UNIVERSIDAD DEL VALLE DE GUATEMALA

FACULTAD DE INGENIERÍA

Implementación de control manual inalámbrico para propulsión acuática y terrestre, sensores de humedad y trasmisión inalámbrica de video en un prototipo de carro anfibio.

Trabajo de graduación en modalidad de trabajo profesional presentado por
Alejandro José Sandoval Acevedo
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala

2014

Implementación de control manual inalámbrico para propulsión acuática y terrestre, sensores de humedad y transmisión inalámbrica de video en un prototipo de carro anfibio.

UNIVERSIDAD DEL VALLE DE GUATEMALA

FACULTAD DE INGENIERÍA

Implementación de control manual inalámbrico para propulsión acuática y terrestre, sensores de humedad y trasmisión inalámbrica de video en un prototipo de carro anfibio.

Trabajo de graduación en modalidad de trabajo profesional presentado por
Alejandro José Sandoval Acevedo
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala

2014

Vo. Bo. :

(f) 

Ing. Gustavo Prera

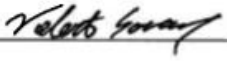
Tribunal Examinador:

(f) 

Ing. Gustavo Prera

(f) 

Msc. Carlos Esquit

(f) 

Msc. Roberto Saravia

Fecha de aprobación: Guatemala 19 de junio del 2014.

CONTENIDO

LISTA DE FIGURAS

Pagina

LISTA DE TABLAS

Pagina

LISTA DE ECUACIONES

Pagina

RESUMEN

Este trabajo fue elaborado y orientado para demostrar cómo puede ser utilizado un dispositivo Android para integrarse con elementos exteriores en este caso utilizando un Arduino ADK Mega, los cuales en conjunto pueden llegar a realizar tareas como manejo de servomotores o proporcionar las señales necesarias para ser implementadas con Drivers de motores DC, lo cual es necesario para proporcionarle movilidad tanto terrestre como acuática al prototipo de carro anfibio.

Se utilizó un dispositivo Android, porque no es necesaria ninguna licencia o pago para poder realizar aplicaciones y también por su gran compatibilidad con los diferentes dispositivos Arduino en este caso Arduino ADK Mega. Utilizando estas dos herramientas se pudo establecer un módulo de comunicación inalámbrica implementando una aplicación diseñada en el software eclipse.

Para establecer una conexión entre un dispositivo Android y una computadora se utilizó una aplicación gratuita llamada Webkey la cual permite clonar la pantalla de nuestro dispositivo hacia nuestra computadora permitiendo realizar tareas como manejar aplicaciones, utilizar la cámara fotográfica para recibir señal de video inalámbricamente, y obtener la posición del prototipo de carro anfibio utilizando GPS.

Los sensores fueron diseñados para detectar presencia de agua en la superficie con la cual tengan contacto, la resistencia del material se obtuvo de manera experimental llevando a cabo pruebas en diferentes ambientes. Estos sensores generaran una respuesta digital la cual podrá ser implementada para realizar un cambio automático de tracción terrestre a tracción acuática o viceversa.

I. INTRODUCCIÓN

El siguiente trabajo de graduación inició siendo parte de un megaproyecto el cual tenía como objetivo realizar un prototipo de carro anfibio. El megaproyecto se desintegró y se decidió retomar el tema de realizar un módulo el cual pueda ser implementado para mejorar el prototipo más adelante.

El uso de alternativas para hacer un proceso de una manera más practica ha crecido conforme la tecnología ha aumentado, un claro ejemplo es la invención de los vehículos anfibios permitiendo facilitar el proceso de cruzar un desastre natural o simplemente desear cruzar un río para explorar distintos terrenos.

La idea surgió de la necesidad constante de la CONRED en asistir inmediatamente a desastres donde sea imposible acceder debido a inundaciones, para comenzar un proceso de investigación se decidió realizar en la UVG un prototipo de carro anfibio con el fin de poder llevar a cabo en escala real con el transcurso del tiempo.

Debido al objetivo deseado en un futuro, es necesaria la implementación de un módulo de control manual inalámbrico el cual tenga larga distancia con el fin de garantizar la seguridad del operario y poder acceder a lugares peligrosos sin problema, esto nos dará la ventaja de proporcionar ayuda desde distancias lejanas.

II. JUSTIFICACIÓN

La idea de realizar un prototipo de carro anfibio surgió de la necesidad planteada por CONRED hacia la UVG, se solicitó realizar un carro anfibio el cual fuera capaz de ingresar a desastres naturales en ríos o inundaciones, debido a que realizar un carro anfibio de escala real es un proyecto sumamente complicado se tomó la decisión de realizar un prototipo el cual pueda servir de base para llevar a cabo el proyecto a escala real.

El siguiente trabajo se enfoca en resolver algunos problemas actuales del prototipo o a la implementación de nuevos módulos los cuales ayuden a proporcionar una forma más amigable de manejo, y una constante retroalimentación de datos sobre la ubicación de nuestro prototipo de carro anfibio.

Actualmente el prototipo de vehículo anfibio se maneja por medio de bluetooth dando libertad de poder maniobrarlo con la principal limitante del alcance, esta limitante se debe corregir ya que el vehículo se utiliza en terrenos donde el operario no puede posicionarse a una distancia corta, como solución se decidió implementar un módulo inalámbrico de mayor alcance el cual proporcione la misma libertad de manejo al vehículo.

Como se mencionó anteriormente la distancia es necesaria para poder maniobrar el vehículo anfibio sin que el operario corra ningún riesgo, pero aumentando la distancia se pierde la visibilidad de nuestro vehículo y se corre el riesgo de caer por pendientes o lugares donde sea imposible recuperar nuestro vehículo, para corregir este problema se implementó un módulo de transmisión de video inalámbrico para tener visibilidad constante del terreno por donde se maniobrara nuestro prototipo.

Para aumentar la visibilidad utilizando una cámara la cual transmitirá video inalámbricamente, se proporcionan dos señales capaces de controlar servomotores los cuales pueden generar movimiento a la base que se desee utilizar como soporte de la cámara de video. En caso no se obtenga video y se desee saber la ubicación del vehículo se acopló la transmisión de las coordenadas geográficas utilizando algún dispositivo GPS.

III. OBJETIVOS

A. General

Implementar un módulo de control manual inalámbrico para mejorar el manejo del prototipo de un carro anfibio. Proporcionar señal de video y su ubicación geográfica de manera inalámbrica y diseñar sensores capaces de identificar el tipo de ambiente en el cual se encuentra.

B. Específicos

- Diseñar una aplicación en Android la cual permita el control de servomotores y proporcione las señales necesarias a los drivers los cuales manejan los motores DC del prototipo.

- Establecer una comunicación inalámbrica entre una computadora y un dispositivo Android para poder utilizar la aplicación diseñada.

- Obtener señal de video y coordenadas geográficas de un dispositivo Android utilizando una aplicación la cual lo permita.

- Diseñar un circuito el cual logre identificar si el objeto con el cual tenga contacto se encuentra húmedo, y así realizar la transición de tracción terrestre a tracción acuática o viceversa.

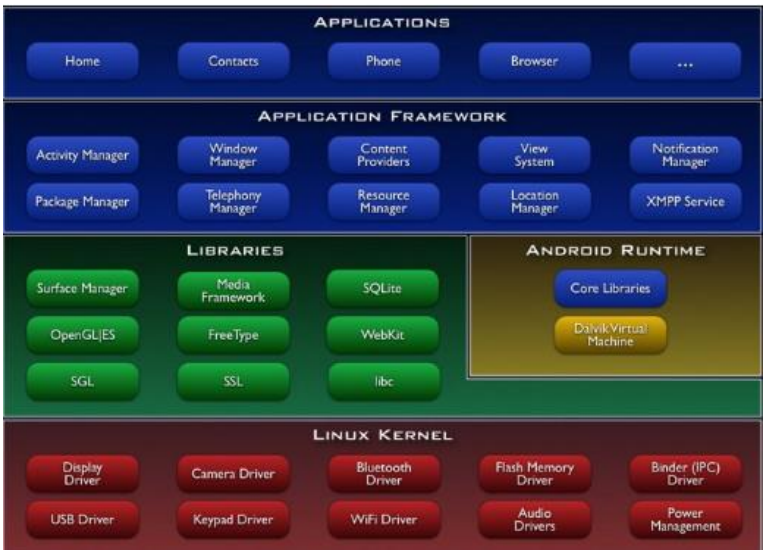
IV. MARCO TEÓRICO

A. ¿Qué es Android?

Android es una plataforma para dispositivos móviles que contiene una pila de software donde se incluye un sistema operativo y aplicaciones básicas para el usuario. Su diseño cuenta, entre otras, con las siguientes características: [1]

- Busca el desarrollo rápido de aplicaciones, que sean reutilizables y verdaderamente portables entre diferentes dispositivos.
- Los componentes básicos de las aplicaciones se pueden sustituir fácilmente por otros.
- Cuenta con su propia máquina virtual, que interpreta y ejecuta código escrito en Java.
- Permite la representación de gráficos 2D y 3D.
- Posibilita el uso de bases de datos.
- Soporta un elevado número de formatos multimedia.
- Servicio de localización GSM.
- Controla los diferentes elementos hardware: Bluetooth, Wi-Fi, cámara fotográfica o de vídeo, GPS, acelerómetro, infrarrojos, etc., siempre y cuando el dispositivo móvil lo contemple.
- Cuenta con un entorno de desarrollo muy cuidado mediante un SDK disponible de forma gratuita.
- Ofrece un plug-in para uno de los entornos de desarrollo más populares, Eclipse, y un emulador integrado para ejecutar las aplicaciones. [1]

Figura No. 1 Arquitectura de Android [1]



B. Partes principales de una aplicación en Android

Las aplicaciones en Android constan de cuatro partes esenciales, las cuales se programan utilizando el lenguaje de programación Java: [4]

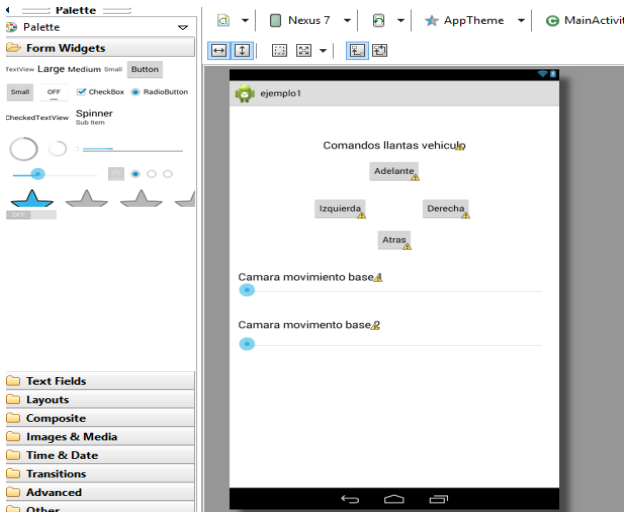
- Actividad (activity).
- Intención (intent).
- Servicio (service).
- Proveedor de contenido (content provider). [4]

No es necesario que todas las aplicaciones utilicen estos cuatro elementos ya que todo depende del objetivo el cual se desee alcanzar, pero en todas las ocasiones se utilizarán combinaciones de estas partes. [4]

El primer paso consta en realizar un esquema de cómo se desea la pantalla de la aplicación, porque en algunas ocasiones es necesario escribir cada línea de código las cuales describan las características de los elementos que se necesiten en la aplicación como por ejemplo botones, textos, imágenes, etc… [1]

En algunos casos se puede utilizar un programa de entorno de desarrollo integrado (IDE), como es en este caso eclipse, el cual cumple con ser un programa de este tipo por ser un programa de aplicación, esto significa que es un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Gracias a estas cualidades en algunos casos no es necesario escribir el código para poder posicionar un objeto en nuestra aplicación, sino que simplemente arrastramos el objeto a nuestra pantalla y de manera automática se generara el código necesario.

Figura No. 2 Ejemplo de la parte grafica de un programa utilizando eclipse



La actividad (activity) es la parte donde se encontrarán todas las funciones que generan un cambio o una acción en la aplicación, por ejemplo si tenemos botones en esta parte del programa se debe escribir todo el código necesario para definir la función Click del botón el cual estemos programando. De acuerdo a esto se podría decir que la actividad es nuestro documento principal, en Android se pueden tener más de una actividad ya que debe existir una por cada página o escritorio que exista en nuestra aplicación. [4]

Las intenciones (intent) es la parte del código que nos describe lo que la aplicación desea hacer, esta sección consta de dos partes importantes las cuales son la acción y lo que desea hacer la aplicación, por ejemplo para ver una página web en el navegador el cual traiga por defecto el dispositivo Android se debe crear una intención con la acción ver y la descripción del contenido al cual se desea ingresar. [4]

Figura No. 3 Ejemplo de una intención [4]

```
new Intent(android.content.Intent.VIEW_ACTION,
      ContentURI.create("http://anddev.org"));
```

Los servicios son programas que siguen corriendo aun después de haber salido o acceder a otra aplicación, como por ejemplo el reproductor de audio cuando el usuario sale de la aplicación, esta continua reproduciendo la música. Los servicios se pueden ordenar por prioridad dependiendo de cómo desee el programador establecer su aplicación y la función que desee darle. [4]

Los proveedores de contenido son todas aquellas aplicaciones las cuales pueden almacenar datos en nuestro dispositivo como por ejemplo SQLite, esta parte es muy utilizada para generar inventarios o poder compartir información con otros dispositivos, lo cual es muy utilizado ya que hoy en día la mayoría de información se almacena de manera digital.[4]

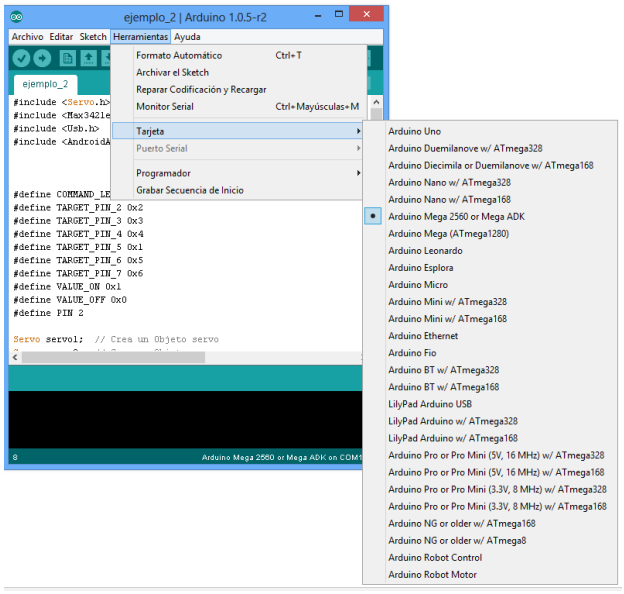
C. ¿Qué es un Arduino ADK?

Un Arduino del tipo ADK (Accessory Development Kit) es básicamente un micro controlador el cual puede integrarse al protocolo Open Accessory Standard, este protocolo es utilizado por los dispositivos Android para establecer una comunicación por medio de USB o Bluetooth, siempre y cuando el dispositivo con el cual se desee conectar tenga la opción de entender este protocolo. [2]

Figura No. 4 Imagen de un Arduino ADK Mega [2]

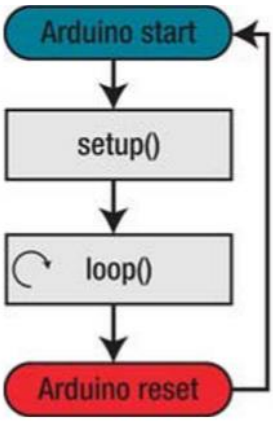
En el programa IDE que se utiliza para programar nuestro Arduino, cada código debe ser escrito para el micro controlador que viene incorporado en nuestro dispositivo, en algunos casos es posible removerlo y el dispositivo se utiliza únicamente para cargar el programa pero en el caso del Arduino ADK Mega viene soldado a nuestra placa como se puede ver en la Figura No. 4. La opción de poder especificar a cual micro controlador corresponde el código escrito es seleccionada en el menú de herramientas del programa. [2]

Figura No. 5 Selección de placa para programar un Arduino



Antes de comenzar a programar se debe conocer el ciclo de vida de un programa de Arduino o como son llamados por el programa (Sketch). Las partes básicas son, el comienzo en donde se definen variables, importan librerías, definen constantes, etc… Luego tenemos la parte de setup donde se especifica los pines que se van a utilizar y si van a hacer entradas o salidas, también se puede configurar si utilizara el protocolo Open Accessory, y otro tipo de comandos que se vayan a utilizar en el programa. Por último se encuentra la función Loop aquí es donde se escribirán todas las funciones las cuales queremos repetir siempre.

Figura No. 6 Ciclo de vida Sketch [2]



El lenguaje que utiliza Arduino está basado en Java, pero su sintaxis es similar a C++, conociendo como debe ser la sintaxis, solo necesitamos conocer las librerías necesarias para establecer una comunicación Open Accessory con un dispositivo Android, las librerías necesarias tienen el nombre de USB Host las cuales podemos obtener de la página oficial de Arduino sin ningún costo. [2]

Cuando ya se tienen las librerías necesarias para establecer la comunicación entre los dispositivos solo queda definir cómo identificar los valores enviados desde Android a Arduino, en este caso está establecido el primer byte como el comando, el segundo el pin objetivo el cual se desea controlar y por último el valor que desea enviar. [2]

Figura No. 7 Definición de bytes en el protocolo establecido [2]

1-Byte	2-Byte	3-Byte
Command 0xF	Target 0xF	Value 0xF

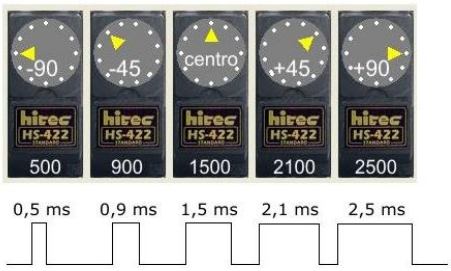
D. Manejo de servo motores e implementación de PWM utilizando Arduino

Un servomotor (Figura No. 8) es la integración de un motor eléctrico, una caja reductora y un sensor el cual controla el movimiento del motor. Estos elementos en conjunto forman un componente sencillo de controlar sin necesidad de tener que implementar un sistema de control para poder mantener su posición al momento de agregar una carga sobre el motor, gracias a esto es un componente sencillo de implementar en aplicaciones de robótica.

Figura No. 8 Ejemplos de servomotores [6]

Dependiendo del tipo y la marca los servomotores pueden tener un movimiento desde 90 grados hasta 360 grados o en algunos casos funcionar por número de vueltas, la ventaja en utilizar este tipo de motores es poder mantener el torque en cualquier posición del eje. Para poder manejar las posiciones de estos motores es necesaria una señal conocida como PWM, la cual consta de una señal cuadra comúnmente de una frecuencia con valor 50Hz aun que puede variar, el valor usado para controlar la posición es el tiempo en alto de la señal o el ciclo de trabajo.

Figura No. 9 Ejemplos de señal PWM para controlar un servomotor [6]



Utilizando un dispositivo Arduino podemos hacer uso de la librería llamada Servo, esta nos permite crear un objeto tipo Servo y definir un pin el cual será una salida de tipo PWM, la ventaja de esta librería es poder obtener valores analógicos en una entrada de nuestro Arduino y traducir estos valores a posiciones en grados utilizando la función map, esta función tiene la siguiente estructura *map("valor obtenido de la entrada analógica","valor minimo que se puede obtener en esa entrada","valor máximo que se puede obtener en esa entrada","posición minima en grados que deseamos","posición máxima en grados")*, definiendo esto el dispositivo Arduino realiza todas las conversiones necesarias para generar un PWM que corresponda a la posición deseada.

En el caso donde se desee generar una salida PWM manual sin ayuda de la función la cual calcula todo por nosotros, basta con seguir el siguiente esquema, *analogWrite("pin que deseamos utilizar","valor entre 0 – 255")*, Se utiliza una salida analógica debido a que intentaremos simular un valor analógico utilizando una señal cuadrada creada por valores digitales, el valor que se encuentra entre 0 – 255 se encargará de variar el ciclo de trabajo de nuestra señal cuadrada.

V. MATERIALES

Tabla No. 1 Listado de materiales electrónicos utilizados

Descripción	Cantidad
Arduino Mega ADK	1
Samsung Galaxy tab 3 7”	1
Amplificador operacional Lm324	1
Laptop Asus G55V	1
Resistencia de 1 MΩ	1
Servomotores	2

Tabla No. 2 Listado de materiales no electrónicos utilizados

Descripción
Aplicación gratuita Webkey.

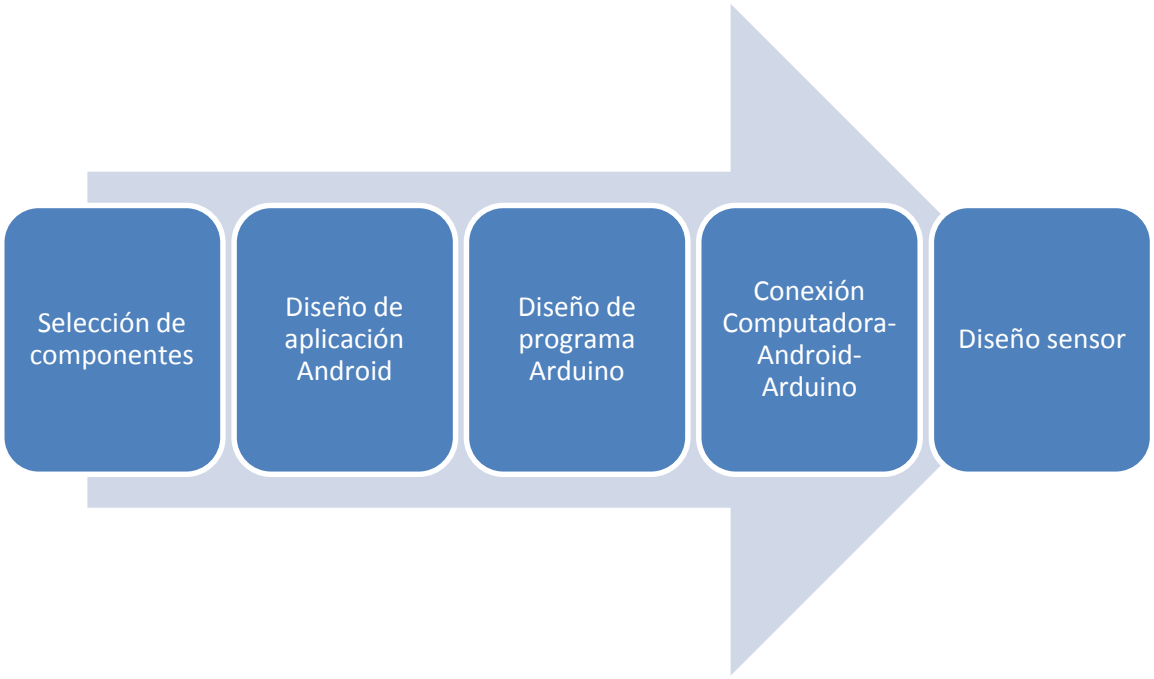
VI. MÉTODOS Y RESULTADOS

El módulo de control manual inalámbrico implementado fue capaz de realizar tareas como proporcionar las señales requeridas para el funcionamiento de dos servomotores los cuales pueden ser implementados en la realización de una base para un dispositivo de grabación de video que en este caso fue una Samsung Galaxy Tab3. Además, permitió el manejo a largas distancias utilizando un programa gratuito llamado Webkey el cual puede ser encontrado en la tienda virtual de aplicaciones de Android.

El sensor identificó la presencia de agua en distintos ambientes a los cuales fue sometido dando una respuesta positiva cuando hubiera una cantidad de agua elevada, obteniendo así una respuesta digital capaz de generar un bit el cual puede ser utilizando para implementar un módulo que permita realizar el cambio de acuático a terrestre o viceversa.

Para alcanzar de manera satisfactoria los objetivos se realizó un proceso el cual es resumido en el siguiente diagrama.

Figura No. 10 Resumen de proceso



A. Selección de componentes

La implementación de un control manual inalámbrico requiere definir dos componentes esenciales el módulo de comunicación inalámbrica y los programas necesarios para generar las señales capaces de controlar servomotores y drivers para control de motores dc.

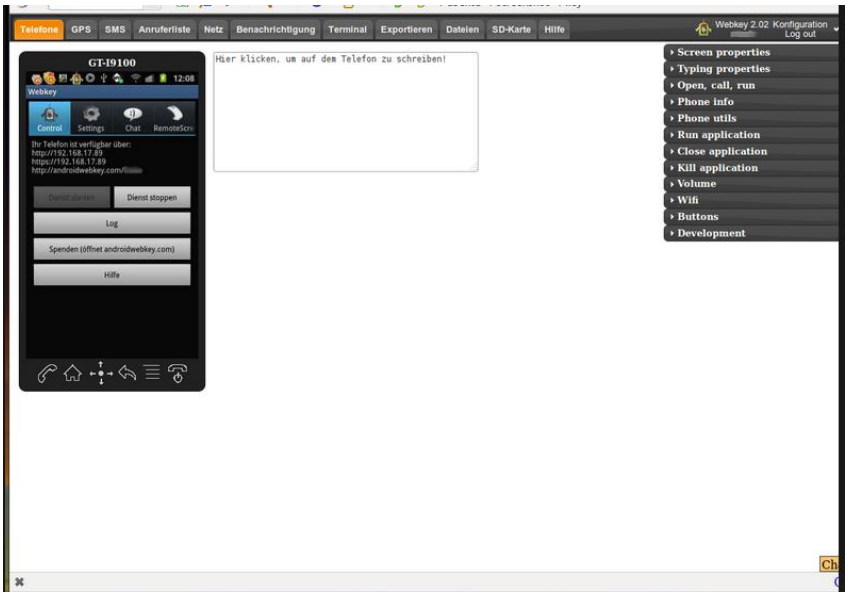
Se realizó una búsqueda para seleccionar el tipo de software a utilizar, para satisfacer el objetivo de poder manejar el prototipo desde largas distancias se decidió implementar un dispositivo el cual tenga conexión constante a internet en este caso una Samsung Galaxy Tab3 (Figura No. 10) con acceso a internet 3G. La ventaja de utilizar un dispositivo Android es la libertad para desarrollar aplicación sin ningún costo.

Figura No. 11 Samsung Galaxy tab3

Para establecer una comunicación de manera inalámbrica se implementó la aplicación Webkey la cual utilizando un navegador web puede clonar la pantalla de nuestro dispositivo (Figura No. 11) junto a todas las funciones para permitir controlar la mayoría de sus funciones de manera inalámbrica. El único procedimiento a realizar para conectar nuestro dispositivo Android con la computadora es seleccionar la opción Iniciar servicio y automáticamente aparecerá una

dirección que se debe copiar en nuestro navegador y así desplegará la pantalla de nuestro dispositivo junto a otras opciones.

Figura No. 12 Imagen de interfaz gráfica aplicación Webkey



La manera de conectar el carro anfibio a nuestro dispositivo Android fue utilizando una placa Arduino ADK Mega, se seleccionó este componente ya que es diseñado para comunicarse utilizando protocolo establecido por Android utilizando la librería USBHost la cual se encuentra en la página oficial de Arduino para descargarse de manera gratuita. [2]

Para poder identificar si existe una alta cantidad de agua en el ambiente con el cual se tenga contacto y así generar una respuesta la cual puede llegar a ser implementada para efectuar la transición de tracción terrestre a tracción acuática, se utilizara una resistencia de 1MΩ en conjunto con dos cables los cuales harán contacto con el ambiente en el que se encuentra el prototipo y un amplificador operacional de referencia a tierra.

B. Diseño de aplicación Android

El prototipo de carro anfibio debe poder ser maniobrado desde largas distancias para no poner en riesgo al operario, para eso se diseñó una aplicación la cual transmite valores por medio de USB y también transmite dos valores en el rango de 0-255 para poder manejar dos servomotores. Las principales funciones de la aplicación son:

- Botón adelante.
- Botón atrás.
- Botón izquierda.

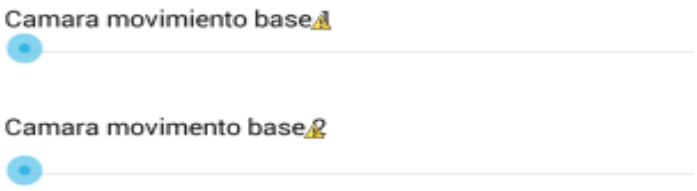
- Botón derecha.
- SeekBar Servo1.
- SeekBar Servo2.

Para proporcionar más comodidad se decidió que el dispositivo mandara un valor de 0x1 mientras se tenga el dedo sobre cualquiera de los cuatro botones (Figura No. 12) lo cual es conocido como la función onTouch, para el evento de retirar el dedo se utilizó la función onClick para mandar el valor de 0x0, se utilizó la función onClick porque es el evento que se activa al momento de levantar el dedo de cualquier botón. El orden de mandar los valores fue, un valor de comando, un valor que tiene la función de opción y el último byte que es el valor mencionado anteriormente.

Figura No. 13 Ejemplo de botones

En la implementación de señales capaces de controlar servomotores se decidió utilizar un objeto llamado SeekBar (Figura No. 13), este objeto permite deslizar con el dedo un círculo sobre una línea recta, esta acción hace variar el valor el cual se encuentra entre los valores de 0 a 255, este valor será mandado por medio de USB.

Figura No. 14 Ejemplo de SeekBars



C. Diseño de programa Arduino

La manera de integrar una aplicación en Android con el prototipo de carro anfibio fue utilizando un Arduino el cual se encargó de traducir los valores enviados por el dispositivo Android. El programa se divide en dos partes la primera es proporcionar las señales necesarias para el manejo de los drivers utilizados para la alimentación de los motores del prototipo de carro anfibio, y la segunda parte se encarga de las señales para controlar dos servomotores.

La primera parte del programa conto con verificar el byte de comando para identificar si el dispositivo Android se desea comunicar con el Arduino, después se necesita identificar el segundo byte el cual se utilizó para identificar el objeto en la aplicación Android con el cual se está interactuando, y por último el byte de valor que establece la acción que llevaremos a cabo con el Arduino.

Tabla No. 3 Valores de los bytes necesarios para el movimiento del prototipo

Función	primer byte enviado	segundo byte enviado	tercer byte enviado	PWM 1	PWM 2
Adelante	0x2	0x2	0x1	255	255
Atrás	0x2	0x1	0x1	0	0
Izquierda	0x2	0x3	0x1	0	255
Derecha	0x2	0x4	0x1	255	0
Sin movimiento	0x2	----	0x0	112	112

Figura No. 15 PWM generado para llevar a cabo la función sin movimiento

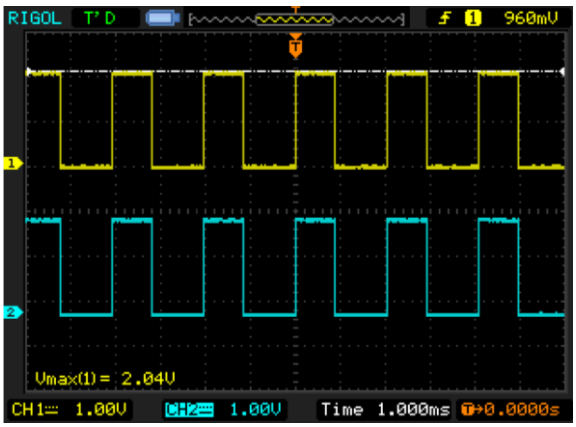


Figura No. 16 PWM generado para llevar a cabo la función izquierda

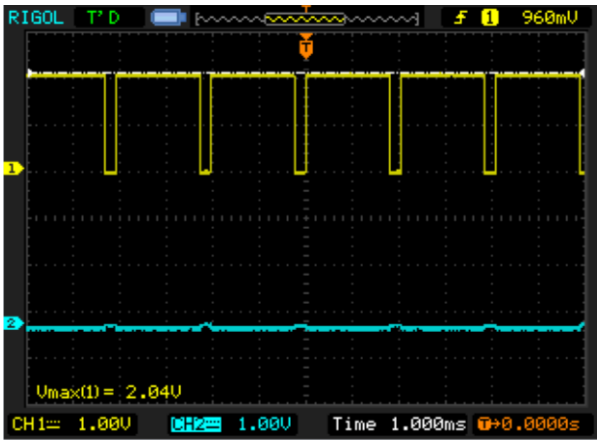


Figura No. 17 PWM generado para llevar a cabo la función derecha

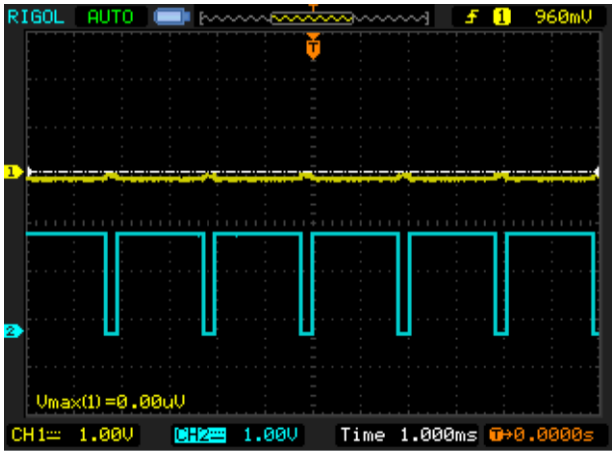


Figura No. 18 PWM generado para llevar a cabo la función atrás

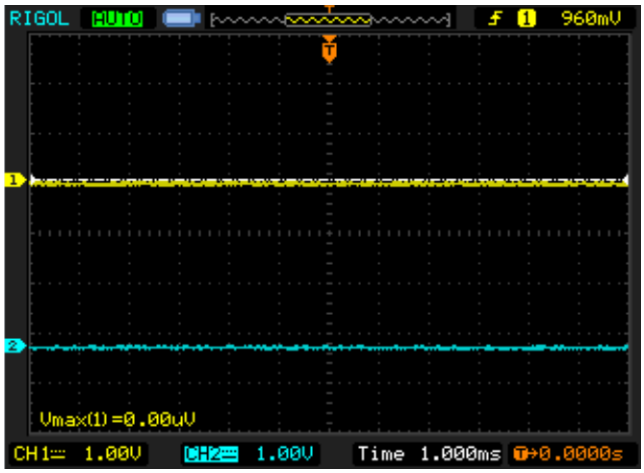
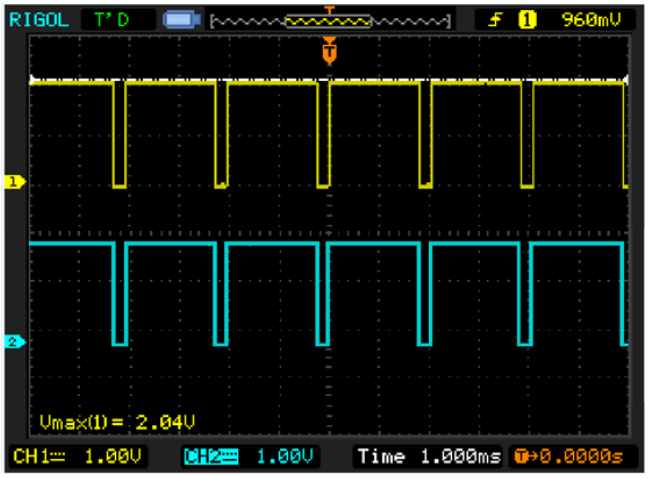


Figura No. 19 PWM generado para llevar a cabo la función adelante



La función sin movimiento no tiene un segundo byte definido debido a que será la acción a realizar si no se cumple ninguna de las cuatro condiciones (Tabla No. 3), esto permite al prototipo estar estático mientras el operario no solicita ninguna función.

En la segunda parte se necesita generar las señales para controlar dos servomotores, este proceso es similar a la parte anterior, se tiene que verificar el byte de comando, luego identificar el objeto en este caso con cual SeekBar se está interactuando y por último se utilizó la función map la cual mapea el valor que se encuentra en el rango de 0-255 enviado desde el dispositivo Android a las posiciones en grados de los servomotores.

Figura No. 20 Primera parte diagrama de flujo programa Arduino

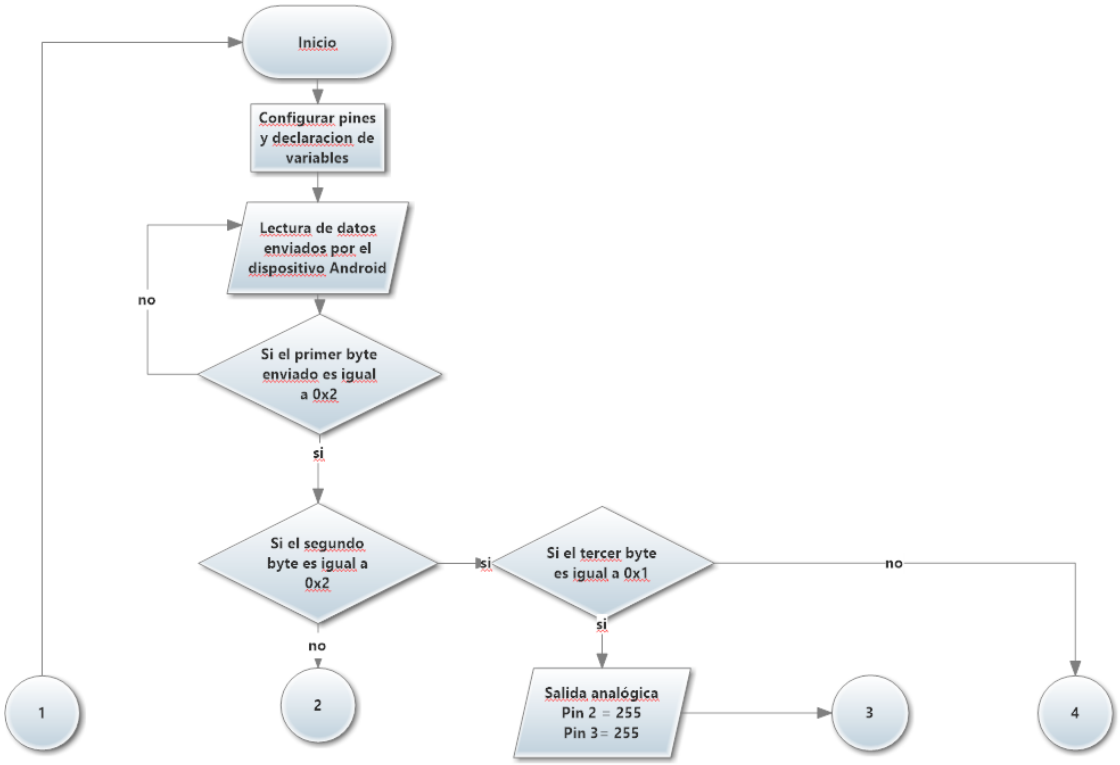
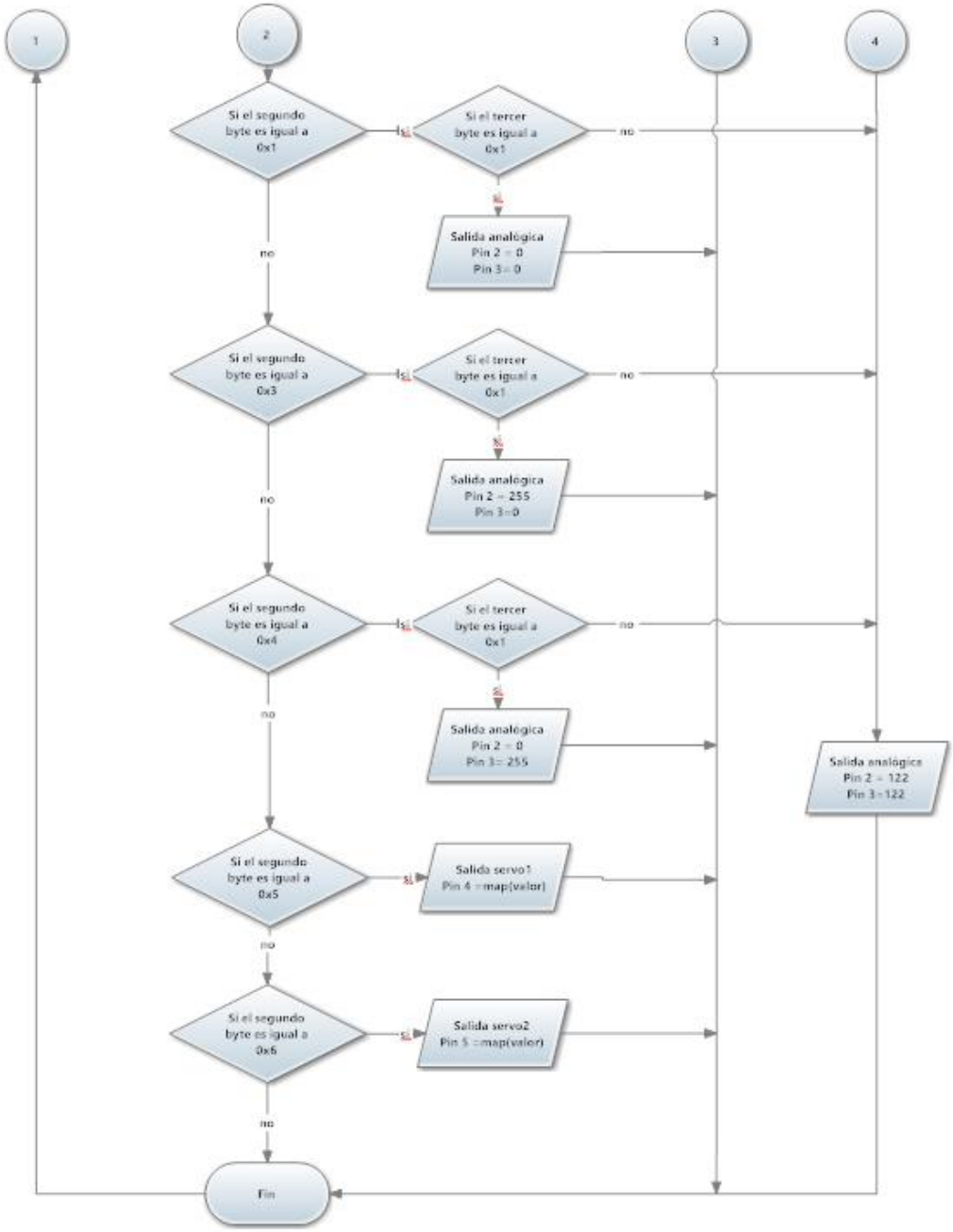


Figura No. 21 Segunda parte diagrama de flujo programa Arduino



Lo siguiente es entender cómo funciona la función map de nuestro Arduino, ya que solo es necesario introducir el valor mínimo y máximo enviado por el dispositivo Android y el valor mínimo y máximo de posición para nuestros servomotores. Para entender el funcionamiento se tomaron los datos de todas las posiciones utilizando la función Serial.print la cual nos permite visualizar todos los valores que nuestro Arduino recibe y manda.

Para fines ilustrativos se seleccionaron diez valores los cuales se encuentran en la Tabla No. 4, luego se efectuó una gráfica para obtener una ecuación utilizando la función de regresión que nos proporciona Excel (Figura No. 21).

Tabla No. 4 Valores obtenidos de la función map

Valor dado por Galaxy Tab3	grados	Ton (ms)	Ciclo de trabajo
0	5	0.60	3.00%
35	27	0.82	4.10%
58	43	1.00	5.00%
89	62	1.18	5.90%
119	82	1.40	7.00%
151	102	1.60	8.00%
181	122	1.80	9.00%
215	142	2.02	10.10%
244	162	2.22	11.10%
255	170	2.30	11.50%

Figura No. 22 Gráfica Valor SeekBar vs Ciclo de trabajo

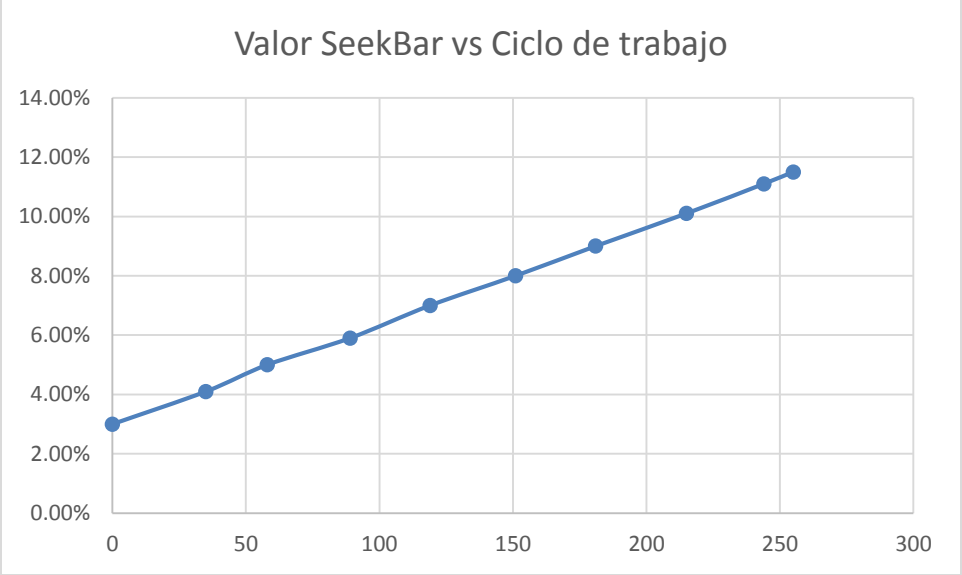


Tabla No. 5 Comparación de tres tipos de regresiones

Tipo de regresión	Ecuación	valor R^2
Lineal	$y = 0.0003x + 0.0299$	0.9998
Exponencial	$y = 0.0353e^{0.005x}$	0.9656
Polinomial grado 2	$y = 3E\text{-}09x^2 + 0.0003x + 0.03$	0.9998

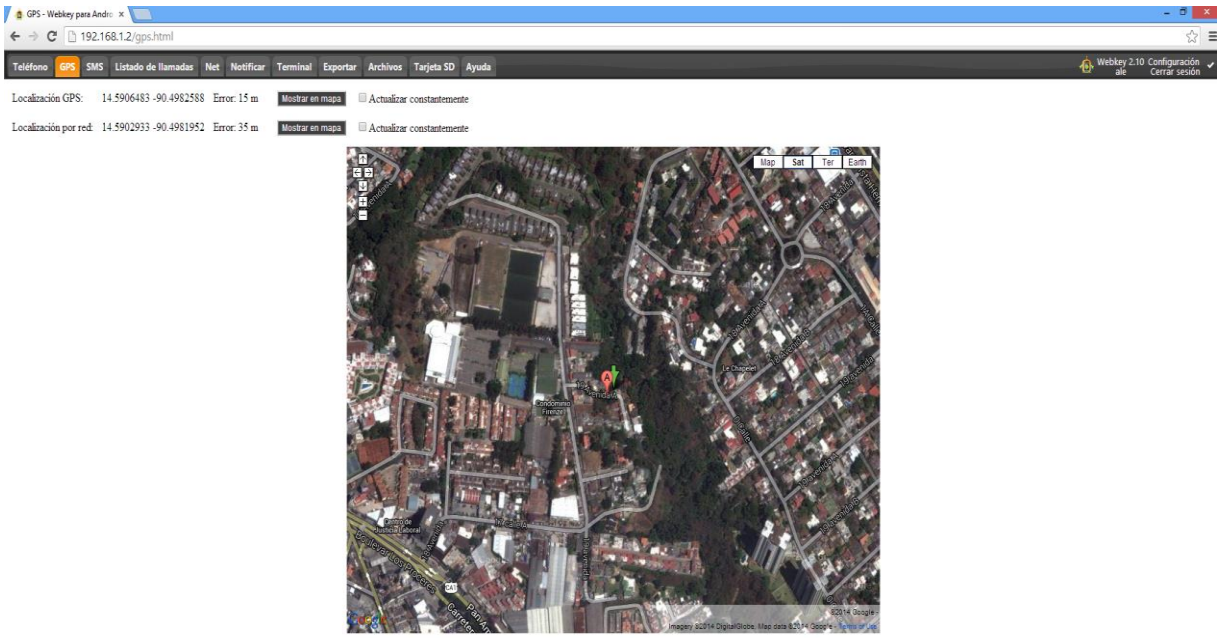
Como se puede observar, los dos tipos de regresiones que describen mejor los datos obtenidos son la regresión lineal, podemos decir esto gracias a el valor de R^2 que es el valor al cual nos describe la capacidad que tiene la función y(x) para describir la serie de puntos solicitados (Tabla No. 5). [5]

No es necesario tomar en cuenta la regresión polinomial grado 2, debido a que si observamos el comportamiento de los datos obtenidos y es multiplicada por un factor muy cercano a cero, debido a esto la regresión lineal la única que tomaremos en cuenta.

D. Conexión Computadora-Android-Arduino

Como se ha mencionado anteriormente la conexión entre la computadora y nuestro dispositivo Android se estableció utilizando la aplicación gratuita Webkey, esta aplicación nos permite tener control total de nuestro dispositivo, también nos permite utilizar la cámara integrada en nuestra Samsung Galaxy Tab3 para obtener una visión del entorno donde se encuentra nuestro prototipo de carro anfibio. Una opción muy útil al momento de no poder identificar el entorno es la opción de GPS la cual permite visualizar las coordenadas en un mapa (Figura No. 22).

Figura No. 23 Visualizando coordenadas GPS utilizando Google Chrome



Para establecer la conexión entre nuestro Arduino ADK Mega y nuestro dispositivo Android, se debe realizar una configuración. Para preparar el dispositivo Arduino es necesario descargar las librerías USBHost y AndroidAccessory, al iniciar el programa necesitamos importar estas dos librerías para poder crear un objeto el cual se encargara de comunicarse con el dispositivo Android el cual se llamara acc:

Android Accessory acc ("Manufacturer","Model","Description","Version","URI","Serial"); [2]

Luego debemos inicializar nuestro objeto creado:

acc.powerOn(); [2]

Para finalizar debemos verificar si el objeto definido se encuentra conecto y guardar el valor enviado en una variable tipo byte la cual en este caso tiene el nombre de rcvmsg:

```
if (acc.isConnected()) {

int len = acc.read(rcvmsg, sizeof(rcvmsg), 1);  [2]
```

El siguiente paso es realizar la configuración necesaria en nuestro dispositivo Android para que establezca la conexión con nuestro Arduino. Debemos agregar a nuestro AndroidManifest.xml las siguientes líneas de código para especificar que nuestra aplicación utilizará el modulo USB para comunicarse con un accesorio externo:

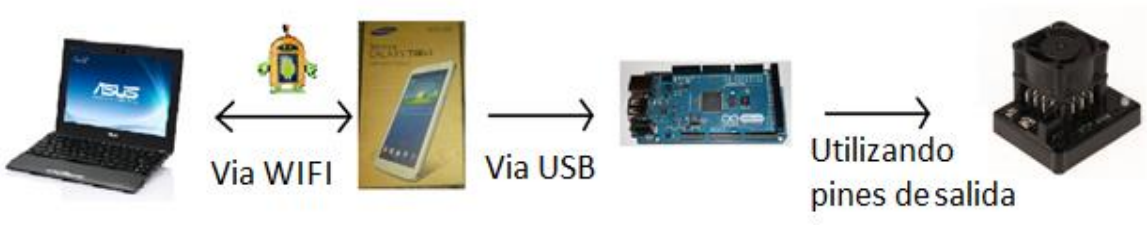
```
<uses-feature android:name="android.hardware.usb.accessory" />

<uses-library android:name="com.android.future.usb.accessory" />

    <meta-data

        android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"andr
        oid:resource="@xml/accessory_filter" /> [2]
```

Figura No. 24 Diagrama de conexión de componentes



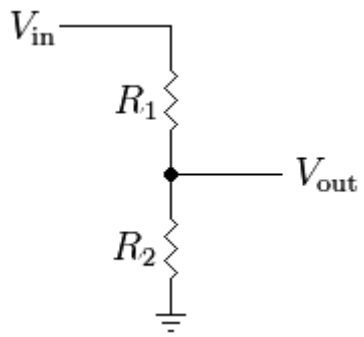
E. Diseño sensor

Para permitir al prototipo de carro anfibio detectar el ambiente donde se encuentra fue diseñado un sensor el cual permite identificar la presencia de agua del ambiente con el cual tenga contacto para generar una respuesta de 5V, si no hubiera agua en el ambiente el sensor generara una respuesta de 0V.

Se decidió diseñar este tipo de circuito ya que en el protocolo se había planteado un sensor de humedad, pero al analizar mejor el problema a resolver se decidió utilizar el sensor de conductividad, el cual utiliza la conductividad eléctrica del agua para generar la respuesta digital.

El primer paso fue la búsqueda de un circuito el cual nos permitiera variar un valor conforme dos cables realizando la función de electrodos tuvieran contacto con diferentes tipos de ambientes. El circuito seleccionado fue el divisor de voltaje (Figura No. 24) porque este circuito es capaz de variar su voltaje dependiendo del valor de las resistencias implementadas.

Figura No. 25 Circuito divisor de voltaje



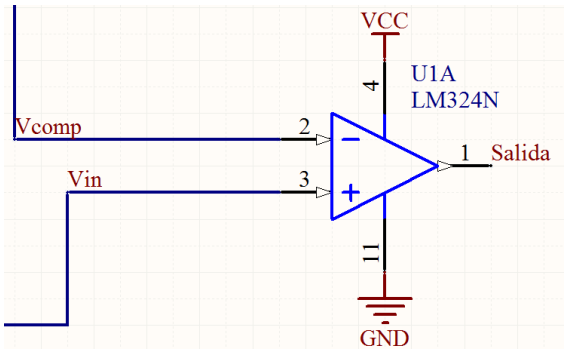
En este caso se utilizó un valor de R_2 fijo el cual fue de 1 MΩ y el valor de R_1 fue la resistencia del ambiente con el cual tuvieran contacto los electrodos, Obteniendo así un valor de V_{out} que fue capaz de variar dependiendo del ambiente con el cual tuvieron contacto los electrodos (Ecuación No. 1).

Ecuación No. 1 Valor V_{out} de un divisor de voltaje

$$V_{\text{out}} = \frac{R_2}{R_1 + R_2} \cdot V_{\text{in}}$$

El proceso para interpretar los valores obtenidos y permitir al sensor dar una respuesta digital es la implementación de un circuito comparador (Figura No. 25), este circuito se realizó utilizando un amplificador operacional de referencia a tierra debido a que solo se necesitaban respuestas positivas.

Figura No. 26 Circuito comparador



Este circuito nos permite establecer un rango el cual nos proporcione un voltaje de 0V y al momento de superar ese rango tendremos una respuesta de 5V (Ecuación No. 2).

Ecuación No. 2 Valor V_{out} de un comparador

$$Vout = \begin{cases} Vcc, Vin > Vcomp \\ GND, Vin < Vcomp \end{cases}$$

Los valores necesarios para comprobar el funcionamiento de nuestro sensor es la resistencia eléctrica de algunos elementos (Tabla No.6) con los cuales tendrán contacto los electrodos. Estos valores fueron obtenido de manera experimental utilizando un multímetro con una distancia de electrodos menor a 5 cm, debido a esto los valores solo serán válidos para el presente experimento ya que pueden verse afectados por distintos factores externos.

Tabla No. 6 Resistencia eléctrica de algunos materiales

Material	Resistencia (MΩ)
Tierra seca	-----
Tierra humeda	0.680
Agua lodosa	0.936
Agua	1.2

En el caso de la tierra seca no se pudo obtener un valor de resistencia debido a que nuestro instrumento de medición carece de rangos de medición mayores a 20MΩ, utilizando estos valores se sustituyó en la Ecuación No. 1 el valor de R_1 por el valor de resistencia que tiene el agua (Ecuación No. 3).

Ecuación No. 3 Sustitución de R_1 por resistencia eléctrica del agua

$$Vout = \frac{1\ M\Omega}{1\ M\Omega + 1.2\ \Omega M} * 5V$$

Como podemos observar en la Ecuación No. 3 se remplazaron los valores de R_1, R_2 y V_{in} con los valores utilizado en las pruebas físicas, al momento de realizar la operación obtenemos que V_{out} tendrá un valor de 2.272V, este valor nos sirvió para establecer el voltaje de comparación ya que si nuestro circuito divisor nos proporciona un voltaje mayor o igual a 2.272V será necesaria realizar la transición de tracción terrestre a tracción acuática, y en el caso inverso si el valor de nuestro circuito divisor de voltaje es menor a 2.272V se realizara la transición de tracción acuática a tracción terrestre.

VII. CONCLUSIONES

De acuerdo a los resultados obtenidos durante el desarrollo del proyecto podemos concluir:

1. Se logró diseñar un módulo de control inalámbrico el cual puede ser implementado en el vehículo anfibio, y se logró establecer una transmisión de video inalámbrica por medio de la aplicación webkey y los dispositivos seleccionados.
2. La aplicación diseñada fue capaz de manejar dos servomotores.
3. Implementando un sensor de conductividad se puede obtener que tanto se ha humedecido un ambiente debido al agua presente, indicando si es necesario llevar a cabo el cambio de tracción.
4. El sensor de conductividad es susceptible a varios componentes externos como el clima, distancia de electrodos, tipos de materiales con el cual tenga contacto; por lo tanto no es la mejor opción para identificar el tipo de ambiente.

VIII. RECOMENDACIONES

Al perderse la comunicación entre el dispositivo Android y Arduino, es necesario establecer una rutina que lleve a cabo una serie de comandos. Se recomienda detener el envío de datos para asegurarse de que al momento de restablecer la comunicación entre estos dos dispositivos, no sean enviados datos erróneos.

La aplicación en Android al momento de interactuar con los objetos inicia el envío de datos constantemente. Esto satura el canal de comunicación, que conecta los dispositivos Android y Arduino, desperdiciando recursos del Android. Para evitar este inconveniente se recomienda diseñar una aplicación la cual sólo envié comandos al Arduino y éste sea el encargado de enviar constantemente los datos a nuestros componentes electrónicos.

Se deben implementar sensores de presión, los cuales podrán identificar si existe una fuerza de flotación actuando sobre la parte baja del vehículo. Utilizando estos sensores en conjunto con lo sensores diseñados se podrá obtener una mejor y precisión al identificar los cambios de ambiente.

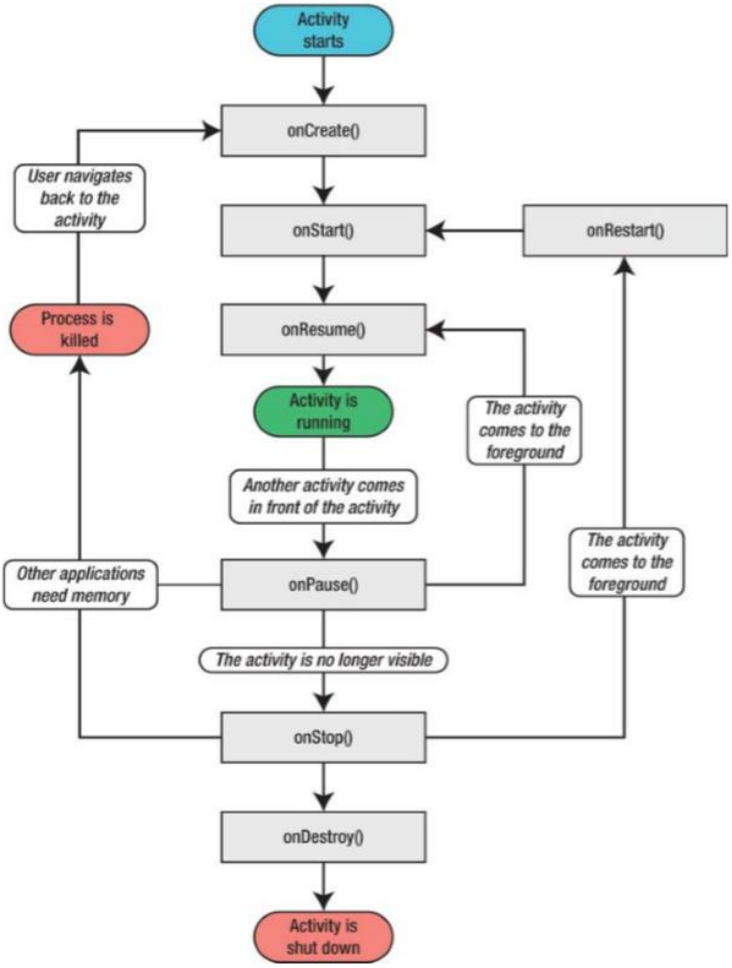
IX. BIBLIOGRAFÍA

1. Aranaz, Jaime.2009; *Desarrollo de aplicaciones para dispositivos moviles sobre la plataforma Android de Google.* Tesis Universidad Carlos III de Madrir. España,Madrid 2009. 199 pp.
2. Böhmer, Mario.2012. *Beginning Android ADK with Arduino.* 1ra ed. Berlin, Technology in action, 310 pp.
3. Evans W. 2007. *Arduino programming notebook*. 1ra ed. California, Creative Commons, 45 pp.
4. Gramlich, Nicolas *andbook! – Android programming*. 1ra ed. Estados Unidos, anddev.org. 62 pp.
5. Peña, Daniel. 2002. *Regresión y diseño de experimentos*. 3ra ed. España Madrid, Alianza Editorial. 744 pp.
6. Pompa, Pablo. Como utilizar los servo motores en nuestros proyectos. *http://www.superrobotica.com/Servosrc.htm*. Visitado: 12 de marzo de 2014.

X. ANEXOS

A. Diagrama ciclo de vida de una aplicación en Android y código fuente programa en Android

Figura No. 27 Diagrama de flujo programa de control [2]



```
package com.example.ejemplo1;

import java.io.FileDescriptor;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

import com.android.future.usb.UsbAccessory;
import com.android.future.usb.UsbManager;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnTouchListener;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.AsyncTask;
import android.os.ParcelFileDescriptor;
import android.util.Log;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;



public class MainActivity extends Activity {

    private static final String TAG =
MainActivity.class.getSimpleName();

    private PendingIntent mPermissionIntent;
    private static final String ACTION_USB_PERMISSION =
"com.android.example.USB_PERMISSION";
    private boolean mPermissionRequestPending;

    private UsbManager mUsbManager;
    private UsbAccessory mAccessory;
    private ParcelFileDescriptor mFileDescriptor;
    private FileInputStream mInputStream;
    private FileOutputStream mOutputStream;

    private static final byte COMMAND_LED = 0x2;
    private static final byte TARGET_PIN_2 = 0x2;
    private static final byte TARGET_PIN_3 = 0x3;
    private static final byte TARGET_PIN_4 = 0x4;
    private static final byte TARGET_PIN_5 = 0x1;
    private static final byte TARGET_PIN_6 = 0x5;
    private static final byte TARGET_PIN_7 = 0x6;

    private static final byte VALUE_ON = 0x1;
```

```
    private static final byte VALUE_OFF = 0x0;

    private Button ArribaButton;
    private Button AbajoButton;
    private Button DerechaButton;
    private Button IzquierdaButton;
    private SeekBar ledIntensitySeekBar;
    private SeekBar ServoSeekBar;


      @Override
      protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

        mUsbManager = UsbManager.getInstance(this);
        mPermissionIntent = PendingIntent.getBroadcast(this, 0, new
Intent(ACTION_USB_PERMISSION), 0);

        IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
        filter.addAction(UsbManager.ACTION_USB_ACCESSORY_DETACHED);
        registerReceiver(mUsbReceiver, filter);

        setContentView(R.layout.activity_main);
            ArribaButton = (Button) findViewById(R.id.button1);
            ArribaButton.setOnTouchListener(new OnTouchListener()
            {
                  @Override
                  public boolean onTouch(View arg0, MotionEvent arg1) {
                        // TODO Auto-generated method stub
                        sendLedSwitchCommand(TARGET_PIN_2,VALUE_ON);
                        return false;
                  }
            });

            ArribaButton.setOnClickListener(new OnClickListener() {

                  @Override
                  public void onClick(View v) {
                        // TODO Auto-generated method stub
                        sendLedSwitchCommand(TARGET_PIN_2,VALUE_OFF);
                  }
            });


            IzquierdaButton = (Button) findViewById(R.id.button2);
            IzquierdaButton.setOnTouchListener(new OnTouchListener()
            {
                  @Override
                  public boolean onTouch(View arg0, MotionEvent arg1) {
                        // TODO Auto-generated method stub
                        sendLedSwitchCommand(TARGET_PIN_3,VALUE_ON);
                        return false;
                  }
            });

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

```
                @Override
                public void onClick(View v) {
                        // TODO Auto-generated method stub
                        sendLedSwitchCommand(TARGET_PIN_3,VALUE_OFF);
                }
        });


        DerechaButton = (Button) findViewById(R.id.button3);
        DerechaButton.setOnTouchListener(new OnTouchListener()
        {
                @Override
                public boolean onTouch(View arg0, MotionEvent arg1) {
                        // TODO Auto-generated method stub
                        sendLedSwitchCommand(TARGET_PIN_4,VALUE_ON);
                        return false;
                }
        });

        DerechaButton.setOnClickListener(new OnClickListener() {

                @Override
                public void onClick(View v) {
                        // TODO Auto-generated method stub
                        sendLedSwitchCommand(TARGET_PIN_4,VALUE_OFF);
                }
        });

        AbajoButton = (Button) findViewById(R.id.button4);
        AbajoButton.setOnTouchListener(new OnTouchListener()
        {
                @Override
                public boolean onTouch(View arg0, MotionEvent arg1) {
                        // TODO Auto-generated method stub
                        sendLedSwitchCommand(TARGET_PIN_5,VALUE_ON);
                        return false;
                }
        });

        AbajoButton.setOnClickListener(new OnClickListener() {

                @Override
                public void onClick(View v) {
                        // TODO Auto-generated method stub
                        sendLedSwitchCommand(TARGET_PIN_5,VALUE_OFF);
                }
        });

        ledIntensitySeekBar = (SeekBar) findViewById(R.id.seekBar1);

ledIntensitySeekBar.setOnSeekBarChangeListener(ledIntensityChangeListen
er);

        ServoSeekBar = (SeekBar) findViewById(R.id.seekBar2);
        ServoSeekBar.setOnSeekBarChangeListener(ServoChangeListener);

    }
```

```
    OnSeekBarChangeListener ledIntensityChangeListener = new
OnSeekBarChangeListener() {
      @Override
      public void onProgressChanged(SeekBar seekBar, int progress,
boolean fromUser) {
            //ledIntensityTextView.setText("LED intensity: "
+ledIntensitySeekBar.getProgress());
            new AsyncTask<Byte, Void, Void>() {

                  @Override
                  protected Void doInBackground(Byte... params) {
                        sendLedSwitchCommand(TARGET_PIN_6, params[0]);
                        return null;
                  }
            }.execute((byte) progress);
      }

      @Override
      public void onStartTrackingTouch(SeekBar seekBar) {
            // not implemented         }
      }
      @Override
      public void onStopTrackingTouch(SeekBar seekBar) {
            // not implemented         }     };
      }
    };

    OnSeekBarChangeListener ServoChangeListener = new
OnSeekBarChangeListener() {
      @Override
      public void onProgressChanged(SeekBar seekBar, int progress,
boolean fromUser) {
            //ledIntensityTextView.setText("LED intensity: "
+ledIntensitySeekBar.getProgress());
            new AsyncTask<Byte, Void, Void>() {

                  @Override
                  protected Void doInBackground(Byte... params) {
                        sendLedSwitchCommand(TARGET_PIN_7, params[0]);
                        return null;
                  }
            }.execute((byte) progress);
      }

      @Override
      public void onStartTrackingTouch(SeekBar seekBar) {
            // not implemented         }
      }
      @Override
      public void onStopTrackingTouch(SeekBar seekBar) {
            // not implemented         }     };
      }
    };
```

```
	/** Called when the activity is resumed from its paused state and
immediately after onCreate(). */
	@Override
	public void onResume() {
		super.onResume();
		if (mInputStream != null && mOutputStream != null) {
			return;
		}
		UsbAccessory[] accessories =
mUsbManager.getAccessoryList();
		UsbAccessory accessory = (accessories == null ? null :
accessories[0]);
		if (accessory != null) {
			if (mUsbManager.hasPermission(accessory)) {
				openAccessory(accessory);
			} else {
				synchronized (mUsbReceiver) {
					if (!mPermissionRequestPending) {

	mUsbManager.requestPermission(accessory, mPermissionIntent);
						mPermissionRequestPending = true;
					}
				}
			}
		} else {
			Log.d(TAG, "mAccessory is null");
		}
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		// Inflate the menu; this adds items to the action bar if
it is present.
		getMenuInflater().inflate(R.menu.main, menu);
		return true;
	}
	/** Called when the activity is paused by the system. */
	@Override
	public void onPause() {
	    super.onPause();
	    closeAccessory();
	}


	/** Called when the activity is no longer needed prior to being
removed from the activity stack. */
	@Override
	public void onDestroy() {
		super.onDestroy();
		unregisterReceiver(mUsbReceiver);
	}
```

```
	private final BroadcastReceiver mUsbReceiver = new
BroadcastReceiver() {
		@Override
		public void onReceive(Context context, Intent intent) {
			String action = intent.getAction();
			if (ACTION_USB_PERMISSION.equals(action)) {
				synchronized (this) {
					UsbAccessory accessory =
UsbManager.getAccessory(intent);
					if
(intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false)) {
						openAccessory(accessory);
					} else {
						Log.d(TAG, "permission denied for
accessory " + accessory);
					}
					mPermissionRequestPending = false;
				}
			} else if
(UsbManager.ACTION_USB_ACCESSORY_DETACHED.equals(action)) {
				UsbAccessory accessory =
UsbManager.getAccessory(intent);
				if (accessory != null &&
accessory.equals(mAccessory)) {
				    closeAccessory();
			}
			}
		}
	};

	private void openAccessory(UsbAccessory accessory) {
		mFileDescriptor = mUsbManager.openAccessory(accessory);
		if (mFileDescriptor != null) {
			mAccessory = accessory;
			FileDescriptor fd =
mFileDescriptor.getFileDescriptor();
			mInputStream = new FileInputStream(fd);
			mOutputStream = new FileOutputStream(fd);


			Log.d(TAG, "accessory opened");
		} else {
			Log.d(TAG, "accessory open fail");
		}
	}

	private void closeAccessory() {
		try {
			if (mFileDescriptor != null) {
				mFileDescriptor.close();
			}
		} catch (IOException e) {
		} finally {
			mFileDescriptor = null;
			mAccessory = null;
		}
	}
```

```
    public void sendLedSwitchCommand(byte target,byte valor) {
      byte[] buffer = new byte[3];
      buffer[0] = COMMAND_LED;
      buffer[1] = target;
      buffer[2] = valor;

      if (mOutputStream != null) {
            try {
                  mOutputStream.write(buffer);
            } catch (IOException e) {
                  Log.e(TAG, "write failed", e);
            }
      }
    }

}
```

B. Código fuente programa en Arduino

```
#include <Servo.h>

#include <Max3421e.h>

#include <Usb.h>

#include <AndroidAccessory.h>




#define COMMAND_LED 0x2

#define TARGET_PIN_2 0x2

#define TARGET_PIN_3 0x3

#define TARGET_PIN_4 0x4

#define TARGET_PIN_5 0x1

#define TARGET_PIN_6 0x5
```

```
#define TARGET_PIN_7 0x6
#define VALUE_ON 0x1
#define VALUE_OFF 0x0
#define PIN 2

Servo servo1;  // Crea un Objeto servo
Servo servo2;  // Crea un Objeto servo
int posicion;    // Variable de la posicion del servo
int posicion2;    // Variable de la posicion del servo

AndroidAccessory acc("Manufacturer","Model","Description","Version","URI","Serial");

byte rcvmsg[3];
boolean bandera= false;

void setup() {
  Serial.begin(19200);
  acc.powerOn();
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  servo1.attach(4);  // Selecionamos el pin 4 como el pin de control para el servo
  servo2.attach(5);  // Selecionamos el pin 5 como el pin de control para el servo
  analogWrite(2,112);
  analogWrite(3,112);
}
```

```
void loop() {

 if (acc.isConnected()) {
  //read the received data into the byte array
  int len = acc.read(rcvmsg, sizeof(rcvmsg), 1);
  if (len > 0) {
   if (rcvmsg[0] == COMMAND_LED) {



  //-------------------adelante-------------
     if (rcvmsg[1] == TARGET_PIN_2){
      //get the switch state
      byte value = rcvmsg[2];
      //set output pin to according state
      if(value == VALUE_ON) {
       //digitalWrite(PIN, HIGH);
        analogWrite(2,255);
        analogWrite(3,255);
      }else{
        analogWrite(2,112);
        analogWrite(3,112);
      }
     }
```

```
//-------------------------atras----------
      if (rcvmsg[1] == TARGET_PIN_5){
       //get the switch state
       byte value = rcvmsg[2];
       //set output pin to according state
       if(value == VALUE_ON) {
         //digitalWrite(PIN, HIGH);
         analogWrite(2,0);
         analogWrite(3,0);
       }else{
         analogWrite(2,112);
         analogWrite(3,112);
       }
      }



//----------------------derecha-------------
      if (rcvmsg[1] == TARGET_PIN_4){
       //get the switch state
       byte value = rcvmsg[2];
       //set output pin to according state
       if(value == VALUE_ON) {
```

```
    //digitalWrite(PIN, HIGH);
    analogWrite(2,225);
    analogWrite(3,0);
   }else{
    analogWrite(2,112);
    analogWrite(3,112);
   }
  }


//----------------------izquierda------------------
  if (rcvmsg[1] == TARGET_PIN_3){
   //get the switch state
   byte value = rcvmsg[2];
   //set output pin to according state
   if(value == VALUE_ON) {
    //digitalWrite(PIN, HIGH);
    analogWrite(2,0);
    analogWrite(3,225);
   }else{
    analogWrite(2,112);
    analogWrite(3,112);
   }
  }
```

```
//-------------------------servo1--------------------

      if (rcvmsg[1] == TARGET_PIN_6){

       //get the switch state

       byte value = rcvmsg[2];

       posicion = value;          // Establecemos el valor de la posicion

       posicion = map(posicion, 0, 255, 5, 170);     // Establecemos la relacion entre los grados de giro y el PWM

       servo1.write(posicion);              // Escribimos la posicion con el mapa de valores al servo

       Serial.print(value);

       Serial.println();

       Serial.print(posicion);

       Serial.println();

       delay(20);

      }



//-------------------------servo2-----------------

      if (rcvmsg[1] == TARGET_PIN_7){

       //get the switch state

       byte value = rcvmsg[2];

       posicion2 = value;          // Establecemos el valor de la posicion

       posicion2 = map(posicion2, 0, 255, 5, 170);     // Establecemos la relacion entre los grados de giro y el PWM

       servo2.write(posicion2);              // Escribimos la posicion con el mapa de valores al servo

       Serial.print(value);
```

```
        Serial.println();
        Serial.print(posicion2);
        Serial.println();
        delay(20);
      }



    }
   }

  }
}
```

C. Diseño de placa sensor

Figura No. 28 Diseño circuito sensor

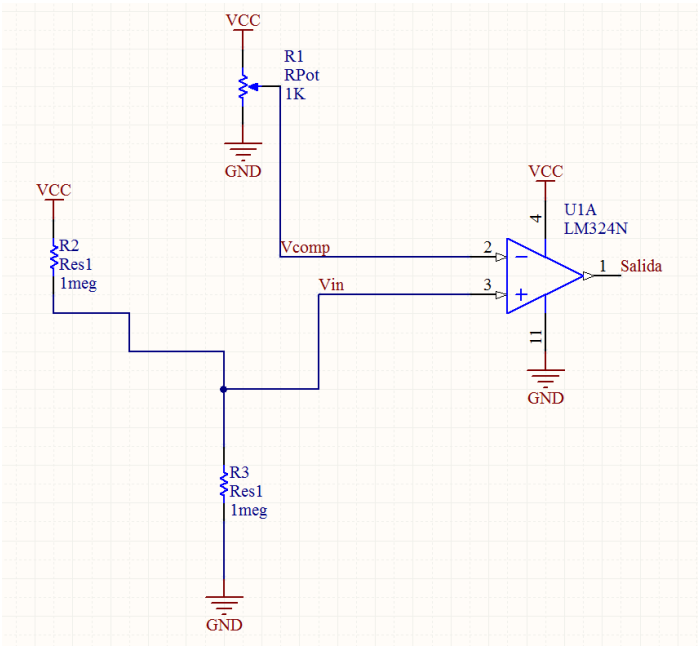


Figura No. 29 Diseño de placa sensor en circuit pro