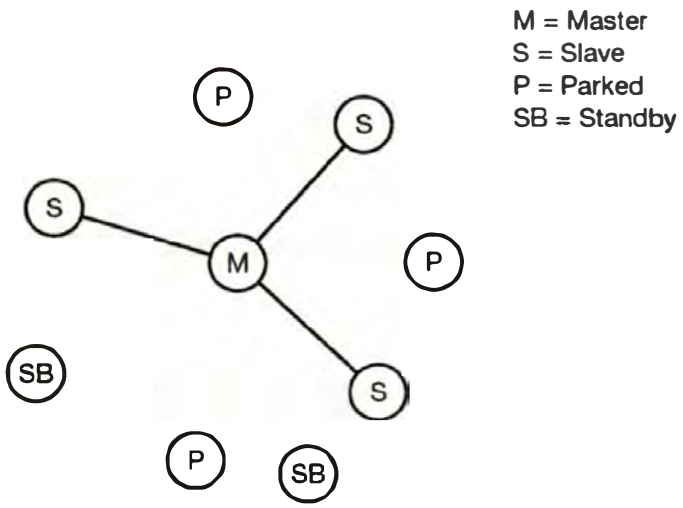


Figura 2.12 Estructura de una red *Piconet*

- ***Scatternet***: Debido a que en una misma red Piconet, todos los usuarios tienen la misma secuencia de salto y comparten el mismo canal de 1 MHz, a mayor cantidad de usuarios que se unan a dicha red, el rendimiento por usuario decae rápidamente ya que una sola red *Piconet* ofrece un poco menos de 1 Mb/s. Entonces tener una sola red dentro de 79 canales de 1 MHz no es muy eficiente, por ello se llegó a la idea de formar grupos de *Piconets* denominado *Scatternet*. *Scatternet* es usado para optimizar el uso del espectro disponible haciendo que todas las unidades compartan dentro de la misma red *Scatternet* el mismo rango de frecuencia pero cada red *Piconet* use

diferente secuencia de saltos determinado por el dispositivo maestro. De esta forma si una unidad, sea maestro o esclavo, desea participar en más de una *Piconet*, debe sincronizar la secuencia de saltos con el maestro de la *Piconet* en la cual desea integrarse y abandonando de manera momentánea la *Piconet* anterior. De esta forma la comunicación entre diferentes *Piconets* se logra mediante los dispositivos saltando continuamente entre las redes *Piconet*.

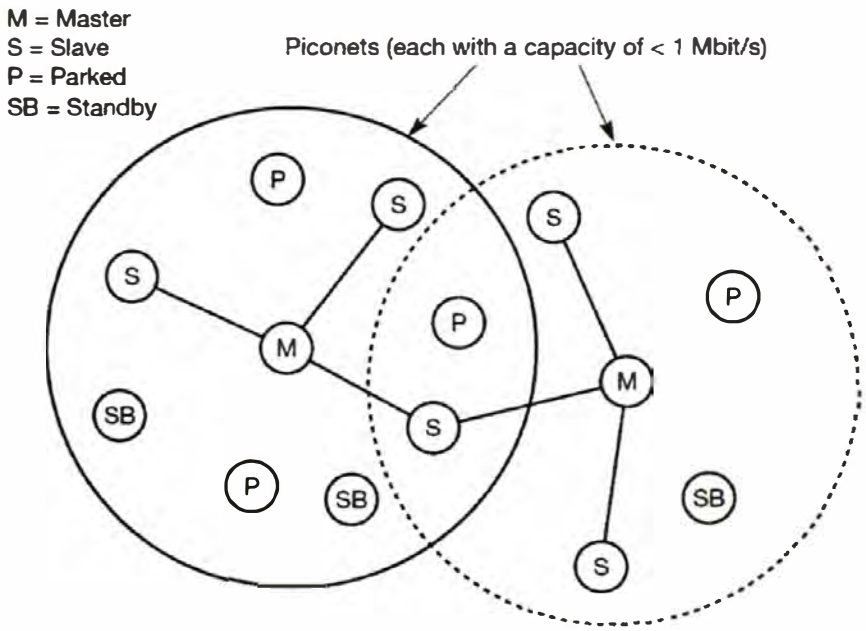


Figura 2.13 Estructura de una red *Scatternet*

2.6 Circuito *dimmer*

Los *dimmers* son dispositivos electrónicos especializados capaces de regular la intensidad de fuentes eléctricas de luz. Los *dimmers* fueron inventados en 1890, por Glanville Woods para evitar los incendios en los teatros debido a la peligrosidad de los métodos utilizados para controlar la intensidad de las lámparas, causaban incendios frecuentemente [15]. Woods desarrolló un método económico y efectivo para regular la intensidad de las luces y así se creó la primera versión del *dimmer* moderno; el cual fue de resistencia variable.

Los primeros *dimmers* resistivos tenían un principio de funcionamiento sencillo, constaba solo de una resistencia variable que se conectaba en serie con la carga haciendo variar la corriente. De esta forma se regulaba la intensidad de las lámparas en los escenarios. Sin embargo existían varios inconvenientes en este tipo de *dimmers* [16]:

- Ocupaban una gran cantidad de espacio en los teatros.

- Disipaban grandes cantidades de energía en forma de calor en la carga la cual era alrededor del 30%.
- Eran fijos para una carga específica, es decir que un *dimmer* de 1000W solo era adecuado para una lámpara de 1000W.

Más adelante se desarrollaron los *dimmers* de reactancia variable para evitar grandes pérdidas de energía pero esto condujo a un alto costo en su construcción mecánica haciendo inaccesibles para muchas compañías de teatros. Posteriormente se diseñaron los *dimmers* con autotransformadores los cuales tienen una salida sinusoidal, evitando así la introducción de armónicos; sin embargo eran grandes, pesados y caros. En la figura 2.14 se muestran ejemplos de los *dimmers* resistivos y de autotransformadores.

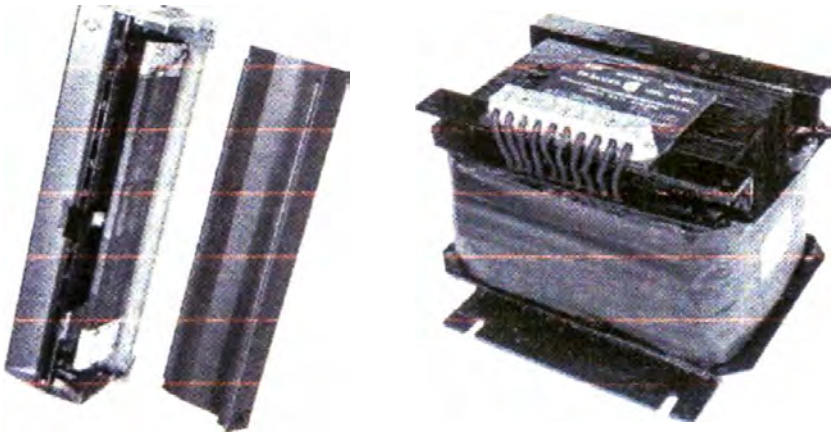


Figura 2.14 *Dimmer* resistivo y de autotransformador

Gracias a la invención del tiristor en los años 50, los nuevos *dimmers* redujeron su tamaño, precio y aumentaron su eficiencia. Los tiristores se usaron en el control de iluminación en la primera parte de la década de los 60's y durante 40 años han formado la base de control de iluminación profesional, ya que son más robustos y tienen la capacidad de soportar altas corrientes repentinas causadas por los fallos en el filamento de las lámparas de tungsteno. Los *dimmers* profesionales están construidos usualmente de manera modular y diseñados para poder resistir diferentes perturbaciones como las producidas por variaciones en la frecuencia de la fuente [17].

Existen solo dos técnicas para limitar el flujo de corriente en la luminaria bajo las cuales operan los sistemas *dimmers*:

- Variación de voltaje

- Variación en el intervalo de tiempo, en el cual la corriente fluye durante cada ciclo de la corriente alterna

En la siguiente sección se explica la segunda técnica por ser la que se aplica en el presente trabajo.

2.6.1 Diseño de *dimmer* con tiristores

Un tiristor es uno de los elementos más importantes de los dispositivos semiconductores y se utiliza de forma extensa en los circuitos electrónicos de potencia. Éste dispositivo es disparado bajo ciertas condiciones pasando de un estado de alta impedancia a uno de baja, estado que se mantiene mientras la corriente y la tensión sean superiores a un valor mínimo denominado niveles de mantenimiento. Por ello, todos los circuitos *dimmer* diseñados con tiristores requieren que el dispositivo se dispare en algún punto predeterminado después que la señal sinusoidal cruza por cero. Esta técnica es conocida como control por ángulo de fase, la cual consiste en controlar el tiempo de disparo o de conducción del tiristor para regular la corriente que se entrega a una carga y de esta manera, controlar la potencia que consume [18]. La figura 2.15 muestra el esquema más básico de un circuito *dimmer* diseñado con tiristores.

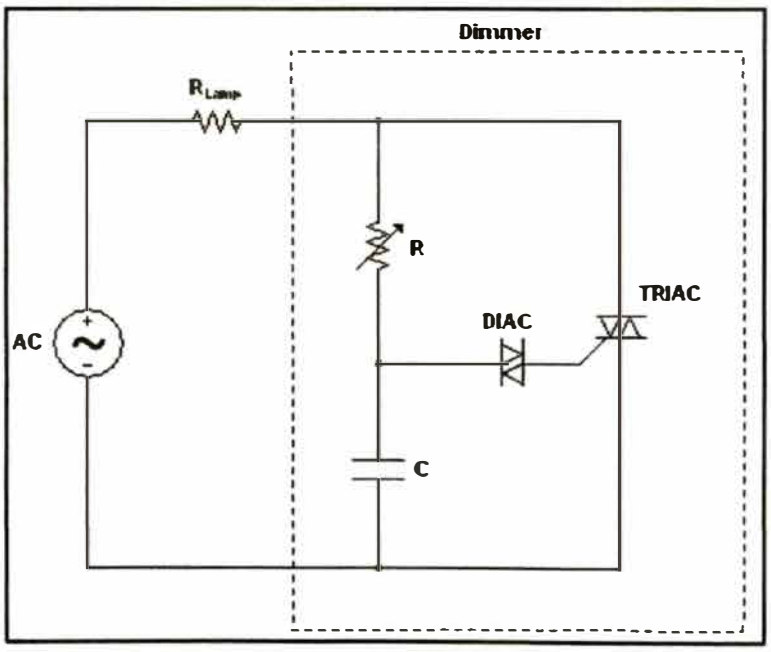


Figura 2.15 Circuito *dimmer* básico

Para desarrollar el esquema, es necesario entender el comportamiento de ciertos componentes electrónicos [19, 20].

a) TRIAC (Triode for Alternating Current)

Un TRIAC es una forma de tiristor que permite que ambos semiciclos de la corriente alterna fluyan a través de la carga. El TRIAC se dispara cuando una señal de baja energía se aplica en su terminal G (*Gate*). Dado que el TRIAC es un dispositivo bidireccional, no es posible identificar sus terminales como ánodo y cátodo, sus terminales se denominan Terminal Principal 1 (MT1) y Terminal Principal 2 (MT2). El electrodo de control se denomina puerta, G. Cuando se alimenta con la señal eléctrica, el semiciclo positivo de la señal de AC pasará por el TRIAC siempre que G sea activo, de esta forma, la corriente circulará de arriba hacia abajo. Mientras que en el semiciclo negativo pasará por el TRIAC siempre y cuando exista una señal de disparo en la entrada G, de esta manera la corriente circulará de abajo hacia arriba. Por ello el TRIAC es especialmente utilizado para controlar el flujo de corriente promedio a una carga, con la particularidad de que conduce en ambos sentidos y puede ser bloqueado por inversión de la tensión o al disminuir la corriente por debajo del valor de mantenimiento. La figura 2.16 muestra la construcción básica y el símbolo de este dispositivo.

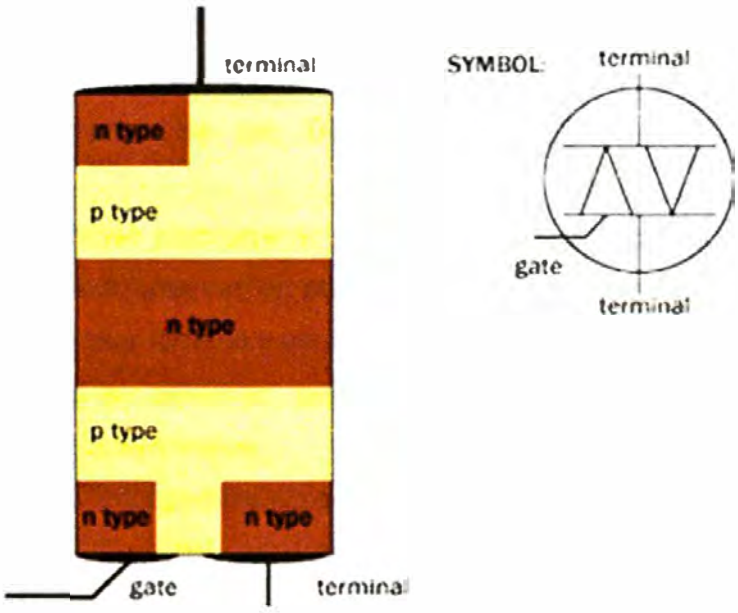


Figura 2.16 Construcción y símbolo del TRIAC

La figura 2.17 muestra la curva característica voltaje VS corriente del TRIAC. Como se observa, el TRIAC puede dispararse en el primer o tercer cuadrante, es decir el procedimiento normal para hacer entrar en conducción a un TRIAC es a través de un pulso de disparo de puerta (positivo o negativo). Para que este dispositivo deje de conducir hay que hacer bajar la corriente por debajo del valor de corriente de mantenimiento del TRIAC.

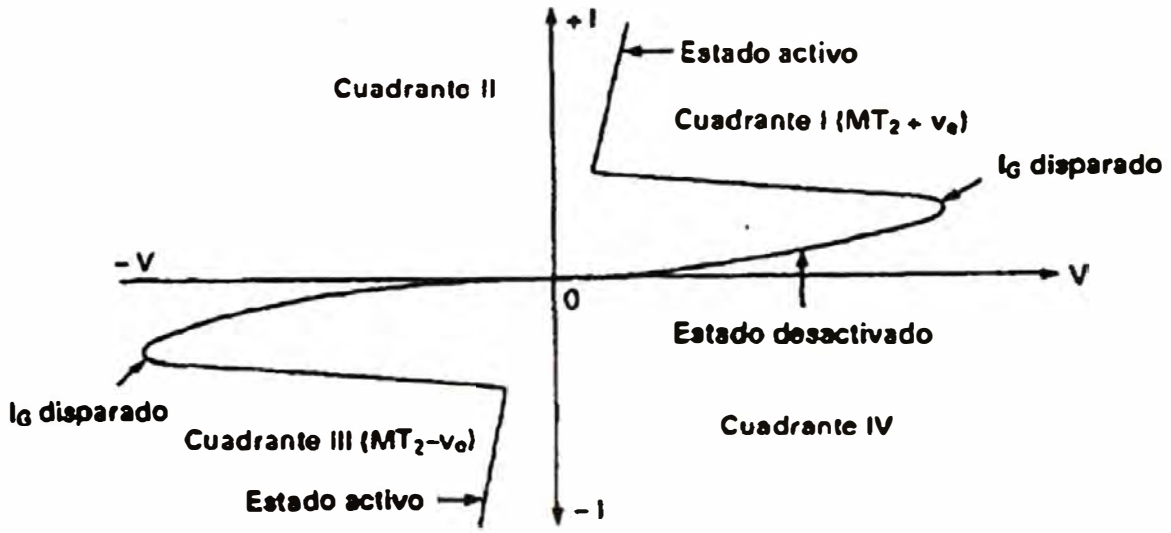


Figura 2.17 Características Voltaje VS Corriente del TRIAC [19]

El uso de estos dispositivos tiene ventajas y desventajas, aunque las ventajas de su utilización exceden a las desventajas. Entre ellas se encuentran [20]:

- Presenta una larga vida útil, pues no sufren deterioros de contactos en cada apertura, por chispas, como si sufren los relés, aun con protección.
- Las corrientes de disparo de los TRIACs son pequeñas lo cual facilita su accionamiento.
- Estas señales no necesitan permanecer en todo el ciclo de trabajo, pues un TRIAC solo necesita ser disparado una vez en cada medio ciclo para conducir durante todo el semiciclo restante. Son muy rápidos y se pueden utilizar en controles por fase.
- Una desventaja es que se calientan, pues penalizan el paso de corriente con una caída de voltaje entre sus terminales.
- No soportan demasiado los cambios de tensión. La variación de tensión máxima que puede soportar un TRIAC es del orden de 100 V/ms, por lo que su uso se restringe prácticamente a aplicaciones de frecuencia de la transición de energía eléctrica (50 ó 60 Hz).

Los TRIACs presentan los siguientes parámetros para tomar en cuenta:

- **I_H (Corriente de mantenimiento):** Es el mínimo valor de corriente necesario para mantener la conducción. Cuando la corriente cae por debajo de IH, el TRIAC cesa de conducir y regresa al estado de bloqueo.

- $I_{T(RMS)}$ **(Corriente en estado de conducción):** Es el máximo valor de corriente eficaz en estado de conducción que puede ser aplicado al dispositivo a través de los 2 terminales del TRIAC.
- I_{GM} **(Corriente máxima de compuerta):** Es la máxima corriente de compuerta pico que puede ser aplicado de un modo seguro para que el dispositivo empiece a conducir.
- V_{DRM} **(Tensión de pico repetitivo en estado de bloqueo):** Es el máximo valor de tensión admitido de tensión inversa, sin que el TRIAC se dañe.
- t_{ON} **(Tiempo de encendido):** Es el tiempo que comprende la permanencia y aumento de la corriente inicial de compuerta hasta que circule la corriente anódica nominal.
- P_{GM} **(Potencia pico de disipación de compuerta):** Es la disipación instantánea máxima permitida en la compuerta.
- **dV/dt (Velocidad critica de crecimiento de tensión en el estado de bloqueo):** Este parámetro indica el ritmo de crecimiento máximo permitido de la tensión en el ánodo antes de que el triac pase al estado de conducción. Se da a una temperatura de 100C y se mide en V/m s.

b) Optoacoplador

Las aplicaciones más comunes de control de potencia, además de requerir el circuito o la etapa de potencia, también requiere la etapa de control que regularmente emplea circuitos digitales con elementos como microprocesadores que consume voltaje y corriente muy bajos. Para evitar que el circuito digital de control sea dañado por la red de alimentación es recomendable aislar ambos sistemas. Las técnicas de aislamiento están basadas en transformadores u optoacopladores. Para esta sección se explicará el segundo dispositivo.

Un optoacoplador también llamado optoaislador, es un dispositivo de aislamiento galvánico que consta en el interior de un fotoemisor y un fotorreceptor, esto quiere decir que la comunicación entre ambos elementos es óptica. De esta forma, existe un aislamiento eléctrico entre los circuitos de entrada y salida. El optoacoplador es unidireccional, es decir que la señal va en un único sentido a diferencia de un transformador que es bidireccional. Además, presenta mayores prestaciones desde el punto de vista de coste, volumen y fiabilidad. La mayoría de los optoacopladores no tiene capacidad de conducir grandes corrientes y por ello son utilizados como circuitos de disparo de TRIACs de mayor potencia.

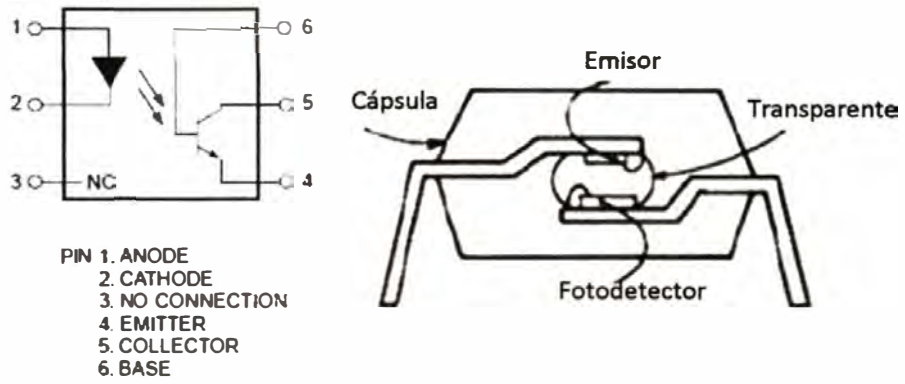


Figura 2.18 Diagramas de un optoacoplador

Existen diferentes tipos de optoacopladores, los cuales se distinguen por la diferencia en la etapa de salida. La figura 2.19 muestra distintos tipos de optoacopladores.

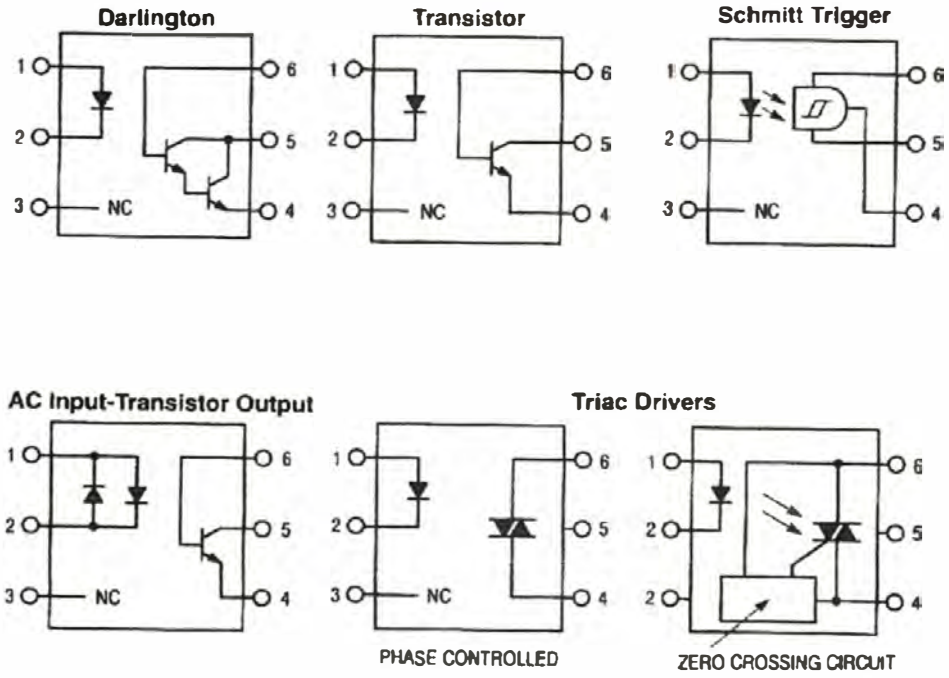


Figura 2.19 Tipos de optoacopladores

Los sistemas *dimmers* basados en TRIAC parten del principio de funcionamiento de estos dispositivos. Estos *dimmers* utilizan la modulación de fase para cortar porciones de la señal eléctrica en AC, por ende reduciendo el voltaje eficaz aplicado a la carga. La figura 2.19 muestra la etapa de potencia con una carga de un foco incandescente a 220 Vac sin detallar el circuito de control necesario para el disparo del TRIAC.

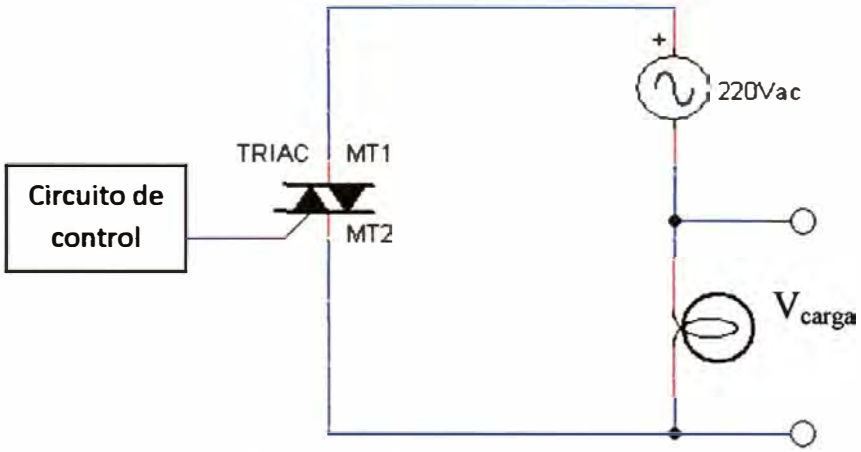


Figura 2.20: Circuito *dimmer*

La figura 2.21 muestra la comparación de la señal recibida en el TRIAC y la carga para dos ángulos de disparo distintos de 30° y 120°.

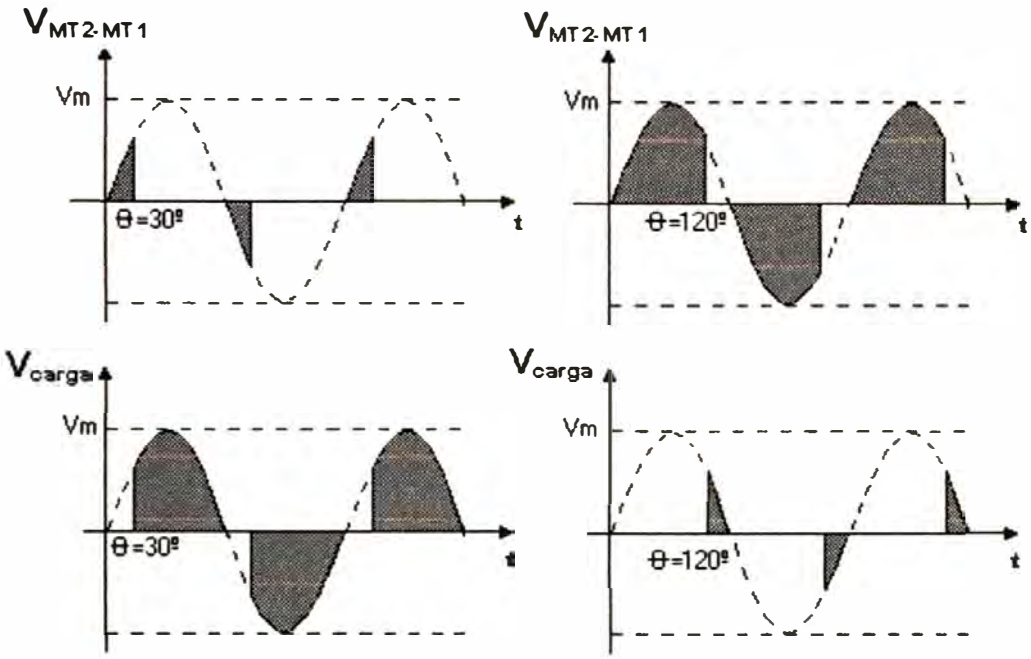


Figura 2.21 Voltajes del TRIAC y la carga para Θ=30° y 120°

Para implementar un circuito de control digital que realice el disparo del TRIAC, es necesario realizar un aislamiento eléctrico del circuito de potencia con el circuito digital. Para ello se hace uso de los optoacopladores vistos previamente. Además es necesario realizar el disparo con el mismo ángulo para cada semiciclo de la onda. Es por ello que se debe detectar el momento que la señal eléctrica cruce los 0 V para que luego de un tiempo determinado (dependiendo del ángulo de disparo) se dispare el TRIAC para lograr la variación de intensidad luminosa de la bombilla. Los circuitos de control y detector de cruce por cero serán desarrollados en el siguiente capítulo.

CAPÍTULO III

DISEÑO E IMPLEMENTACIÓN

3.1 Introducción

El presente capítulo describe de manera detallada el diseño del sistema completo por etapas. En primer lugar se diseñará la etapa de la conectividad entre el Arduino y *Android* mediante *Bluetooth*, para ello se programará la transmisión en el dispositivo *Android* al módulo *Bluetooth*, luego la recepción a un puerto serial del Arduino y la lógica para enviar los pulsos a la siguiente etapa del sistema, es decir la etapa de potencia. Finalmente, se desarrollará la última etapa, diseñando los valores y tipos de dispositivos electrónicos a usar para el control de la luminaria.

3.2 Diseño general

El sistema implementado consiste en un dispositivo móvil con sistema operativo *Android,* dicho dispositivo tiene la aplicación que establece la conexión al módulo de *Bluetooth* JY-MCU, el cual está conectado a la tarjeta Arduino que debe transmitir los datos que recibe del *Android* de forma serial. Una vez recibida las instrucciones, la tarjeta procesa los datos para encender, apagar y/o regular la intensidad de las bombillas, dependiendo de la instrucción. Un factor importante para el control de intensidad es que el Arduino además de recibir los datos, también debe realizar una interrupción cada vez que la red eléctrica pase por cero, sin esto la luz iluminaría de forma intermitente. Como última etapa del sistema, se encuentran los elementos electrónicos que manejarán los pulsos recibidos de los puertos de salida digital del Arduino. Para el caso de encendido y apagado, se utiliza un relé de 5V, mientras que para el caso del control de intensidad, se requieren del circuito detector de cruce por cero y el circuito con un TRIAC que será disparado por el Arduino.

La figura 3.1 muestra de manera simple y gráfica la implementación del sistema.

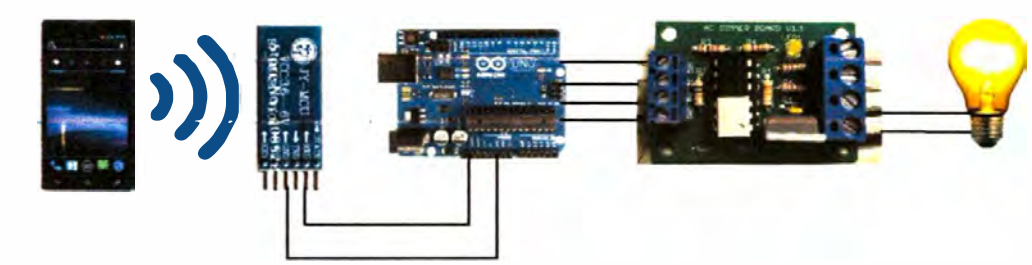


Figura 3.1 Esquema general del sistema

3.3 Diseño específico

3.3.1 Diseño de la aplicación móvil

Para diseñar la aplicación móvil, se dividirá en tres partes principales. El primero es el archivo xml donde se define la interfaz principal de la aplicación. En segundo lugar, el archivo xml llamado *AndroidManifest* donde se define la configuración de la aplicación. Finalmente, el archivo en *Java* donde estará definida toda la lógica de la aplicación.

a) Interfaz de la aplicación

En el directorio *res/layout* se crea el archivo xml donde se define los elementos gráficos a inicializar. Para el caso de una bombilla de encedido y apagado, se necesita un botón (<*Button*>) y una imagen (<*ImageView*>) de una bombilla encendida o apagada, dependiendo el estado actual del foco, que se encuentran en el directorio *res/drawable*. Para el caso de una bombilla de intensidad variable, se usa igualmente una imagen pero en lugar de un botón, se coloca una barra variable (<*SeekBar*>) con valor mínimo 0 y valor máximo de 125. La interfaz de la aplicación se muestra en la figura 3.2:

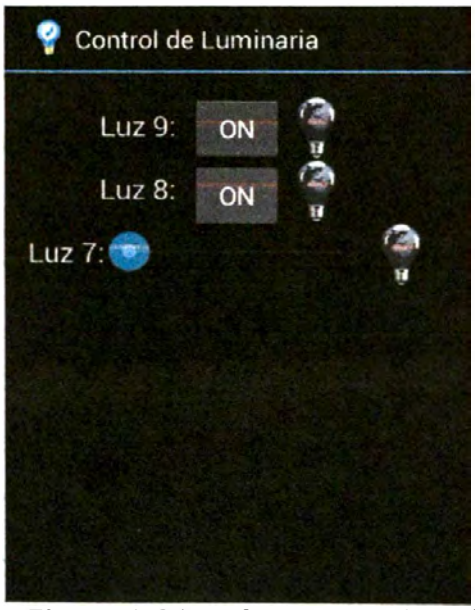


Figura 3.2 Interfaz de la aplicación

b) Archivo de configuración de la aplicación

Esta aplicación requiere permisos para utilizar *Bluetooth* para poderlo alinear con el módulo JY-MCU. Se definen los siguientes permisos para poder conectarse y alinearse con otro dispositivo *Bluetooth* de la siguiente forma:

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

En este archivo también se define el valor mínimo de la versión de *Android* que la aplicación puede soportar. Para este caso, la versión mínima aceptable es la 8, la cual es *Android* 2.2 (*Froyo*). De esta forma, se logra cubrir la gran mayoría de celulares existentes y todas las versiones de *tablets*.

```
<uses-sdk  android:minSdkVersion="8" />
```

c) Lógica principal de la aplicación

La lógica de la aplicación estará en una clase que herede la clase Activity. La siguiente figura muestra el flujo que sigue al momento de iniciar la aplicación, es decir antes que el usuario interactúe con la aplicación para poder controlar la luminaria.

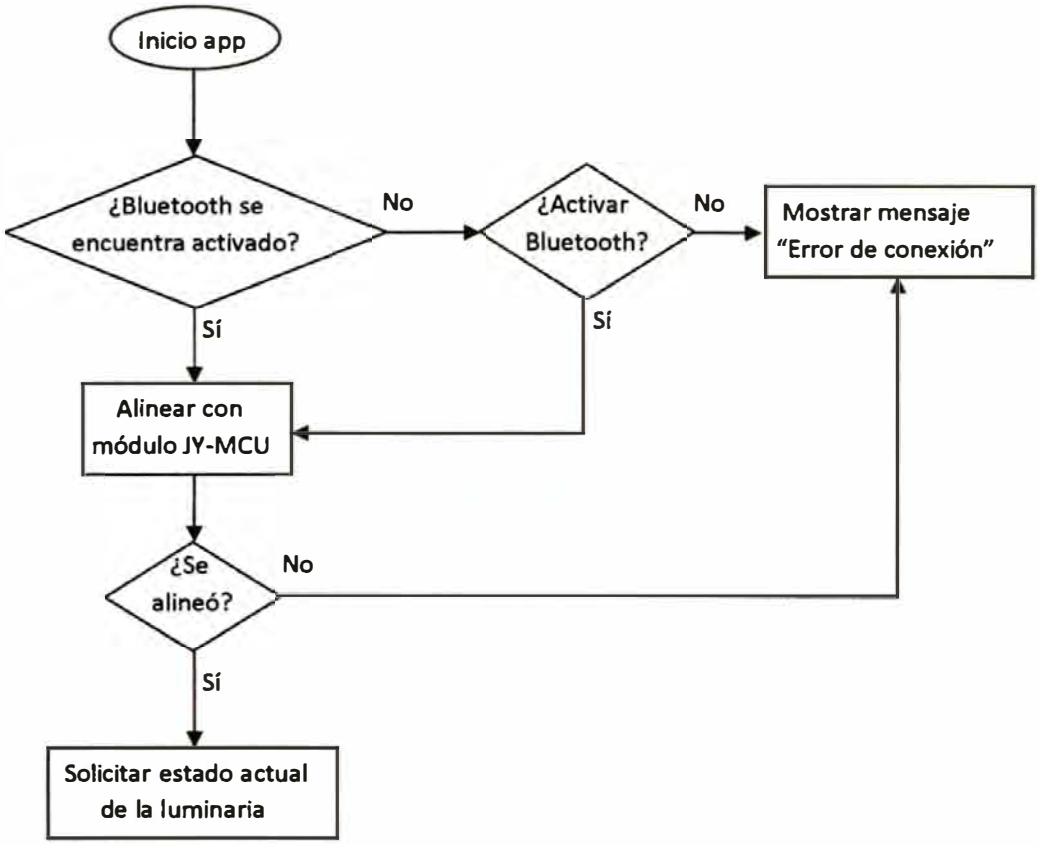


Figura 3.3 Flujo al iniciar la aplicación

Para explicar con más detalle el programa en *Java*, se mencionará el código más importante por partes empezando por los objetos que se declaran además de variables estáticas que se requerirán en toda la clase, siguiendo con los métodos que provienen de la clase *Activity* (*onCreate*, *onResume* y *onPause*) y finalmente con los métodos generados en la clase. Se tomará como referencia 2 luces *on/off* (9 y 8) y una luz de intensidad variable (7). Se dejará de lado el código referente a la estética o modificación de la interfaz de usuario para explayarse en lo funcional.

- **Variables de la clase principal**

Se declara los objetos de las clases externas para el funcionamiento de *Bluetooth*. El primero proviene de la clase *BluetoothAdapter* el cual es el encargado de verificar si existe o está apagado el adaptador *Bluetooth* del dispositivo móvil. De igual forma se declara un objeto de la clase *BluetoothSocket* la cual realizará la conexión con el módulo JY-MCU para luego, una vez emparejados los módulos, iniciar la transmisión y recepción de datos. Dado que el módulo es único, no es necesario ver la lista de dispositivos con *Bluetooth* para luego emparejarlos. Por esa razón se realizará una conexión directa solo con la MAC de dicho módulo. Para descubrir el valor de su dirección MAC, se energizó el módulo *Bluetooth* y luego se ejecutó el siguiente comando en Linux para ver los módulos que se encontraban disponibles:

```
root@localhost:~# hcitool scan
Scanning ...
20:13:06:13:45:69  BlueArd                    //MAC del módulo JY-MCU
```

Además para crear el *socket* RFCOMM, es necesario también el identificador único universal (*UUID*) el cual para el servicio de *Bluetooth* como Puerto Serial es el valor de 16 bits 0x1101 [21]. Sin embargo la clase *UUID* de *Java*, dicho valor es de 128 bits. De acuerdo a las especificaciones en la página oficial de *Android*, se debe asignar el valor de *UUID* "00001101-0000-1000-8000-00805F9B34FB" [22]. Las variables mencionadas se declaran de la siguiente manera:

```
private BluetoothAdapter btAdapter;
private BluetoothSocket btSocket;
private static final String MAC = "20:13:06:13:45:69";
private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-8000-
00805F9B34FB");
```

- ***OnCreate***

En este método se inicializan los objetos de la interfaz de usuario (*UI*), es decir los textos, botones, barras e imágenes. También se inician los listeners que son los códigos

encargados de realizar una acción luego de algún evento externo como lo es la interacción del usuario con la interfaz.

Se inicializan los elementos de la interfaz de tipo *Button*, *SeekBar* e *Imageview* con su respectivo índice (9, 8, 7). La inicialización es de la siguiente forma dependiendo el tipo de foco a controlar.

```
// Para el caso de un foco on/off
Button btn9 = (Button) findViewById(R.id.btn9);
ImageView iv9 = (ImageView) findViewById(R.id.iview9);

// Para el caso de un foco de intensidad variable
SeekBar sb7 = (SeekBar) findViewById(R.id.sb7);
ImageView iv7 = (ImageView) findViewById(R.id.iview7);
```

Ahora es necesario crear los listeners para los botones y barras. Para el ejemplo de btn9 que es de tipo *on/off*, al tocar el botón, dependiendo de su texto, encenderá o apagará la luz 9. Análogamente para la barra7, se enviará el número 7 y el valor de la intensidad entre 0 y 125.

```
// Para el caso de un foco on/off
btn9.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
     if (btn9.getText().equals("ON")) {
       cbt.enviarMsj(transBytes(9, 1));
     } else if(btn9.getText().equals("OFF")) {
       cbt.enviarMsj(transBytes(9, 0));
     }
   }
});
```

```
// Para el caso de un foco de intensidad variable
sb7.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {
  @Override
  public void onProgressChanged(SeekBar seekBar,int progress,boolean fromUser){
    cbt.enviarMsj(transBytes(7, progress));
  }
});
```

La función enviarMsj de la clase creada ConexionBT, que se desarrollará más adelante, tiene como entrada un arreglo de bytes por lo que es necesaria una función que transforme los parámetros a bytes. La función transBytes devolverá 3 bytes, el primero es el valor del índice de la luz a controlar, el segundo el valor de intensidad y el último es el *byte* 127 (0x7F) que indica que el mensaje ha terminado.

```
private byte[] transBytes(String s, int i){
  ByteBuffer bb = ByteBuffer.allocate(3);
  int ns = Integer.parseInt(s);
  bb.put((byte)ns);
  bb.put((byte)i);
  bb.put((byte)127);
  return bb.array();
}
```

- ***OnResume***

Este método de la clase *Activity* realizará el emparejamiento con el módulo *Bluetooth* conectado al Arduino para el posterior envío de sentencias. Como se estableció previamente, el objeto btAdapter se conectará directamente mediante la dirección MAC del módulo y asignará a una variable de la clase *BluetoothDevice* de la siguiente forma:

```
BluetoothDevice device = btAdapter.getRemoteDevice(MAC);
```

Luego de tener el enlace con el módulo, se creará una socket de comunicación entre ambos dispositivos con el protocolo RFCOMM siendo necesario el identificador único universal (*UUID*) definido anteriormente. Posteriormente se inicializará el objeto declarado en *onCreate* con el parámetro btSocket y se iniciará el hilo para que transmita y reciba los mensajes. Este es el código necesario para realizar lo indicado:

```
btSocket = device.createRfcommSocketToServiceRecord(MY_UUID);
btSocket.connect();
cbt = new ConexionBT(btSocket);
cbt.start();
cbt.enviarMsj(transBytes());
```

La última sentencia define el envío del mensaje que solicita el estado de todos los focos, para ello se envían los bytes 126 y 127 que está indicado en el método transBytes(). En la tarjeta Arduino se encuentra la lógica que realiza para el manejo de mensajes.

```
private byte[] transBytes(){
     ByteBuffer bb = ByteBuffer.allocate(2);
     bb.put((byte)126);
     bb.put((byte)127);
     return bb.array();
}
```

- ***OnPause***

Como se indicó en el capítulo anterior, esta función se iniciará cuando la aplicación deje de ser visible y por ello es sumamente importante cerrar la conexión del socket creado. De lo contrario, al desear ingresar nuevamente a la aplicación, no permitiría la conexión

ya que existiría una conexión del mismo tipo. La sentencia para cerrar el *socket* es la siguiente:

```
btSocket.close();
```

- **Clase ConexionBT**

Esta clase creada realiza la tarea de realizar el envío de los mensajes al Arduino y también de recibir los mensajes provenientes de la conexión generada. Esta clase tiene a *Thread* como clase padre ya que necesita ejecutarse en su propio hilo y no en el mismo hilo de la interfaz de la aplicación (*UI thread*). Esto es debido a que el hilo de la aplicación no puede mantenerse en espera con procesos largos como conexiones externas (*Bluetooth*, Internet) o temporizadores, de ser así, lanzaría una excepción que terminaría la aplicación [23]. A continuación se explicará las 2 funciones principales de la clase, la primera la que envía el mensaje y la segunda es la que recibe lo que envía Arduino.

Para empezar, se define el constructor de la clase que tiene como entrada el socket de *Bluetooth* inicializado en el método *onResume*. Se declara una variable que será el flujo de datos que envía llamada *mmOutStream* y otra de nombre *mmInStream* que recibe los datos. Dado que en la clase dichas variables deben de ser finales, estos harán referencia a unas variables temporales que son las que reciben o forman el flujo de datos como se muestra a continuación en el código:

```
private final InputStream mmInStream;
private final OutputStream mmOutStream;
public ConexionBT(BluetoothSocket socket) {
  InputStream tmpIn = null;
  OutputStream tmpOut = null;
  tmpIn = socket.getInputStream();
  tmpOut = socket.getOutputStream();
  mmInStream = tmpIn;
  mmOutStream = tmpOut;
}
```

Como se había especificado anteriormente, el envío de datos los realiza la función enviarMsj. Como único parámetro que recibe es un arreglo de *bytes* y es dada de la forma:

```
public void enviarMsj(byte[] message) {
  mmOutStream.write(message);
}
```

Mientras que el envío de datos se define de manera sencilla, la recepción es más compleja. El diagrama de la figura 3.4 explica el flujo para la recepción.

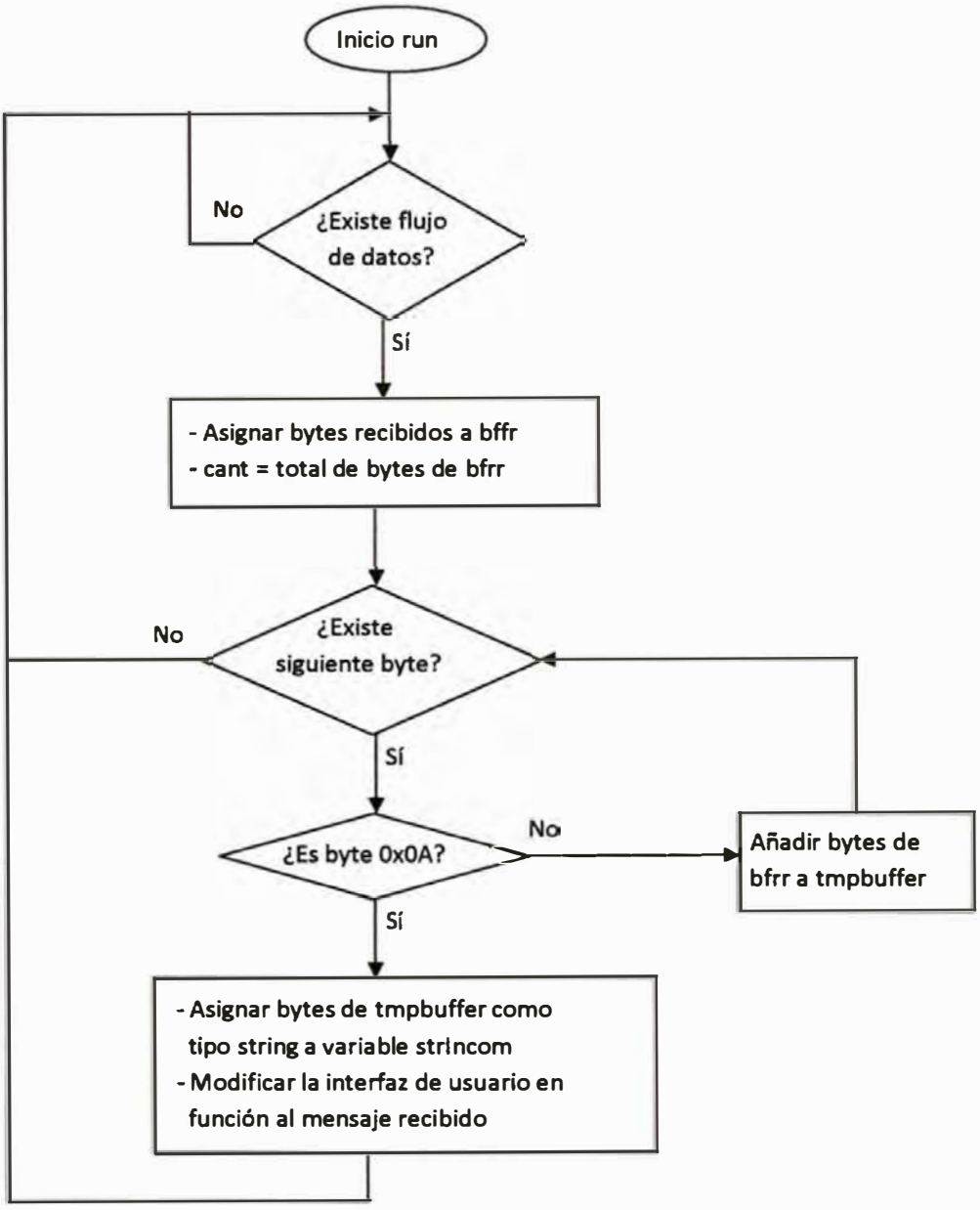


Figura 3.4 Flujo del método que recibe los datos del Arduino

El método *run* que viene de la clase padre, se ejecuta al iniciar el hilo, inicializándose así las variables bffr y tmpbuffer necesarias para almacenar los bytes recibidos.

Inicialmente se tuvo inconvenientes con la recepción de las cadenas de mensajes que enviaba el módulo. Experimentalmente se verificó que toda la cadena que envía el módulo *Bluetooth* no se realiza en un solo flujo, en ocasiones se enviaba el mensaje por

partes, por lo que era necesario crear una variable temporal que almacene los *bytes* hasta que llegue el *byte* 0x0A que indicaba el término del mensaje.

```
public void run() {
  byte[] bffr = new byte[8];
  byte[] tmpbuffer = new byte[8];
  int j=0;
  int cant;
  while (true) {
    cant = mmInStream.read(bffr);
    for(int i=0;i<cant;i++){
      if(bfrr[i]==10){
        String strIncom = new String(tmpbuffer,0,j);
        h.obtainMessage(1, 1, -1, strIncom).sendToTarget();
        j=0;
      } else {
        System.arraycopy(bffr, i, tmpbuffer, j, 1);
        j++;
      }
    }
  }
}
```

Luego de obtener el mensaje correctamente, ya era posible procesarlo para modificar los elementos gráficos de la aplicación mediante los *handlers* que manejan las modificaciones de las imágenes o textos de la aplicación. El código de esta función que se llama mediante el método *obtainMessage*() se muestra a continuación.

```
h = new Handler() {
  public void handleMessage(android.os.Message msg) {
    String[] words = msg.obj.toString().split(",");
    if(words[0]==7) {
      tmmp="usd";
      sb7.setProgress(Integer.valueOf(words[1]));
      if (words[1].equals("0")) {
        iv7.setImageResource(R.drawable.off);
      } else {
        iv7.setImageResource(R.drawable.on);
      }
    } else {
      if(words[1].equals("1")) {
        btn9.setText("OFF");
        iv9.setImageResource(R.drawable.on);
      } else {
        btn9.setText("ON");
        iv9.setImageResource(R.drawable.off);
      }
    }
  };
};
```

Como se observa en el código, el mensaje recibido está separado por comas, siendo el primer valor el número del puerto (9, 8, 7) y el segundo su estado. Para la luminaria de tipo *on/off*, si es 1 su estado, cambiará el texto del botón a *OFF* y cambiará la imagen a la

bombilla encendida. Para el caso de una bombilla *dimmer*, se coloca la imagen de la bombilla apagada para el valor del *SeekBar* de 0 y para el resto se pone la bombilla encendida.

3.3.2 Configuración del Módulo *Bluetooth*

Para entablar la comunicación entre el dispositivo *Android* y la placa Arduino, utilizó el módulo *Bluetooth* JY-MCU el cual es un módulo económico, fácil de usar y tiene un rango aceptable.

La comunicación para este dispositivo de manera serial a través de sus pines Rx y Tx y transparente ya que solo actúa como un puente entre el *Android* y el Arduino. Este dispositivo presenta cuatro pines [ver Anexo A], los cuales irán conectados al Arduino teniendo en cuenta que el pin de transmisión (Tx) debe ir conectado al puerto serial del Arduino de recepción (Rx) y viceversa como se muestra en la figura 3.5.

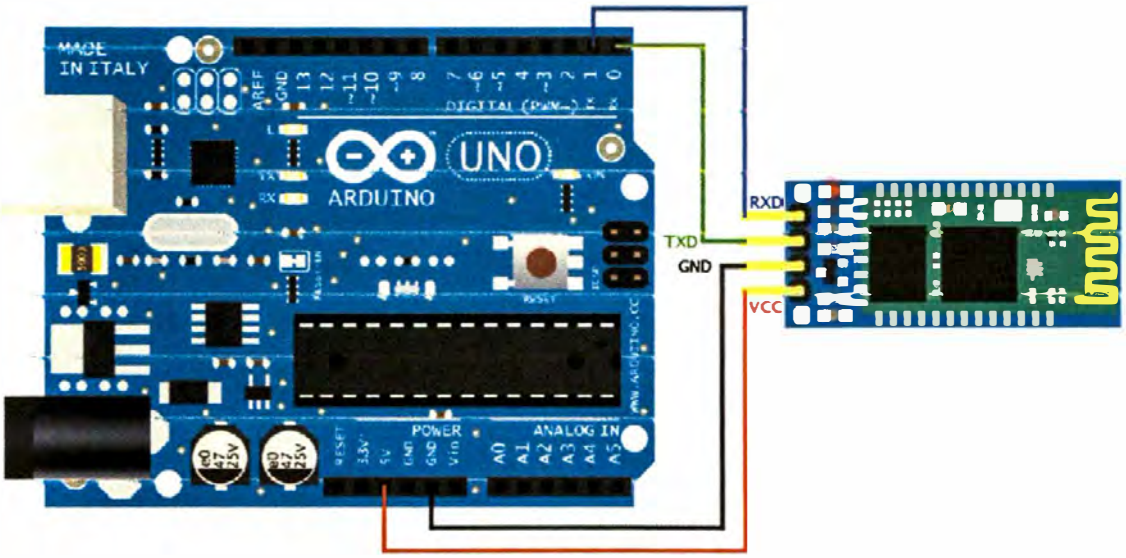


Figura 3.5 Conexión del módulo *Bluetooth* y el Arduino

Luego de conectar los pines correspondientes, es necesario configurar el módulo a los parámetros que se desean. Para modificar sus valores por defecto del dispositivo, se utilizan los comandos AT, los cuales son enviados a través del Monitor Serial de Arduino. En la consola se ejecutan los comandos necesarios como se muestra en la figura 3.6.

Figura 3.6 Interfaz de Monitor Serial de Arduino

Previamente se debe cargar el *sketch* de comunicación serial [24]. En primer lugar, el *sketch* llama a la librería que emula puertos de salida digital en puertos seriales (pines 10 y 11), luego se asignan valores de tasa de transferencia de 9600 baudios a ese puerto serial y al puerto serial en *hardware* (pines 0 y 1) El *sketch* se limita a enviar lo que se escribe en el Monitor Serial y mostrar en el monitor lo que se recibe del módulo. Dicho *sketch* se muestra en las siguientes líneas:

```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(10, 11); // RX, TX

void setup() {
  Serial.begin(9600);
  mySerial.begin(9600);
}
void loop() {
  if (mySerial.available())
    Serial.write(mySerial.read());
  if (Serial.available())
    mySerial.write(Serial.read());
}
```

TABLA N° 3.1 Comandos AT disponibles para el módulo

Comando	Descripción	Opciones	Respuesta
AT+VERSION	Devuelve la versión del módulo		OKlinvorV1.x
AT+BAUDx	Configura la velocidad de trasmisión del modulo según el valor de x	1 >> 1200 2 >> 2400 3 >> 4800 4 >> 9600 (por defecto) 5 >> 19200 6 >> 38400 7 >> 57600 8 >> 115200 9 >> 230400	OKx

AT+NAMEx	Configura el nombre con el que se visualizara el módulo	Cualquier nombre pero soporta como máximo 20 caracteres	OKsetname
AT+PINxxxx	Configura el Pin de Acceso al modulo	Cualquier número de 4 digitos es aceptado. Por defecto el PIN es 1234	OKsetPIN

3.3.3 Diseño de la lógica del Arduino

Para poder definir la lógica que procesará el Arduino, en primer lugar es necesario tener en cuenta cuales serán las entradas y salidas. Respecto a las entradas, existen únicamente lo que recibirá por su puerto serial Rx del módulo *Bluetooth* y los pulsos de 5V que recibirá del circuito detector de cruce por cero. En cuanto a las salidas, el puerto serial Tx enviará el estado de todas las bombillas; esto es de vital importancia para el usuario ya que al cerrar y abrir nuevamente la aplicación móvil, se puede ver en la interfaz los valores de encendido, apagado e intensidad de todas las luces. El programa se dividirá en las siguientes partes:

a) Definición de variables: En primer lugar, se definirá las variables de los puertos conectados a las bombillas como dos arreglo, una para encendido y apagado y otra para los que se regulan. Además se define el número total de focos a controlar:

```
#define numL 2                    // numero de focos on/off
#define numdL 1                   // numero de focos regulados
int LED[numL] = {9,8};            // puertos de focos on/off
int dLED[numdL] = {7,6};          // puertos de focos regulados
int valdLED[numdL] = {0,0};       // valores de intensidad iniciales
```

b) Definición de la función *setup*: En esta función, se inicializa el puerto Serial Rx y Tx de la tarjeta Arduino con una velocidad de 9600 baudios la cual es la misma con la que el módulo *Bluetooth* viene configurada por defecto. Luego de ello, se asignan todos los pines asignados a los focos como salida digital. Finalmente es en esta función donde se asigna la interrupción cuando la señal eléctrica pase por cero.

```
void setup() {
  Serial.begin(9600);
  for(int k=0;k<numL;k++){
    pinMode(LED[k],OUTPUT);
  }
   for(int k=0;k<numdL;k++){
     pinMode(dLED[k],OUTPUT);
  }
  attachInterrupt(0, det_cruce_cero, RISING);
}
```

c) **Definición de la función *loop*:** Esta función de bucle infinito, se utiliza para detectar todos los *bytes* que sean recibidos en el puerto serial Rx de la tarjeta y de acuerdo al mensaje recibido, la tarjeta procede a realizar la tarea asignada de acuerdo a la programación del microcontrolador. La lógica de esta función se muestra en el diagrama de flujo de la figura 3.7.

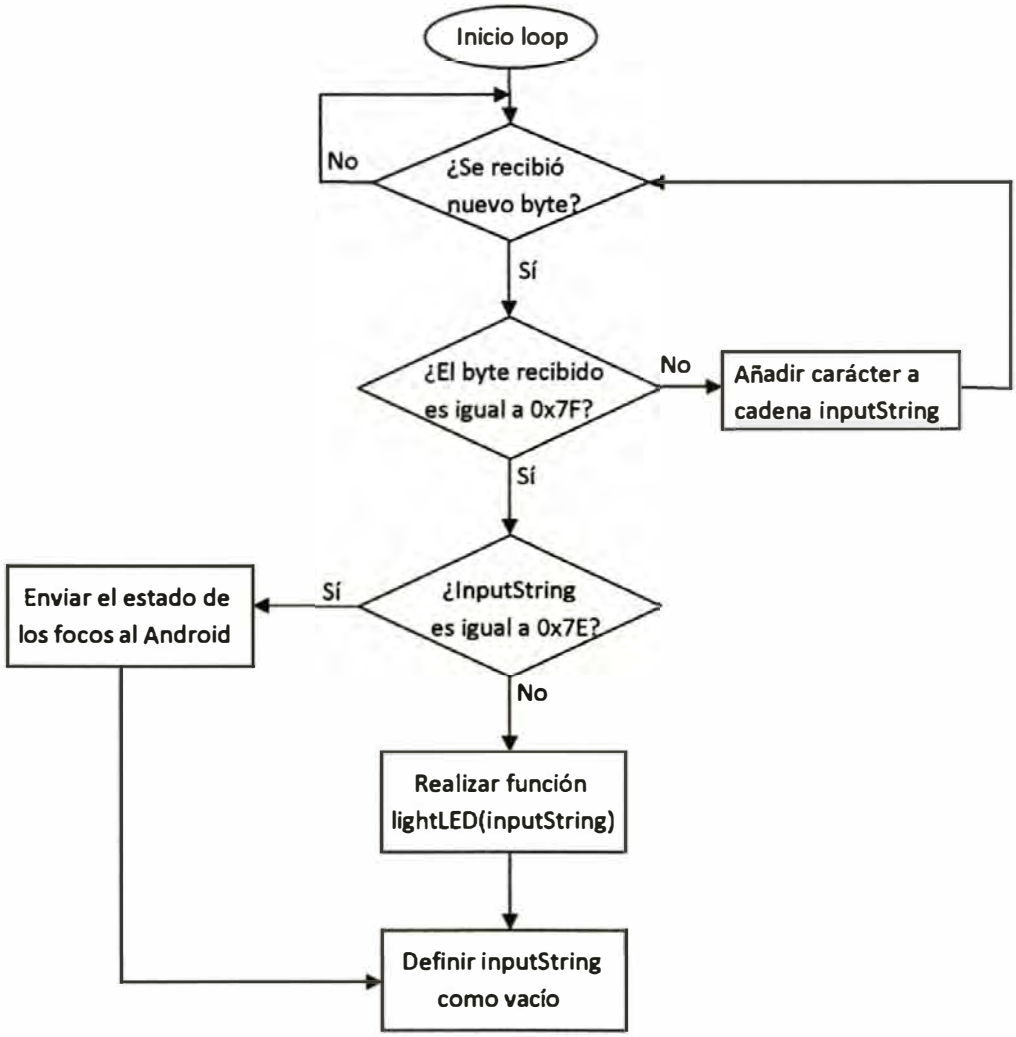


Figura 3.7 Diagrama de flujo de la función *loop*

Como se indica en el diagrama anterior, el *byte* 0x7F (127) será el separador que indicará que el mensaje ha concluido. A partir de ahí, analizará el valor de *inputString*. El *byte* 0x7E indica que la aplicación ha solicitado el estado actual de todos los puertos, el cual será enviado como caracteres imprimibles de tipo *string*. El mensaje constará del valor del puerto y su valor actual separado por una coma como se indicó en el diseño del programa de *Android*.

La variable *inputString* estará conformada por 2 *bytes*, el primer *byte* indica el valor del puerto a controlar. El segundo *byte* estará el valor siendo los valores 0 o 1 para los puertos *on/off*, mientras que para los puertos que se regulan, los valores varían en el rango de 0 a 125 siendo 0 el valor de intensidad más baja y 125 la más alta. La lógica de la función *lightLED* se muestra en el diagrama de flujo de la figura 3.8.

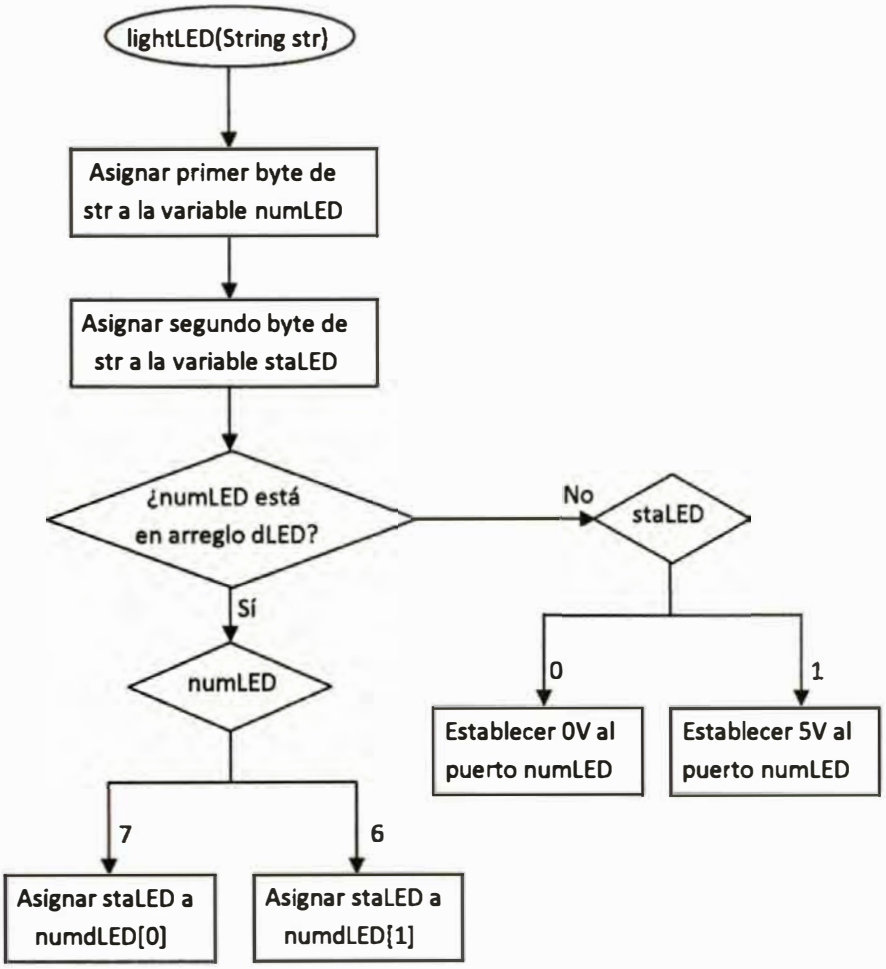


Figura 3.8 Diagrama de flujo de la función *lightLED*

d) Definición de la función de interrupción: Como se indicó en la función setup, existe un comando de interrupción que se define en la función *setup*() y que ejecutará la función det_cruce_cero() cada vez que la señal eléctrica cruce por cero. A continuación se muestra el código de la función de interrupción.

```
void det_cruce_cero() {
  for(int i=134;i>0;i--){
    delayMicroseconds(60);
    if(valdLED[0]==i){
      digitalWrite(dLED[0], HIGH);
      delayMicroseconds(4);
      digitalWrite(dLED[0], LOW);
    }
  }
}
```

El código se ejecuta en un *for* una cantidad determinada de veces desde el valor más alto hasta el valor de 1. Cada vez que recorra el código interno del *for*, habrá un tiempo en espera de 60 microsegundos. Esto define el valor de tiempo que el puerto esperará para realizar el disparo del TRIAC.

Además, en cada recorrido del *for*, verificará el valor de intensidad del foco dado por la variable valdLED[0] y solo cuando se iguale con el índice i, se realizará el disparo. Esto se realizó para hacerlo un sistema escalable con la posibilidad de añadir más bombillas *dimmer* tan solo validando el índice en el que se encuentra con el valor de los otros focos valdLED[1], valdLED[2], etc. Esto se debe a que al ejecutar un retardo con el comando delay, el microprocesador solo se queda esperando sin poder realizar alguna otra tarea. Por ello se divide en varios retardos en lugar de una espera continua de varios microsegundos para disparar solo un TRIAC.

También se debe considerar que el tiempo total que ejecuta el for, no debe superar el tiempo del semiciclo de la señal eléctrica. Dado que la frecuencia de la red eléctrica es de 60 Hz, el tiempo que dura cada medio ciclo es de 8333.33 µs aproximadamente. Entonces el valor máximo del índice i (N) multiplicado por 60 debe ser menor que 8333.33.

$$N \times 60\mu s < 8333.33$$
$$N < 138.888$$

Se elige el valor de N=134 ya que se necesita tiempo disponible del microprocesador para recibir los datos del *Android* y también debido a que la intensidad luminosa para valores de ángulo de disparo cercanos a 180, no es notoria y puede ser dejado de lado esos valores.

La lógica de la función det_cruce_cero se ejecuta en cada momento que el pin 2 del Arduino, detecto el pulso enviado por el optoacoplador. Dicha lógica se muestra en el diagrama de flujo de la figura 3.9.

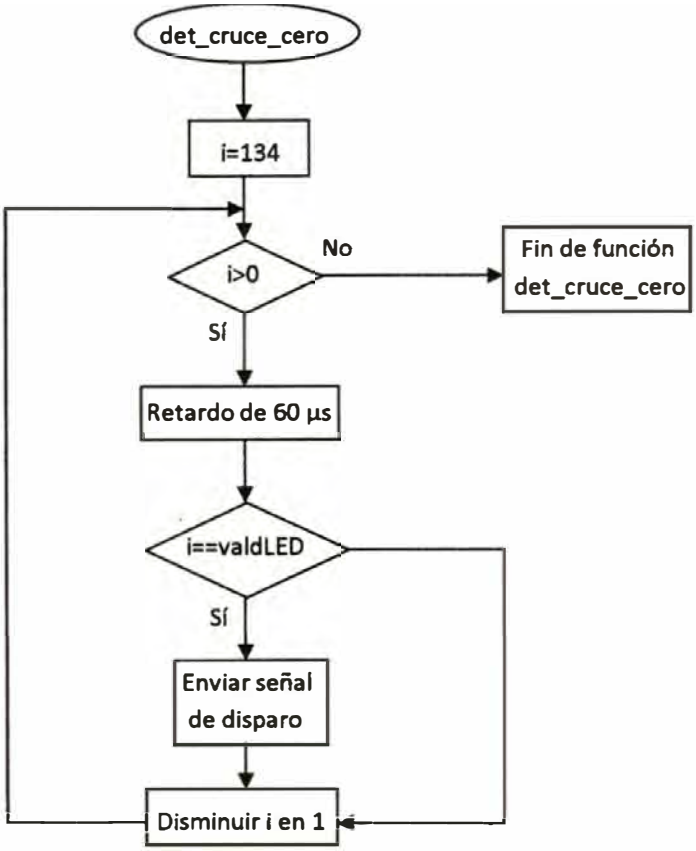


Figura 3.9 Diagrama de flujo de la función det_cruce_cero

3.3.4 Diseño de la etapa de potencia

Luego de tener implementado las etapas anteriores y verificar que los puertos de salida digitales proveen el voltaje requerido al momento de manipular la aplicación en el dispositivo móvil, ya se puede diseñar la última etapa del sistema, teniendo 2 diseños separados, la primera para el simple encendido y apagado de la bombilla, y la segunda para la regulación de la intensidad de la luz.

a) Encendido y apagado

Por defecto, la luminaria se encuentra directamente conectada a la red eléctrica, pasando por ella una corriente del orden de mA, dependiendo de la potencia del foco. Para este trabajo, se probó con un foco de 60W por lo que la corriente está dada por:

$$P = VI \tag{3.1}$$

$$I = \frac{P}{V} \approx \frac{60W}{220V} \approx 272mA$$

Requiriendo simplemente un *switch* que conduzca al recibir del Arduino 5V y dado que un foco incandescente no es una carga inductiva [4], se opta por el uso de un relé de 5V que será activado al enviarse la orden del *Android*, el esquema está dada por la figura 3.10:

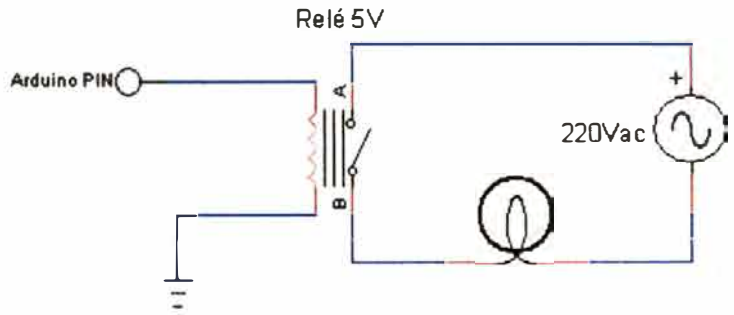


Figura 3.10 Esquema del circuito de encendido y apagado

b) Regulación de intensidad

Para lograr el control de la intensidad de la luz, se requiere dos etapas:

- **Detector de cruce por cero**

Para poder disparar el TRIAC, es necesario hacerlo luego que la señal de la red eléctrica haya pasado por 0V en cada medio ciclo para lograr la onda cortada en todo el periodo que recibirá la bombilla, con ello disminuyendo su intensidad.

Dado que se requiere el momento en que cruza 0V tanto en polaridad positiva como negativa, es decir para ambos semiciclos del periodo, se utilizó el optoacoplador de tipo fototransistor H11AA1, el cual posee en la entrada dos diodos antiparalelos según la hoja de datos (ver Anexo B) para la onda positiva y negativa, de esta forma se detectaran ambos cruces y no conducirá el transistor a la salida. El esquema general de esta etapa se puede apreciar en la Figura 3.11:

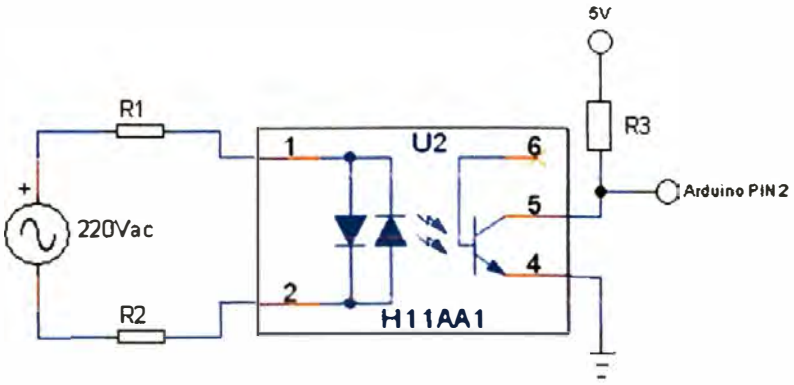


Figura 3.11 Esquema general del detector de cruce por cero

Las resistencias R1 y R2 tendrán la labor de disipar la energía de la red eléctrica. Las resistencias a utilizar son de 0.5W por lo que ambas resistencias se igualan a R para consumir como máximo medio Watt cada uno. De la ley de Ohm y de la ecuación (3.1), se calcula la potencia en función del voltaje y resistencia:

$$P = \frac{V^2}{R} \tag{3.2}$$

Dado que la potencia máxima es 1W, y de (3.2) se calcula:

$$P_{max} = \frac{220^2}{R_{min}} \rightarrow 2R_{min} = \frac{220^2}{1W}$$

$$R_{min} = 24.2\ K\Omega$$

Eligiendo un valor comercial, se asigna 30 KΩ a las resistencias.

$$\boxed{R1 = R2 = 30\ K\Omega}$$

El objetivo es detectar el momento en el cual el valor de la corriente de entrada es muy próximo a cero, es decir el transistor del H11AA1 está en corte, y asegurar que luego se encuentre en saturación, con ello el voltaje de salida sea $V_{ce(sat)}$ aproximadamente 0V, obteniendo como voltaje de salida la forma de la figura 3.12. Dicho voltaje irá al pin 2 del Arduino para realizar las interrupciones en cada cruce por cero.

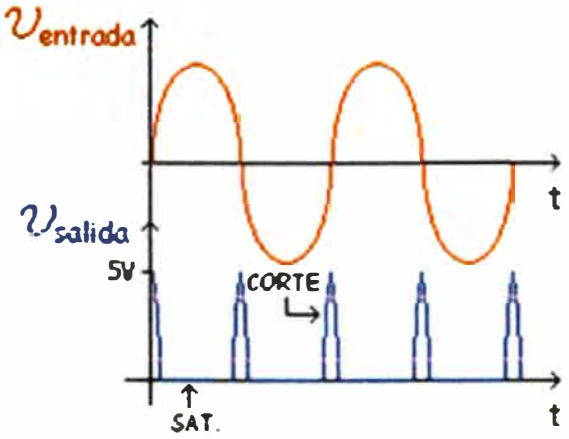


Figura 3.12 Gráfica Vce vs Ic

De acuerdo a la hoja de datos del optoacoplador, se probó con valores de corriente en colector de 0.5 mA y corriente directa de 10 mA, por lo que se diseñará la resistencia R3 con dicho valor de Ic:

$$5V = Ic \times R3 + V_{ce(sat)}$$

$$\boxed{R3 = \frac{5 - 0.4}{0.5} \approx 10K\Omega}$$

- **Circuito de conexión a la carga**

Luego de haber obtenido los momentos en que la señal eléctrica pasa por cero, ya es posible controlar el ángulo de disparo del TRIAC para regular la intensidad luminosa. Debido a que se controlará el circuito en AC, se utilizará el optoacoplador de tipo fototriac MOC3020. Además, dado que la carga que se utilizará es un foco incandescente, es decir puramente resistiva, no hay necesidad de colocar una red Snubber para proteger el TRIAC a los cambios bruscos de voltaje producidas por una carga inductiva, por ello solo se usará una resistencia entre un terminal del TRIAC y un pin del integrado como se muestra en la gráfica 3.13:

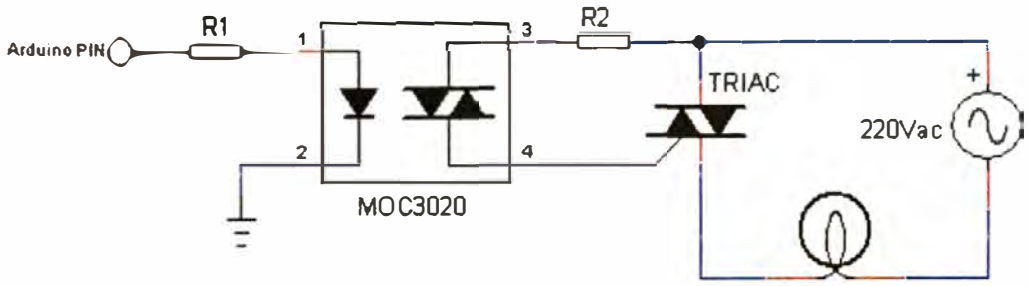


Figura 3.13 Esquema general del circuito para la carga

Según la hoja de datos del optoacoplador MOC3020 (ver Anexo C), la corriente mínima para disparar el fototriac del integrado es de 15mA teniendo una tensión del LED en polarización directa de 1.2 V. Dado que el pin de salida digital del Arduino alimentará con 5 V, de esta forma se puede hallar el valor de R1:

$$V_{pin} = I_{FT} \times R1 + V_F$$
$$5V = 15 \times R1_{max} + 1.2V$$
$$R1_{max} = 250\ \Omega$$

Eligiendo un valor comercial, se asigna 220 Ω a la resistencia R1.

$$\boxed{R1 = 220\ \Omega}$$

El pin de salida digital del Arduino, envía un pulso de corriente luego de un tiempo del cruce por cero (ángulo de disparo) definido por la lógica en el Arduino en cada de la red eléctrica para disparar el TRIAC que se encuentra en la salida del MOC3020. Con este valor de resistencia en la entrada del fototriac, la corriente máxima que entrega el Arduino es de aproximadamente 17 mA, con el cual se obtendría una forma de onda de corriente aplicada de la siguiente forma:

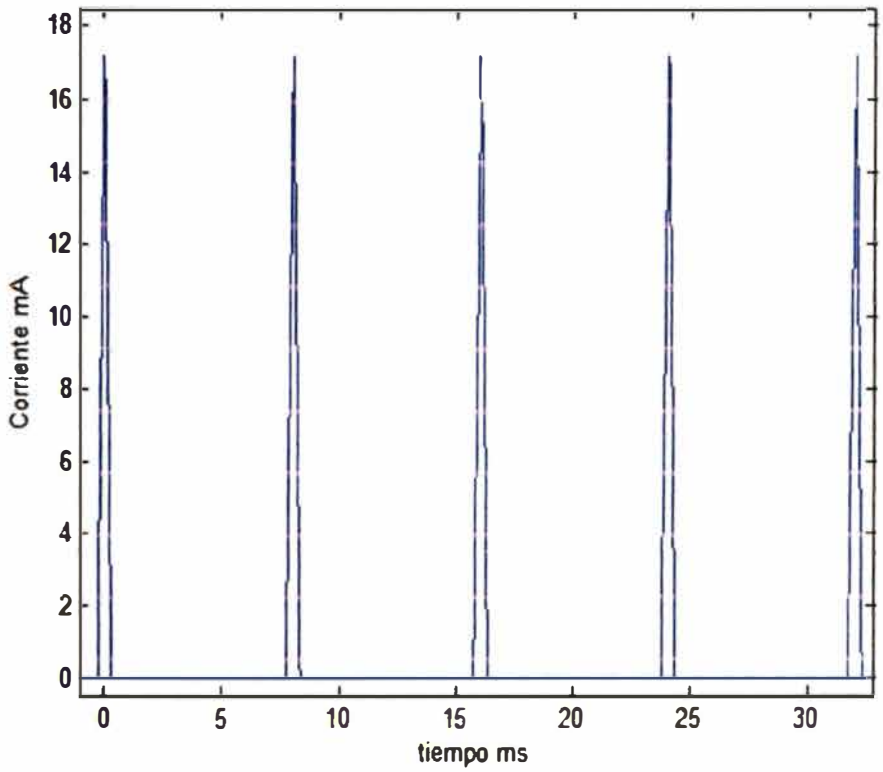


Figura 3.14 Forma de onda de corriente aplicada al MOC3020

El valor de la resistencia R2 está dada en la hoja de datos del MOC3020, en ella se asigna un valor de 180 Ω para una carga resistiva con señal eléctrica de 220Vac a 60Hz.

$$\boxed{R2 = 180\ \Omega}$$

CAPÍTULO IV

ANÁLISIS Y PRESENTACIÓN DE RESULTADOS

4.1 Introducción

En este capítulo se presenta los resultados obtenidos y el análisis de cada etapa del sistema implementado tomando en cuenta los procesos y dificultades que se tuvieron para desarrollar dicho sistema. En primer lugar se entrará en detalle al análisis de ciertas optimizaciones realizadas al código en *Java* de la aplicación. Luego, en la siguiente sección, se analizará lo realizado en la etapa de la placa Arduino y el inconveniente que se obtuvo al momento de enviar la señal de disparo. Como tercer punto, se analizará y observarán los resultados de la última etapa de potencia. Finalmente se revisará el costo final que se obtuvo para implementar este trabajo.

4.2 Etapa de la Aplicación Móvil

Respecto a la aplicación móvil, se realizaron las pruebas de conexión al módulo por *Bluetooth* usando las librerías de *Bluetooth* en *Android* con éxito y sin presentar mayores inconvenientes. El código escrito para las pruebas con los dos distintos tipos de luminaria a controlar, es decir con un foco *on/off* y otro *dimmer*, no resultaba extenso. Sin embargo, si se quisiera incrementar el número de bombillas, las líneas de código se incrementarían notablemente, ya que cada bombilla debe contar con un botón o barra (dependiendo el caso de foco) con un *listener* que actuará al momento de ser pulsado por el usuario. Además del *listener*, también se tiene el código que modifica los elementos de la interfaz de usuario (botones e imágenes) al momento de presionar los botones de encendido y variar la barra de intensidad que son manejados por el *Handler* que se vio en la sección 3.3.1.

Para solucionar ese inconveniente se crearon las clases que definen los elementos de la interfaz para una luz determinada, por ejemplo para la luz número 9, que es de tipo on/off, está contenida por la imagen iv9 y el botón btn9, estos elementos conforman el objeto de la clase *KeyImageBtn*. Para el caso de una luz *dimmer* como la del puerto 7, contiene la imagen iv7 y la barra sb7, estos elementos conforman el objeto de la clase *KeyImageSeek*. Luego se instancian dos objetos más de la clase *Map* para asociar el

número del puerto como tipo de variable *String* junto al objeto de la clase como se muestra en las siguientes líneas:

```
KeyImageBtn obj9;
KeyImageSeek obj7;
Map<String, KeyImageBtn> sw = new HashMap<String, KeyImageBtn>();
Map<String, KeyImageSeek> dim = new HashMap<String, KeyImageSeek>();
```

Finalmente en la función *onCreate* se inicializan los objetos y se les asigna al objeto de la clase *Map* que pertenece. De esta forma teniendo posibilidad de agregar más puertos a controlar añadiendo solamente unas líneas más de la siguiente.

```
obj9 = new KeyImageBtn(iv9,btn9);
obj7 = new KeyImageSeek(iv7, sb7);
sw.put("9", obj9);
dim.put("7", obj7);
```

Esto permite crear los *listeners* de manera recursiva como muestra las siguientes líneas de código, mediante la función que crea un *listener* que se vio en la sección 3.3.1. Es decir que los *listeners* de los botones y barras se crearán de acuerdo a los elementos que estén añadidos en los objetos sw y dim.

```
for (String st : sw.keySet()) {
     switchListener(st);
}
for (String st : dim.keySet()) {
     dimListener(st);
}
```

Respecto a la comunicación por *Bluetooth*, se realizaba la verificación de la cadena recibida por el módulo *Bluetooth* sin mayores problemas como muestra la herramienta de depuración *LogCat* (ver Fig 4.1), en él se aprecia los mensajes encerrados en los cuadros rojos, la cadena de recepción (*strIncom*) y lo que se envía. Lo que se envía al módulo son *bytes* y no caracteres imprimibles, por ello no se ve un mensaje entendible sino la interpretación de *Java* de esa cadena enviada. El mensaje que se recibe si presenta un texto entendible que es el número de puerto y su estado separado por comas como ya se había visto en el capítulo anterior.

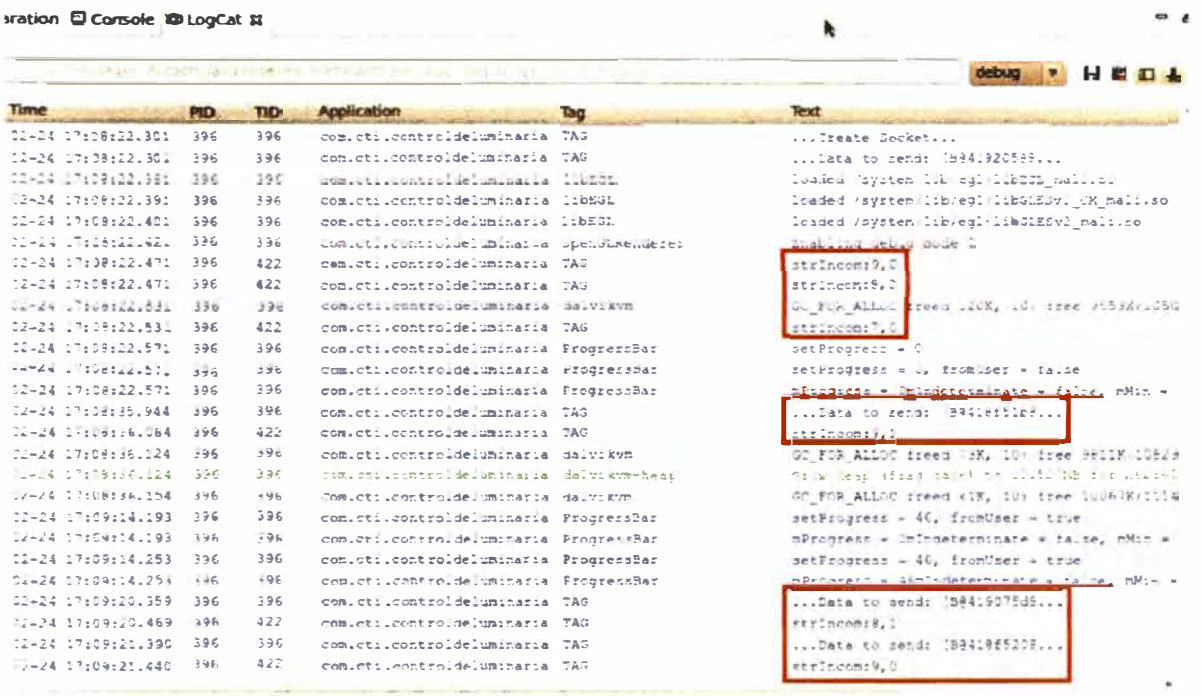


Figura 4.1 Depuración de mensajes transmitidos y recibidos

4.3 Etapa de Arduino

Respecto a la implementación de la tarjeta Arduino al sistema, no se tuvieron mayores inconvenientes, tanto al momento de modificar los parámetros del módulo *Bluetooth*, así como también al momento de cargar el código final que procesa los datos recibos y enviados a través de sus puertos seriales. No obstante, al momento de acoplarlo con el circuito de potencia, se tuvieron problemas al momento de variar el valor de intensidad con valores cercanos a 125. Al momento de modificar la barra a valores mayores a 120, la bombilla controlada se apagaba, es decir que el disparo no se lograba. Para solucionar esto, se ajustó los valores de retardo para el momento inicial en el código correspondiente a la función de detección de cruce por cero visto en la sección 3.3.3.

Este inconveniente presentado es debido en mayor parte a la etapa de potencia ya que el circuito de detección de cruce por cero, manda el pulso al pin 2 de la placa Arduino como se mostró en la figura 3.12. La interrupción en Arduino se realizará en el momento que se esté elevando el voltaje (opción *RISING*), es decir que el optoacoplador envía el pulso varios microsegundos antes que la señal eléctrica llegue realmente el cero, luego se mantiene en 5 V hasta que nuevamente vuelva a caer a cero como se muestra en la figura 4.1.

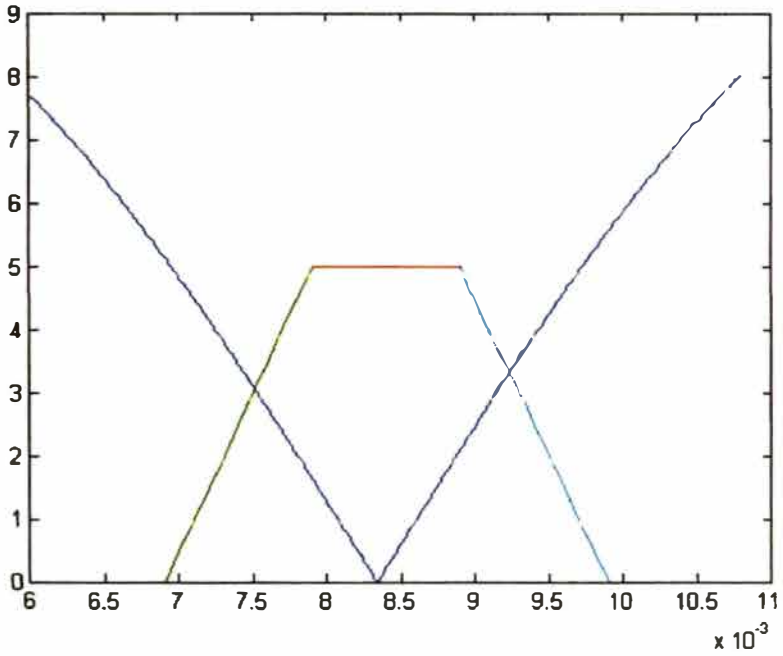


Figura 4.2 Gráfica de la señal del Pin 2 del Arduino

Experimentalmente se ajustó el valor hasta que no se obtuvieron fallas en el control de la luminaria, siendo este valor aproximadamente 750 microsegundos. Para evitar dar un solo retardo de ese valor, el cual implica el bloqueo del microprocesador durante ese tiempo, se extendió el índice i del *for* en la función para dar 13 retardos más de 60 microsegundos, como se vio en la sección 3.3.3.

4.4 Etapa de Potencia

Luego de tener implementada la comunicación inalámbrica de los dispositivos, se implementó en un *protoboard* primero antes de realizar la tarjeta. Los parámetros de diseño se mencionaron en la sección 3.3.4. Sin embargo se probaron algunos integrados como el optoacoplador MOC3020 y MOC3021 con los mismos resultados, el único requerimiento es que sea un foto-TRIAC y existen varios de la misma serie. La diferencia entre ellos está en la cantidad de corriente que activa el foto-TRIAC siendo el MOC3020 la que necesita mayor corriente para activar el tiristor en la salida. Se escogió el MOC3020 dejando abierta la posibilidad de escoger cualquier otro de la serie.

Igualmente el optoacoplador H11AA1 se escogió ya que contiene en la entrada dos diodos para evitar el uso de un rectificador de onda completa para detectar el cruce por cero de ambos ciclos positivo y negativo. Igualmente se probaron dos TRIAC diferentes, el BTA-12600 y el BT136 y 137 con iguales resultados, el tiempo de disparo no es determinante, solo varía el valor de cantidad de corriente que soportan ambos, siendo el TRIAC BTA-12600 la que permite mayor corriente pero esto implica mayor costo del

dispositivo electrónico. Por ello, dado que la carga es fija no implica una cantidad alta de corriente, se opta por el TRIAC BT136 de menor costo.

4.4 Presupuesto

En este trabajo, existen dos tipos de componentes de acuerdo al costo, los que son de una sola adquisición como la tarjeta Arduino con los componentes del detector de cruce por cero y los que se requerirán de manera proporcional a la cantidad de puertos a controlar como los optoacopladores MOC3020. Por lo tanto, se especificará el costo de cada dispositivo en diferentes secciones separándolos por los tipos de costo.

4.4.1 Costo único

Los componentes que cuentan con un único costo son la placa Arduino, el módulo *Bluetooth* JY-MCU y los dispositivos electrónicos que intervienen en el circuito detector de cruce por cero. El costo de cada componente requerido se muestra en la tabla 4.1.

TABLA N° 4.1 Precio de componentes de costo único

Dispositivo	Costo
Arduino UNO	S/. 85.00
Módulo JY-MCU	S/. 25.00
H11AA1	S/. 5.00
1 Resistencia 10KΩ	S/. 0.05
2 Resistencias 30KΩ	S/. 0.10
TOTAL	**S/. 115.15**

4.4.2 Costo recurrente

De acuerdo a la cantidad de luminaria a controlar y su modo de funcionamiento, sea *on/off* o *dimmer*, variará el costo. Se especificará el costo para un solo foco para cada uno de los casos, requiriendo solo un relé de 5V para el modo de encendido y apagado y el resto de componentes para el *dimmer*.

TABLA N° 4.2 Precio de componentes de costo recurrente

Dispositivo	Costo
Relé 5V	S/. 2.00
MOC3020	S/. 1.50
BT 136	S/. 1.00
1 Resistencia 180Ω	S/. 0.05
1 Resistencias 220Ω	S/. 0.05
TOTAL	**S/. 4.6**

En caso de requerirse controlar N focos on/off y M focos *dimmer*, el costo total en soles sería:

$$\boxed{Total = 2N + 2.6M}$$

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Observando los objetivos que se tuvieron al empezar el trabajo, se puede concluir que todos los puntos se han logrado obtener de manera satisfactoria. La aplicación móvil cuenta con la compatibilidad de la mayoría de dispositivos *Android*. Además de la implementación de bajo costo junto al uso de tecnologías actuales y vigentes como *Bluetooth* y la plataforma Arduino. Igualmente, se destaca la escalabilidad del sistema en el caso de aumentar focos para controlar. Otro punto a favor es el bajo consumo energético de la comunicación por *Bluetooth*, por lo cual la aplicación no realiza un desgaste significativo de la batería.

Existe también una gran ventaja en la implementación de este sistema debido a que a pesar de ser utilizado solo para el control de luminaria, se podrían desarrollar otras aplicaciones domóticas. Manteniendo la etapa de la aplicación móvil, la comunicación inalámbrica y las instrucciones que ejecuta Arduino al recibir los mensajes del dispositivo *Android*, se podrían controlar otros periféricos. Por ejemplo, la apertura de las puertas electrónicas que requieren un pulso de 12 V, igualmente la apertura y cierre de persianas o el control de la puerta de un garaje, podrían ser implementadas de manera directa con lo realizado en las etapas previas, solo se tendría que diseñar la última etapa para elevar el voltaje ya que los puertos de salida digital del Arduino están limitadas a 5 V.

Finalmente aunque el sistema implementado no posee todos los elementos que posee un sistema domótico, este sistema está abierto a aumentar su capacidad de controlar otros elementos con los puertos libres que tiene la tarjeta Arduino como se explicó en el párrafo anterior y presenta un coste de implementación bajo y fácil de implementar.

Recomendaciones

De acuerdo al sistema presentado en este trabajo, se puede desarrollar y mejorar ciertos aspectos para trabajos futuros para crear un sistema más completo, con mayor funcionalidad y abierto a mayor cantidad de dispositivos, los cuales se exponen a continuación:

- La topología de este trabajo es de tipo centralizada siendo el módulo *Bluetooth* conectado al Arduino el nodo central que se comunica con el dispositivo *Android*. Esto limita el tamaño de la vivienda dado que el rango de comunicación por *Bluetooth* es limitado. Una forma de superar este inconveniente es utilizar otros tipos de tecnología de comunicación inalámbrica como por ejemplo el estándar *ZigBee* en la cual se puede implementar una topología de malla con distintos nodos, de esta forma ampliando la cobertura a lo largo del recinto. Además de tener redundancia en caso que uno de los nodos fallen. Sin embargo, implementar esto conlleva a un incremento de costo y también se debe considerar el inconveniente de tener todos los módulos de radiofrecuencia *Xbee* energizados todo el tiempo.

- La implementación de la aplicación móvil en este trabajo se limita solo a los dispositivos *Android*. Si bien es cierto que es el sistema operativo de mayor utilización a nivel mundial y nacional, se deja de lado otros sistemas operativos como los de *iOS* con el lenguaje de programación *Objective*-C, *Blackberry*, etc. Se recomendaría implementar las aplicaciones en el lenguaje nativo de cada sistema o también utilizar el *framework* de desarrollo de aplicaciones móviles *PhoneGap*. Esta *software* permite desarrollar las aplicaciones móviles utilizando solo los lenguajes de programación web (*HTML*, *CSS* y *JavaScript*) para todos los sistemas operativos móviles que existen.

- Se podría ampliar la funcionalidad del sistema a otras aplicaciones domóticas adicionales dado que la tarjetas Arduino utilizada, posee puertos con entrada analógica las cuales no se utilizan en este trabajo pero que podrían ser utilizadas con sensores conectados a la placa para controlar otras variables del ambiente en un hogar como por ejemplo la temperatura.

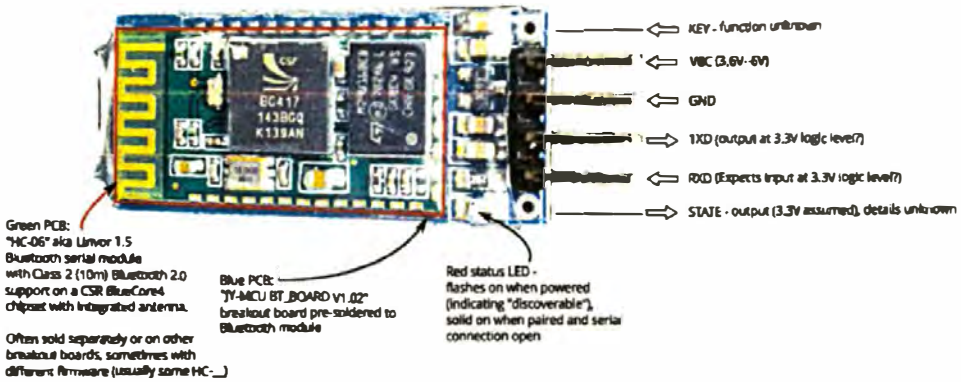
ANEXO A

Diagrama de Módulo *Bluetooth* JY-MCU

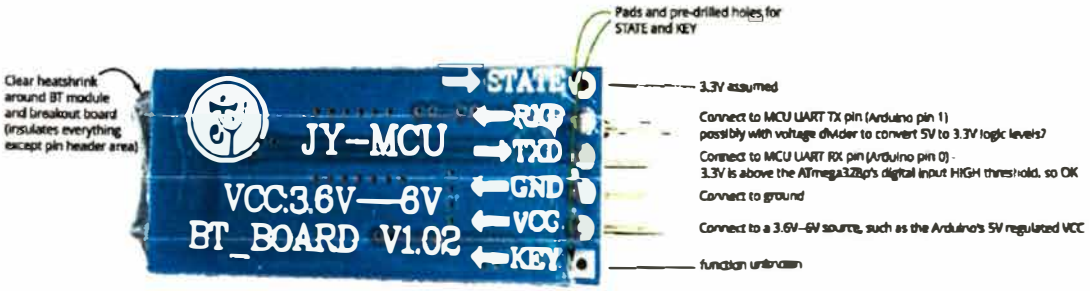
Annotated Diagram of Bluetooth Module and Breakout

From DealExtreme: "JY-MCU Arduino Bluetooth Wireless Serial Port Module", SKU 104299
This one came with the HC-06 (aka Linvor 1.5) firmware installed: default serial setting 9600/N/1, pairing code 1234

Breakout Board and BT Module Front Side



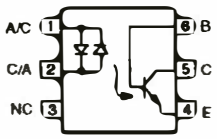
Breakout Board Back Side



ANEXO B

Hoja de Datos de Optoacoplador H11AA1

Optocoupler, Phototransistor Output, AC Input, With Base Connection



FEATURES

- AC or polarity insensitive input
- Built-in reverse polarity input protection
- I/O compatible with integrated circuits
- Industry standard DIP package
- Isolation test voltage: 5300 V_{RMS}
- Lead (Pb)-free component
- Component in accordance to RoHS 2002/95/EC and WEEE 2002/96/EC

APPLICATIONS

- Telephone line detection
- AC line motor
- PLC
- Instrumentation

DESCRIPTION

The H11AA1 is a bi-directional input optically coupled isolator consisting of two inverse parallel Gallium Arsenide Infrared LEDs coupled to a silicon NPN phototransistor in a 6-pin DIP package. The H11AA1 has a minimum CTR of 20 %, a CTR symmetry of 1:3 and is designed for applications requiring detection or monitoring of AC signals.

AGENCY APPROVALS

- UL1577, file no. E52744 system code H or J, double protection
- CSA 93751
- BSI IEC60950 IEC60065
- DIN EN 60747-5-2 (VDE0884)
  DIN EN 60747-5-5 pending
  available with option 1
- FIMKO

ORDER INFORMATION	
PART	REMARKS
H11AA1	CTR > 20 %, DIP-6
H11AA1-X006	CTR > 20 %, DIP-6 400 mil (option 6)
H11AA1-X007	CTR > 20 %, SMD-6 (option 7)
H11AA1-X009	CTR > 20 %, SMD-6 (option 9)

Note
For additional information on the available options refer to Option Information.

H11AA1

Vishay Semiconductors — Optocoupler, Phototransistor Output, AC Input, With Base Connection

VISHAY.

ABSOLUTE MAXIMUM RATINGS				
PARAMETER	**TEST CONDITION**	**SYMBOL**	**VALUE**	**UNIT**
INPUT				
Forward continuous current		I_F	60	mA
Power dissipation		P_{diss}	100	mW
Derate linearly from 25 °C			1.3	mW/°C
OUTPUT				
Power dissipation		P_{diss}	200	mW
Derate linearly from 25 °C			2.6	mW/°C
Collector emitter breakdown voltage		BV_{CEO}	30	V
Emitter base breakdown voltage		BV_{EBO}	5	V
Collector base breakdown voltage		BV_{CBO}	70	V
COUPLER				
Isolation test voltage (RMS)	between emitter and detector, referred to standard climate 23 °C/50% RH, DIN 50014	V_{ISO}	5300	V_{RMS}
Creepage distance			≥ 7	mm
Clearance distance			≥ 7	mm
Comparative tracking index	per DIN IEC 112/VDE 0303, part 1		175	
Isolation resistance	V_{IO} = 500 V, T_{amb} = 25 °C	R_{IO}	≥ 10^{12}	Ω
	V_{IO} = 500 V, T_{amb} = 100 °C	R_{IO}	≥ 10^{11}	Ω
Storage temperature range		T_{stg}	- 55 to + 150	°C
Operating temperature range		T_{amb}	- 55 to + 100	°C
Lead soldering time at 260 °C		T_{sld}	10	s

Note

T_{amb} = 25 °C, unless otherwise specified.
Stresses in excess of the absolute Maximum Ratings can cause permanent damage to the device. Functional operation of the device is not implied at these or any other conditions in excess of those given in the operational sections of this document. Exposure to absolute Maximum Rating for extended periods of the time can adversely affect reliability.

ELECTRICAL CHARACTERISTCS						
PARAMETER	**TEST CONDITION**	**SYMBOL**	**MIN.**	**TYP.**	**MAX.**	**UNIT**
INPUT						
Forward voltage	I_F = 10 mA	V_F		1.2	1.5	V
OUTPUT						
Collector emitter breakdown voltage	I_C = 1 mA	BV_{CEO}	30			V
Emitter base breakdown voltage	I_E = 100 µA	BV_{EBO}	5			V
Collector base breakdown voltage	I_C = 100 µA	BV_{CBO}	70			V
Collector emitter leakage current	V_{CE} = 10 V	I_{CEO}		5	100	nA
COUPLER						
Collector emitter saturation voltage	I_F = 10 mA, I_C = 0.5 mA	V_{CEsat}			0.4	V

Note

T_{amb} = 25 °C, unless otherwise specified.
Minimum and maximum values were tested requirements. Typical values are characteristics of the device and are the result of engineering evaluations. Typical values are for information only and are not part of the testing requirements.

CURRENT TRANSFER RATIO						
PARAMETER	**TEST CONDITION**	**SYMBOL**	**MIN.**	**TYP.**	**MAX.**	**UNIT**
DC current transfer ratio	I_F = 10 mA, V_{CE} = 10 V	CTR_{DC}	20			%
Symmetry (CTR at + 10 mA)/(CTR at - 10 mA)			0.33	1	3	

TYPICAL CHARACTERISTICS

T_{amb} = 25 °C, unless otherwise specified

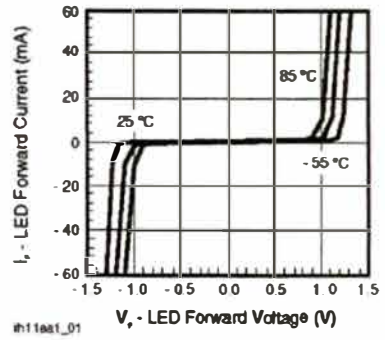


Fig. 1 - LED Forward Current vs. Forward Voltage

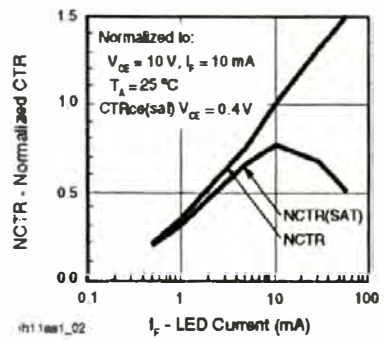


Fig. 2 - Normalized Non-Saturated and Saturated CTR vs. LED Current

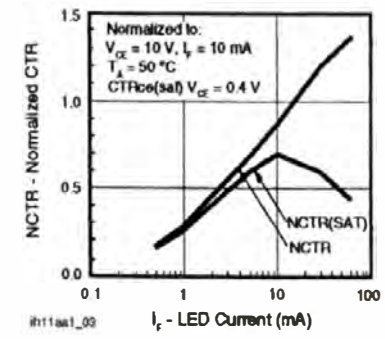


Fig. 3 - Normalized Non-Saturated and Saturated CTR vs. LED Current

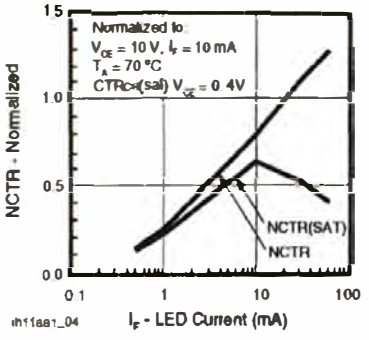


Fig. 4 - Normalized Non-Saturated and Saturated CTR vs. LED Current

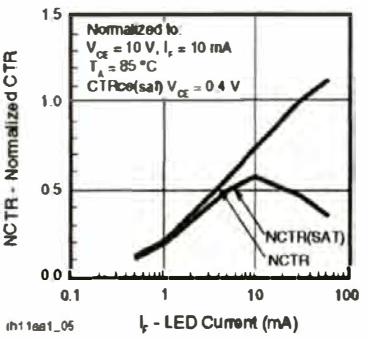


Fig. 5 - Normalized Non-Saturated and Saturated CTR vs. LED Current

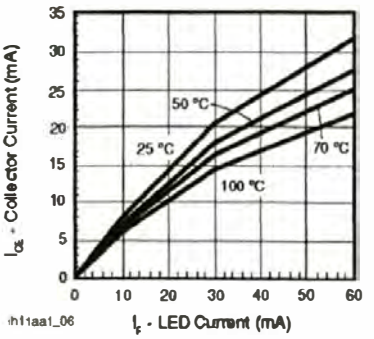


Fig. 6 - Collector Emitter Current vs. Temperature and LED Current

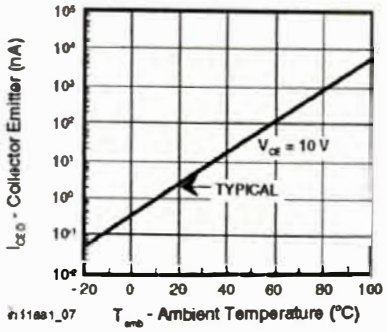


Fig. 7 - Collector Emitter Leakage Current vs. Temperature

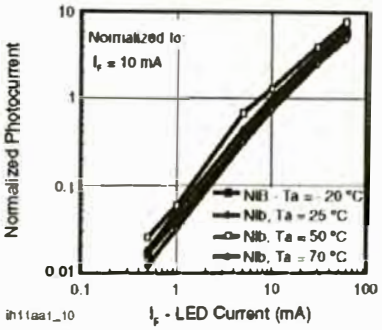


Fig. 10 - Normalized Photocurrent vs. LED Current

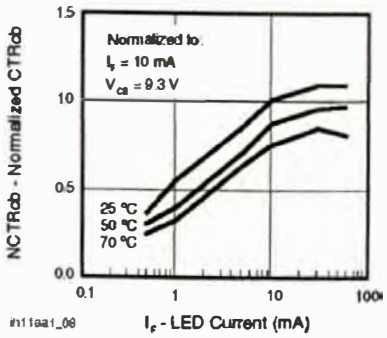


Fig. 8 - Normalized CTRcb vs. LED Current and Temperature

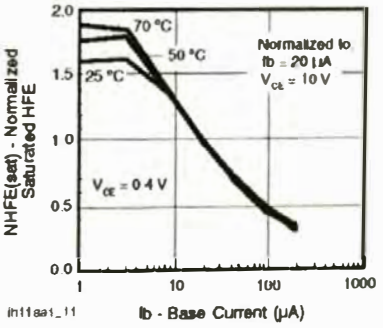


Fig. 11 - Normalized Saturated HFE vs. Base Current and Temperature

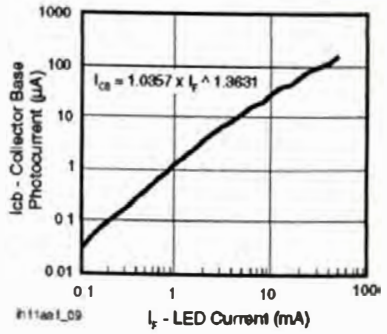


Fig. 9 - Collector Base Photocurrent vs. LED Current

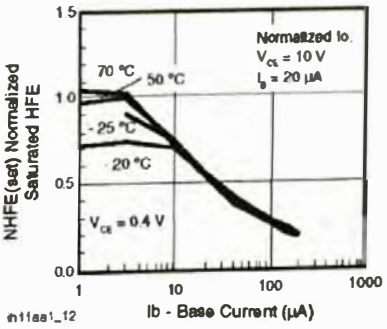
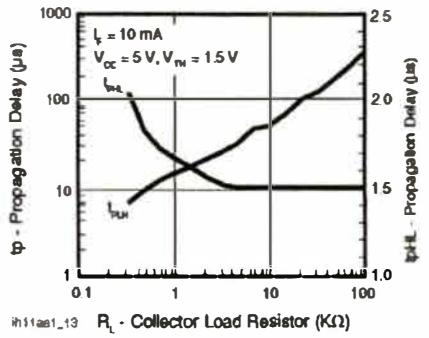
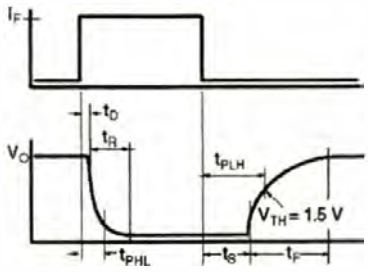


Fig. 12 - Normalized Saturated HFE vs. Base Current and Temperature



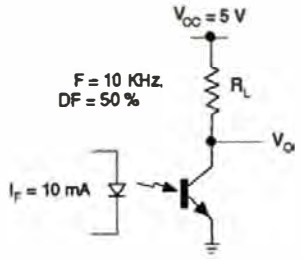
ih11aa1_13

Fig. 13 - Propagation Delay vs. Collector Load Resistor



ih11aa1_14

Fig. 14 - Switching Waveform



ih11aa1_15

Fig. 15 - Switching Schematic

ANEXO C

Hoja de Datos de Optoacoplador MOC3020

MOC3020 THRU MOC3023
OPTOCOUPLERS/OPTOISOLATORS

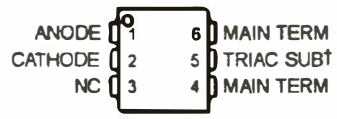
SOES025A – OCTOBER 1998 – REVISED APRIL 1998

- **400 V Phototriac Driver Output**
- **Gallium-Arsenide-Diode Infrared Source and Optically-Coupled Silicon Triac Driver (Bilateral Switch)**
- **UL Recognized . . . File Number E65085**
- **High Isolation . . . 7500 V Peak**
- **Output Driver Designed for 220 Vac**
- **Standard 6-Terminal Plastic DIP**
- **Directly Interchangeable with Motorola MOC3020, MOC3021, MOC3022, and MOC3023**

typical 115/240 Vac(rms) applications

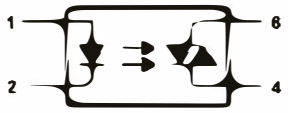
- **Solenoid/Valve Controls**
- **Lamp Ballasts**
- **Interfacing Microprocessors to 115/240 Vac Peripherals**
- **Motor Controls**
- **Incandescent Lamp Dimmers**

MOC3020 – MOC3023 . . . PACKAGE
(TOP VIEW)



† Do not connect this terminal
NC – No internal connection

logic diagram



absolute maximum ratings at 25°C free-air temperature (unless otherwise noted)†

Input-to-output peak voltage, 5 s maximum duration, 60 Hz (see Note 1)	7.5 kV
Input diode reverse voltage	3 V
Input diode forward current, continuous	50 mA
Output repetitive peak off-state voltage	400 V
Output on-state current, total rms value (50-60 Hz, full sine wave): T_A = 25°C	100 mA
T_A = 70°C	50 mA
Output driver nonrepetitive peak on-state current (t_w = 10 ms, duty cycle = 10%, see Figure 7)	1.2 A
Continuous power dissipation at (or below) 25°C free-air temperature:	
Infrared-emitting diode (see Note 2)	100 mW
Phototriac (see Note 3)	300 mW
Total device (see Note 4)	330 mW
Operating junction temperature range, T_J	–40°C to 100°C
Storage temperature range, T_{stg}	–40°C to 150°C
Lead temperature 1,6 (1/16 inch) from case for 10 seconds	260°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

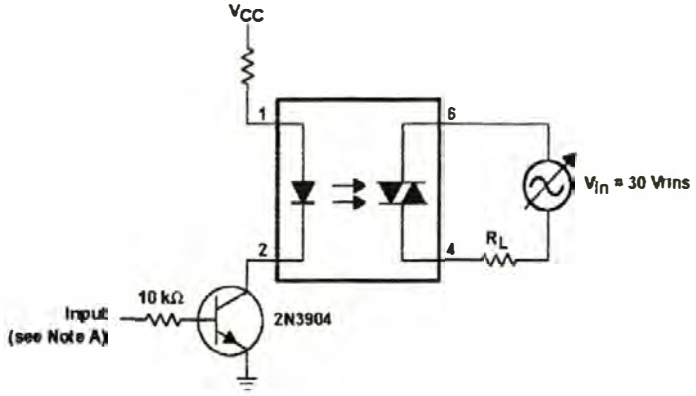
NOTES: 1. Input-to-output peak voltage is the internal device dielectric breakdown rating.
2. Derate linearly to 100°C free-air temperature at the rate of 1.33 mW/°C.
3. Derate linearly to 100°C free-air temperature at the rate of 4 mW/°C.
4. Derate linearly to 100°C free-air temperature at the rate of 4.4 mW/°C.

electrical characteristics at 25°C free-air temperature (unless otherwise noted)

PARAMETER			TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_R	Static reverse current		V_R = 3 V		0.05	100	µA
V_F	Static forward voltage		I_F = 10 mA		1.2	1.5	V
$I_{(DRM)}$	Repetitive off-state current, either direction		$V_{(DRM)}$ = 400 V, See Note 5		10	100	nA
dv/dt	Critical rate of rise of off-state voltage		See Figure 1		100		V/µs
dv/dt(c)	Critical rate of rise of commutating voltage		I_O = 15 mA, See Figure 1		0.15		V/µs
I_{FT}	Input trigger current, either direction	MOC3020	Output supply voltage = 3 V		15	30	mA
		MOC3021			8	15	
		MOC3022			5	10	
		MOC3023			3	5	
V_{TM}	Peak on-state voltage, either direction		I_{TM} = 100 mA		1.4	3	V
I_H	Holding current, either direction				100		µA

NOTE 5: Test voltage must be applied at a rate no higher than 12 V/µs.

PARAMETER MEASUREMENT INFORMATION



NOTE A. The critical rate of rise of off-state voltage, dv/dt, is measured with the input at 0 V. The frequency of V_{in} is increased until the phototriac turns on. This frequency is then used to calculate the dv/dt according to the formula:

$$dv/dt = 2\sqrt{2}\,\pi f V_{in}$$

The critical rate of rise of commutating voltage, dv/dt(c), is measured by applying occasional 5-V pulses to the input and increasing the frequency of V_{in} until the phototriac stays on (latches) after the input pulse has ceased. With no further input pulses, the frequency of V_{in} is then gradually decreased until the phototriac turns off. The frequency at which turn-off occurs may then be used to calculate the dv/dt(c) according to the formula shown above.

Figure 1. Critical Rate of Rise Test Circuit

TYPICAL CHARACTERISTICS

EMITTING-DIODE TRIGGER CURRENT (NORMALIZED)
vs
FREE-AIR TEMPERATURE

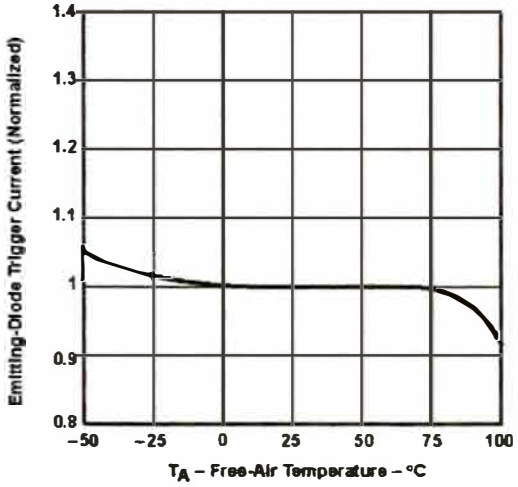


Figure 2

ON-STATE CHARACTERISTICS

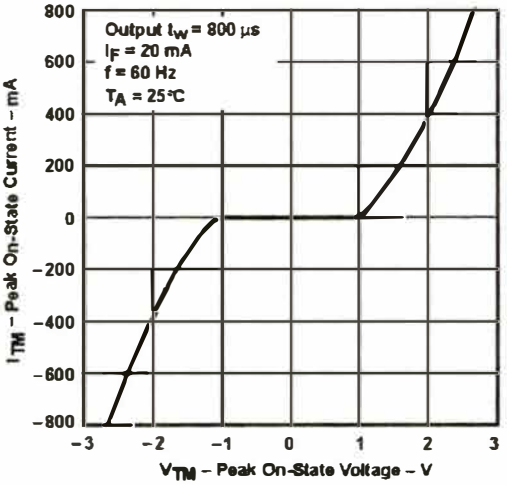


Figure 3

NONREPETITIVE PEAK ON-STATE CURRENT
vs
PULSE DURATION

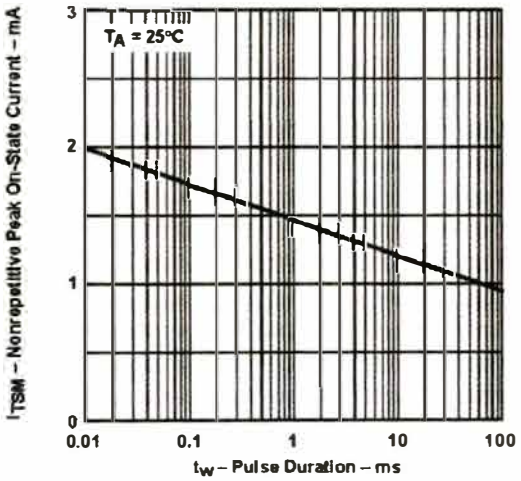


Figure 4

APPLICATIONS INFORMATION

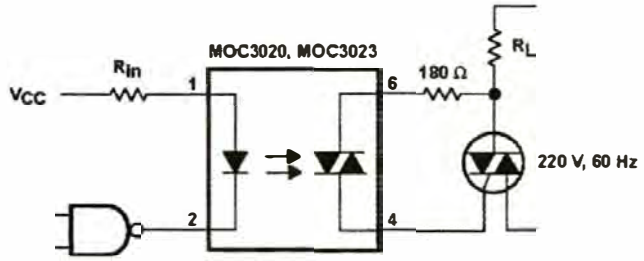
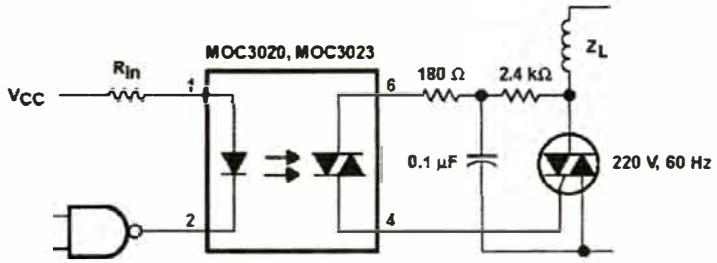
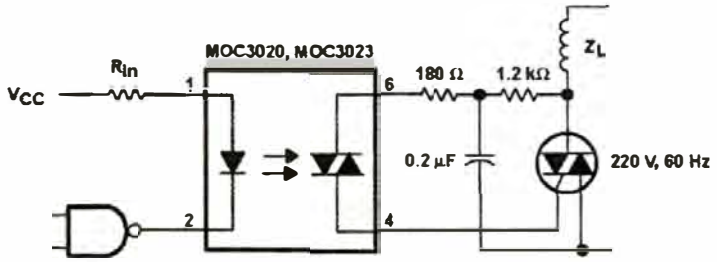


Figure 5. Resistive Load



$I_{GT} \leq 15$ mA

Figure 6. Inductive Load With Sensitive-Gate Triac



$15 \text{ mA} < I_{GT} < 50 \text{ mA}$

Figure 7. Inductive Load With Nonsensitive-Gate Triac

BIBLIOGRAFÍA

1. Renzo Bertuzzi L., Patricio Guarda M. y Juan Pablo Salazar F, "Automatización del Hogar Usando el Protocolo de Comunicación X10", Universidad Austral de Chile, 2004
2. Rahul Balani, "Energy Consumption Analysis for Bluetooth, WiFi and Cellular Networks", University of California at Los Angeles, 2007
3. Lorena Aguirre Solvez, "Estudio de una red de sensores sin hilos basada en la tecnología Arduino bajo protocolos de comunicaciones Zigbee", Universidad Politécnica de Catalunya - España, 2009.
4. IDC Worldwide Quarterly Mobile Phone Tracker (2014): Worldwide Smartphone Market Expected. [Online]. Disponible en http://www.idc.com/getdoc.jsp?containerId=prUS24442013
5. Diario Gestión, "Los usuarios peruanos prefieren Android, por encima del IOS", 2014, [Online]. Disponible en http://gestion.pe/tecnologia/usuarios-peruanos-prefieren-android-encima-ios-2085273
6. Satya Komatineni y Dave MacLean, "Pro Android 4", Apress, 2012
7. Sitio oficial de desarrolladores de Android - Arquitectura. [Online]. Disponible en http://developer.android.com/images/system-architecture.jpg
8. Sitio oficial de desarrolladores de Android - Actividad. [Online]. Disponible en http://developer.android.com/reference/android/app/Activity.html
9. Sitio oficial de desarrolladores de Android - Servicio. [Online]. Disponible en http://developer.android.com/guide/components/services.html
10. Sitio oficial de Arduino. [Online]. Disponible en http://www.arduino.cc/es/
11. Sitio oficial de Arduino - Introducción. [Online]. Disponible en http://www.arduino.cc/es/Guide/Introduction
12. Sitio oficial de Arduino - AttachInterrupt. [Online]. Disponible en http://arduino.cc/en/Reference/attachInterrupt
13. Sitio oficial de Arduino - Productos. [Online]. Disponible en http://arduino.cc/en/Main/Products
14. Malik Zaka Ullah, "An Analysis of the Bluetooth Technology", Blekinge Institute of Technology – Suecia, 2009

15. Jochen Schiller, "Mobile Communications", 2da Edición, Addison-Wesley, 2003
16. J. Smith, J. Speakes, M. H. Rashid, "An overview of the modems light dimmer: Design, Operation and Application", IEEE, 2005
17. R. Simpson, "Lighting Control: Technology and Applications", Focal Press, 2003
18. Kevin Vick, "Characterizing Dimming of an LED Light Bulb for Incandescent Replacement", Case Western Reserve University – USA, 2011
19. Muhammad H. Rashid, "Electrónica de Potencia", 2da Edición, Prentice Hall, 1993
20. Juan Carlos Malataxi Amagua, "Construcción de un Circuito de Luces de Seis Canales Accionadas por Sonido", Escuela Politécnica Nacional - Ecuador, 2007
21. Sitio oficial de Bluetooth – Service Discovery. [Online]. Disponible en https://www.bluetooth.org/en-us/specification/assigned-numbers/service-discovery
22. Sitio oficial de desarrolladores de Android - Bluetooth. [Online]. Disponible en http://developer .android.com/reference/android/bluetooth/BluetoothDevice.html
23. Grant Allen, "Beginning Android", Apress, 2012
24. Tutorial de Conexión Arduino con Módulo JY-MCU. [Online]. Disponible en http://ecno92.blogspot.com/2012/11/jy-mcu-linvor-at-commands-change-name.html
25. Trevor Stork y Moira Mathers, "The Basics of Efficient Lighting", National Framework for Energy Efficiency – Australia, 1ra Edición, 2009. [Online]. Disponible en http://www.energyrating.gov.au/wp-content/uploads/2011/02/2009-ref-manual-lighting.pdf