Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Proyecto de Integración

Manipulación de un brazo robótico a través de una interfaz para teléfono móvil

Alumno: Marco Antonio Bucio Martínez

Matrícula: 208202307

Asesor: Ricardo Godínez Bravo

Profesor asistente

Departamento de Electrónica

Trimestre 2014 Primavera

28 de agosto de 2014

Yo, ***Ricardo Godínez Bravo***, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma: 

Yo, ***Marco Antonio Bucio Martínez***, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma: 

Resumen

Para el desarrollo de este proyecto se diseñó un sistema embebido con capacidad de conexión a Internet a través de un punto de acceso Wi-Fi[1] para el envío de datos entre el sistema y un teléfono celular, de forma que el teléfono haga la función de un control remoto.

El dispositivo está basado en una microcontrolador AT89C51 fabricado por Atmel, que posee una CPU de 8 bits, este microcontrolador recibirá las instrucciones desde el teléfono celular, de forma que una vez que reciba los datos, éste los interpretará para que pueda manipular los motores del brazo robótico.

El circuito que llevará montado el microcontrolador también llevará montado el módulo RN-XV Wi-Fly, que es un dispositivo que nos permite conectarnos a un punto de acceso por medio de una señal Wi-Fi.

Para el diseño del control remoto se realizó una aplicación para dispositivos electrónicos con sistema operativo Android[2] (Smartphone, Tablet) en versiones 4.0.4 o menores. La aplicación fue desarrollada en lenguaje Java utilizando el IDE[3] de programación Eclipse[4], empleando el SDK[5] de Android y el plug-in Android Development Tools para integrar el desarrollo de aplicaciones para Android dentro de un proyecto de Eclipse.

Por último el brazo robótico que se utilizará será el modelo K-682 de Steren™, el cual tiene cinco grados de libertad, teniendo tres diferentes movimientos verticales, uno horizontal, y un movimiento de apertura y cierre de una tenaza.

[1] Mecanismo inalámbrico de conexión
[2] Sistema operativo diseñado principalmente para dispositivos móviles con pantalla táctil
[3] Por sus siglas en inglés, Integrated Development Environment, es un conjunto de herramientas de programación.
[4] Programa compuesto por un conjunto de herramientas de programación de código abierto.
[5] Por sus siglas en inglés, Software Development Kit, conjunto de herramientas para desarrollar aplicaciones

Tabla de contenido

Índice de diagramas, tablas y figuras

1. Introducción

Durante el desarrollo de este proyecto se ha estudiado como desarrollar algunos sistemas embebidos, que nos pueden servir para crear desde pequeños controles remotos, hasta robots articulados con sensores que permitan hacer trabajos donde no es necesaria la intervención del hombre.

Se ha estudiado como se relacionan un microcontrolador con uno o varios dispositivos vía UART[6] y cómo se puede enviar información a través del protocolo TCP/IP.

Una de las partes más importantes en el desarrollo de este proyecto fue el estudio del dispositivo RN-XV Wi-Fly, que es el módulo que nos permitirá hacer la conexión entre el teléfono celular y el circuito que manipula el brazo robótico, éste módulo tiene diferentes modos de trabajar y la configuración se puede hacer de distintas formas, pero lo más importante fue el modo de asociación a un punto de acceso y la forma de configurarlo para trabajar por medio del protocolo TCP/IP[7].

Se ha elegido el ambiente de desarrollo eclipse para programar la aplicación para Android, esto gracias a la gran cantidad de información y tutoriales que existen para guiarse en el desarrollo de la misma.

Una de las ventajas de trabajar con el sistema operativo Android es que el manejo de sockets[8] TCP es muy parecido al manejo de sockets en Java, por lo cual se utilizó un programa de escritorio en Java como plantilla, y a partir de ese programa se diseñó la aplicación final.

Por último una parte importante es la alimentación de los diferentes circuitos utilizados, un problema con el que se tuvo que lidiar fue el hecho de que el módulo Wi-.Fly se alimenta con 3.3 volts y no con 5 volts como el microcontrolador, por lo cual se utilizó un circuito extra que nos permitiera alimentar todos los componentes con 5 volts.

[6] Por sus siglas en inglés, Universal Asynchronous Receiver-Transmitter, Transmisor/Receptor Asíncrono Universal, es el que controla los puertos y dispositivos seriales.
[7] Familia de protocolos de Internet que permiten la transmisión de datos entre dispositivos.
[8] Mecanismo para la entrega de paquetes entre dispositivos.

2. Antecedentes

Un sistema embebido es un sistema de computación diseñado para realizar un conjunto de funciones o tareas específicas en tiempo real. Las tareas que debe de realizar van desde control de motores, procesamiento de datos a pequeñas escala, entre otras.

El componente central de un sistema distribuido es la CPU. Y es la que controla todo el sistema. Gracias a él se pueden conectar otros dispositivos que realizan funciones de medición, control, etc.

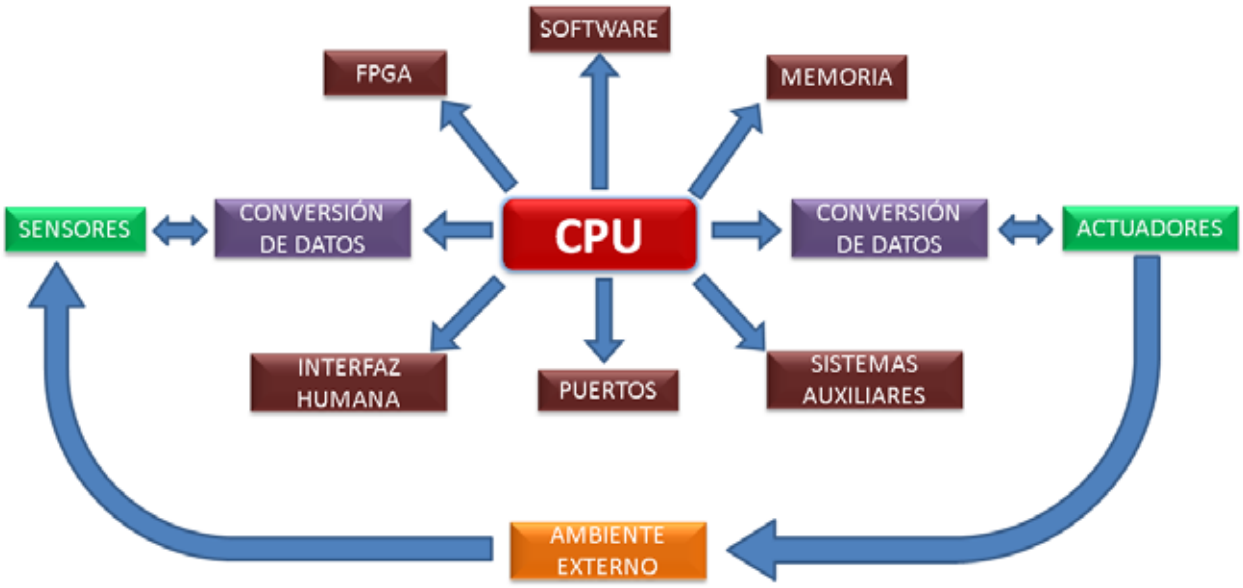

Diagrama 1.Funcionamiento de la CPU de un sistema distribuido

El primer sistema embebido reconocido fue el sistema de guía de Apolo desarrollado por el laboratorio de desarrollo del MIT[9] para las misiones Apolo hacia la luna. Cada vuelo hacia la luna tenía dos de estos sistemas. La función era manejar el sistema de guía inercial de los módulos de excursión lunar. En un comienzo fue considerado como el elemento que más riesgo presentaba en el proyecto Apolo. Este sistema de cómputo fue el primero en utilizar circuitos integrados y utilizaba una memoria RAM magnética, con un tamaño de palabra de 16 bits. El software fue escrito en el lenguaje

[9] Siglas en inglés de Massachusetts Institute of Technology

ensamblador propio y constituía en el sistema operativo básico, pero capaz de soportar hasta ocho tareas simultáneas.

El primer sistema embebido producido en masa, fue la computadora guía del misil norteamericano Minuteman II en 1962. El principal aspecto de diseño de la computadora del Minuteman, es que además de estar construido con circuitos integrados, permitía reprogramar los algoritmos de guía del misil para la reducción de errores, y permitía realizar pruebas sobre el misil ahorrando así el peso de los cables y conectores.

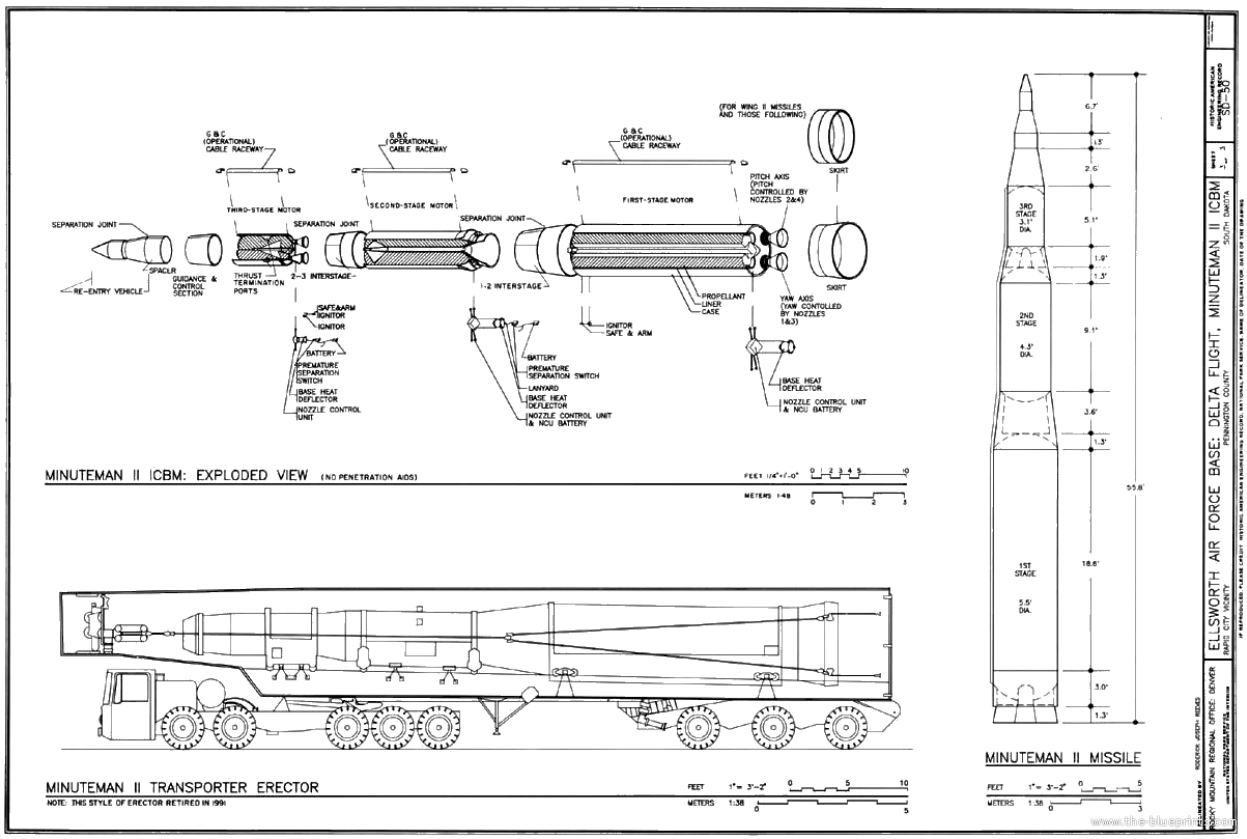

Figura 1. Misil Minuteman II

3. Justificación

Aunque la comunicación Wi-Fi es relativamente nueva (finales de 1999 y principios de 2000), ya es un mecanismo que se utiliza mucho hoy en día, no sólo en cuestiones de ingeniería o de investigación, sino también en la vida diaria. En la actualidad hay muchos campos más en donde cobra mucha más fuerza, como la robótica, mecatrónica, electrónica, mecánica o computación.

La manipulación de dispositivos electrónicos a través de interfaces gráficas es un tema de actualidad que todo ingeniero que esté familiarizado con la computación y electrónica debe conocer. En este proyecto se aplicarán técnicas que muestren un panorama de lo que se está trabajando en dichos campos de estudio, así mismo el presente proyecto mostrará la capacidad del alumno para desarrollar proyectos similares en el campo laboral.

Un sistema como el que se desarrollará en este proyecto puede servir para diferentes objetivos, dependiendo de la situación, en éste caso es para manipular un brazo, pero bien podrían manipularse otro tipo de dispositivos, como sistemas de seguridad, sistemas de transporte de materiales, o incluso cosas tan sencillas como abrir una puerta o encender las luces en una casa, todo esto basándose en la misma idea, un dispositivo con interfaz gráfica que manipule otros a través de comunicación inalámbrica.

La principal diferencia de este proyecto a otros con objetivos similares, es el hecho de que en el presente proyecto el control remoto será a través de un dispositivo móvil (teléfono celular), lo cual nos permitirá una mayor portabilidad para la manipulación del brazo ya que no será necesario utilizar una computadora para poder utilizar el sistema.

3.1 Descripción del proyecto

Para el desarrollo de este proyecto se utilizarán varios módulos, cada uno con diferentes funciones y objetivos, estos módulos deberán integrarse a un solo sistema que nos permitirán alcanzar el correcto funcionamiento de la aplicación deseada

El diagrama 1 muestra la interacción entre los módulos que manipularán el brazo robótico.

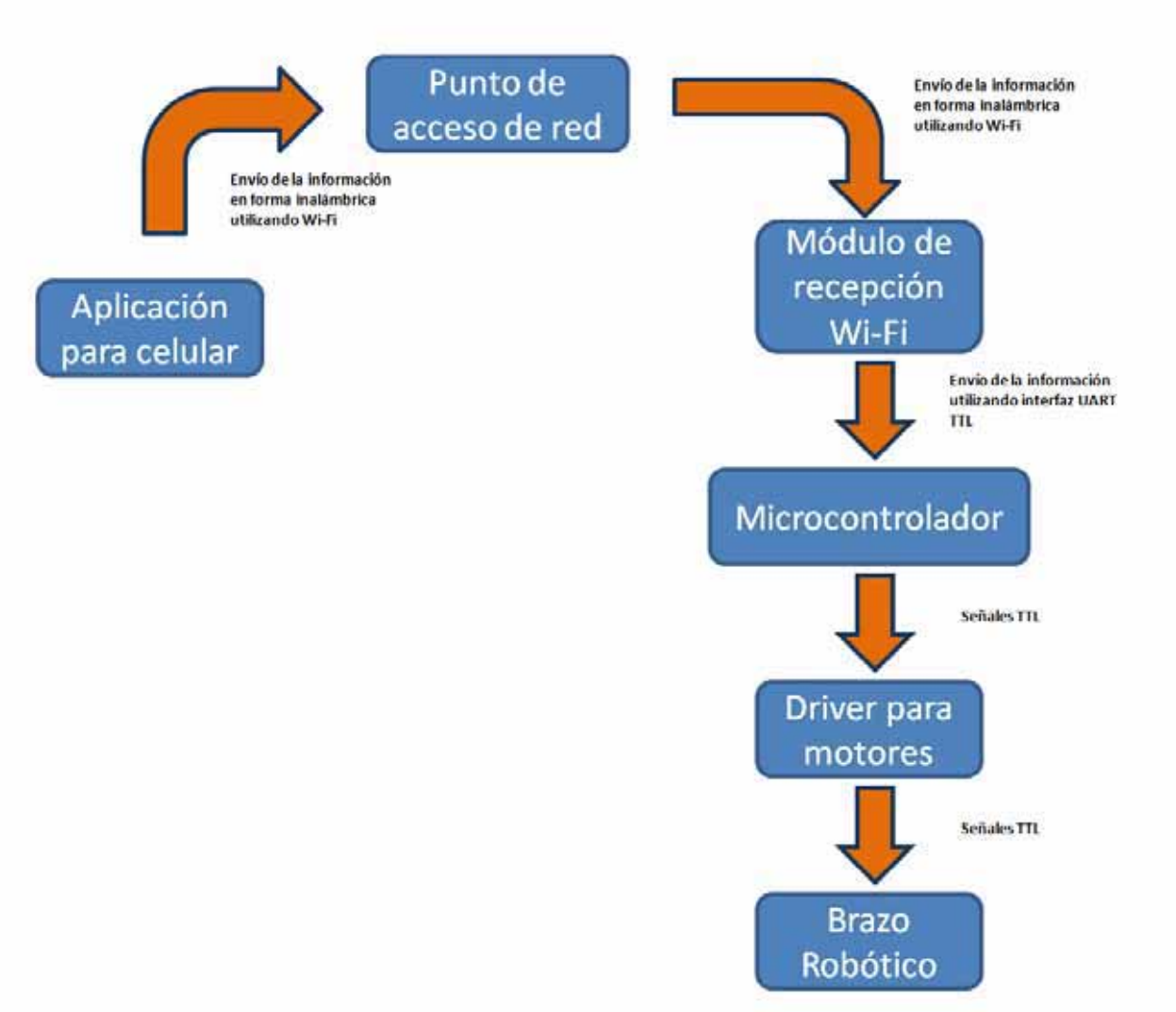

Diagrama 2. Proceso de manipulación del brazo

4. Objetivos

Objetivo general: Diseñar un sistema electrónico que permita la manipulación de un brazo robótico a través de un teléfono celular por medio de una conexión inalámbrica.

Objetivos específicos

- Crear un sistema electrónico basado en un microcontrolador1 que permita el manejo y control de un brazo robótico.
- Desarrollar una aplicación para un teléfono celular que funcione como interfaz gráfica para el control del brazo robótico.
- Crear un programa para el microcontrolador que manipulará el brazo.

5. Marco teórico

Como parte del desarrollo y documentación del presente trabajo, a continuación se presenta información donde se tienen algunos conceptos y definiciones técnicas que permitirán conocer más a fondo el proyecto que se presenta.

5.1 MICROCONTROLADOR AT89C51

Un microcontrolador es un circuito integrado programable, que es capaz de ejecutar las instrucciones grabadas en su memoria. Así mismo es el dispositivo encargado de realizar el procesamiento de datos a través de sus puertos de entrada/salida.

En el caso de este proyecto el microcontrolador AT89C51 almacenará el programa principal que contendrá las instrucciones de mando para el funcionamiento del brazo robótico.

Figura 2. Microcontrolador AT89C51

5.2 CONEXIÓN INALÁMBRICA

En la comunicación inalámbrica existen diferentes tecnologías, dentro de todas ellas se escogió la comunicación Wi-Fi. A continuación se presenta la información de los medios inalámbricos con los que se trabaja en el proyecto.

Las redes inalámbricas se pueden clasificar por su alcance; en el primer orden se encuentran las redes WPAN (Wireless Personal Area Network, por sus siglas en inglés)

que tienen un alcance de hasta 10 m generalmente. Dentro de esta primera clasificación se encuentra la tecnología Bluetooth. El Bluetooth trabaja con radiofrecuencia en la banda de los 2,4 GHz.

Una opción diferente es la tecnología Wi-Fi que se basa en un estándar de radiofrecuencia en la banda de los 2.4 GHz.

El Wi-Fi que es una de las tecnologías de comunicación inalámbrica mediante ondas más utilizada hoy en día, también llamada WLAN (Wireless Lan, red inalámbrica) o estándar IEEE 802.11. Puede ofrecer desde 11 Mbit/s hasta 54 Mbit/s, y sus distintas aplicaciones dependiendo del estándar que se utilice. El Wi-Fi tiene un alcance de 100 hasta 150 m por lo que tiene una perfecta funcionalidad con este proyecto.

5.3 PUNTO DE ACCESO A LA RED

Un punto de acceso inalámbrico, es un dispositivo que interconecta dispositivos alámbricos para formar una red inalámbrica

Diagrama 3. Esquema de un punto de acceso

5.4 MÓDULO Wi-Fly

Se estableció como medio de comunicación la vía inalámbrica, utilizando el estándar 802.11 g/b (Wi-Fi). Esto se logró gracias al módulo Wi-Fly el cual utiliza el estándar mencionado para la comunicación con los dispositivos.

El módulo de RN-XV fabricado por Roving Networks™ es un dispositivo con certificación Wi-Fi, diseñado especialmente para clientes que desean migrar su actual arquitectura 802.15.4(estándar de redes inalámbricas de área personal con tasas bajas de transmisión de datos) a una plataforma estándar basada en el protocolo TCP / IP sin tener que rediseñar el hardware existente.

El módulo de RN-XV se basa en el módulo Wi-Fi RN-171 e incorpora radio 802.11 b/g y 32 bits, la pila TCP / IP, reloj en tiempo real, acelerador criptográfico, unidad de administración de energía y sensores analógicos. En la configuración más simple, el hardware sólo requiere de cuatro conexiones (PWR, TX, RX y GND) para crear una conexión de datos inalámbrica.

Características:

- Consumo de energía ultra bajo: el modo de suspensión 4uA, para recepción 40mA, y para transmisión 180 mA en 10dBm.
- Potencia de transmisión: 0 dBm a 12dBm.
- Interfaces de hardware: TTL UART.
- 8 pines digitales de E/S de propósito general.
- 3 entradas de sensor analógico.
- Fuente de alimentación regulada 3.3VDC
- Dimensiones: 3.5cm X 2.5 cm

Figura 3. RN-XV Wi-Fly Module

5.5 MÓDULO Xbee Explorer Regulated

El Xbee Explorer Rugulated se encarga de la regulación a 3.3V, el acondicionamiento de señales y los indicadores de actividad básicas (alimentación, RSSI y actividad en los datos de entrada DIN y salida DOUT). Convierte las señales seriales de 5V a 3.3V para poder conectar sistemas de 5V a cualquier módulo del tipo Wi-Fly. La tarjeta se encuentra diseñada convenientemente para acoplarse perfectamente al módulo Wi-Fly. Al conectar esta tarjeta con el módulo Wi-Fly se podrá alimentar el sistema con 5V.

Figura 4. Xbee Explorer Regulated

5.6 MÓDULO Xbee Explorer USB

Unidad USB a serial que nos sirve para conectar el Wi-Fly. Esta unidad funciona con todos los módulos Wi-Fly. Solo se conecta la unidad Xbee Explorer USB, se inserta un cable mini USB y se tiene acceso directo a los pins seriales y de programación de la unidad Wi-Fly.

Figura 5. Xbee Explorer USB

5.7 BRAZO ROBÓTICO

El brazo que se utilizará será el modelo K-682 de Steren™, el cual tiene las siguientes especificaciones:

- Abertura máxima de la tenaza: 1,77 pulgadas (a)
- Movimiento vertical de la tenaza: 120° (b)
- Movimiento vertical de la parte superior del brazo: 120° (c)
- Movimiento vertical de la parte inferior del brazo: 180° (d)
- Movimiento horizontal de la base del brazo: 270° (e)
- Alimentación: 6 Vcc

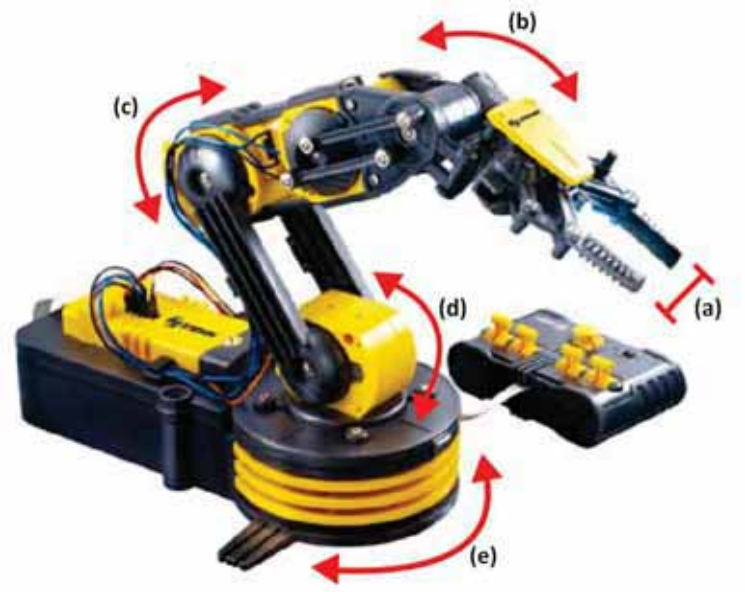

Figura 6. Brazo robótico modelo K-682

6. Desarrollo del proyecto

Para la realización del proyecto se ha seguido un desarrollo interactivo y creciente. Las etapas se han diferenciado y separado en análisis, desarrollo y resultados.

6.1 PRIMERA FASE (ANÁLISIS)

Consistió en una fase de toma de contacto con el entorno y el hardware inicialmente mostrado. Esta consistió básicamente en los siguientes puntos:

6.1.1 Instalación del software para configuración del Wi-Fly

Para poder configurar y analizar el Wi-Fly es necesario un emulador de terminal, la empresa que fabrica el módulo recomienda el uso de *TeraTerm* como emulador, pero cualquier otro software parecido funciona de la misma forma, en este caso se utilizó el software *Putty*, que es de distribución libre y que funciona para poder hacer conexiones seriales, vía Telnet o SHH.

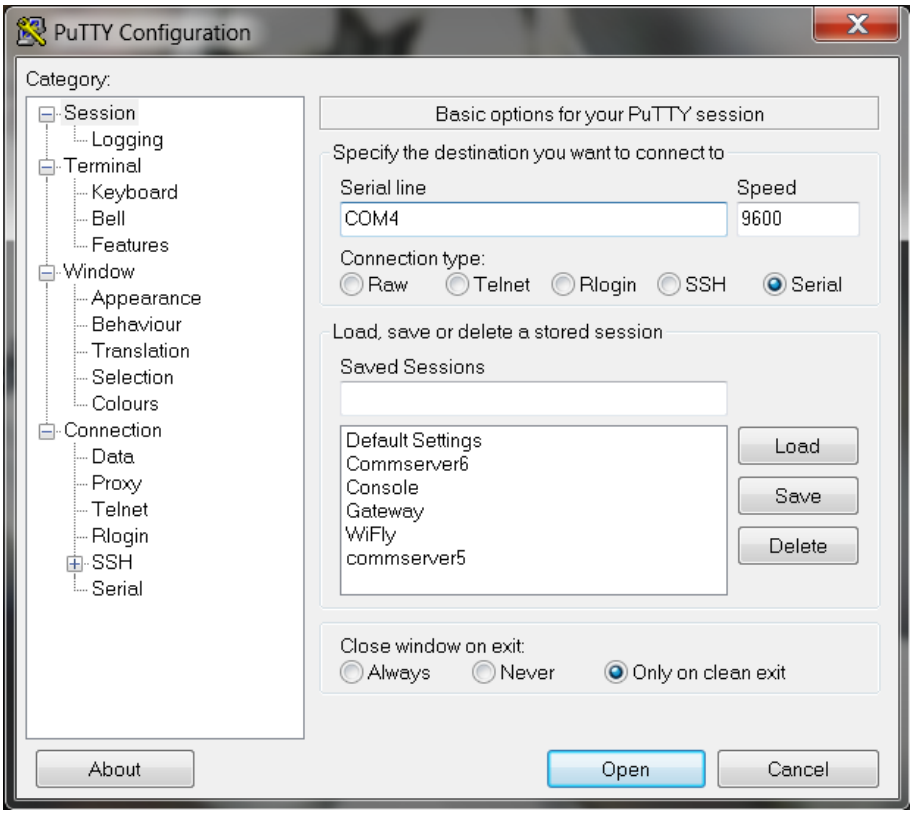

Figura 7. Pantalla inical del software Putty

6.1.2 Conexión y configuración del Wi-Fly

Existen dos formas para poder configurar el módulo Wi-Fly, una es de forma remota, que es utilizando el modo ADHOC, y la otra es por medio del Xbee Explorer USB. Se describen a continuación las dos formas de configuración, aunque en el desarrollo de este proyecto la configuración se realizó utilizando el Xbee Explorer USB.

6.1.2.1 Configuración en modo ADHOC

En el modo ADHOC se elimina la necesidad de asociar el Wi-Fly a un punto de acceso, ya que se crea una conexión punto a punto entre la PC y el Wi-Fly. Para esto es necesario primero habilitar el modo ADHOC en el Wi-Fly, esto se logra vía hardware, conectando la entrada PIO9 a Vcc (3.3V)

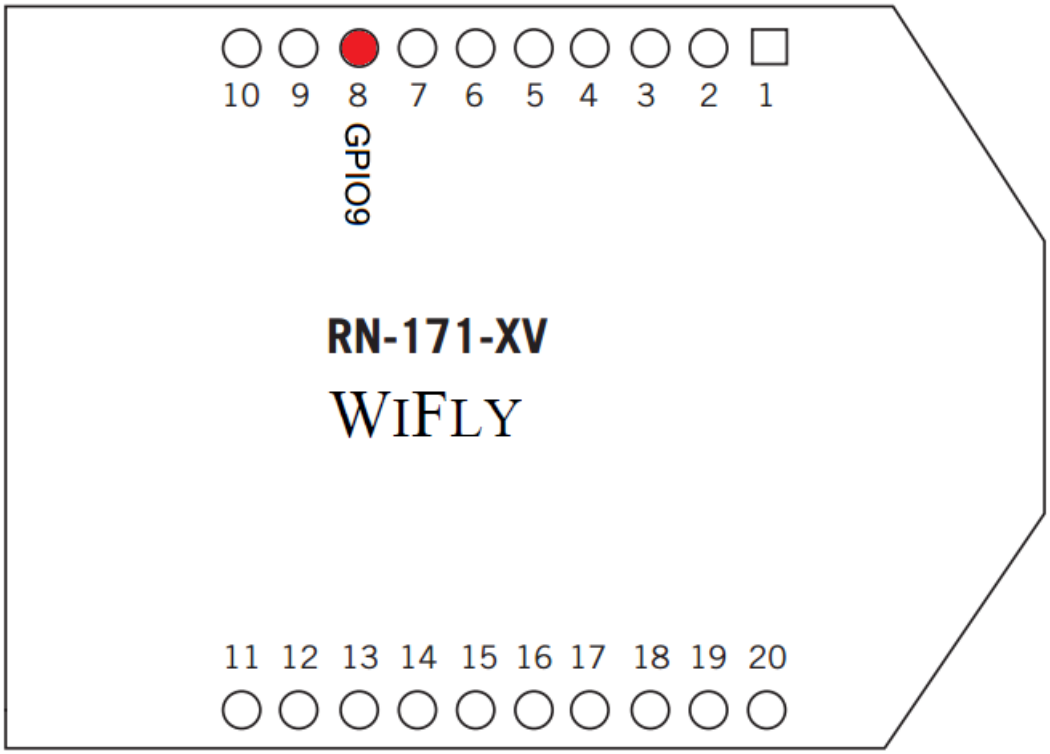

Figura 8. Vista superior del Wi-Fly

Una vez que habilitamos el modo ADHOC, el Wi-Fly tomará los siguientes valores:

SSID: WiFly-GSX-XX, donde XX son los dos bytes finales de la dirección MAC del dispositivo
Channel: 1
DHCP: OFF
IP address: 169.254.1.1
Netmask: 255.255.0.0

De esta forma ya solo es necesario hacer una conexión vía telnet hacia la dirección IP 169.254.1.1 en el puerto 2000.

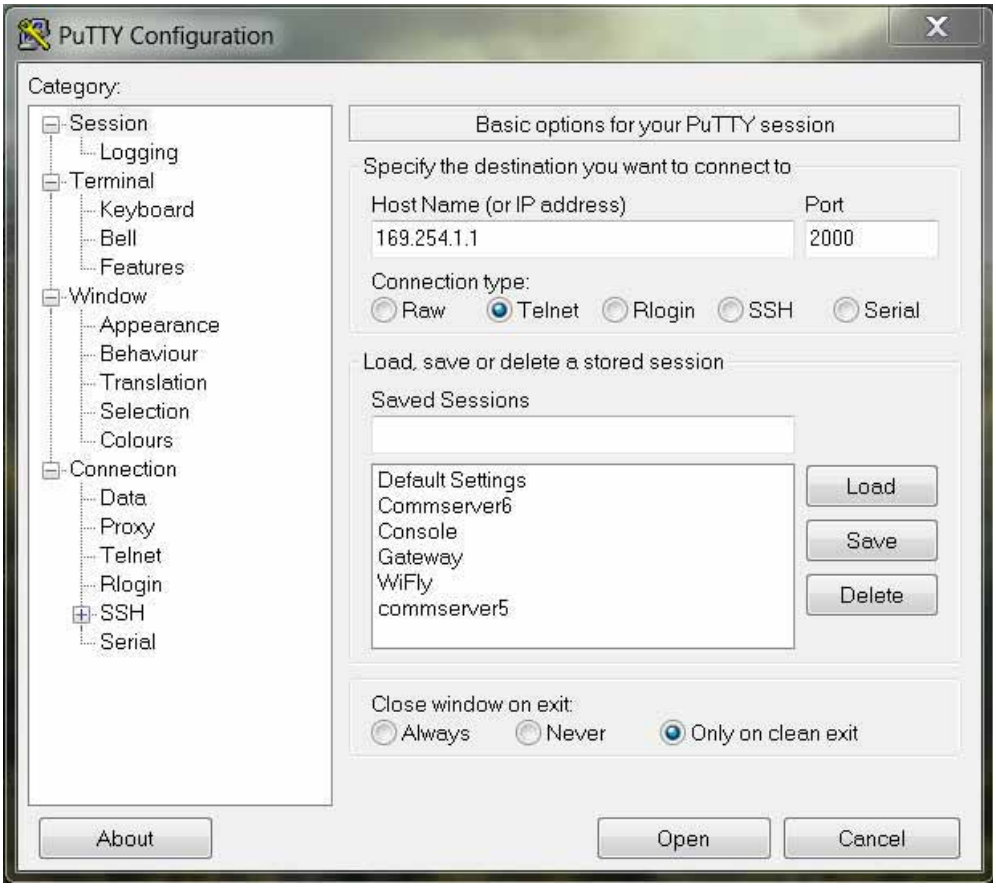

Figura 9. Conexión vía Telnet a través del Putty

Una vez hecha la conexión el Wi-Fly responderá con un
HELLO

Figura 10. Respuesta del Wi-Fly a una conexión ADHOC

Una vez que el dispositivo contesta con la cadena ***HELLO*,** se puede entrar en el modo de configuración escribiendo la secuencia **$$$**.

6.1.2.2 Configuración con el módulo Xbee Explorer USB

Una forma más sencilla de poder configurar el módulo Wi-.Fly, es utilizando el módulo Xbee Explorer USB, de ésta forma ya sólo es necesario montar el Wi-Fly sobre el Explorer USB y conectarlo directamente en la PC.

Figura 11. Conexión del Wi-Fly a una PC usando el Xbee Explorer USB

Una vez conectado a la PC, solo abrimos el puerto COM al cual está conectado y escribimos la secuencia **$$$** para activar el modo de configuración.

Figura 12. Pantalla en modo comando

6.1.2.3 Conexión física del Wi-Fly

Para poder trabajar con el Wi-Fly es necesario montarlo en la placa Xbee Explorer Regulated, esto para que el sistema completo pueda ser alimentado con 5V, el diagrama de conexión del Xbee Explorer se muestra a continuación:

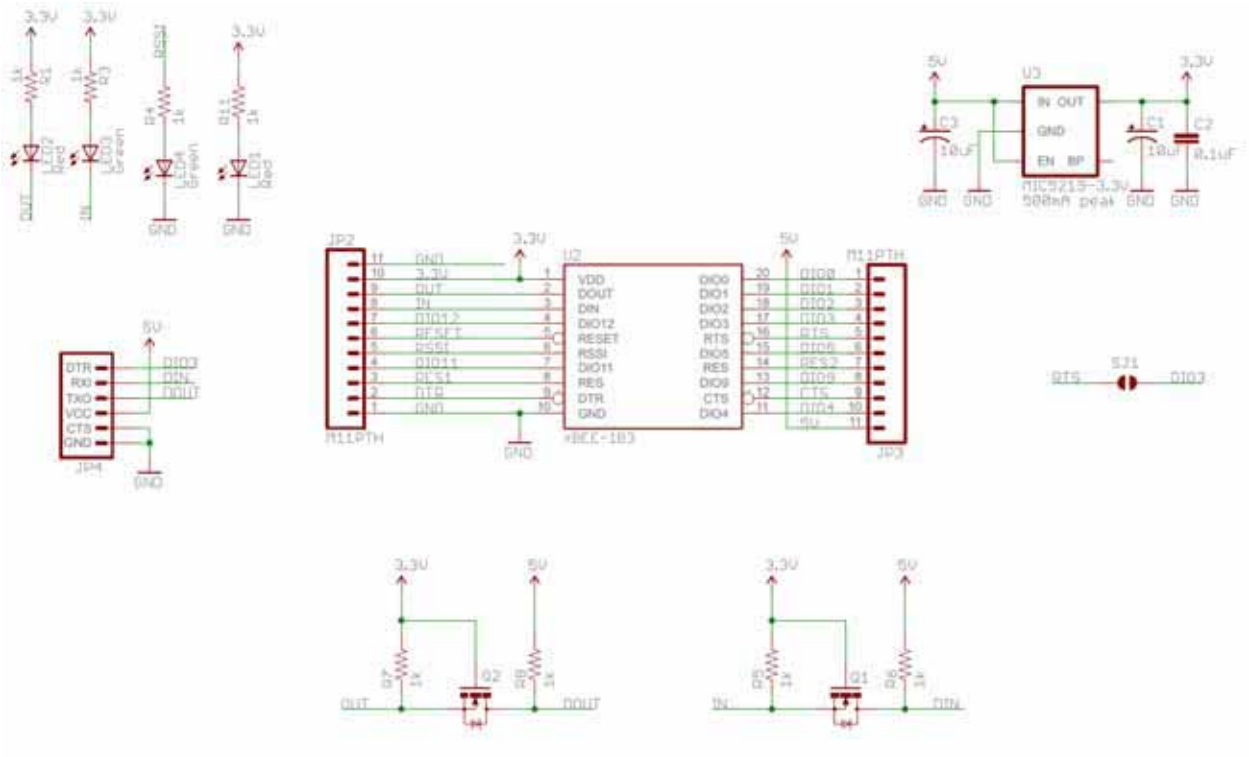

Figura 13. Diagrama eléctrico del Xbee Explorer Regulated

Una vez montado el Wi-Fly sobre el Xbee Explorer Regulated, solamente es necesario alimentarlo y ponerle una conexión a tierra en los pines 1 y 10 respectivamente.

6.1.3 Análisis del IDE de desarrollo para la aplicación

Para el desarrollo de la aplicación para teléfono celular se utilizó el IDE de programación Eclipse, el cual es una herramienta de programación de código abierto.

Para poder trabajar con aplicaciones Android, es necesario instalar el plug-in Android Development Tools.

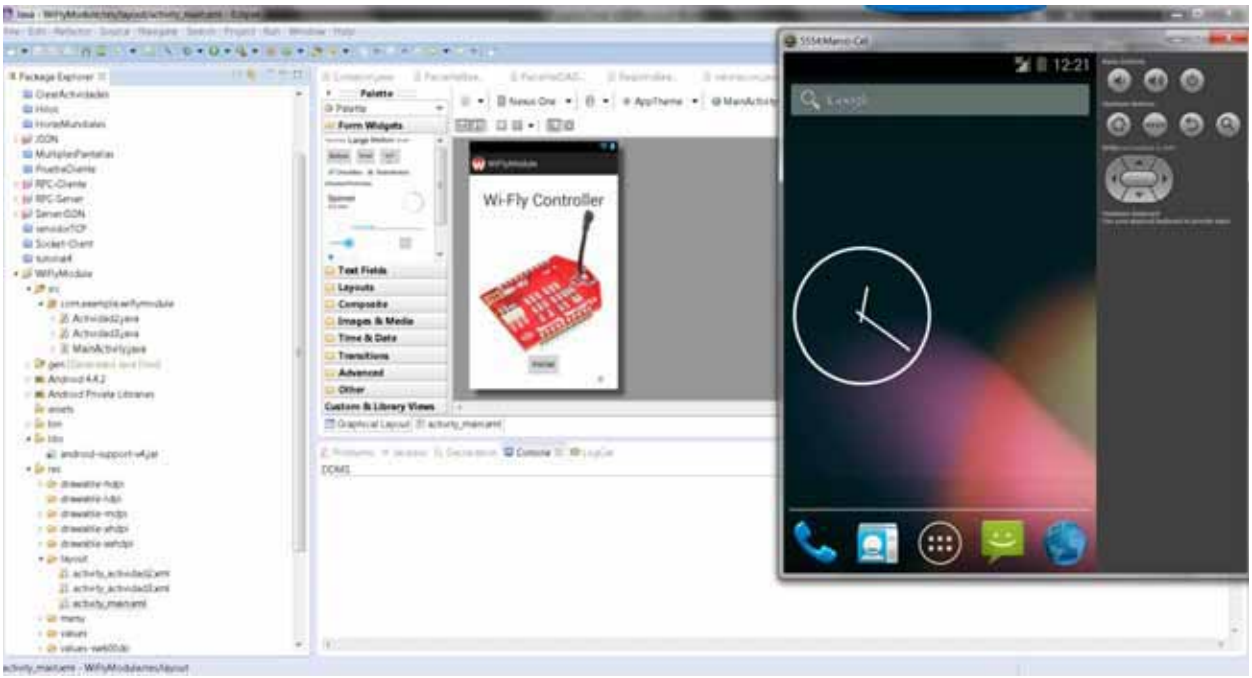

Figura 14. Entorno de desarrollo eclipse

6.1.4 Software para programación del microcontrolador AT89C51

El microcontrolador AT89C51se programa en lenguaje ensamblador, por lo que el software utilizado será el ENSAMBLADOR MCS-51 Versión 2004.

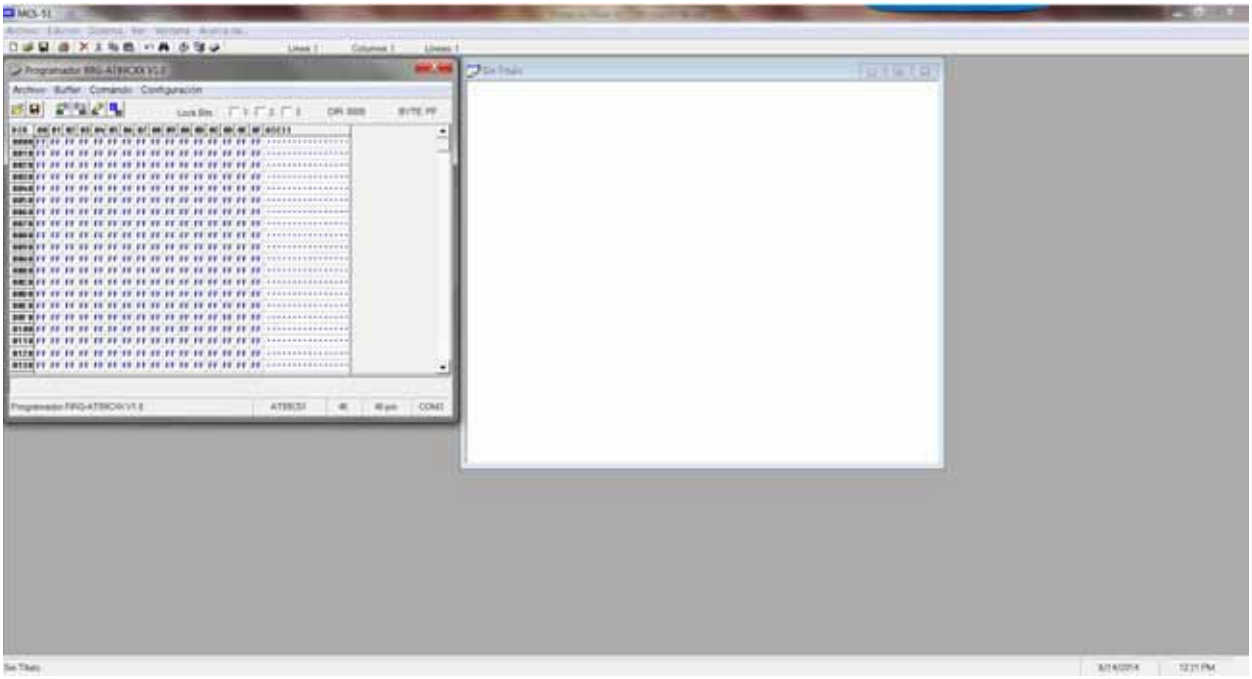

Figura 15. Pantalla principal del MCS-51

6.2 SEGUNDA FASE (DESARROLLO)

Para un mejor desempeño en esta segunda fase del proyecto, se decidió dividir el trabajo en tres partes principales:

1. Desarrollo de la aplicación.
2. Conexión a un punto de acceso y configuración del Wi-Fly como cliente TCP.
3. Hardware necesario para la manipulación del brazo.
4. Programación del microcontrolador.

6.2.1 Desarrollo de la aplicación

Para la programación de esta aplicación se siguieron los siguientes pasos:

- Lógica de la programación.
- Diagrama de flujo del programa
- Desarrollo del programa

6.2.1.1 Lógica de programación

En el desarrollo de aplicaciones Android, es muy común el uso de diferentes pantallas (en ocasiones llamadas actividades), esto nos permite navegar a través de la aplicación de una forma dinámica sin tener toda la información en una sola ventana. En el caso de nuestra aplicación, se utilizan tres pantallas o actividades diferentes.

En la primera actividad o pantalla principal, solamente se debe ver la presentación de la aplicación, y un botón que nos llevará a la segunda pantalla.

Figura 16. Pantalla principal de la aplicación

La segunda pantalla es donde empezará la interacción con el usuario, aquí solamente se desplegarán dos opciones de introducir texto, las cuales serán introducir la IP, y el número de puerto al que se quiere conectar el dispositivo.

También se debe mostrar un botón que haga la conexión, así como también el cambio de actividad, esto nos llevará a la tercera pantalla.

Figura 17. Segunda pantalla de la aplicación

Por último, la tercera pantalla mostrará los controles con los cuales manipulará el brazo robótico, de la siguiente forma:

Figura 18. Tercera pantalla de la aplicación

6.2.1.2 Diagrama de flujo

A continuación se muestra el diseño del programa y la lógica que este sigue a través de un diagrama de flujo.

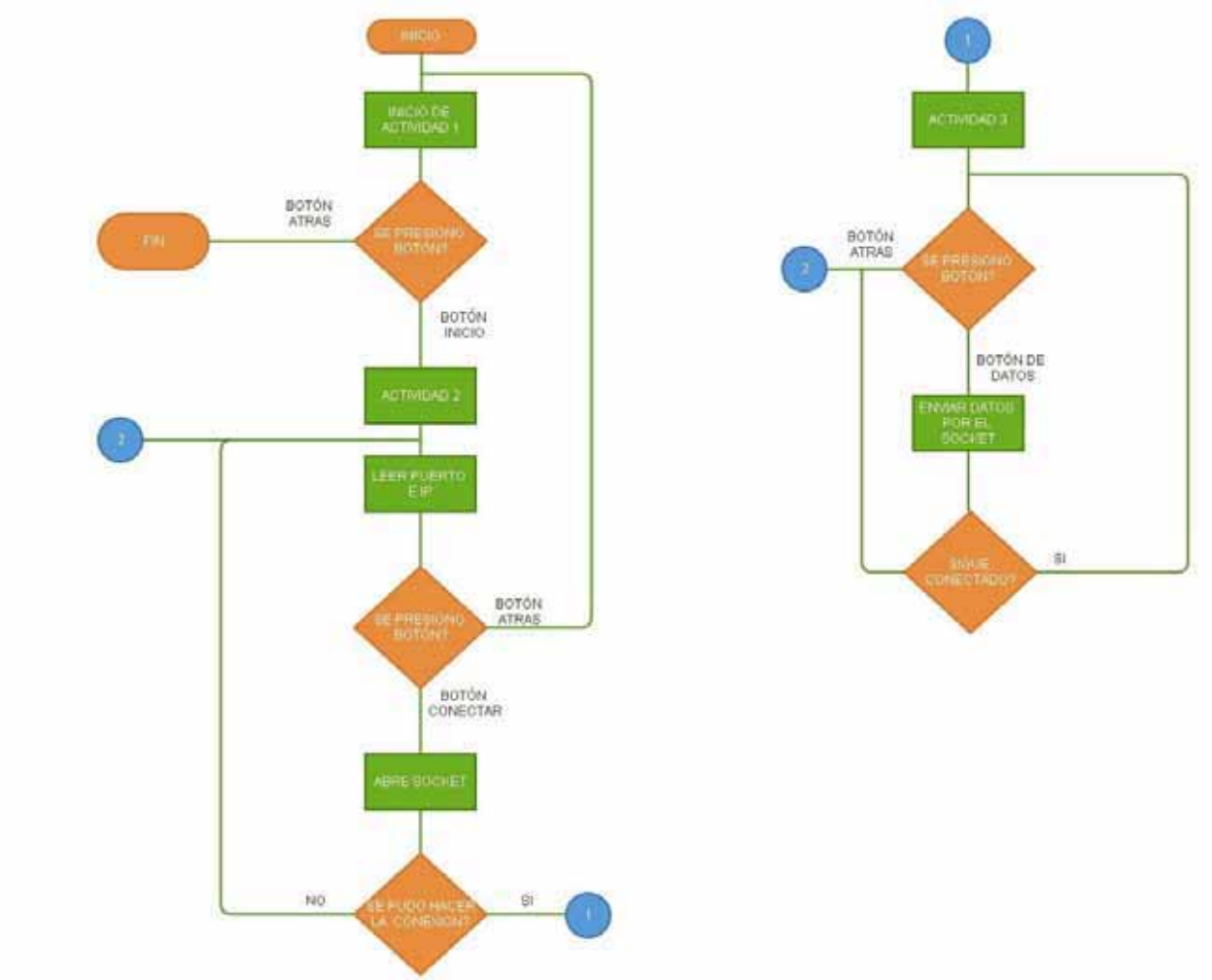

Diagrama 4. Diagrama de flujo para programar la aplicación

6.2.1.3 Desarrollo del programa

Al realizar el código del programa en ambiente de desarrollo eclipse, se puede notar que al crear un nuevo proyecto para una aplicación Android genera diferentes archivos que es necesario tener para que el proyecto pueda compilarse y ejecutarse, todos estos archivos se generan de forma automática.

En la siguiente imagen se puede ver el árbol de directorios que se genera al crear un nuevo proyecto.

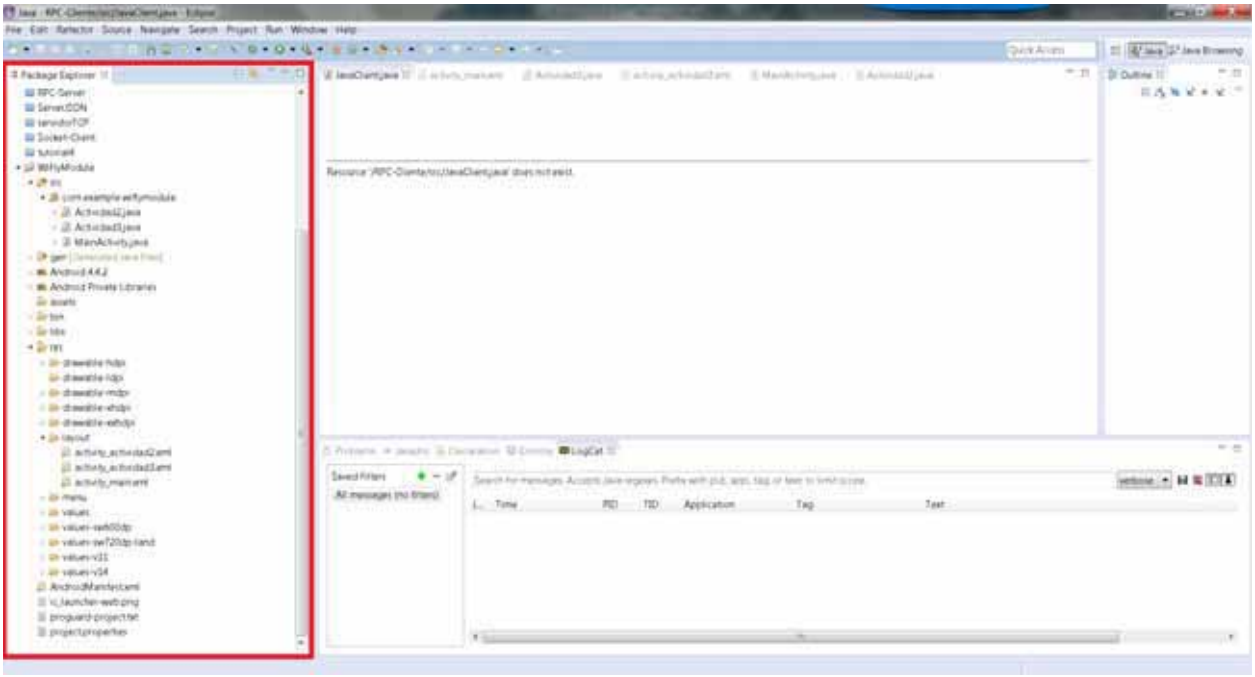

Figura 19. Árbol de directorios creados en un nuevo proyecto

Para efectos del desarrollo de este proyecto solo nos interesan dos de los directorios creados en el proyecto:

- src
- res

En el directorio src es en donde vamos a generar nuestras actividades, es decir dónde vamos poner todo nuestro código Java para el manejo de las pantallas, por lo cual debemos tener tres archivos con extensión .java ya que, de acuerdo al diseño, tendremos tres pantallas.

Es importante especificar que para el paso de datos entre el teléfono celular y el módulo Wi-Fly se utilizaron sockets TCP, en la modalidad cliente y servidor, en donde el cliente se ejecutará en el celular y el servidor en el módulo Wi-Fly.

Actividad 1

```
package com.example.wiflymodule;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;

public class MainActivity extends Activity {                /*Se crea por defecto*/

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void pasarActividad2(View v) {                   /*Método para moverse entre pantallas*/
        Intent act = new Intent(this, Actividad2.class);

         startActivity(act);
    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {         /*Se crea por defecto*/
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

La mayoría de este código fue generada en automático por el IDE de desarrollo, a excepción de las siguientes líneas:

```
import android.app.Activity;
import android.content.Intent;
```

Estas líneas importan las bibliotecas necesarias para el manejo de actividades o pantallas en la aplicación.

Además se agregó el siguiente método:

```
public void pasarActividad2(View v) {
    Intent act = new Intent(this, Actividad2.class);

     startActivity(act);
}
```

Este método se utiliza para pasar de la primera pantalla a la segunda, esto se logra de la siguiente forma:

1. Se crea un objeto “act2 de tipo “Intent” el cual guardará la referencia hacia la nueva actividad a la cual nos queremos mover.
2. Como parámetro le pasamos la activad a la que nos queremos mover, en este caso “Actividad2”

3. Al final se le indica que se inicie la actividad referenciada por el objeto “act”.

Actividad 2.

```
package com.example.wiflymodule;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class Actividad2 extends Activity {

    private EditText ipinput, portinput;
    private Button btconect;
    String  info1, info2;

    final static String INTENT_INFO = "com.example.wiflymodule.Actividad3";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_actividad2);
        ipinput = (EditText) findViewById(R.id.editText1);
        portinput = (EditText) findViewById(R.id.editText2);
        btconect = (Button) findViewById(R.id.button1);


        btconect.setOnClickListener(new OnClickListener() {
            @Override
            // conectar
            public void onClick(View v) {
                    info1 = ipinput.getText().toString();
                    Intent intent = new Intent();
                    intent.setClass(Actividad2.this, Actividad3.class);
                    intent.putExtra(INTENT_INFO, info1);

                    info2 = portinput.getText().toString();
                    Intent intent2 = new Intent();
                    intent.setClass(Actividad2.this, Actividad3.class);
                    intent2.putExtra(INTENT_INFO, info2);
                    startActivity(intent);
                    }
        });
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.actividad2, menu);
        return true;
    }

}
```

En ésta actividad, las líneas que se agregan son las siguientes:

```
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
```

Las cuales nos sirven para el manejo de botones y cajas de texto en la pantalla.

Se declaran las variables que se utilizarán en esta actividad:

```
private EditText ipinput, portinput;
private Button btconect;
String  info1, info2;
```

Las variables “ipinput” y “portinput” se utilizan para el manejo de las cadenas que el usuario va a escribir en las cajas de texto en la pantalla, y almacenarán el valor de la IP¨y el puerto a los que se quiere conectar la aplicación.

La variable “btconnect”, sirve para manejar la acción del botón “conectar”.

Las variables “info1” e “info2”, son las que nos van a permitir el paso de los valores de la IP y el puerto hacia una tercera actividad.

Las siguientes líneas nos permiten asignar los valores que escribe el usuario a las variables creadas:

```
ipinput = (EditText) findViewById(R.id.editText1);
portinput = (EditText) findViewById(R.id.editText2);
btconect = (Button) findViewById(R.id.button1);
```

La línea:

```
btconect.setOnClickListener(new OnClickListener() {
```

Es el manejador del click del botón, lo que haya dentro del cuerpo de éste manejador se ejecutará cuando se presione el botón identificado por “btnconect”.

Por último las líneas:

```
info1 = ipinput.getText().toString();
Intent intent = new Intent();
intent.setClass(Actividad2.this, Actividad3.class);
intent.putExtra(INTENT_INFO, info1);
```

Son las que hacen el paso de la información de la IP y el puerto hacia la actividad 3, utilizando nuevamente un objeto de tipo intent.

Actividad 3

```
package com.example.wiflymodule;

import java.io.DataOutputStream;
import java.io.OutputStream;
import java.net.Socket;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class Actividad3 extends Activity {

    Socket miCliente;
    private Button botonA1, botonA2,botonA3,
                   botonB1, botonB2, botonB3,
                   botonC1, botonC2, botonC3,
                   botonD1, botonD2, botonD3,
                   botonE1, botonE2, botonE3,
                   botonLedON, botonLedOFF, botonStop;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_actividad3);
        botonA1 = (Button) findViewById(R.id.button1);
        botonB1 = (Button) findViewById(R.id.button2);
        botonC1 = (Button) findViewById(R.id.button3);
        botonD1 = (Button) findViewById(R.id.button4);
        botonE1 = (Button) findViewById(R.id.button5);

        botonA2 = (Button) findViewById(R.id.button6);
        botonB2 = (Button) findViewById(R.id.Button7);
        botonC2 = (Button) findViewById(R.id.Button8);
        botonD2 = (Button) findViewById(R.id.Button9);
        botonE2 = (Button) findViewById(R.id.Button10);

        botonA3 = (Button) findViewById(R.id.button11);
        botonB3 = (Button) findViewById(R.id.button12);
        botonC3 = (Button) findViewById(R.id.button13);
        botonD3 = (Button) findViewById(R.id.button14);
        botonE3 = (Button) findViewById(R.id.button15);

        botonLedOFF = (Button) findViewById(R.id.button16);
        botonLedON = (Button) findViewById(R.id.button17);
        botonStop = (Button) findViewById(R.id.button18);

        botonA1.setOnClickListener(new OnClickListener(){
            public void onClick(View v){
                        try{
                             OutputStream aux = miCliente.getOutputStream();
                             DataOutputStream flujo= new DataOutputStream( aux );
                             flujo.writeUTF( "9" );

                            } catch( Exception e ) {
                                System.out.println( e.getMessage() );
                                }
            }
        });

        botonA2.setOnClickListener(new OnClickListener(){
            public void onClick(View v){
                        try{
                             OutputStream aux = miCliente.getOutputStream();
                             DataOutputStream flujo= new DataOutputStream( aux );
                             flujo.writeUTF( "0" );

                            } catch( Exception e ) {
                                System.out.println( e.getMessage() );
                                }
            }
        });
```

```
botonA3.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
                try{
                     OutputStream aux = miCliente.getOutputStream();
                     DataOutputStream flujo= new DataOutputStream( aux );
                     flujo.writeUTF( "e" );

                    } catch( Exception e ) {
                         System.out.println( e.getMessage() );
                         }
    }
});

botonB1.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
                try{
                     OutputStream aux = miCliente.getOutputStream();
                     DataOutputStream flujo= new DataOutputStream( aux );
                     flujo.writeUTF( "7" );

                    } catch( Exception e ) {
                         System.out.println( e.getMessage() );
                         }
    }
});

botonB2.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
                try{
                     OutputStream aux = miCliente.getOutputStream();
                     DataOutputStream flujo= new DataOutputStream( aux );
                     flujo.writeUTF( "8" );

                    } catch( Exception e ) {
                         System.out.println( e.getMessage() );
                         }
    }
});


botonB3.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
                try{
                     OutputStream aux = miCliente.getOutputStream();
                     DataOutputStream flujo= new DataOutputStream( aux );
                     flujo.writeUTF( "d" );

                    } catch( Exception e ) {
                         System.out.println( e.getMessage() );
                         }
    }
});

botonC1.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
                try{
                     OutputStream aux = miCliente.getOutputStream();
                     DataOutputStream flujo= new DataOutputStream( aux );
                     flujo.writeUTF( "5" );

                    } catch( Exception e ) {
                         System.out.println( e.getMessage() );
                         }
    }
});

botonC2.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
                try{
                     OutputStream aux = miCliente.getOutputStream();
                     DataOutputStream flujo= new DataOutputStream( aux );
                     flujo.writeUTF( "6" );

                    } catch( Exception e ) {
                         System.out.println( e.getMessage() );
                         }
    }
});
```

```
botonC3.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
                try{
                     OutputStream aux = miCliente.getOutputStream();
                     DataOutputStream flujo= new DataOutputStream( aux );
                     flujo.writeUTF( "c" );

                    } catch( Exception e ) {
                         System.out.println( e.getMessage() );
                         }
    }
});

botonD1.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
                try{
                     OutputStream aux = miCliente.getOutputStream();
                     DataOutputStream flujo= new DataOutputStream( aux );
                     flujo.writeUTF( "3" );

                    } catch( Exception e ) {
                         System.out.println( e.getMessage() );
                         }
    }
});

botonD2.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
                try{
                     OutputStream aux = miCliente.getOutputStream();
                     DataOutputStream flujo= new DataOutputStream( aux );
                     flujo.writeUTF( "4" );

                    } catch( Exception e ) {
                         System.out.println( e.getMessage() );
                         }
    }
});

botonD3.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
                try{
                     OutputStream aux = miCliente.getOutputStream();
                     DataOutputStream flujo= new DataOutputStream( aux );
                     flujo.writeUTF( "b" );

                    } catch( Exception e ) {
                         System.out.println( e.getMessage() );
                         }
    }
});

botonE1.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
                try{
                     OutputStream aux = miCliente.getOutputStream();
                     DataOutputStream flujo= new DataOutputStream( aux );
                     flujo.writeUTF( "1" );

                    } catch( Exception e ) {
                         System.out.println( e.getMessage() );
                         }
    }
});

botonE2.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
                try{
                     OutputStream aux = miCliente.getOutputStream();
                     DataOutputStream flujo= new DataOutputStream( aux );
                     flujo.writeUTF( "2" );

                    } catch( Exception e ) {
                         System.out.println( e.getMessage() );
                         }
    }
});
```

```
        botonE3.setOnClickListener(new OnClickListener(){
            public void onClick(View v){
                        try{
                             OutputStream aux = miCliente.getOutputStream();
                             DataOutputStream flujo= new DataOutputStream( aux );
                             flujo.writeUTF( "a" );

                            } catch( Exception e ) {
                                 System.out.println( e.getMessage() );
                                 }
            }
        });

        botonLedOFF.setOnClickListener(new OnClickListener(){
            public void onClick(View v){
                        try{
                             OutputStream aux = miCliente.getOutputStream();
                             DataOutputStream flujo= new DataOutputStream( aux );
                             flujo.writeUTF( "O" );

                            } catch( Exception e ) {
                                 System.out.println( e.getMessage() );
                                 }
            }
        });

        botonLedON.setOnClickListener(new OnClickListener(){
            public void onClick(View v){
                        try{
                             OutputStream aux = miCliente.getOutputStream();
                             DataOutputStream flujo= new DataOutputStream( aux );
                             flujo.writeUTF( "o" );

                            } catch( Exception e ) {
                                 System.out.println( e.getMessage() );
                                 }
            }
        });

        botonStop.setOnClickListener(new OnClickListener(){
            public void onClick(View v){
                        try{
                             OutputStream aux = miCliente.getOutputStream();
                             DataOutputStream flujo= new DataOutputStream( aux );
                             flujo.writeUTF( "s" );

                            } catch( Exception e ) {
                                 System.out.println( e.getMessage() );
                                 }
            }
        });

    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.actividad3, menu);
        return true;
    }

}
```

Para la actividad 3 hay algunas líneas de código más, sobre todo para el envío de la información hacia el dispositivo al cual se ha conectado la aplicación.

Lo primero son las bibliotecas que se tiene que añadir para que se puedan crear los sockets, estos sockets nos permitirán el envío de bytes desde la aplicación al Wi-Fly.

```
import java.io.DataOutputStream;
import java.io.OutputStream;
import java.net.Socket;
```

Una vez importadas las bibliotecas necesarias, vemos que se declaran las variables que se necesitan para el manejo de todos los botones, y después está el siguiente código:

```
botonA1.setOnClickListener(new OnClickListener(){
    public void onClick(View v){
                try{
                     OutputStream aux = miCliente.getOutputStream();
                     DataOutputStream flujo= new DataOutputStream( aux );
                     flujo.writeUTF( "9" );

                    } catch( Exception e ) {
                         System.out.println( e.getMessage() );
                         }
    }
});
```

En este caso es el código que se aplicará en caso de que se presione el botón identificado como “botonA1”, pero es prácticamente el mismo código para cada uno de los botones.

Lo primero es el manejador del click en el botón, como ya se había visto anteriormente. Después las líneas:

```
OutputStream aux = miCliente.getOutputStream();
DataOutputStream flujo= new DataOutputStream( aux );
flujo.writeUTF( "9" );
```

Nos sirven para convertir los datos en ASCII en bytes, para así pode renviarlos por el socket.

6.2.2 Conexión a un punto de acceso y configuración del Wi-Fly como servidor TCP

Asociar el Wi-Fly con un punto de acceso a la red es muy sencillo, solamente hay seguir los siguientes pasos:

1. Conectar el Wi-Fly a la Pc usando el Xbee Explorer USB.
2. Ejecutar el programa Putty
3. Seleccionar el puerto COM que se le asignó al Xbee Explorer USB
4. Abrir

5. Teclear la secuencia **$$$**, que nos permitirá entrar al Wi-Fly en modo de comando, esto lo podemos notar ya que el Wi-Fly responderá con la palabra ***CMD*.**
6. Asociamos el Wi-Fly al punto de acceso con el comando ***set wlan ssid <nombre>***, donde <nombre> es el nombre del punto de acceso al que nos queremos conectar, por ejemplo “Cisco18399”.
7. Para conectarnos al punto de acceso, hay que saber que tipo de seguridad tiene, ya que esto cambia un poco la configuración para poder establecer la conexión. De Igual forma el comando utilizado es *set* ***wlan auth <valor>,*** donde valor puede ser un número del 0 al 8, dependiendo del tipo de seguridad:

Value	Authentication Mode
0	Open (Default)
1	WEP-128
2	WPA1
3	Mixed WPA1 & WPA2-PSK
4	WPA2-PSK
5	Not Used
6	Adhoc, Join any Adhoc network
8	WPE-64

Tabla 1. Valores que puede tomar el comando wlan auth

8. Para este proyecto el punto de acceso que se usó (Cisco18399) tiene seguridad abierta, por lo que para conectarse no es necesario algún comando para guardar la contraseña, solamente es necesario el valor ***0*** en el comando ***set wlan auth***.
9. Ahora hay que dar el comando ***set ip dhcp 1***. Con este comando el Wi-Fly obtendrá su dirección IP por medio de DHCP.
10. Para que el Wi-Fly se conecte al punto de acceso en cuanto lo alimentemos, solo es necesario el comando ***set wlan join 1***.

De esta forma queda configurado el módulo Wi-Fly para unirse a un punto de acceso, se puede obtener toda esta información con el comando ***get wlan*** y ***get ip***.

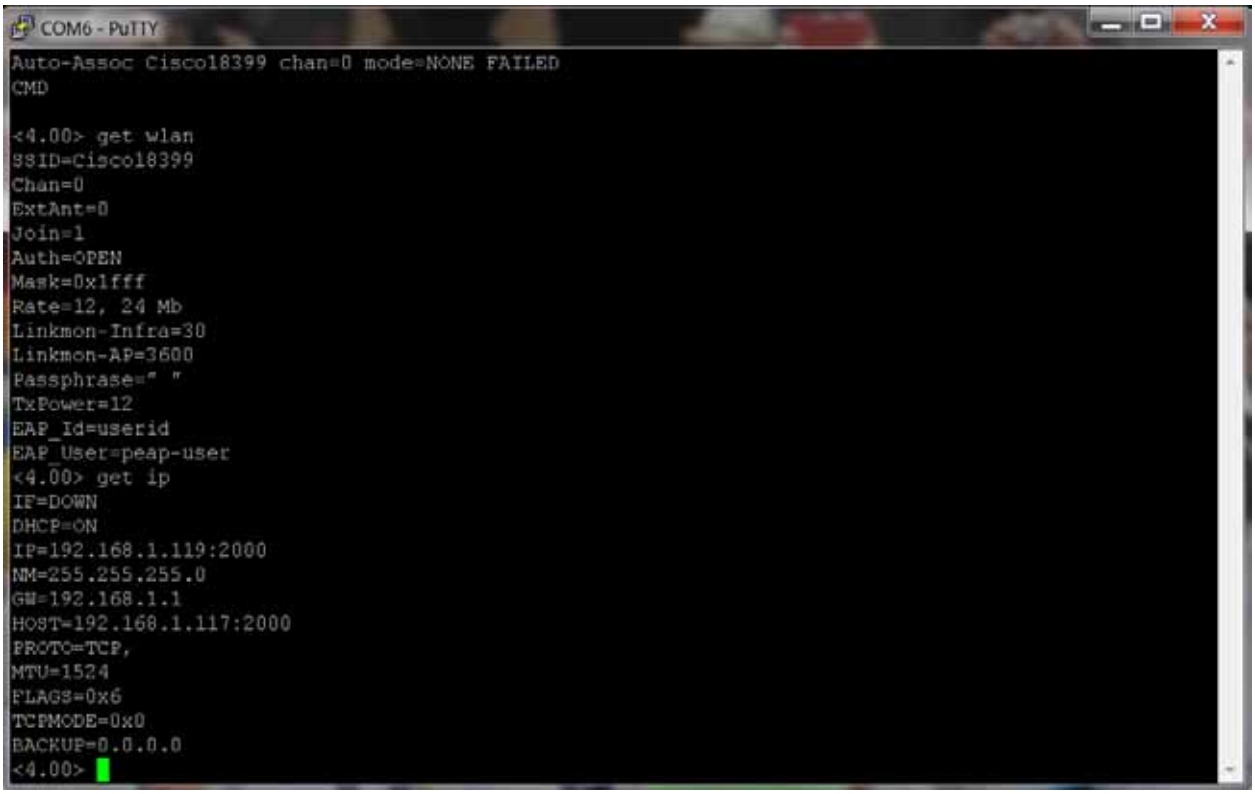

Figura 20. Pantalla del Putty después de usar los comandos get wlan y get ip

Para poder establecer una conexión entre el Wi-Fly y la aplicación para celular, el paso de los datos tendrá que ser a través de sockets TCP. Para el manejo de sockets es necesario crear un cliente y un servidor, para este proyecto se tiene pensado que el servidor será el módulo Wi-Fly y el cliente se ejecutará en el teléfono celular.

Por lo cual, lo siguiente es configurar el Wi-Fly en modo servidor TCP, esto se hace de la siguiente forma:

1. Conectar el módulo Wi-Fly a la PC usando el Xbee Explorer USB
2. Ejecutar el programa Putty
3. Escribir la secuencia $$$ para poder entrar en modo de comandos al Wi-Fly.
4. Una vez en modo CMD, introducimos el comando ***set ip protocol <valor>***, donde ***<valor>*** está determinado por la siguiente tabla:

Bit Position	Protocol
0	UDP
1	TCP Server & Client (Default)
2	Secure (only receive packets with IP address matches the store host IP)
3	TCP Client only
4	HTTP client mode

Tabla 2. Valores que puede tomar el comando set ip protocol

Por lo cual, si se quiere que el Wi-Fly esté en modo servidor, se deba activar el bit número 1, por lo que ***<valor>*** será igual a ***2***.

5. Para que el Wi-Fly esté recibiendo conexiones de clientes, una buena opción es usar el comando ***set sys autoconn <valor>***, donde en ***<valor>*** podemos poner un número entre 2-254, éste número permitirá al Wi-Fly abrir una conexión TCP en intervalos de cada ***<valor>*** segundos.
6. Con el comando ***set ip host <IP address>*** le decimos al Wi-Fly cuál será el dispositivo que va permitir que se conecte a través de la IP del cliente.
7. Por último con el comando ***set ip remote <port>*** seleccionamos el purto a través del cual se hará la conexión TCP.

Una vez hecha esta configuración, solamente se necesita alimentar el Wi-Fly para que se conecte al punto de acceso y comience a abrir la conexión TCP, quedando en espera de un cliente que se conecte.

Mientras no haya un cliente el Wi-Fly estará mandando mensajes de ***CLOSED***, indicando que intentó abrir una conexión TCP, pero que ningún cliente se conectó. En cuanto un cliente se conecte el Wi-Fly responderá con el mensaje ***OPEN***, indicando que se ha establecido la conexión, y en ese momento podrá recibir los datos que mandé la aplicación.

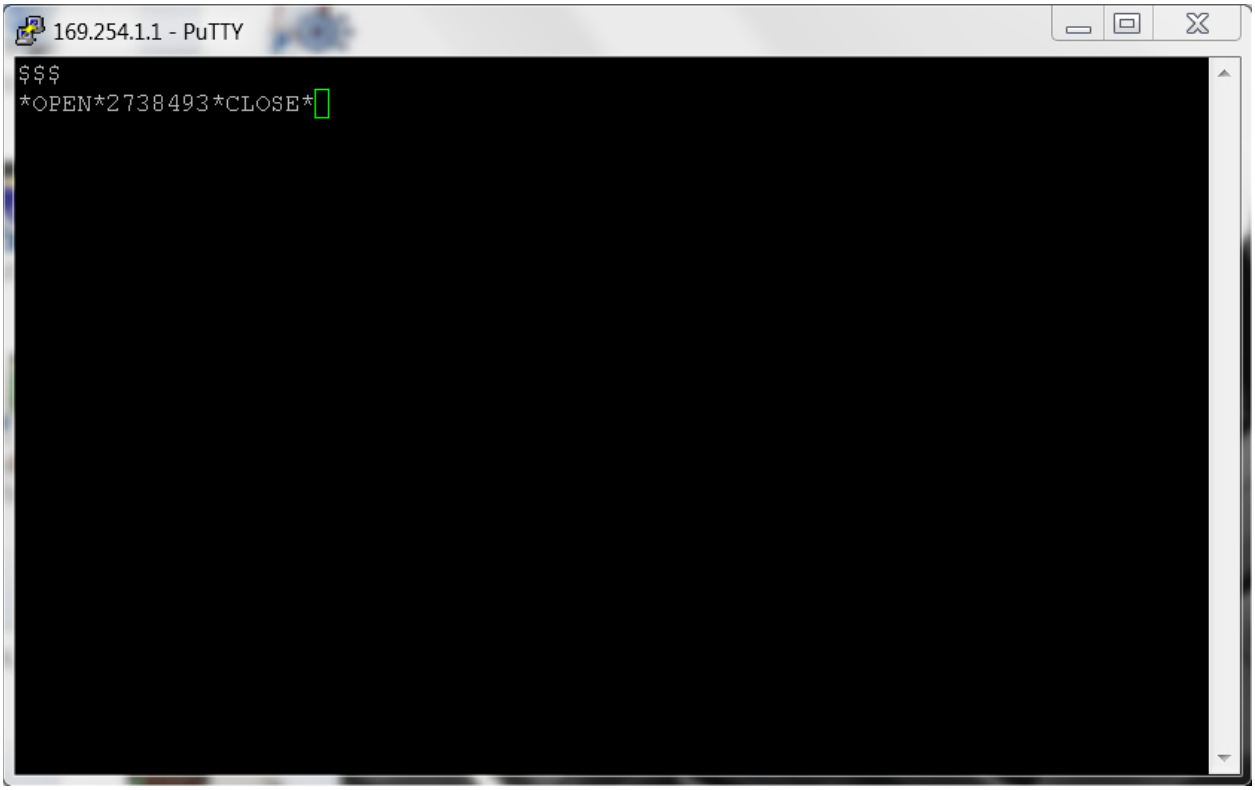

Figura 22. Conexión por medio de sockets

6.2.3 Hardware para la manipulación del brazo

Para poder manipular el brazo robótico fue necesario desarrollar el hardware necesario para el control de los motores así como para la comunicación con el módulo de recepción Wi-Fly.

La comunicación entre el Wi-Fly y el circuito que moverá el brazo se hizo mediante UART, como el Wi-Fly ya viene configurado de fábrica con un puerto UART ya no es necesario hacer alguna configuración extra.

Como el objetivo de este proyecto no es el diseño del circuito los detalles del mismo no son necesarios, solamente se muestran los diagramas básicos del diseño así como el circuito terminado.

El diseño del ciruito es el siguiente:

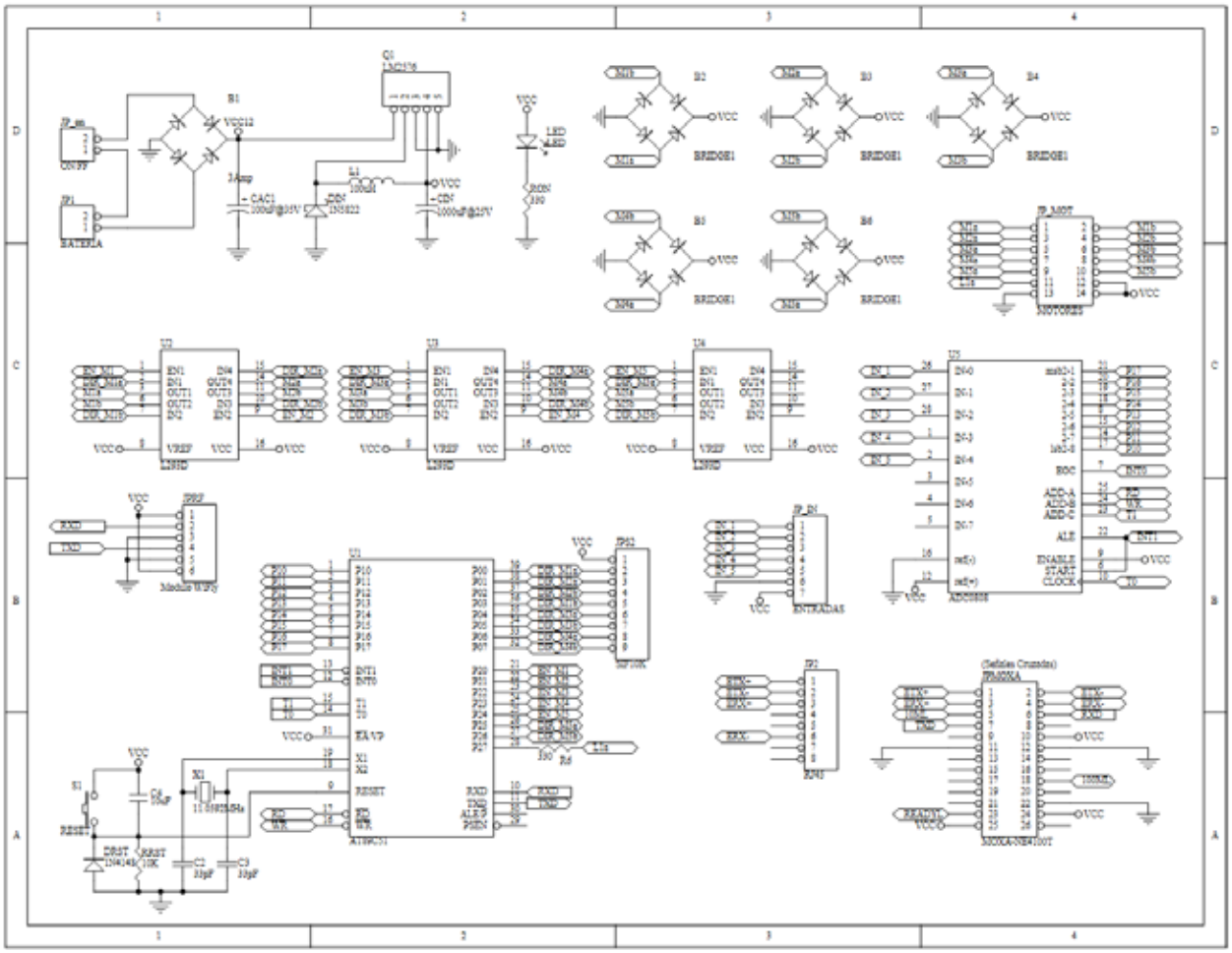

Figura 23. Diseño del circuito para la manipulación del brazo

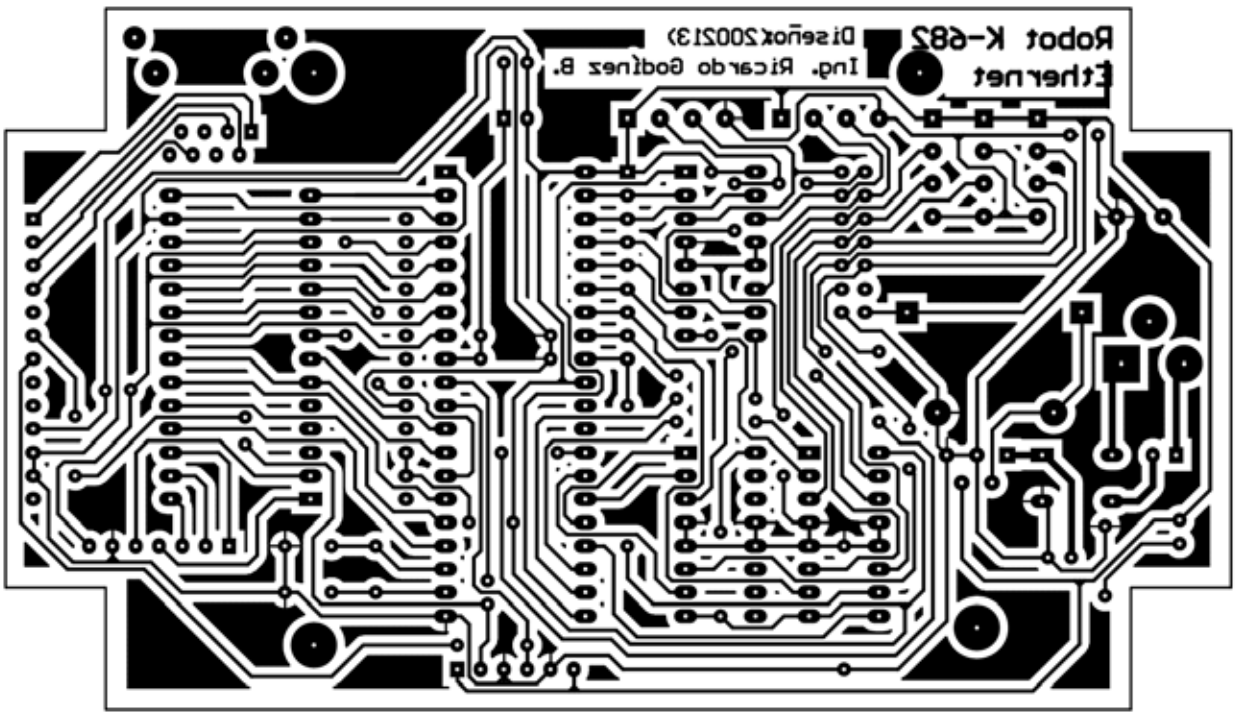

Figura 24. Diagrama PCB del circuito

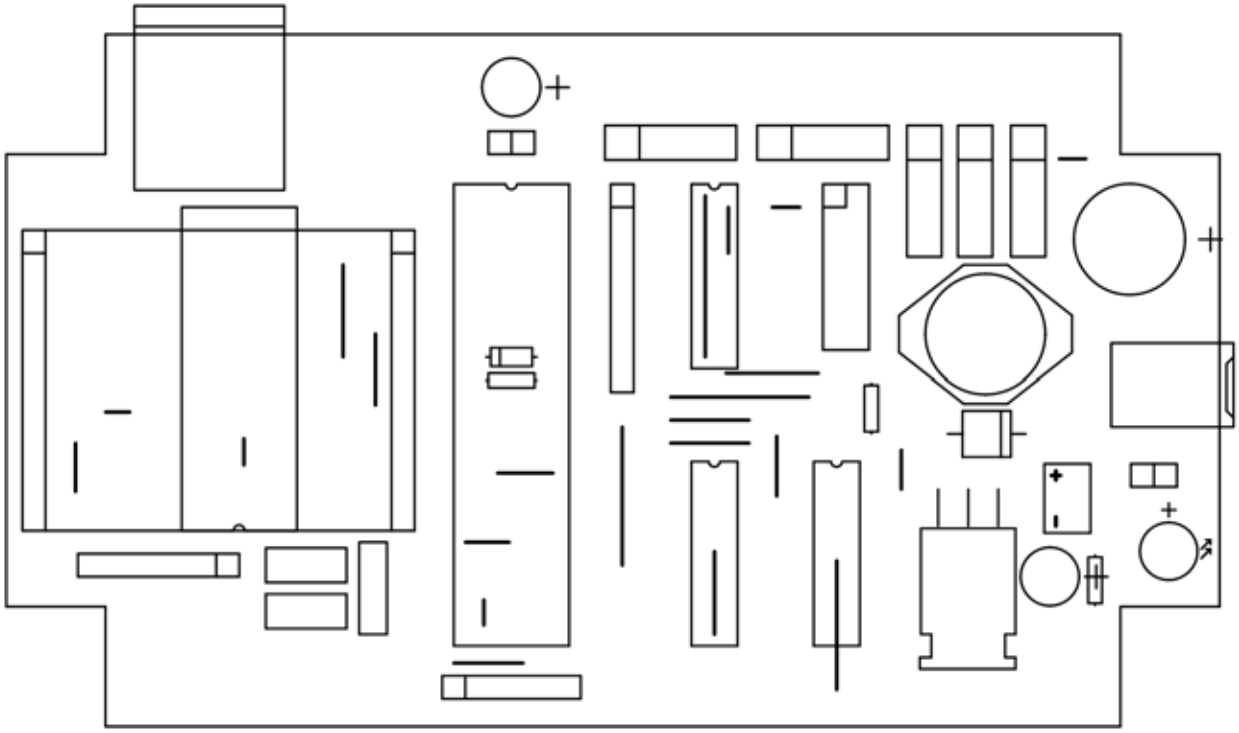

Figura 25. Máscara de componentes del circuito

Figura 26. Circuito terminado

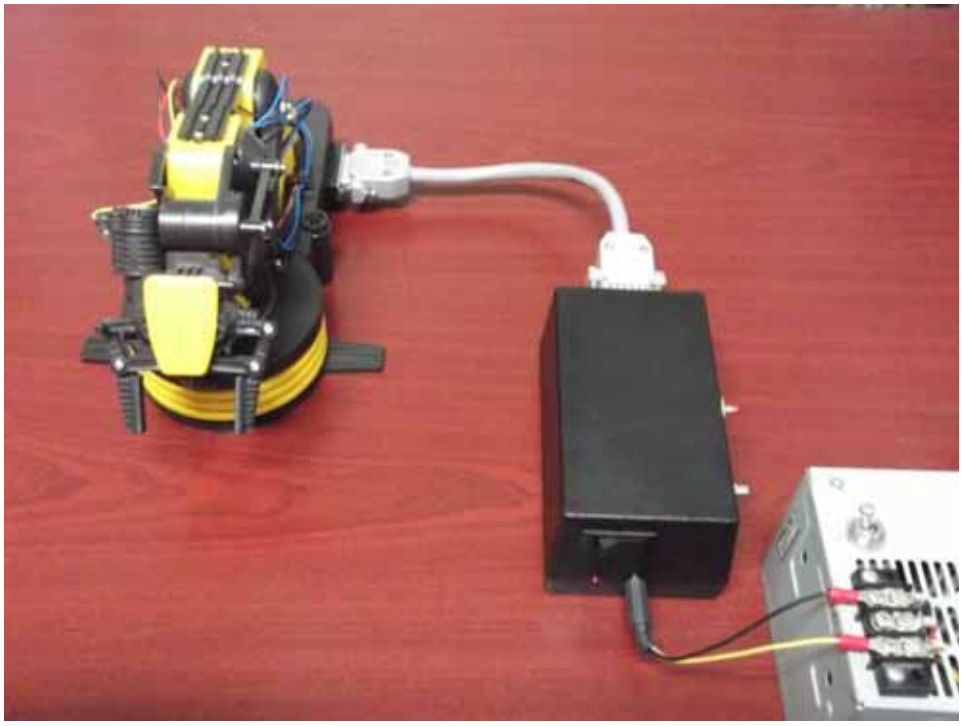

Figura 27. Circuito conectado al brazo robótico

6.2.4 Programación del microcontrolador

```
;-------------------------------------------------------------------------
;PROGRAMA MICROCONTROLADOR
;-------------------------------------------------------------------------

;Declaración de variables

MOVER        equ        7FH        ;contiene la dirección de giro
ENABLE       equ        7EH        ;contiene el motor a mover

V_DIR        equ        7DH
V_EN         equ        7CH

;Declaración de constantes puerto 0

;    P0.7         P0.6        P0.5        P0.4        P0.3        P0.2        P0.1
P0.0
;        |        |        |        |        |        |        |        |-----> DIR_M1I
;        |        |        |        |        |        |        |-----------> DIR_M2D
;        |        |        |        |        |        |-----------------> DIR_M2I
;        |        |        |        |        |-----------------------> DIR_M1D
;        |        |        |        |-----------------------------> DIR_M3D
;        |        |        |-----------------------------------> DIR_M3I
;        |        |-------------------------------------------> DIR_M4D
;        |-----------------------------------------------> DIR_M4I

M1D        equ        08H
M1I                equ        01H
M2D                equ        02H
M2I                equ        04H
M3D                equ        10H
M3I                equ        20H
M4D                equ        40H
M4I                equ        80H

STP_M1        equ        09H                ;detiene motor1 fast dir0=dir1=1
STP_M2        equ        06H                ;detiene motor2 fast dir0=dir1=1
STP_M3        equ        30H                ;detiene motor3 fast dir0=dir1=1
STP_M4        equ        0C0H               ;detiene motor4 fast dir0=dir1=1

;Declaracion de constantes puerto 2

;    P2.7         P2.6        P2.5        P2.4        P2.3        P2.2        P2.1
P2.0
;        |        |        |        |        |        |        |        |-----> EN_M1
;        |        |        |        |        |        |        |-----------> EN_M2
;        |        |        |        |        |        |-----------------> EN_M3
;        |        |        |        |        |-----------------------> EN_M4
;        |        |        |        |-----------------------------> EN_M5
;        |        |        |-----------------------------------> DIR_M5D
;        |        |-------------------------------------------> DIR_M5I
;        |-----------------------------------------------> LED
```

```
EN_M1        equ        81H        ;incluye LED_off
EN_M2        equ        82H
EN_M3        equ        84H
EN_M4         equ        88H

NEN_M1        equ        0FEH               ;deshabilita motor1
NEN_M2        equ        0FDH               ;deshabilita motor2
NEN_M3        equ        0FBH               ;deshabilita motor3
NEN_M4        equ        0F7H               ;deshabilita motor4

M5D              equ        20H
M5I              equ        40H
EN_M5            equ        90H               ;incluye LED_off
STP_M5           equ        60H               ;detiene motor5 fast dir0=dir1=0
NEN_M5           equ        0EFH              ;deshabilita motor5

STOP_M           equ        0FFH
STOP_E           equ        0E0H

LED_on           equ        7FH
LED_off          equ        80H

;*************************************************************************
;                              INICIO DE PROGRAMA PRINCIPAL
;*************************************************************************

               org        0

INICIO:        acall      COND_INI

GET_PC:        jnb        RI,$
               clr        RI
               mov        A,SBUF

MotStop:       cjne       A,#'s',Mot1D
               mov        P0,#STOP_M
               mov        P2,#STOP_E
               sjmp       GET_PC

Mot1D:         cjne       A,#'1',Mot1I
               setb       P0.3
               clr        P0.0
               setb       P2.0
               sjmp       GET_PC

Mot1I:         cjne       A,#'2',Mot2D
               clr        P0.3
               setb       P0.0
               setb       P2.0
               sjmp       GET_PC

Mot2D:         cjne       A,#'3',Mot2I
```

```
                setb       P0.1
                clr        P0.2
                setb       P2.1
                sjmp       GET_PC

Mot2I:           cjne       A,#'4',Mot3D
                 clr        P0.1
                 setb       P0.2
                 setb       P2.1
                 sjmp       GET_PC

Mot3D:           cjne       A,#'5',Mot3I
                 setb       P0.4
                 clr        P0.5
                 setb       P2.2
                 sjmp       GET_PC

Mot3I:           cjne       A,#'6',Mot4D
                 clr        P0.4
                 setb       P0.5
                 setb       P2.2
                 sjmp       GET_PC

Mot4D:           cjne       A,#'7',Mot4I
                 setb       P0.6
                 clr        P0.7
                 setb       P2.3
                 sjmp       GET_PC

Mot4I:           cjne       A,#'8',Mot5D
                 clr        P0.6
                 setb       P0.7
                 setb       P2.3
                 sjmp       GET_PC

Mot5D:           cjne       A,#'9',Mot5I
                 setb       P2.5
                 clr        P2.6
                 setb       P2.4
                 sjmp       GET_PC

Mot5I:           cjne       A,#'0',StpMot1
                 clr        P2.5
                 setb       P2.6
                 setb       P2.4
                 ajmp       GET_PC

                 ;**************

StpMot1:         cjne        A,#'a',StpMot2
                 setb        P0.3
                 setb        P0.0
                 clr         P2.0
```

```
                ajmp        GET_PC

StpMot2:        cjne        A,#'b',StpMot3
                setb        P0.1
                setb       P0.2
                clr        P2.1
                ajmp       GET_PC

StpMot3:        cjne       A,#'c',StpMot4
                setb       P0.4
                setb       P0.5
                clr        P2.2
                ajmp       GET_PC

StpMot4:        cjne       A,#'d',StpMot5
                setb       P0.6
                setb       P0.7
                clr        P2.3
                ajmp       GET_PC

StpMot5:        cjne       A,#'e',L_on
                setb       P2.5
                setb       P2.6
                clr        P2.4
                ajmp       GET_PC

                ;**************

L_on:           cjne       A,#'o',L_off
                clr        P2.7
                ajmp       GET_PC

L_off:          cjne        A,#'O',NO_PC
                setb        P2.7
NO_PC:          ajmp        GET_PC

;*********************************************************************
;                ------- Rutina de configuracion del sistema -------
;---------------------------------------------------------------------

COND_INI:       mov        P0,#STOP_M        ;deshabilita motores y apaga luz
                mov        P2,#STOP_E
                mov        SCON,#50H
                mov        TMOD,#20H
                mov        TH1,#0FAH
                mov        TL1,#0FAH
                setb        R1
                ret
;*********************************************************************

                end
```

7. Resultados

La aplicación para celular se ejecuta de forma correcta en cualquier celular que tenga instalada una versión de sistema operativo Android en su versión 4.0 o menor. Esta aplicación puede hacer una conexión como cliente TCP a un servidor, siempre y cuando se introduzcan los valores de IP y puerto correctos. Una vez hecha la conexión, es capaz de enviar valores de caracteres ASCII en forma de bytes a través de sockets TCP.

El módulo de recepción Wi-Fly es capaz de conectarse a un punto de acceso a la red, y una vez conectado abre conexiones TCP en forma de servidor, es capaz de aceptar la conexión de un cliente y recibir los datos que éste le envía. Ya que recibe los datos los transmite por medio de su puerto UART para que sean recibidos por otro sistema.

El circuito que lleva montado el microcontrolador AT89C51 puede recibir la información que le envía el módulo Wi-Fly a través del puerto serie. El microcontrolador es capaz de procesar esta información y convertirla en instrucciones en lenguaje ensamblador que le permiten manipular los motores que mueven el brazo robótico.

8. Análisis y discusión de los resultados

La aplicación desarrollada no permite el cierre de los sockets de forma manual, es decir no hay una opción dentro de ella que te permita desconectarte del servidor TCP, pero no es necesario esta opción ya que la configuración del Wi-Fly cierra la conexión si después de diez segundos no recibe información, por lo que solamente es necesario reconectarse por medio del cliente y la conexión se volverá a establecer.

Si la IP o el puerto que se introducen en la segunda pantalla de la aplicación no son los correctos, la aplicación no cambiará a la tercera pantalla, pero no manda algún mensaje que le diga al usuario que estos datos no son los correctos.

El programa en el microcontrolador solamente activa y desactiva los motores del brazo por lo que si un motor es activado por una señal que se manda desde el celular, este motor quedará activado mientras no se le mande otra instrucción ya sea de paro o de cambio de dirección, esto es un problema ya que un motor puede estar tratando de

mover alguna parte del brazo incluso cuando éste ya llegó al límite, lo que pondría en riesgo todo el sistema.

9. Conclusiones

Después de llevar a cabo el desarrollo y las pruebas de este proyecto, se presenta una alternativa al manejo de circuitos y sistemas de forma inalámbrica, utilizando tecnología Wi-Fi, se ha propuesto un prototipo portable ya que que fue llevado al plano de las aplicaciones para teléfonos celulares lo cual permite seguir innovando y mejorando el sistema.

La información sobre el dispositivo Wi-Fly, que fue uno de los principales componentes de este proyecto, es muy escasa, solamente se cuenta con el manual de usuario y algunos modos básicos de configuración, por lo que este proyecto puede servir como referencia para posteriores proyectos.

Las aplicaciones para teléfonos celulares son un campo enorme que permite amplias posibilidades de desarrollo de programas que pueden ser fácilmente integrados a sistemas embebidos, lo cual nos da las herramientas para que nuestros sistemas sean cada vez más portables.

10. Referencias bibliográficas

1. E. Rivera López, “Agente SNMPv1 integrado en un microcontrolador”, Proyecto Terminal, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México.

2. H. J. Mendoza Sosa, “Automatización de un brazo mecánico”, Proyecto Terminal, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México, 2012.

3. D. Hernández Reyes, “Brazo manipulador autómata con reconocimiento de colores para procesos industriales”, Proyecto Terminal, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México, 2008.

4. A. Suárez Zapata & J. Vila Francés, “Sistema de adquisición inalámbrico de señales basado en el módulo eZ430-RF2500 de Texas Instruments”, Mundo Electrónico No. 440/441, no. pp.25-28, 2012.

5. G.S. Gupta, S. C. Mukhopadhyay, M. Finnie, “WiFi-based Control of a Robotic Arm with Remote Vision”, in Conf. I2MTC '09, IEEE, p.p. 557 – 562.

6. Z. Annan, L. Peijie, C. Shuying, “Design and Realization of Home Appliances Control System Based on the Android Smartphone”, in (ICCECT) International Conference, Liaoning, 2012, p.p. 56 – 59.

7. J. A. Martínez Zermeño, “Interfaz para un robot de 4 grados de libertad accionado mediante radiofrecuencia vía computadora personal”, Propuesta de Proyecto Terminal, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México, 2011.

8. Microcontrolador AT89C51 Datasheet http://www.atmel.com/images/doc0265.pdf

9. RN-XV Datasheet
http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Wireless/WiFi/WiFly-RN-XV-DS.pdf

10. Wi-Fly Command reference, advanced features and applications user’s guide
http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Wireless/WiFi/rn-wiflycr-ug-v1.2r.pdf