SEGUIMIENTO DE TRAYECTORIAS EN UN ROBOT DIFERENCIAL POR MEDIO DE UN CONTROLADOR PID Y COMPÁS

Reporte de práctica 3.1

Asignatura : Robótica

Profesor: Dr. José Hugo Barrón Zambrano

Alumno: Angel Arturo Ramírez Suárez

Maestría en Ingeniería

Fecha: Martes 19 de mayo de 2015

Cd. Victoria, Tam.

Índice

ÍNDICE TEMÁTICO **2**

1. Introducción **1**

2. Desarrollo **2**
2.1. Elaboración de robot diferencial . . . 2
2.2. Control . . . 3
2.3. Etapa de potencia . . . 3
2.4. Alimentación . . . 4
2.5. Sensado . . . 4
2.6. GPS . . . 5
2.7. Codificador óptico . . . 7
2.7.1. Cálculo de desplazamiento, velocidad y aceleración . . . 7
2.8. Captura de orientación . . . 7
2.8.1. Comunicación por bluetooth . . . 9

3. Resultados **10**

4. Conclusiones **11**

A. Código fuente de captura de información de codificadores ópticos y seguimiento de trayectoria por control PID **14**

B. Código para captura de información de compás de dispositivo Android **17**
B.0.2. Actividad principal . . . 17
B.0.3. CompassView . . . 26
B.0.4. LowPassFilter . . . 27
B.1. Código para la recepción de información por bluetooth . . . 28
B.2. Accel.cpp . . . 30
B.3. Accel.h . . . 31
B.4. Motor.cpp . . . 31
B.5. Motor.h . . . 32
B.6. Navcontrol.cpp . . . 33
B.7. NavControl.h . . . 34
B.8. Ultrasonic.cpp . . . 34
B.9. Ultrasonic.h . . . 35

1. Introducción

La robótica representa una ciencia multidisciplinaria que comprende campos diversos de la ciencia y la ingeniería como son ingeniería mecánica y de materiales, electrónica, ciencia computacional y control, además de otras disciplinas como economía, inteligencia artificial o psicología que contribuyen a facilitar su objetivo, el estudio y creación de sistemas autómatas capaces de realizar diversas acciones de forma autónoma [1].

Los robots tienen una gran cantidad de aplicaciones diferentes entre las que se encuentran la manipulación de sustancias peligrosas para el ser humano o la realización de actividades en condiciones extremas (cambios de temperatura bruscos o alta radiación), además de la automatización de tareas repetititvas o de alta precisión [2].

Debido a la gran cantidad de elementos que componen a un sistema robótico, su elaboración resulta una tarea de alto costo. Esto se vuelve particularmente notable en los robots industriales o de propósitos muy específicos como es el caso de los robots médicos en los cuales se puede requerir una gran cantidad de recursos y tiempo para su elaboración y prototipado [2].

Entre las diversas aplicaciones que se da a la robótica se encuentra aplicaciones de rescate y exploración, en las cuales los robots requieren trasladarse a través de zonas peligrosas o no aptas para los seres humanos y realizar diversas tareas desde tomar muestras de elementos locales, transmitir información sobre el estado de las áreas, o en el caso de robots de rescate más complejos, transportar material delicado o seres humanos por medio de un conjunto de actuadores especializados.

Otra de las aplicaciones en las cuales la robótica y el control tienen gran uso es la industria para la manipulación de elementos peligrosos para el ser humano o la realización de tareas tediosas y repetitivas. En estas tareas los robots se destacan por su capacidad de mantenerse operando aún bajo condiciones extremas y sin sufrir los efectos de la fatiga que afectan a un ser humano.

Para la correcta navegación de los dispositivos robóticos se hace uso de sistemas de sensores que les permiten conocer su posición y orientación en el espacio. Entre éstos, los compases son utilizados con el propósito de conocer la posición absoluta de un dispositivo con respecto al eje terrestre. Esto es fundamental para los robots que requieren adquirir una orientación determinada para alcanzar un punto en el espacio.

Para la realización de esta práctica se hizo uso de la plataforma Arduino debido a las ventajas provistas por las librerías que permiten emplear lenguajes de alto nivel para interactuar con los elementos del sistema y programar los algoritmos de control.

Para la captura de información del compás se hizo uso de la plataforma Android para obtener información del dispositivo sensor compás interno de una tablet Samsung Galaxy S7 y se transmitió esta información por medio del módulo para comunicación bluetooth interno.

2. Desarrollo

2.1. Elaboración de robot diferencial

Se realizó la adquisición de una base de pruebas de un robot diferencial para la implementación de los algoritmos ilustrados. Esta base consta de dos ruedas diferenciales controladas por motores de corriente directa de 6 volts encargados de dar impulso a la plataforma, además de una rueda loca que sirve como soporte al resto de la estructura y que se ubica en la parte trasera del robot.

En la figura 1 se puede observar la plataforma adquirida en su estructura básica ensamblada y cada una de las partes que la componen.

Figura 1: Plataforma robótica adquirida.

Esta base fue modificada añadiendo una estructura superior en la cual se plantea colocar el resto de sensores y microcontrolador a utilizar en proyectos futuros. En la figura 2 se puede observar el robot con cada uno de los componentes y la segunda plataforma elaborada en cartón.

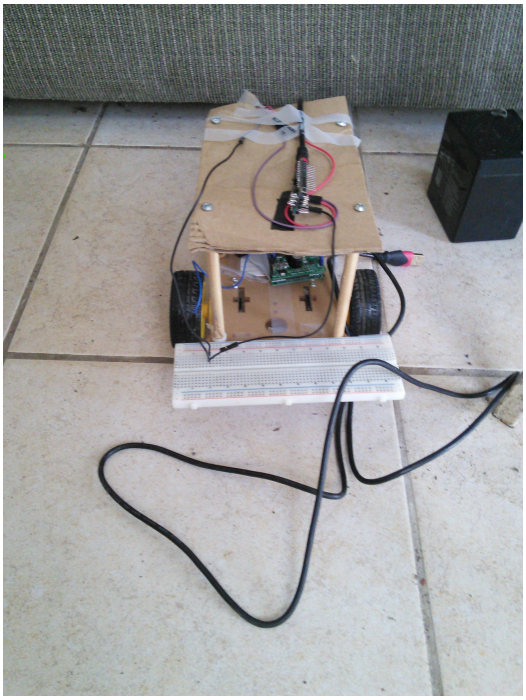


Figura 2: Robot elaborado para la implementación del controlador PID y de lazo abierto.

2.2. Control

Para el control del dispositivo se optó por implementar un microcontrolador, siendo en este caso seleccionado el dispositivo Arduino nano debido a las ventajas que ofrece respecto a dimensiones, costo y prestaciones ya que permite la implementación de señales PWM para el control de velocidad de los motores, puertos digitales para entrada y salida de señales digitales además de puertos analógicos para la captura de información por parte de los sensores instalados en la plataforma.

En la figura 3 se observa el arduino nano utilizado.

Cuadro 1: Principales características del Arduino nano

Característica	Valor
Microcontrolador	ATmega328
Voltaje de operación	5 volts
Entradas/Salidas digitales	14 con 6 de tipo PWM
Entradas analógicas	8 entradas
Corriente máxima por pin	40 mA
Memoria flash	32kB
EEPROM	1kB
Frecuencia de reloj	16 MHz

2.3. Etapa de potencia

Para la conexión entre la etapa de control compuesta por el Arduino uno en el cual se cargaron las aplicaciones, es necesario establecer una etapa de potencia encargada de suministrar la corriente requerida por los motores para operar. Para ello se utilizó una tarjeta controladora de motores de corriente directa Polulu T-REX cuyas características de operación se describen a continuación:

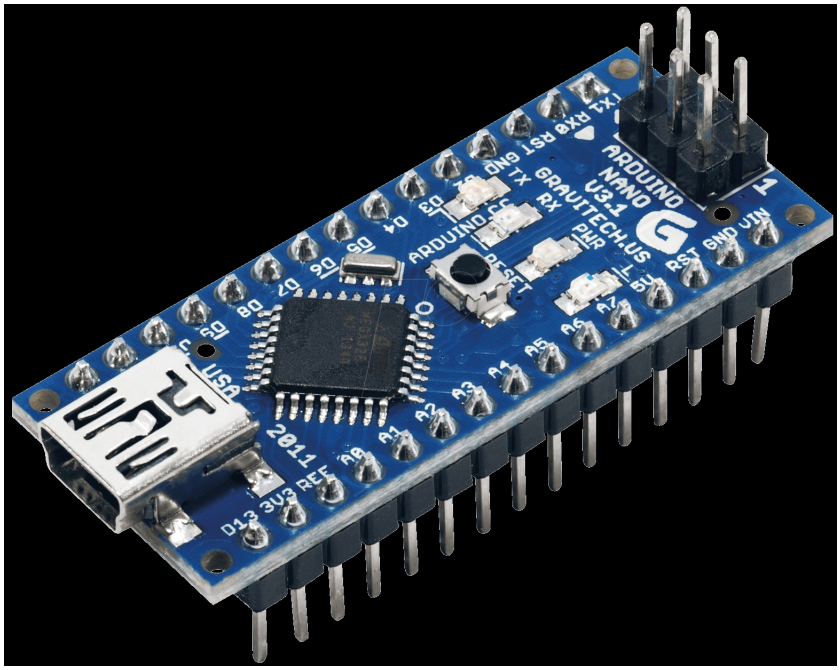


Figura 3: Microcontrolador para usos diversos Arduino nano.

Cuadro 2: Principales características del Arduino nano

Característica	Valor
Voltaje de operación	6 a 18 volts dc
Corriente máxima de operación	13 A por motor o 30 Amperes total
Puerto serial	RS-232 y TTL
Entradas analógicas	5 entradas
Entradas RC	5 entradas

Esta tarjeta que puede ser observada en la figura 4 fue elegida debido a la cantidad de potencia que es capaz de soportar y las diversas capacidades de control que permite. En este caso fue empleado el control por medio del envio de comandos por el puerto serial de la misma.

2.4. Alimentación

Para suministrar la corriente requerida para los motores se utilizó una batería recargable sellada de ácido-plomo con una capacidad de voltaje de 12 volts y 1.2 Amperes que puede observarse en la figura 5.

2.5. Sensado

Con el objetivo de mejorar la interacción del robot con el ambiente y reducir la incertidumbre en el control del mismo, se vuelve necesario el proveer al robot de la capacidad de percibir su ambiente. Para ello se hace uso de sensores que capturen información del ambiente y permitan al robot actuar en base a la información entregada por éstos.

Para la retroalimentación de la posición se hizo uso de un sensor ultrasónico modelo HC-SR04 que permite el cálculo de la distancia del robot por medio de la medición del tiempo de retorno de una señal sonora. El sensor opera al emitir un pulso a través del sensor y esperar la señal de retorno. Una vez que ha regresado,

Figura 4: Tarjeta de potencia para el control de los dos motores de corriente directa.

Figura 5: Bateria de ácido-plomo utilizada para la alimentación del robot.

calcula el tiempo de retorno de la señal, a través del cual obtiene la distancia considerando la velocidad de retorno del sonido.

En la figura 6 se puede observar el sensor utilizado para esta aplicación.

2.6. GPS

El sistema de posicionamiento global o GPS (Global Positioning System) es un sistema creado para permitir a los usuarios el contar con información respecto a la posición absoluta de un objeto relativo al eje terrestre, velocidad y tiempo de lectura por medio de la captura de información transmitida por un conjunto de satélites orbitando en la atmósfera terrestre.

Entre éstos, los más famosos son el sistema GALILEO y BAIDU que se encuentran en desarrollo por parte de la unión europea y China, además del sistema más utilizado conocido como GPSIII que continúa en desarrollo con cerca de 32 satélites [3].

Figura 6: Sensor ultrasónico HC-SR04 utilizado para la implementación del controlador PID.

Figura 7: Módulo GPS GY-GPS6MV1.

El sistema GPS opera a través de un conjunto de satélites que viajan a través de la atmósfera terrestre cuya posición es conocida dentro de un margen de tolerancia aceptado. Estos dispositivos transmiten sus coordenadas hacia la tierra en un intervalo conocido, señal que es capturada por los dispositivos pasivos en la tierra.

Los dispositivos pasivos se encargan de recibir la señal y calcular con respecto a la posición del GPS en el espacio, las coordenadas globales en x y y de los dispositivos con respecto al eje global terrestre. Esto tiene la ventaja de permitir, a diferencia del caso de acelerómetros y codificadores ópticos, el obtener una referencia absoluta que puede ser usada para mantener la posición con respecto al planeta y sin error acumulado.

El dispositivo utilizado para esta práctica es el sensor GY-GPS6MV1, el cual consiste en un módulo GPS de alta resolución que permite la captura de coordenadas absolutas con respecto al eje terrestre haciendo uso del servicio provisto por el sistema GPSIII.

En la figura 7 se puede observar el dispositivo utilizado.

Las especificaciones de operación del dispositivo se muestran en la tabla 3.

El módulo fue conectado al Arduino haciendo uso de la librería TinyGPS, la cual provee una serie de funciones abstractas que permiten simplificar el proceso de captura y cálculo de las coordenadas del dispositivo. El código implementado se puede observar en A.

Para la conexión del dispositivo se hizo uso de los pines 4 y 5 que corresponden a RX y TX respectivamente en el Arduino.

Cuadro 3: Especificaciones de operación del sensor GPS GY-GPS6MV1.

Tipo	Módulo
Uso	Universal
Dimensiones	10.2x2.5x0.8 cm
Peso neto	0.017g
Voltaje de operación	3.3 a 5 volts

2.7. Codificador óptico

Un codificador óptico es un dispositivo de sensado que permite la captura de la posición y orientación de un elemento rotatorio por medio de la transmisión de pulsos eléctricos codificados que representan el desplazamiento del elemento. Consisten en un fotodiodo que transmite un haz de fotones a través de un disco ranurado y que es recibido por un dispositivo fotoreceptor.

El desplazamiento del haz a través de las ranuras del disco permite al dispositivo el conocer por medio de la cantidad de pulsaciones obtenidas, la presencia o no de desplazamiento en el actuador. Estos dispositivos suelen ser acoplados a dispositivos actuadores angulares como es el caso de motores.

El codificador óptico y su modo de conexión se describen en la figura **??** en la cual se muestra los pines y características principales del motor de corriente directa utilizado.

Para la interacción entre la aplicación y el codificador óptico se hizo uso de la librería **Encoder.h**, la cual permite la captura y procesamiento de información por parte de los codificadores ópticos sin requerir de la interacción directa con los elementos del microcontrolador, facilitando así la tarea de su implementación.

2.7.1. Cálculo de desplazamiento, velocidad y aceleración

El codificador óptico otorga al usuario pulsos a través de los cuales es posible inferir variables como desplazamiento, velocidad y aceleración. Para ello se requiere implementar una serie de fórmulas matemáticas considerando las condiciones de operación del dispositivo.

El cálculo del desplazamiento es logrado por medio de la fórmula:

$$Distancia = dimetroderueda * (\frac{numerodepulsos}{pulsosporrevolucin}) \tag{1}$$

A partir de esto y conociendo el tiempo de desplazamiento es posible calcular la velocidad del robot por medio de la fórmula:

$$velocidad = \frac{distancia}{tiempo} \tag{2}$$

2.8. Captura de orientación

Para la captura de la orientación se hizo uso del sensor compás del dispositivo tablet Samsung Galaxy S7, el cual permite extraer información de la orientación absoluta del dispositivo con respecto al eje magnético terrestre. La información capturada por el compás fue procesada a través de un filtro pasa bandas con el objetivo de eliminar aquellos valores fuera de rango y posteriormente se aplicó operaciones de procesamiento con el objetivo de corregir el error en el dispositivo.

Estas operaciones fueron necesarias para lograr una lectura confiable de información y obtener el valor real debido a que los sensores compás siempre obtienen información relativa a el polo magnético de la tierra, el cual presenta una diferencia de 1.5 grados con respecto al polo geográfico real [4].

La aplicación de Android se puede observar en la sección B. Esta fue elaborada para inicializar el módulo de compás en el dispositivo y capturar la información de la orientación del mismo. El sistema es capaz además de obtener información respecto a las coordenadas GPS del dispositivo y la aceleración utilizando un acelerómetro interno, sin embargo esta información no fue transmitida al robot debido a que se encuentra fuera del alcance de esta práctica y a que ya se cuenta con sensores que realizan estas acciones.

Motorreductor con Encoder

Motorreductor con encoder

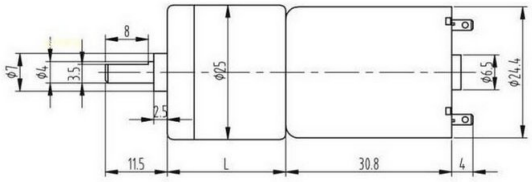
Este motorreductor cuenta con un motor de DC capaz de operar en un rango de 3V a 12V y una reducción de 75/2662 que nos multiplica el par del motor de DC en casi 36 veces y capaz de alcanzar un par de hasta 12 Kg*cm.
Su encoder incremental proporciona 334 pulsos por revolución, una cifra útil en aplicaciones donde se requiere conocer con presición la posición angular de su eje.
Su forma compacta y liviana le permite ser utilizado en robotica movil y prototipos accionados con ruedas.

Especificaciones

- Voltaje de operación de motor DC	3-12V
- Voltaje de operación de encoder	3-5V
- Pulsos por revolución	334
- Reducción	75/2662
- Par de torsión máximo	12 kg*cm
- Velocidad angular máxima	126 RPM
- Consumo @ 12V	1A

Posee 6 conexiones:

Negro:	GND Encoder
Blanco:	Salida A de encoder
Rojo	VCC Encoder
Verde:	Salida B de encoder
Naranja:	Alimentación del motor
Amrillo:	Alimentación del motor



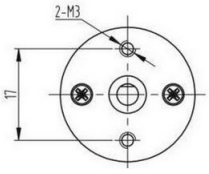


Figura 8: Principales características y modo de conexión del motor de corriente directa con codificador óptico acoplado.

Figura 9: Tableta Samsung Galaxy S7.

2.8.1. Comunicación por bluetooth

Para la transferencia de información entre el dispositivo tablet y el sistema robótico se hizo uso de un módulo de transferencia de datos por el protocolo bluetooth debido al bajo costo de implementación y la facilidad de establecer la comunicación entre los dispositivos.

Los módulos bluetooth tienden a tener un rango de hasta 100 metros y una tasa de transferencia de 3MB [5].

El sensor utilizado para la transmisión de información por bluetooth es el HC 06, el cual es un módulo de comunicación de bajo costo y bajo peso que puede ser conectado de manera sencilla al dispositivo Arduino.

En la figura 10 se puede observar el sensor y el método de conexión del mismo.

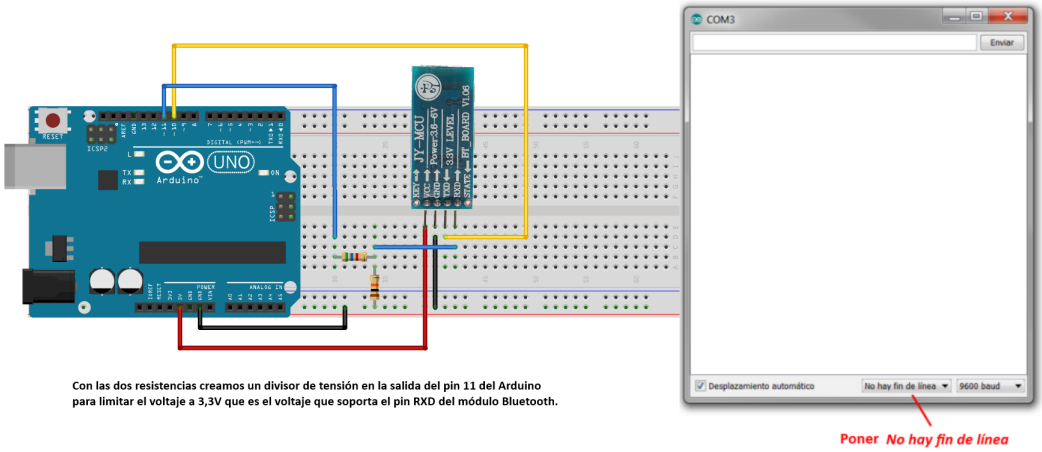


Figura 10: Módulo bluetooth HC-06 con diagrama de conexión.

El código utilizado para la transmisión en el dispositov Android realiza la recolección de la información de orientación en grados y lo transmite al dispositivo por medio del módulo HC-06 que se describe en B.1.

3. Resultados

Se implementó de forma exitosa una aplicación en Android que realiza la captura de información del compás interno de los dispositivos móviles que utilizan este sistema operativo y que transfiere esta información al robot en una tasa de transferencia de un segundo.

En la figura 11 se puede observar la interfaz de la aplicación desarrollada.

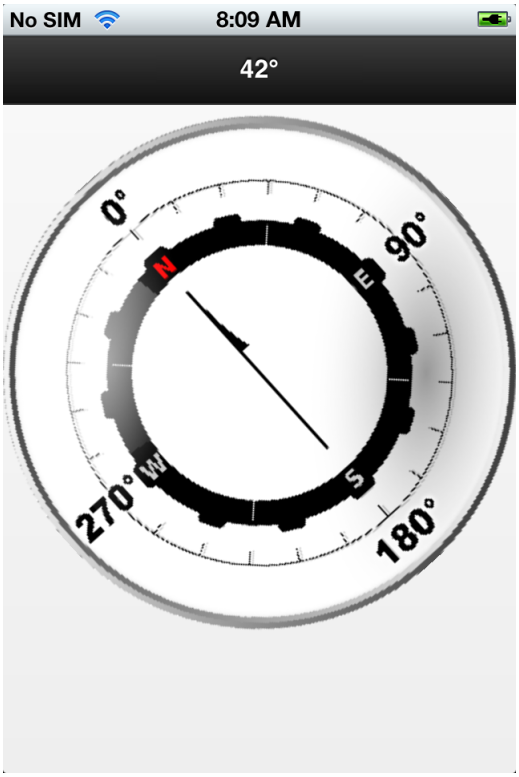


Figura 11: Aplicación de Android desarrollada para la captura de información del compás interno de los dispositivos móviles.

Se desarrolló e implementó además una aplicación utilizando la plataforma y el lenguaje de programación Arduino que permite el seguimiento de una trayectoria recta en una distancia especificada utilizando la información de dirección del compás recibida por medio de un módulo de comunicación bluetooth modelo HC-06. Este código fue puesto a prueba realizando un recorrido de 150 centímetros utilizando el robot navegador elaborado tal y como se observa en la figura 13. El resultado del recorrido final puede observarse en la figura **??** en el cual se obtuvo una media de error de 25 centímetros.

4. Conclusiones

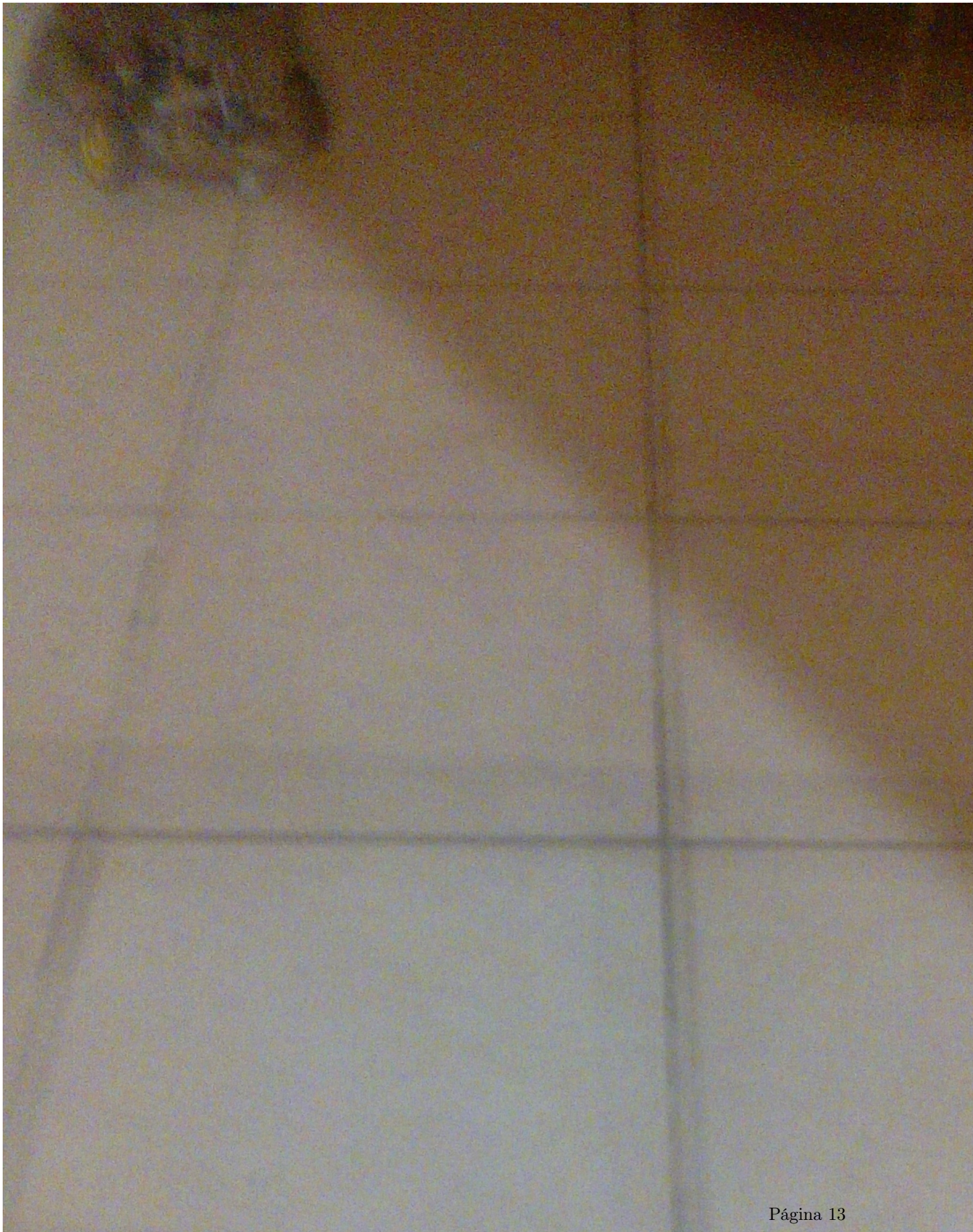
Por medio de esta práctica se elaboró e implementó una aplicación en Android que realiza la captura de información del compás del dispositivo tablet Samsung Galaxy S7 y la transmisión de la orientación en grados por medio del dispositivo bluetooth interno.

Esta información fue recibida de forma exitosa por parte del robot navegador móvil elaborado en la práctica utilizando el módulo de comunicación bluetooth HC-06. Este dispositivo recibió la información del dispositivo móvil y lo transmitió al Arduino para operar los algoritmos de control en base a la información de orientación obtenida.

A través de esto fue posible que el dispositivo mantuviera una trayectoria en línea recta en base a la información obtenida del compás. La distancia fue calculada utilizando la información que se ilustra en la Práctica 2.2 en la cual se hizo uso del codificador óptico para medir la distancia recorrida.

Como trabajo a futuro se pretende mejorar el algoritmo de captura de información en el dispositivo móvil y transferirlo a un hilo alterno al hilo principal debido a que la aplicación tiene una tendencia a congelarse durante la operación de forma aleatoria. Esto además permitirá reducir la tasa de transferencia de información en el módulo bluetooth para reducirla a valores menores a un segundo, lo cual es una limitación actual debido a que reducir el tiempo causa el fallo de la aplicación.

Respecto al algoritmo utilizado en el robot, se pretende mejorar el control implementando un controlador difuso con el objetivo de optimizar las acciones de control y reducir el sobretiro en la respuesta, caso que se presenta con el dispositivo actual.



A. Código fuente de captura de información de codificadores ópticos y seguimiento de trayectoria por control PID

```
/* CONTROLADOR DE ROBOT DIFERENCIAL.
Autor: Angel Arturo Ramirez Suarez
Asignatura: Robotica
Carrera: Maestria en Ingenieria
Universidad Politecnica de Victoria
*/
//Libreria para comunicacion serial. Solo permite una senal a la vez.
#include <SoftwareSerial.h>
//Libreria para el codificador optico.
#include <Encoder.h>
Encoder knobLeft(3, 10);
Encoder knobRight(2, 11);
long positionLeft  = 0;
long positionRight = 0;

float distanciaRecorrida = 0;
float wheelDiam = 6.5 / 100.0;
float countsPRev = 145.0;

int leftSpeed = 20;
int rightSpeed = 20;

unsigned long initTime;
float tiempo;

#define TXPIN 6
#define RXPIN 7
#define maxRange 200

float prev_error = 0, int_error = 0;
float proportional_gain = 0.5;
float integral_gain = 0.1;
float derivative_gain = 0.2;
float error = 0;
float dt = 1, diff;
int ddeseada = 20;
int P, I, D, PID;
int counter = 0;

SoftwareSerial pololu(RXPIN, TXPIN);

void setup() {
  Serial.begin (115200);
  pololu.begin(19200);
  delay(10);
  pinMode(RXPIN, INPUT);
  pinMode(TXPIN, OUTPUT);
  delay(3000);
  Serial.println("Starting");
}

void loop() {
```

```
  if(counter < 0){
    pololu.end();
  }
  else if(counter == 0){
    SetSpeed(0, true, leftSpeed);
    SetSpeed(1, true, rightSpeed);
    counter = counter + 1;
    initTime = millis();
  }
  else if(counter < 3){
    long newLeft, newRight;
    newLeft = knobLeft.read();
    newRight = knobRight.read();
    if (newLeft != positionLeft || newRight != positionRight) {
      Serial.print("Left = ");
      Serial.print(newLeft);
      Serial.print(", Right = ");
      Serial.print(newRight);
      Serial.println();
      positionLeft = newLeft;
      positionRight = newRight;

      //Correct error in left and right motors.
      error = newLeft - newRight;
      if(error < 0){
        //Right motor is moving faster.
        calcularPID(newLeft, newRight);
        leftSpeed = abs(leftSpeed+PID);
        if(leftSpeed > 50){
          leftSpeed = 30;
        }
        rightSpeed = abs(rightSpeed+PID);
        if(rightSpeed > 50){
          rightSpeed = 30;
        }
        SetSpeed(0, true, leftSpeed);
        SetSpeed(1, true, rightSpeed);
        Serial.print("PID: "); Serial.println(PID, 2);
        Serial.print("Nueva velocidad: "); Serial.println(leftSpeed);
      }
      else if(error > 0){
        //Left motor is moving faster.
        calcularPID(newLeft, newRight);
        rightSpeed = abs(rightSpeed+PID);
        if(rightSpeed > 50){
          rightSpeed = 30;
        }
        leftSpeed = abs(leftSpeed+PID);
        if(leftSpeed > 50){
          leftSpeed = 30;
        }
        SetSpeed(0, true, leftSpeed);
        SetSpeed(1, true, rightSpeed);
        Serial.print("PID: "); Serial.println(PID, 2);
        Serial.print("Nueva velocidad: "); Serial.println(rightSpeed);
      }

      //Average of both encoders to get middle counts.
      float averageCounts = (newLeft + newRight) / 2.0;
```

```
      distanciaRecorrida = (wheelDiam) * (averageCounts / countsPRev);
      Serial.print("Distancia recorrida: "); Serial.println(distanciaRecorrida, 5);

      //Calculate speeed.
      tiempo = (float)(millis() - initTime) / 1000.0;
      Serial.print("Tiempo: "); Serial.println(tiempo);
      float rspeed = distanciaRecorrida / tiempo;
      Serial.print("Velocidad: "); Serial.println(rspeed, 5);

      if(distanciaRecorrida >= 250){
        SetSpeed(0, true, 0);
        SetSpeed(1, true, 0);
        counter = -1;
      }
    }
  }
}

float calcularPID(int val1, int val2){
  error = (val1 - val2) / 300;
  if(error < 0){
        diff = (prev_error - error) / dt;
        P = proportional_gain * error;
        I = 0;
        D = derivative_gain * diff;
        prev_error = error;
        PID = P + I + D;
        return abs(PID);
    }
    else if(error > 0){
        diff = (prev_error - error) / dt;
        P = proportional_gain * error;
        I = 0;
        D = derivative_gain * diff;
        prev_error = error;
        PID = P + I + D;
        return abs(PID);
    }
}

void SetSpeed(int MotorIndex, boolean Forward, int Speed)
{
  // Validate motor index
  if(MotorIndex < 0 || MotorIndex > 2)
    return;

  // Validate speed
  if(Speed < 0)
    Speed = 0;
  else if(Speed > 127)
    Speed = 127;

  // Send the "set" command based on the motor
  // Note that we do not accelerate to the
  // speed, we just instantly set it
  unsigned char SendByte = 0;
  if(MotorIndex == 0)
    SendByte = 0xC2;
  else if(MotorIndex == 1)
```

```
      SendByte = 0xCA;
    else if(MotorIndex == 2)
      SendByte = 0xF0;

    // If we go backwards, the commands are the same
    // but minus one
    if(!Forward)
      SendByte--;

    // Send the set speed command byte
    //pololu.write(SendByte);
    pololu.write(SendByte);


    // Send the speed data byte
    pololu.write(Speed);
  }
```

B. Código para captura de información de compás de dispositivo Android

El código desarrollado permite la captura de información del sensor interno de un dispositivo Android y transmite de forma periódica esta información a través del módulo de comunicación bluetooth del dispositivo móvil. Esta información tiene una tasa de transferencia de un segundo debido a que una tasa de transferencia menor en el hilo principal del dispositivo tiende a generar el fallo de la aplicación. Se ha planteado dentro de la sección de trabajo a futuro el implementar la función de forma externa en un hilo independiente, con lo cual se espera sea posible utilizar una tasa de transferencia menor.

B.0.2. Actividad principal

```
package arthur.robotics.compass;

import java.io.IOException;
import java.io.OutputStream;
import java.lang.reflect.Method;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import java.text.NumberFormat;
import java.util.Calendar;
import java.util.TimeZone;
import java.util.UUID;
import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.hardware.GeomagneticField;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Build;
```

```
import android.os.Bundle;
import android.os.CountDownTimer;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.SystemClock;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.WindowManager;
import android.widget.TextView;
import android.widget.Toast;
//Libreria bluetooth.
import android.bluetooth.*;

public class MainActivity extends Activity implements SensorEventListener,
    LocationListener{

        public static final String NA = "N/A";
        public static final String FIXED = "FIXED";

        //Bluetooth
        private static final String TAG = "bluetooth1";
        private BluetoothAdapter bluetooth = null;
        private BluetoothSocket btSocket = null;
        private OutputStream outStream = null;
        // SPP UUID service
        private static final UUID MY_UUID = UUID.fromString("
            00001101-0000-1000-8000-00805F9B34FB");

        // MAC-address of Bluetooth module (you must edit this line)
        private static String address = "98:D3:31:40:2F:99";

        //Tiempo de localizacion minimo.
        private static final int LOCATION_MIN_TIME = 30 * 1000;
        //Distancia de localizacion minima.
        private static final int LOCATION_MIN_DISTANCE = 10;
        //Gravedad para datos de acelerometro.
        private float[] gravity = new float[3];
        //Datos magneticos.
        private float[] geomagnetic = new float[3];
        //Datos de rotacion.
        private float[] rotation = new float[9];
        //Orientacion (azimuth).
        private float[] orientation = new float[3];
        //Valores suavizados.
        private float[] smoothed = new float[3];
        //Sensor manager.
        private SensorManager sensorManager;
        //Sensor gravity.
        private Sensor sensorGravity;
        private Sensor sensorMagnetic;
        private LocationManager locationManager;
        private Location currentLocation;
        private GeomagneticField geomagneticField;
        private double bearing = 0;
        private TextView textDirection, textLat, textLong;
        private CompassView compassView;

        DecimalFormatSymbols dfs = new DecimalFormatSymbols();
```

```
        NumberFormat formatter = new DecimalFormat("#0.00", dfs);

        //Bluetooth handler.
        final Handler handler = new Handler();

        Calendar cal = Calendar.getInstance(TimeZone.getTimeZone("UTC"));
        long startTime = 0;
        long currTime = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textLat = (TextView)findViewById(R.id.latitude);
        textLong = (TextView)findViewById(R.id.longitude);
        textDirection = (TextView)findViewById(R.id.text);
        compassView = (CompassView)findViewById(R.id.compass);

        //Inicializa clase bluetooth.
        bluetooth = BluetoothAdapter.getDefaultAdapter();
        checkBTState();
        BTConnect();

        //Mantener pantalla encendida.
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
    }

    private void checkBTState() {
        if(bluetooth != null)
        {
                if (bluetooth.isEnabled()) {
                    // Enabled. Work with Bluetooth.
                Toast.makeText(getBaseContext(), "Turn on LED", Toast.LENGTH_SHORT)
                    .show();
                }
                else
                {
                    // Disabled. Do something else.
                        /*AlertDialog.Builder builder1 = new AlertDialog.Builder(
                            this);
                builder1.setMessage("Modulo bluetooth se encuentra desactivado.");
                AlertDialog alert11 = builder1.create();
                alert11.show();*/
                        Intent turnOn = new Intent(BluetoothAdapter.
                            ACTION_REQUEST_ENABLE);
                startActivityForResult(turnOn, 0);
                }
        }
    }

    private BluetoothSocket createBluetoothSocket(BluetoothDevice device) throws
        IOException {
        if(Build.VERSION.SDK_INT >= 10){
            try {
                final Method  m = device.getClass().getMethod("
                    createInsecureRfcommSocketToServiceRecord", new Class[] { UUID.
                    class });
                return (BluetoothSocket) m.invoke(device, MY_UUID);
            } catch (Exception e) {
```

```
                Log.e(TAG, "Could not create Insecure RFComm Connection",e);
            }
        }
        return  device.createRfcommSocketToServiceRecord(MY_UUID);
    }

    private void errorExit(String title, String message){
        Toast.makeText(getBaseContext(), title + " - " + message, Toast.LENGTH_LONG).show();
        finish();
      }

      private void sendData(String message) {
        byte[] msgBuffer = message.getBytes();

        Log.d(TAG, "...Send data: " + message + "...");

        try {
          outStream.write(msgBuffer);
        } catch (IOException e) {
          String msg = "In onResume() and an exception occurred during write: " + e.getMessage();
          if (address.equals("00:00:00:00:00:00"))
            msg = msg + ".\n\nUpdate your server address from 00:00:00:00:00:00 to the correct address on line 35 in the java code";
            msg = msg +  ".\n\nCheck that the SPP UUID: " + MY_UUID.toString() + " exists on server.\n\n";

            errorExit("Fatal Error", msg);
        }
      }

    @Override
    protected void onStart(){
        super.onStart();
        startTime = System.currentTimeMillis();
        sendData(bearing + "\n");

        sensorManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
        sensorGravity = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        sensorMagnetic = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);

        //Esperar datos de los sensores.
        sensorManager.registerListener(this, sensorGravity, SensorManager.SENSOR_DELAY_NORMAL);
        sensorManager.registerListener(this, sensorMagnetic, SensorManager.SENSOR_DELAY_NORMAL);

        //Solicitar datos del GPS.
        locationManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
                        LOCATION_MIN_TIME, LOCATION_MIN_DISTANCE, this);

        //Obtener ultima posicion conocida.
        Location gpsLocation = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);

```

```
        if(gpsLocation != null){
                currentLocation = gpsLocation;
        }else{
                //Intentar por medio de internet.
                Location networkLocation = locationManager.getLastKnownLocation(
                    LocationManager.NETWORK_PROVIDER);
                if(networkLocation != null){
                        currentLocation = networkLocation;
                }else{
                        //Fijar posicion.
                        currentLocation = new Location(FIXED);
                        currentLocation.setAltitude(1);
                        currentLocation.setLatitude(1000);
                        currentLocation.setLongitude(1000);
                }
        }
        //Establecer posicion.
                onLocationChanged(currentLocation);
    }

    @Override
    protected void onStop(){
        super.onStop();
        //Eliminar listeners.
        sensorManager.unregisterListener(this, sensorGravity);
        sensorManager.unregisterListener(this, sensorMagnetic);
        locationManager.removeUpdates(this);

        //Desactivar bluetooth.
        //bluetooth.disable();
        Toast.makeText(getApplicationContext(),"Turned off" ,
                Toast.LENGTH_LONG).show();
    }

    @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;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

        @Override
        public void onLocationChanged(Location location) {
                currentLocation = location;
                //Actualiza localizacion en pantalla.
                updateLocation(location);

```

```
                geomagneticField = new GeomagneticField(
                                (float)currentLocation.getLatitude(),
                                (float)currentLocation.getLongitude(),
                                (float)currentLocation.getAltitude(),
                                System.currentTimeMillis());
        }

        private void updateLocation(Location location){
                if(FIXED.equals(location.getProvider())){
                        textLat.setText("N/A");
                        textLong.setText("NA");
                }

                dfs.setDecimalSeparator('.');
                textLat.setText("Lat: " + formatter.format(location.getLatitude()))
                    ;
                textLong.setText("Lon: " + formatter.format(location.getLongitude()
                    ));
        }

         @Override
          public void onResume() {
            super.onResume();

            /*Log.d(TAG, "...onResume - try connect...");

            // Set up a pointer to the remote node using it's address.
            BluetoothDevice device = bluetooth.getRemoteDevice(address);

            // Two things are needed to make a connection:
            //   A MAC address, which we got above.
            //   A Service ID or UUID.  In this case we are using the
            //     UUID for SPP.

            try {
                btSocket = createBluetoothSocket(device);
            } catch (IOException e1) {
                errorExit("Fatal Error", "In onResume() and socket create failed: "
                     + e1.getMessage() + ".");
            }

            // Discovery is resource intensive.  Make sure it isn't going on
            // when you attempt to connect and pass your message.
            bluetooth.cancelDiscovery();

            // Establish the connection.  This will block until it connects.
            Log.d(TAG, "...Connecting...");
            try {
              btSocket.connect();
              Log.d(TAG, "...Connection ok...");
            } catch (IOException e) {
              try {
                btSocket.close();
              } catch (IOException e2) {
                errorExit("Fatal Error", "In onResume() and unable to close socket
                    during connection failure" + e2.getMessage() + ".");
              }
            }

```

```
            // Create a data stream so we can talk to server.
            Log.d(TAG, "...Create Socket...");

            try {
              outStream = btSocket.getOutputStream();
            } catch (IOException e) {
              errorExit("Fatal Error", "In onResume() and output stream creation failed:" + e.getMessage() + ".");
            }*/
          }

          @Override
          public void onPause() {
            super.onPause();

            Log.d(TAG, "...In onPause()...");

            if (outStream != null) {
              try {
                outStream.flush();
              } catch (IOException e) {
                errorExit("Fatal Error", "In onPause() and failed to flush output stream: " + e.getMessage() + ".");
              }
            }

            try     {
              btSocket.close();
            } catch (IOException e2) {
              errorExit("Fatal Error", "In onPause() and failed to close socket." + e2.getMessage() + ".");
            }
          }

        @Override
        public void onStatusChanged(String provider, int status, Bundle extras) {
                // TODO Auto-generated method stub

        }

        @Override
        public void onProviderEnabled(String provider) {
                // TODO Auto-generated method stub

        }

        @Override
        public void onProviderDisabled(String provider) {
                // TODO Auto-generated method stub

        }

        @Override
        public void onSensorChanged(SensorEvent event) {
                boolean accelOrMagnetic = false;

                //Obtener informacion de acelerometro.
                if(event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
                        //Filtro pasa bajas para suavizar informacion.
```

```
                        smoothed = LowPassFilter.filter(event.values, gravity);
                        gravity[0] = smoothed[0];
                        gravity[1] = smoothed[1];
                        gravity[2] = smoothed[2];
                        accelOrMagnetic = true;
                }else if(event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD){
                        smoothed = LowPassFilter.filter(event.values, geomagnetic);
                        geomagnetic[0] = smoothed[0];
                        geomagnetic[1] = smoothed[1];
                        geomagnetic[2] = smoothed[2];
                        accelOrMagnetic = true;
                }

                //Obtener matriz de rotacion de datos de gravedad y magneticos.
                SensorManager.getRotationMatrix(rotation, null, gravity,
                    geomagnetic);
                SensorManager.getOrientation(rotation, orientation);
                //Correccion del norte verdadero.
                bearing = orientation[0];
                //Conversion de radianes a grados.
                bearing = Math.toDegrees(bearing);
                //Correccion (FUNCIONA!).
                if(geomagneticField != null){
                        bearing += geomagneticField.getDeclination();
                }

                //Normalizacion 0 a 360 grados.
                if(bearing < 0){
                        bearing += 360;
                }

                //Actualizar compas.
                compassView.setBearing((float)bearing);

                if(accelOrMagnetic){
                        compassView.postInvalidate();
                }

                //Mostrar direccion en pantalla.
                updateTextDirection(bearing);
        }

        private void updateTextDirection(double bearing){
                int range = (int) (bearing / (360f / 16f));
                String dirTxt = "";
                if(range == 15 || range == 0)dirTxt = "N";
                if(range == 1 || range == 2)dirTxt = "NE";
                if(range == 3 || range == 4)dirTxt = "E";
                if(range == 5 || range == 6)dirTxt = "SE";
                if(range == 7 || range == 8)dirTxt = "S";
                if(range == 9 || range == 10)dirTxt = "SW";
                if(range == 11 || range == 12)dirTxt = "W";
                if(range == 13 || range == 14)dirTxt = "NW";

                //sendData("A"+String.valueOf(bearing)+"B"+String.valueOf(gravity
                    [0]));
                //sendData("A");
                currTime = System.currentTimeMillis();
                //sendData(String.valueOf((currTime - startTime) / 1000l)+"\n");
```

```
                //if(((currTime - startTime) / 1000l) > 1){
                if(((currTime - startTime) / 1000l) > 1){
                        sendData(bearing + "\n");
                        startTime = System.currentTimeMillis();
                }
                textDirection.setText("" + ((int)bearing) + ((char)176) + " " + dirTxt);
        }

        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
                if(sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD && accuracy == SensorManager.SENSOR_STATUS_UNRELIABLE){
                        //Datos de compas son poco confiables. Puede implementar filtro.
                }
        }

        public void BTConnect(){
                // Set up a pointer to the remote node using it's address.
            BluetoothDevice device = bluetooth.getRemoteDevice(address);

            // Two things are needed to make a connection:
            //   A MAC address, which we got above.
            //   A Service ID or UUID.  In this case we are using the
            //     UUID for SPP.

            try {
                btSocket = createBluetoothSocket(device);
            } catch (IOException e1) {
                errorExit("Fatal Error", "In onResume() and socket create failed: " + e1.getMessage() + ".");
            }

            // Discovery is resource intensive.  Make sure it isn't going on
            // when you attempt to connect and pass your message.
            bluetooth.cancelDiscovery();

            // Establish the connection.  This will block until it connects.
            Log.d(TAG, "...Connecting...");
            try {
              btSocket.connect();
              Log.d(TAG, "...Connection ok...");
            } catch (IOException e) {
              try {
                btSocket.close();
              } catch (IOException e2) {
                errorExit("Fatal Error", "In onResume() and unable to close socket during connection failure" + e2.getMessage() + ".");
              }
            }

            // Create a data stream so we can talk to server.
            Log.d(TAG, "...Create Socket...");

            try {
              outStream = btSocket.getOutputStream();
            } catch (IOException e) {
```

```
                errorExit("Fatal Error", "In onResume() and output stream creation failed:" + e.getMessage() + ".");
            }
        }
}
```

B.0.3. CompassView

Esta sección se encarga de establecer un compás gráfico en la aplicación con el objetivo de visualizar los elementos que operan en ésta.

```
package arthur.robotics.compass;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.view.View;

/*
 * CompassView.
 * */

public class CompassView extends View{
        private static final Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG);
        private int width = 0;
        private int height = 0;
        private Matrix matrix;  //Rotacion del compas.
        private Bitmap bitmap;
        private float bearing;  //Angulo de rotacion al norte magnetico.

        public CompassView(Context context){
                super(context);
                initialize();
        }
        public CompassView(Context context, AttributeSet attr){
                super(context, attr);
                initialize();
        }
        private void initialize(){
                matrix = new Matrix();
                //Crear bitmap para icono del compas.
                bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.icon);
        }
        public void setBearing(float b){
                bearing = b;
        }
        @Override
        protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec){
                super.onMeasure(widthMeasureSpec, heightMeasureSpec);
                width = MeasureSpec.getSize(widthMeasureSpec);
                height = MeasureSpec.getSize(heightMeasureSpec);
                setMeasuredDimension(width, height);
        }
```

```
        @Override
        protected void onDraw(Canvas canvas){
                int bitmapWidth = bitmap.getWidth();
                int bitmapHeight = bitmap.getHeight();
                int canvasWidth = canvas.getWidth();
                int canvasHeight = canvas.getHeight();

                if(bitmapWidth > canvasWidth || bitmapHeight > canvasHeight){
                        //Cambiar tamanio de icono para adecuarlo a canvas.
                        bitmap = Bitmap.createScaledBitmap(bitmap, (int) (
                            bitmapWidth*0.85),
                                        (int) (bitmapHeight*0.85), true);
                }
                //Centrar.
                int bitmapX = bitmap.getWidth() / 2;
                int bitmapY = bitmap.getHeight() / 2;
                int parentX = width / 2;
                int parentY = height / 2;
                int centerX = parentX - bitmapX;
                int centerY = parentY - bitmapY;
                //Calcular angulo de rotacion.
                int rotation = (int) (360 - bearing);
                //Resetear matriz.
                matrix.reset();
                matrix.setRotate(rotation, bitmapX, bitmapY);
                //Centrar bitmap en canvas.
                matrix.postTranslate(centerX, centerY);
                //Dibujar bitmap.
                canvas.drawBitmap(bitmap, matrix, paint);
        }
}
```

B.0.4. LowPassFilter

Por último un filtro pasa bajas se encarga de eliminar las frecuencias fuera de rango para disminuir en la medida de lo posible los errores presentes en la captura del sensor. Esto es necesario debido a que el sensor produce información errónea que de no ser eliminada puede generar comportamientos extraños en el dispositivo robótico, desde entregar una lectura que altere el algoritmo de navegación del dispositivo, hasta generar una indeterminación en un algoritmo de control y con ello causar el fallo total del sistema.

```
package arthur.robotics.compass;

public class LowPassFilter {
        private static final float alpha = 0.2f;

        private LowPassFilter(){

        }
        public static float[] filter(float[] input, float[] output){
                if(output == null)
                        return input;

                for(int i=0; i<input.length; i++){
                        output[i] = output[i] + alpha * (input[i] - output[i]);
                }
                return output;
        }
```

```
}
```

B.1. Código para la recepción de información por bluetooth

```
//Base robot de usos multiples.
/*
Autor: Angel Arturo Ramirez Suarez
Asignatura: Robotica
Carrera: Maestria en Ingenieria
Universidad Politecnica de Victoria
*/
#include <SoftwareSerial.h>

//Control de tarjeta de motores pololu.
#include "Motor.h"
Motor *motor;
#define motor_rx 6
#define motor_tx 7
//Sensor ultrasonico.
#include "Ultrasonic.h"
Ultrasonic *ultra;
#define trigPin 8
#define echoPin 9
//Codificadores opticos.
#include <Encoder.h>
#include "NavControl.h"
NavControl *leftE;
NavControl *rightE;
#define lftA 2
#define lftB 4
#define rgtA 3
#define rgtB 5
float dR = 0;
float dDeseada = 1500;
//Acelerometro MPU 6050.
#include "Accel.h"
#include<Wire.h>
Accel *ax;
//Serial pins.
int ledPin = 13;  // use the built in LED on pin 13 of the Uno
String state;
int flag = 0;        // make sure that you return the state on
float bearing = 0;
float Ang = 0;
//PID Library.
#include <PID_v1.h>

//*********************************************************
int i=0;

void setup() {
  //Inicializacion de comunicacion serial.
  Serial.begin(9600);
  delay(10);

  //Inicializacion de tarjeta de potencia.
  motor = new Motor(motor_rx,motor_tx);
```

```
  motor->startMotor();

  //Inicializacion de sensor ultrasonico.
  ultra = new Ultrasonic(trigPin, echoPin);

  //Inicializacion de codificadores opticos.
  leftE = new NavControl(lftA, lftB);
  rightE = new NavControl(rgtA, rgtB);

  //Inicializacion de acelerometro.
  ax = new Accel();

  //Alerta de inicio de operaciones.
  Serial.println("E.L.F.E.O. - Iniciando.");
  delay(1000);
}

void loop() {
  //Serial.println("En movimiento.");
  dR = leftE->calcDistance(leftE->getEncoder(), rightE->getEncoder());
  //Serial.print("Ultrasonico: "); Serial.println(ultra->getDistance(trigPin,
      echoPin));
  //Serial.print("Distancia recorrida: "); Serial.print(dR); Serial.println(" cm");
  if(i <= 0){
    if(Serial.available() > 0){
      state = Serial.readString();
      delay(1000);
      state = Serial.readString();
      state = Serial.readString();
      char carray[state.length() + 1]; //determine size of the array
      state.toCharArray(carray, sizeof(carray)); //put readStringinto an array
      bearing = atof(carray);
      Serial.print("Angulo inicial: "); Serial.println(bearing);
      i = i+1;
    }
  }
  if(i > 0){
      if(dR < dDeseada){
      float AngAccel = ax->getAccel();

      if(Serial.available() > 0){
        state = Serial.readString();
        char carray[state.length() + 1]; //determine size of the array
        state.toCharArray(carray, sizeof(carray)); //put readStringinto an array
        Ang = atof(carray);
        Serial.print("Angulo inicial: "); Serial.println(bearing);
        Serial.print("Angulo actual: "); Serial.println(Ang);
      }
      float Control = leftE->calcPID(bearing, Ang) / 100.0;
      float error = getError(abs(bearing), abs(Ang));
      Serial.print("Error: "); Serial.println(error);
      Serial.print("PID: ");Serial.println(Control, 4);
      Serial.print("Acelerometro: "); Serial.println(Ang);
      if(Control > 50){
        Control = 30;
      }
      if(Control < 5){
        Control = 15;
      }
```

```
      if(error >= 7){
        Serial.println("Derecha");
        motor->setSpeed(0, true, 8);
        motor->setSpeed(1, false, 8);
      }
      else if(error <= -7){
        Serial.println("Izquierda");
        motor->setSpeed(0, false, 8);
        motor->setSpeed(1, true, 8);
      }
      else{
        Serial.println("Adelante");
        motor->setSpeed(0, true, Control);
        motor->setSpeed(1, true, Control);
      }
    }

    if(dR > dDeseada){
      motor->setSpeed(0, true, 0);
      motor->setSpeed(1, true, 0);
      delay(100);
      motor->endMotor();
    }
    i = i+1;
  }
}

float getError(float val1, float val2){
  float error = val1 - val2;
  return error;
}
```

B.2. Accel.cpp

Este módulo se encarga de obtener información del sensor acelerómetro conectado al Arduino.

```
#include "Arduino.h"
#include "Accel.h"
#include<Wire.h>
const int MPU=0x68;  // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;

Accel::Accel(){
  Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B);  // PWR_MGMT_1 register
  Wire.write(0);     // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
}

float Accel::getAccel(){
  Wire.beginTransmission(MPU);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU,14,true);  // request a total of 14 registers
  AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
```

```
  AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
  /*Serial.print("AcX = "); Serial.print(AcX);
  Serial.print(" | AcY = "); Serial.print(AcY);
  Serial.print(" | AcZ = "); Serial.print(AcZ);
  Serial.print(" | Tmp = "); Serial.print(Tmp/340.00+36.53);  //equation for
      temperature in degrees C from datasheet
  Serial.print(" | GyX = "); Serial.print(GyX);
  Serial.print(" | GyY = "); Serial.print(GyY);
  Serial.print(" | GyZ = "); Serial.println(GyZ);*/
  delay(333);
  return GyX;
}
```

B.3. Accel.h

Declaración de la librería de acelerómetro.

```
#ifndef Accel_h
#define Accel_h
#include "Arduino.h"
#include<Wire.h>

class Accel{
  public:
    Accel();
    float getAccel();
};

#endif
```

B.4. Motor.cpp

Contiene las funciones para control de la tarjeta de potencia que moviliza los motores del robot.

```
#include "Arduino.h"
#include "Motor.h"

SoftwareSerial *pololu;

Motor::Motor(int RXPIN, int TXPIN){
  pololu = new SoftwareSerial(RXPIN, TXPIN);
  pinMode(RXPIN, INPUT);
  pinMode(TXPIN, OUTPUT);
  delay(10);
  _RXPIN = RXPIN;
  _TXPIN = TXPIN;
}

void Motor::endMotor(){
  pololu->end();
}

```

```
void Motor::startMotor(){
  pololu->begin(19200);
  delay(10);
}

void Motor::setSpeed(int MotorIndex, boolean Forward, int Speed){
  // Validate motor index
  if(MotorIndex < 0 || MotorIndex > 2)
    return;

  // Validate speed
  if(Speed < 0)
    Speed = 0;
  else if(Speed > 127)
    Speed = 127;

  // Send the "set" command based on the motor
  // Note that we do not accelerate to the
  // speed, we just instantly set it
  unsigned char SendByte = 0;
  if(MotorIndex == 0)
    SendByte = 0xC2;
  else if(MotorIndex == 1)
    SendByte = 0xCA;
  else if(MotorIndex == 2)
    SendByte = 0xF0;

  // If we go backwards, the commands are the same
  // but minus one
  if(!Forward)
    SendByte--;

  // Send the set speed command byte
  //pololu.write(SendByte);
  pololu->write(SendByte);


  // Send the speed data byte
  pololu->write(Speed);
}
```

B.5. Motor.h

Contiene la declaración de funciones de la librería de Motor.cpp.

```
#ifndef Motor_h
#define Motor_h

#include "Arduino.h"
#include <SoftwareSerial.h>

class Motor{
  public:
    Motor(int RXPIN, int TXPIN);
    void setSpeed(int MotorIndex, boolean Forward, int Speed);
    void endMotor();
    void startMotor();
```

```
  private:
    int _RXPIN;
    int _TXPIN;
};
#endif
```

B.6. Navcontrol.cpp

Contiene los algoritmos de navegación de la aplicación, incluyendo el controlador PID.

```
#include "Arduino.h"
#include "NavControl.h"
#include <Encoder.h>

Encoder *enc;

long positionLeft  = 0;
long positionRight = 0;

float distanciaRecorrida = 0;
float wheelDiam = 6.5 / 100.0;
float countsPRev = 145.0;

unsigned long initTime;
float tiempo;


float prev_error = 0, int_error = 0;
float proportional_gain = 10;
float integral_gain = 0.1;
float derivative_gain = 0.2;
float error = 0;
float dt = 1, diff;
int ddeseada = 20;
int P, I, D, PID;
int counter = 0;

NavControl::NavControl(int pinA, int pinB){
  enc = new Encoder(pinA, pinB);
}

long NavControl::getEncoder(){
  long Pulses = enc->read();
  return Pulses;
}

float NavControl::calcDistance(long pulsesLeft, long pulsesRight){
  float averageCounts = (pulsesLeft + pulsesRight) / 2.0;
  distanciaRecorrida = (wheelDiam) * (averageCounts / countsPRev);
  return distanciaRecorrida;
}

float NavControl::calcPID(int val1, int val2){
  error = (val1 - val2);// / 300;
  if(error < 0){
    diff = (prev_error - error) / dt;
    P = proportional_gain * error;
```

```
      I = 0;
      D = derivative_gain * diff;
      prev_error = error;
      PID = P + I + D;
      return abs(PID);
    }
    else if(error > 0){
      diff = (prev_error - error) / dt;
      P = proportional_gain * error;
      I = 0;
      D = derivative_gain * diff;
      prev_error = error;
      PID = P + I + D;
      return abs(PID);
    }
  }
```

B.7. NavControl.h

Contiene las funciones de navegación del módulo NavControl.cpp.

```
#ifndef NavControl_h
#define NavControl_h

#include "Arduino.h"
#include <Encoder.h>

class NavControl{
  public:
    NavControl(int pinA, int pinB);
    long getEncoder();
    float calcDistance(long pulsesLeft, long pulsesRight);
    float calcPID(int val1, int val2);
  private:
};

#endif
```

B.8. Ultrasonic.cpp

Contiene las funciones para operar el sensor ultrasónico y calcular la distancia de los objetos cercanos con respecto al robot.

```
#include "Arduino.h"
#include "Ultrasonic.h"

Ultrasonic::Ultrasonic(int trigPin, int echoPin){
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  delay(10);
  _trigPin = trigPin;
  _echoPin = echoPin;
}

float Ultrasonic::getDistance(int trigPin, int echoPin){
  long duration, distance;
```

```
  //Realiza conteo de distancia.
  digitalWrite(trigPin, LOW);  // Added this line
  delayMicroseconds(2); // Added this line
  digitalWrite(trigPin, HIGH);
  //  delayMicroseconds(1000); - Removed this line
  delayMicroseconds(10); // Added this line
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distance = (duration/2) / 29.1;
  return distance;
}
```

B.9. Ultrasonic.h

Contiene la declaración de funciones utilizada por el módulo Ultrasonic.cpp.

```
#ifndef Ultrasonic_h
#define Ultrasonic_h

#include "Arduino.h"

class Ultrasonic{
  public:
    Ultrasonic(int trigPin, int echoPin);
    float getDistance(int trigPin, int echoPin);
  private:
    int _trigPin, _echoPin;
};
#endif
```

Referencias

[1] J. Angelo, *Robotics: A Reference Guide to the New Technology.* Greenwood Press, 2007.

[2] K. Boyer, L. Stark, and H. Bunke, *Applications of AI, Machine Vision and Robotics.* Series in machine perception and artificial intelligence, World Scientific, 1995.

[3] E. Kaplan and C. Hegarty, *Understanding GPS: Principles and Applications, Second Edition.* Artech House mobile communications series, Artech House, 2005.

[4] "La diferencia entre el norte geográfico y el norte magnético." `http://www.ehowenespanol.com/diferencia-norte-geografico-norte-magnetico-info_427839/`. Visto: 2015-06-20.

[5] I. Management Association, *Networking and Telecommunications: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications.* No. v. 1-3, IGI Global, 2010.