



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

Reporte de práctica 3.2

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
2.9. Acelerómetro	9
3. Resultados	12
4. Conclusiones	13
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.0.1. Actividad principal	17
B.0.2. CompassView	26
B.0.3. 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 repetitivas 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 encuentran 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 acelerómetros son dispositivos que permiten al usuario conocer el cambio en la posición del robot en un momento determinado, lo cual puede ser utilizado con objeto de corregir el error presente en el sistema al detectar la desviación del dispositivo con respecto a la trayectoria establecida.

En esta práctica se hizo uso del sensor acelerómetro MPU-6050 que fue implementado sobre el sistema de la plataforma Arduino Nano utilizando la librería MPU provista por la empresa InvenSense. Esta librería permite acceder a las funciones del dispositivo sin necesidad de realizar programación a nivel bit, con lo cual se vuelve posible el obtener información del dispositivo de manera sencilla y sin requerir un gran número de instrucciones y su equivalente tiempo de desarrollo.

Se realizó el seguimiento de una trayectoria en línea recta y se calculó la distancia media que el sistema es capaz de navegar haciendo uso del sensor acelerómetro, con lo cual se obtuvo una idea de su operación y los requisitos para su implementación.

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.

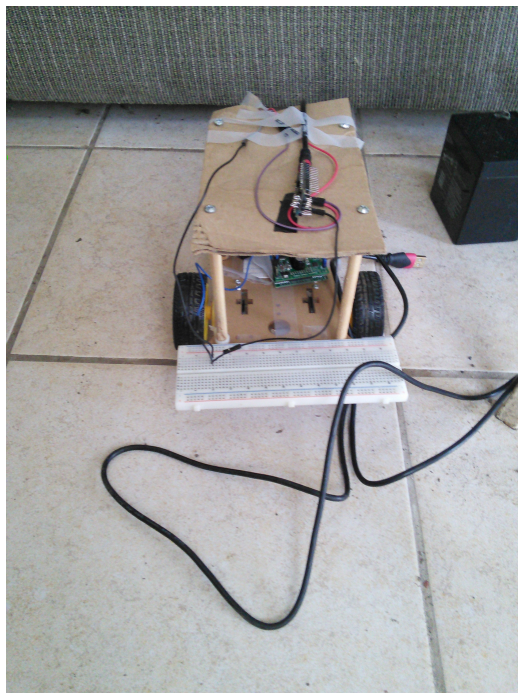


Figura 2: Robot elaborado para la implementación del controlador PID y de lazo abierto.

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:

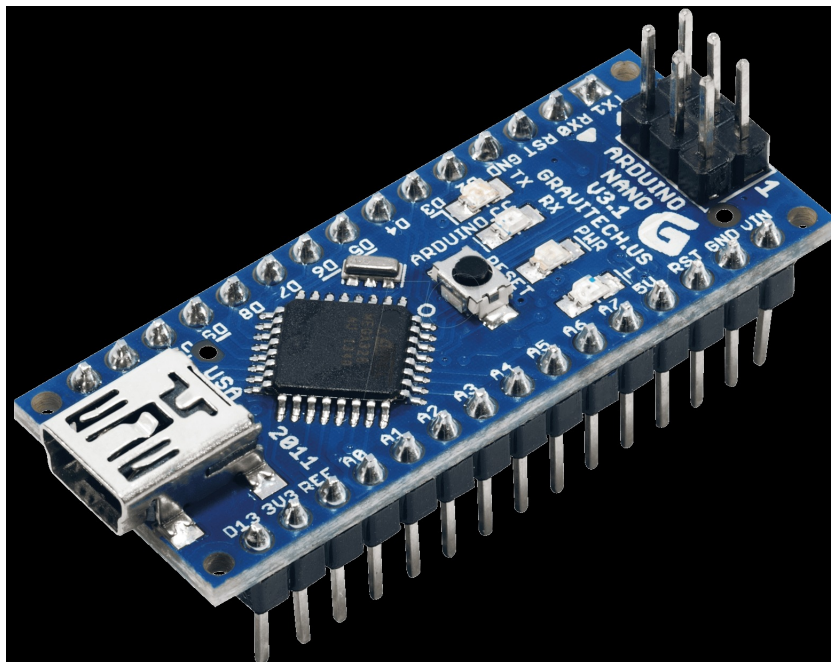


Figura 3: Microcontrolador para usos diversos Arduino nano.

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 envío 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: Batería 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:

$$Distancia = dimetroderueda * \left(\frac{numerodepulsos}{pulsosporrevolucion} \right) \quad (1)$$

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

$$velocidad = \frac{distancia}{tiempo} \quad (2)$$

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 precisión la posición angular de su eje.

Su forma compacta y liviana le permite ser utilizado en robótica móvil 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:

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

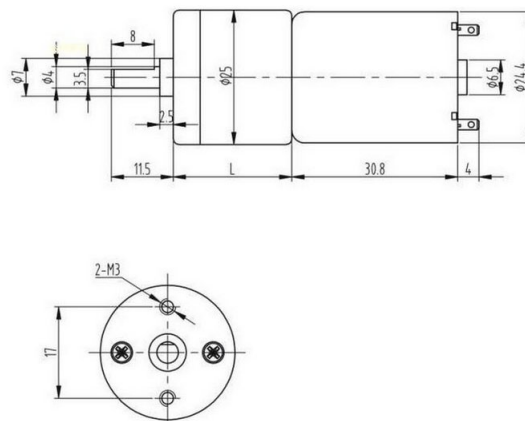


Figura 8: Principales características y modo de conexión del motor de corriente directa con codificador óptico acoplado.



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.

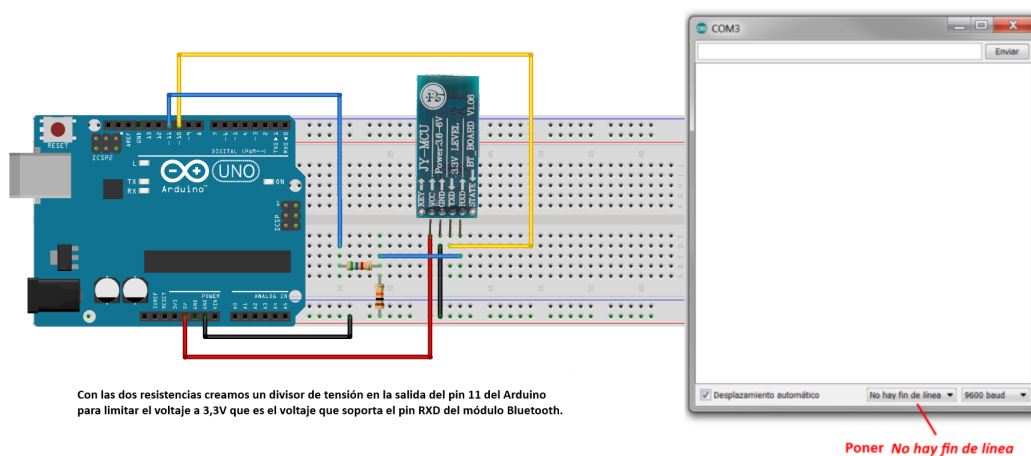


Figura 10: Módulo bluetooth HC-06 con diagrama de conexión.

2.9. Acelerómetro

Un acelerómetro es un dispositivo electromecánico utilizado para medir fuerzas de aceleración. Estas fuerzas pueden ser estáticas, como es el caso de la fuerza de gravedad, o dinámicas como es el caso del movimiento y las vibraciones [6].

Se define **aceleración** como el cambio en velocidad con respecto al tiempo y opera mediante la detección del cambio de elementos internos en el sensor por medio del efecto de la energía cinética producida durante su cambio de posición. El efecto es logrado de forma usual por medio de cristales piezoeléctricos sensibles a

los cambios en la posición. Estos cristales generan una diferencia de potencial al recibir energía cinética de forma proporcional a la velocidad con que tiene lugar este movimiento.

Este voltaje tras ser acondicionado y procesado, puede ser utilizado para conocer la aceleración del dispositivo y en base a éste, conocer otros aspectos como son la velocidad y cambio de posición del sistema.

Para la realización de esta práctica se hizo uso del sensor MPU-6050 de la empresa InvenSense, el cual consiste en un sensor acelerómetro y giroscopio de tres grados de libertad que puede ser utilizado para detectar la diferencia en posición con respecto al tiempo. En la figura 11 se puede observar el sensor utilizado [7].



Figura 11: Sensor acelerómetro MPU-6050.

El sensor fue conectado tal y como se observa en la figura 12 considerando que el Arduino Nano utiliza la siguiente configuración de pines 13. Esto debido a que no se indica en la hoja de datos oficial del Arduino Nano la configuración correcta.

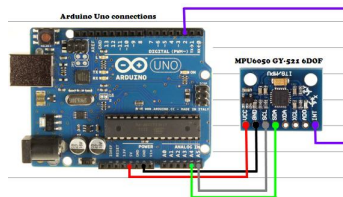


Figura 12: Conexi.

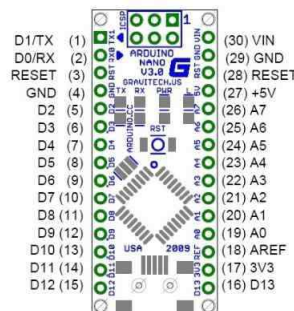


Figura 13: Configuración de pines verdadera del Arduino Nano.

El medio de comunicación del sensor es I2C, la cual es un protocolo de control que soporta hasta 100kHz de

tasa de transmisión y es comúnmente utilizado por los sistemas automotrices e industriales. Posee dos puertos SCL y SCA que permiten sincronizar la señal de reloj y realizar la transmisión de datos respectivamente.

3. Resultados

Por medio de la elaboración de la práctica se implementó de forma exitosa el sensor de aceleración MPU-6050, lográndose trasladar el robot una distancia de aproximadamente 47 centímetros de forma exitosa sin perder la dirección. A partir de este punto el robot comienza a tener un error mayor a 10 centímetros debido a que la tasa de corrección del sensor es incapaz de mantener la posición del sistema como se desea.

En la figura 14 se puede observar la posición inicial del sistema y su posición final en la figura 15.



Figura 14: Posición inicial del sistema tras la realización de la prueba..

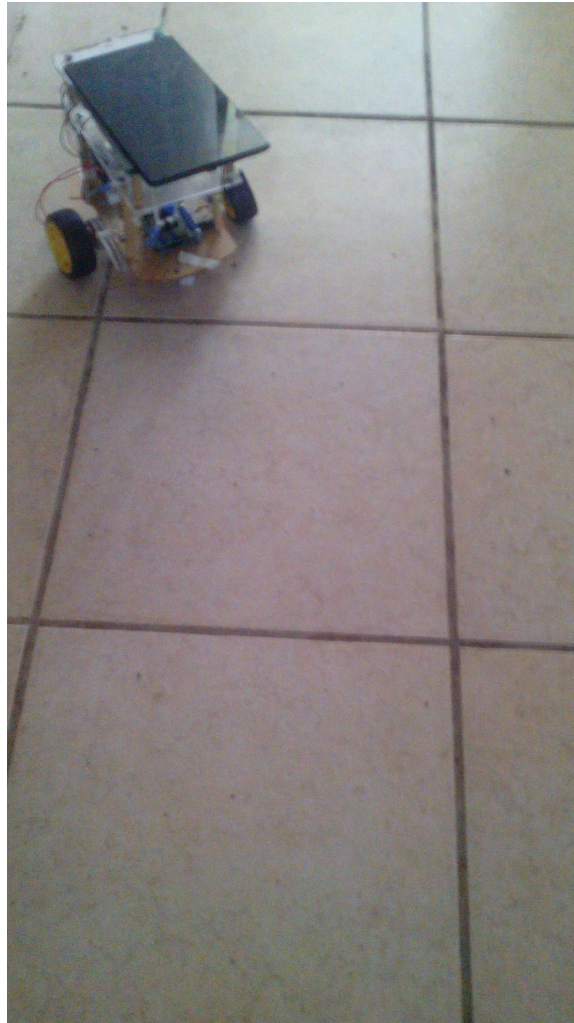


Figura 15: Posición final del sistema tras la realización de la prueba.

4. Conclusiones

Por medio de esta práctica se elaboró e implementó una aplicación en la plataforma Arduino que permite la captura de información del sensor acelerómetro y giroscopio MPU-6050 utilizando la librería creada por la empresa InvenSense. Este sistema hace uso de la detección en los cambios de posición del sistema para corregir el error producido en el caso del robot navegando una línea recta en una distancia determinada.

Como principal reto se encontró el hecho de que el sistema requiere realizar una corrección rápida en el controlador para lograr disminuir el error producido ya que al ser un sensor relativo que cambia su eje de referencia con respecto al movimiento del robot durante su ejecución, al estabilizarse la acción de control se vuelve nula y por tanto provoca que el margen de error vaya incrementándose conforme el tiempo de ejecución aumenta.

Esto lleva a la necesidad de implementar sensores absolutos con el objetivo de corregir la posición del sistema y evitar que el error continúe incrementándose como es el caso del compás, el cual permitiría corregir la orientación del navegador.

Esto se establece como objetivo a futuro para integrar los distintos sensores disponibles.

A.

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

```
1  /* CONTROLADOR DE ROBOT DIFERENCIAL.
2  Autor: Angel Arturo Ramirez Suarez
3  Asignatura: Robotica
4  Carrera: Maestria en Ingenieria
5  Universidad Politecnica de Victoria
6  */
7  //Libreria para comunicacion serial. Solo permite una senal a la vez.
8  #include <SoftwareSerial.h>
9  //Libreria para el codificador optico.
10 #include <Encoder.h>
11 Encoder knobLeft(3, 10);
12 Encoder knobRight(2, 11);
13 long positionLeft = 0;
14 long positionRight = 0;
15
16 float distanciaRecorrida = 0;
17 float wheelDiam = 6.5 / 100.0;
18 float countsPRev = 145.0;
19
20 int leftSpeed = 20;
21 int rightSpeed = 20;
22
23 unsigned long initTime;
24 float tiempo;
25
26 #define TXPIN 6
27 #define RXPIN 7
28 #define maxRange 200
29
30 float prev_error = 0, int_error = 0;
31 float proportional_gain = 0.5;
32 float integral_gain = 0.1;
33 float derivative_gain = 0.2;
34 float error = 0;
35 float dt = 1, diff;
36 int ddeseada = 20;
37 int P, I, D, PID;
38 int counter = 0;
39
40 SoftwareSerial pololu(RXPIN, TXPIN);
41
42 void setup() {
43     Serial.begin (115200);
44     pololu.begin(19200);
45     delay(10);
46     pinMode(RXPIN, INPUT);
47     pinMode(TXPIN, OUTPUT);
48     delay(3000);
49     Serial.println("Starting");
50 }
51
52 void loop() {
```

```

53   if(counter < 0){
54       pololu.end();
55   }
56   else if(counter == 0){
57       SetSpeed(0, true, leftSpeed);
58       SetSpeed(1, true, rightSpeed);
59       counter = counter + 1;
60       initTime = millis();
61   }
62   else if(counter < 3){
63       long newLeft, newRight;
64       newLeft = knobLeft.read();
65       newRight = knobRight.read();
66       if (newLeft != positionLeft || newRight != positionRight) {
67           Serial.print("Left = ");
68           Serial.print(newLeft);
69           Serial.print(", Right = ");
70           Serial.print(newRight);
71           Serial.println();
72           positionLeft = newLeft;
73           positionRight = newRight;
74
75           //Correct error in left and right motors.
76           error = newLeft - newRight;
77           if(error < 0){
78               //Right motor is moving faster.
79               calcularPID(newLeft, newRight);
80               leftSpeed = abs(leftSpeed+PID);
81               if(leftSpeed > 50){
82                   leftSpeed = 30;
83               }
84               rightSpeed = abs(rightSpeed+PID);
85               if(rightSpeed > 50){
86                   rightSpeed = 30;
87               }
88               SetSpeed(0, true, leftSpeed);
89               SetSpeed(1, true, rightSpeed);
90               Serial.print("PID: "); Serial.println(PID, 2);
91               Serial.print("Nueva velocidad: "); Serial.println(leftSpeed);
92           }
93           else if(error > 0){
94               //Left motor is moving faster.
95               calcularPID(newLeft, newRight);
96               rightSpeed = abs(rightSpeed+PID);
97               if(rightSpeed > 50){
98                   rightSpeed = 30;
99               }
100              leftSpeed = abs(leftSpeed+PID);
101              if(leftSpeed > 50){
102                  leftSpeed = 30;
103              }
104              SetSpeed(0, true, leftSpeed);
105              SetSpeed(1, true, rightSpeed);
106              Serial.print("PID: "); Serial.println(PID, 2);
107              Serial.print("Nueva velocidad: "); Serial.println(rightSpeed);
108          }
109
110          //Average of both encoders to get middle counts.
111          float averageCounts = (newLeft + newRight) / 2.0;
  
```

```

112     distanciaRecorrida = (wheelDiam) * (averageCounts / countsPRev);
113     Serial.print("Distancia recorrida: "); Serial.println(distanciaRecorrida, 5);
114
115     //Calculate speeded.
116     tiempo = (float)(millis() - initTime) / 1000.0;
117     Serial.print("Tiempo: "); Serial.println(tiempo);
118     float rspeed = distanciaRecorrida / tiempo;
119     Serial.print("Velocidad: "); Serial.println(rspeed, 5);
120
121     if(distanciaRecorrida >= 250){
122         SetSpeed(0, true, 0);
123         SetSpeed(1, true, 0);
124         counter = -1;
125     }
126 }
127 }
128 }
129
130 float calcularPID(int val1, int val2){
131     error = (val1 - val2) / 300;
132     if(error < 0){
133         diff = (prev_error - error) / dt;
134         P = proportional_gain * error;
135         I = 0;
136         D = derivative_gain * diff;
137         prev_error = error;
138         PID = P + I + D;
139         return abs(PID);
140     }
141     else if(error > 0){
142         diff = (prev_error - error) / dt;
143         P = proportional_gain * error;
144         I = 0;
145         D = derivative_gain * diff;
146         prev_error = error;
147         PID = P + I + D;
148         return abs(PID);
149     }
150 }
151
152 void SetSpeed(int MotorIndex, boolean Forward, int Speed)
153 {
154     // Validate motor index
155     if(MotorIndex < 0 || MotorIndex > 2)
156         return;
157
158     // Validate speed
159     if(Speed < 0)
160         Speed = 0;
161     else if(Speed > 127)
162         Speed = 127;
163
164     // Send the "set" command based on the motor
165     // Note that we do not accelerate to the
166     // speed, we just instantly set it
167     unsigned char SendByte = 0;
168     if(MotorIndex == 0)
169         SendByte = 0xC2;
170     else if(MotorIndex == 1)
  
```

```

171     SendByte = 0xCA;
172   else if(MotorIndex == 2)
173     SendByte = 0xF0;
174
175   // If we go backwards, the commands are the same
176   // but minus one
177   if(!Forward)
178     SendByte--;
179
180   // Send the set speed command byte
181   //pololu.write(SendByte);
182   pololu.write(SendByte);
183
184
185   // Send the speed data byte
186   pololu.write(Speed);
187 }

```

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.0.1. Actividad principal

```

1  package arthur.robotics.compass;
2
3  import java.io.IOException;
4  import java.io.OutputStream;
5  import java.lang.reflect.Method;
6  import java.text.DecimalFormat;
7  import java.text.DecimalFormatSymbols;
8  import java.text.NumberFormat;
9  import java.util.Calendar;
10 import java.util.TimeZone;
11 import java.util.UUID;
12 import java.util.concurrent.TimeUnit;
13
14 import android.app.Activity;
15 import android.content.Context;
16 import android.content.Intent;
17 import android.hardware.GeomagneticField;
18 import android.hardware.Sensor;
19 import android.hardware.SensorEvent;
20 import android.hardware.SensorEventListener;
21 import android.hardware.SensorManager;
22 import android.location.Location;
23 import android.location.LocationListener;
24 import android.location.LocationManager;
25 import android.os.Build;

```



```
26 import android.os.Bundle;
27 import android.os.CountDownTimer;
28 import android.os.Handler;
29 import android.os.HandlerThread;
30 import android.os.SystemClock;
31 import android.util.Log;
32 import android.view.Menu;
33 import android.view.MenuItem;
34 import android.view.WindowManager;
35 import android.widget.TextView;
36 import android.widget.Toast;
37 //Libreria bluetooth.
38 import android.bluetooth.*;
39
40 public class MainActivity extends Activity implements SensorEventListener,
    LocationListener{
41
42     public static final String NA = "N/A";
43     public static final String FIXED = "FIXED";
44
45     //Bluetooth
46     private static final String TAG = "bluetooth1";
47     private BluetoothAdapter bluetooth = null;
48     private BluetoothSocket btSocket = null;
49     private OutputStream outputStream = null;
50     // SPP UUID service
51     private static final UUID MY_UUID = UUID.fromString("
        00001101-0000-1000-8000-00805F9B34FB");
52
53     // MAC-address of Bluetooth module (you must edit this line)
54     private static String address = "98:D3:31:40:2F:99";
55
56     //Tiempo de localizacion minimo.
57     private static final int LOCATION_MIN_TIME = 30 * 1000;
58     //Distancia de localizacion minima.
59     private static final int LOCATION_MIN_DISTANCE = 10;
60     //Gravedad para datos de acelerometro.
61     private float[] gravity = new float[3];
62     //Datos magneticos.
63     private float[] geomagnetic = new float[3];
64     //Datos de rotacion.
65     private float[] rotation = new float[9];
66     //Orientacion (azimuth).
67     private float[] orientation = new float[3];
68     //Valores suavizados.
69     private float[] smoothed = new float[3];
70     //Sensor manager.
71     private SensorManager sensorManager;
72     //Sensor gravity.
73     private Sensor sensorGravity;
74     private Sensor sensorMagnetic;
75     private LocationManager locationManager;
76     private Location currentLocation;
77     private GeomagneticField geomagneticField;
78     private double bearing = 0;
79     private TextView textDirection, textLat, textLong;
80     private CompassView compassView;
81
82     DecimalFormatSymbols dfs = new DecimalFormatSymbols();
```



```

83     NumberFormat formatter = new DecimalFormat("#0.00", dfs);
84
85     //Bluetooth handler.
86     final Handler handler = new Handler();
87
88     Calendar cal = Calendar.getInstance(TimeZone.getTimeZone("UTC"));
89     long startTime = 0;
90     long currTime = 0;
91
92     @Override
93     protected void onCreate(Bundle savedInstanceState) {
94         super.onCreate(savedInstanceState);
95         setContentView(R.layout.activity_main);
96         textLat = (TextView)findViewById(R.id.latitude);
97         textLong = (TextView)findViewById(R.id.longitude);
98         textDirection = (TextView)findViewById(R.id.text);
99         compassView = (CompassView)findViewById(R.id.compass);
100
101         //Inicializa clase bluetooth.
102         bluetooth = BluetoothAdapter.getDefaultAdapter();
103         checkBTState();
104         BTConnect();
105
106         //Mantener pantalla encendida.
107         getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
108     }
109
110     private void checkBTState() {
111         if(bluetooth != null)
112         {
113             if (bluetooth.isEnabled()) {
114                 // Enabled. Work with Bluetooth.
115                 Toast.makeText(getApplicationContext(), "Turn on LED", Toast.LENGTH_SHORT)
116                     .show();
117             }
118             else
119             {
120                 // Disabled. Do something else.
121                 /*AlertDialog.Builder builder1 = new AlertDialog.Builder(
122                     this);
123                 builder1.setMessage("Modulo bluetooth se encuentra desactivado.");
124                 AlertDialog alert11 = builder1.create();
125                 alert11.show();*/
126                 Intent turnOn = new Intent(BluetoothAdapter.
127                     ACTION_REQUEST_ENABLE);
128                 startActivityForResult(turnOn, 0);
129             }
130         }
131     }
132
133     private BluetoothSocket createBluetoothSocket(BluetoothDevice device) throws
134     IOException {
135         if(Build.VERSION.SDK_INT >= 10){
136             try {
137                 final Method m = device.getClass().getMethod("
138                     createInsecureRfcommSocketToServiceRecord", new Class[] { UUID.
139                         class });
140                 return (BluetoothSocket) m.invoke(device, MY_UUID);
141             } catch (Exception e) {

```

```

136         Log.e(TAG, "Could not create Insecure RFComm Connection",e);
137     }
138 }
139 return device.createRfcommSocketToServiceRecord(MY_UUID);
140 }
141
142 private void errorExit(String title, String message){
143     Toast.makeText(getApplicationContext(), title + " - " + message, Toast.LENGTH_LONG
144         ).show();
145     finish();
146 }
147
148 private void sendData(String message) {
149     byte[] msgBuffer = message.getBytes();
150
151     Log.d(TAG, "...Send data: " + message + "...");
152
153     try {
154         outputStream.write(msgBuffer);
155     } catch (IOException e) {
156         String msg = "In onResume() and an exception occurred during write: " + e
157             .getMessage();
158         if (address.equals("00:00:00:00:00:00"))
159             msg = msg + ".\n\nUpdate your server address from 00:00:00:00:00:00 to
160                 the correct address on line 35 in the java code";
161         msg = msg + ".\n\nCheck that the SPP UUID: " + MY_UUID.toString() + "
162             exists on server.\n\n";
163
164         errorExit("Fatal Error", msg);
165     }
166 }
167
168 @Override
169 protected void onStart(){
170     super.onStart();
171     startTime = System.currentTimeMillis();
172     sendData(bearing + "\n");
173
174     sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
175     sensorGravity = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
176     sensorMagnetic = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)
177         ;
178
179     //Esperar datos de los sensores.
180     sensorManager.registerListener(this, sensorGravity, SensorManager.
181         SENSOR_DELAY_NORMAL);
182     sensorManager.registerListener(this, sensorMagnetic, SensorManager.
183         SENSOR_DELAY_NORMAL);
184
185     //Solicitar datos del GPS.
186     locationManager = (LocationManager) getSystemService(Context.
187         LOCATION_SERVICE);
188     locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
189         LOCATION_MIN_TIME, LOCATION_MIN_DISTANCE, this);
190
191     //Obtener ultima posicion conocida.
192     Location gpsLocation = locationManager.getLastKnownLocation(LocationManager
193         .GPS_PROVIDER);

```

```

186         if(gpsLocation != null){
187             currentLocation = gpsLocation;
188         }else{
189             //Intentar por medio de internet.
190             Location networkLocation = locationManager.getLastKnownLocation(
191                 locationManager.NETWORK_PROVIDER);
192             if(networkLocation != null){
193                 currentLocation = networkLocation;
194             }else{
195                 //Fijar posicion.
196                 currentLocation = new Location(FIXED);
197                 currentLocation.setAltitude(1);
198                 currentLocation.setLatitude(1000);
199                 currentLocation.setLongitude(1000);
200             }
201             //Establecer posicion.
202             locationManager.onLocationChanged(currentLocation);
203         }
204
205         @Override
206         protected void onStop(){
207             super.onStop();
208             //Eliminar listeners.
209             sensorManager.unregisterListener(this, sensorGravity);
210             sensorManager.unregisterListener(this, sensorMagnetic);
211             locationManager.removeUpdates(this);
212
213             //Desactivar bluetooth.
214             //bluetooth.disable();
215             Toast.makeText(getApplicationContext(),"Turned off" ,
216                 Toast.LENGTH_LONG).show();
217         }
218
219         @Override
220         public boolean onCreateOptionsMenu(Menu menu) {
221             // Inflate the menu; this adds items to the action bar if it is present.
222             getMenuInflater().inflate(R.menu.main, menu);
223             return true;
224         }
225
226         @Override
227         public boolean onOptionsItemSelected(MenuItem item) {
228             // Handle action bar item clicks here. The action bar will
229             // automatically handle clicks on the Home/Up button, so long
230             // as you specify a parent activity in AndroidManifest.xml.
231             int id = item.getItemId();
232             if (id == R.id.action_settings) {
233                 return true;
234             }
235             return super.onOptionsItemSelected(item);
236         }
237
238         @Override
239         public void onLocationChanged(Location location) {
240             currentLocation = location;
241             //Actualiza localizacion en pantalla.
242             updateLocation(location);
243         }
  
```

```

244         geomagneticField = new GeomagneticField(
245             (float)currentLocation.getLatitude(),
246             (float)currentLocation.getLongitude(),
247             (float)currentLocation.getAltitude(),
248             System.currentTimeMillis());
249     }
250
251     private void updateLocation(Location location){
252         if(FIXED.equals(location.getProvider())){
253             textLat.setText("N/A");
254             textLong.setText("NA");
255         }
256
257         dfs.setDecimalSeparator('.');
258         textLat.setText("Lat: " + formatter.format(location.getLatitude()))
259         ;
260         textLong.setText("Lon: " + formatter.format(location.getLongitude()
261         ));
262     }
263
264     @Override
265     public void onResume() {
266         super.onResume();
267
268         /*Log.d(TAG, "...onResume - try connect...");
269
270         // Set up a pointer to the remote node using it's address.
271         BluetoothDevice device = bluetooth.getRemoteDevice(address);
272
273         // Two things are needed to make a connection:
274         //   A MAC address, which we got above.
275         //   A Service ID or UUID. In this case we are using the
276         //   UUID for SPP.
277
278         try {
279             btSocket = createBluetoothSocket(device);
280         } catch (IOException e1) {
281             errorExit("Fatal Error", "In onResume() and socket create failed: "
282                 + e1.getMessage() + ".");
283         }
284
285         // Discovery is resource intensive. Make sure it isn't going on
286         // when you attempt to connect and pass your message.
287         bluetooth.cancelDiscovery();
288
289         // Establish the connection. This will block until it connects.
290         Log.d(TAG, "...Connecting...");
291         try {
292             btSocket.connect();
293             Log.d(TAG, "...Connection ok...");
294         } catch (IOException e) {
295             try {
296                 btSocket.close();
297             } catch (IOException e2) {
298                 errorExit("Fatal Error", "In onResume() and unable to close socket
299                     during connection failure" + e2.getMessage() + ".");
300             }
301         }
302     }
  
```

```

299         // Create a data stream so we can talk to server.
300         Log.d(TAG, "...Create Socket...");
301
302         try {
303             outputStream = btSocket.getOutputStream();
304         } catch (IOException e) {
305             errorExit("Fatal Error", "In onResume() and output stream creation
306                 failed:" + e.getMessage() + ".");
307         }*/
308     }
309
310     @Override
311     public void onPause() {
312         super.onPause();
313
314         Log.d(TAG, "...In onPause()...");
315
316         if (outStream != null) {
317             try {
318                 outputStream.flush();
319             } catch (IOException e) {
320                 errorExit("Fatal Error", "In onPause() and failed to flush output
321                     stream: " + e.getMessage() + ".");
322             }
323         }
324
325         try {
326             btSocket.close();
327         } catch (IOException e2) {
328             errorExit("Fatal Error", "In onPause() and failed to close socket." +
329                 e2.getMessage() + ".");
330         }
331     }
332
333     @Override
334     public void onStatusChanged(String provider, int status, Bundle extras) {
335         // TODO Auto-generated method stub
336     }
337
338     @Override
339     public void onProviderEnabled(String provider) {
340         // TODO Auto-generated method stub
341     }
342
343     @Override
344     public void onProviderDisabled(String provider) {
345         // TODO Auto-generated method stub
346     }
347
348     @Override
349     public void onSensorChanged(SensorEvent event) {
350         boolean accelOrMagnetic = false;
351
352         //Obtener informacion de acelerometro.
353         if(event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
354             //Filtro pasa bajas para suavizar informacion.

```

```

355         smoothed = LowPassFilter.filter(event.values, gravity);
356         gravity[0] = smoothed[0];
357         gravity[1] = smoothed[1];
358         gravity[2] = smoothed[2];
359         accelOrMagnetic = true;
360     }else if(event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD){
361         smoothed = LowPassFilter.filter(event.values, geomagnetic);
362         geomagnetic[0] = smoothed[0];
363         geomagnetic[1] = smoothed[1];
364         geomagnetic[2] = smoothed[2];
365         accelOrMagnetic = true;
366     }
367
368     //Obtener matriz de rotacion de datos de gravedad y magneticos.
369     SensorManager.getRotationMatrix(rotation, null, gravity,
370         geomagnetic);
371     SensorManager.getOrientation(rotation, orientation);
372     //Correccion del norte verdadero.
373     bearing = orientation[0];
374     //Conversion de radianes a grados.
375     bearing = Math.toDegrees(bearing);
376     //Correccion (FUNCIONA!).
377     if(geomagneticField != null){
378         bearing += geomagneticField.getDeclination();
379     }
380
381     //Normalizacion 0 a 360 grados.
382     if(bearing < 0){
383         bearing += 360;
384     }
385
386     //Actualizar compas.
387     compassView.setBearing((float)bearing);
388
389     if(accelOrMagnetic){
390         compassView.postInvalidate();
391     }
392
393     //Mostrar direccion en pantalla.
394     updateTextDirection(bearing);
395 }
396
397 private void updateTextDirection(double bearing){
398     int range = (int) (bearing / (360f / 16f));
399     String dirTxt = "";
400     if(range == 15 || range == 0)dirTxt = "N";
401     if(range == 1 || range == 2)dirTxt = "NE";
402     if(range == 3 || range == 4)dirTxt = "E";
403     if(range == 5 || range == 6)dirTxt = "SE";
404     if(range == 7 || range == 8)dirTxt = "S";
405     if(range == 9 || range == 10)dirTxt = "SW";
406     if(range == 11 || range == 12)dirTxt = "W";
407     if(range == 13 || range == 14)dirTxt = "NW";
408
409     //sendData("A"+String.valueOf(bearing)+"B"+String.valueOf(gravity
410         [0]));
411     //sendData("A");
412     currTime = System.currentTimeMillis();
413     //sendData(String.valueOf((currTime - startTime) / 1000l)+"\n");
  
```

```

412         //if(((currTime - startTime) / 10001) > 1){
413         if(((currTime - startTime) / 10001) > 1){
414             sendData(bearing + "\n");
415             startTime = System.currentTimeMillis();
416         }
417         textDirection.setText("" + ((int)bearing) + ((char)176) + " " +
            dirTxt);
418     }
419
420     @Override
421     public void onAccuracyChanged(Sensor sensor, int accuracy) {
422         if(sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD && accuracy ==
423             SensorManager.SENSOR_STATUS_UNRELIABLE){
424             //Datos de compas son poco confiables. Puede implementar
425                 filtro.
426         }
427     }
428
429     public void BTConnect(){
430         // Set up a pointer to the remote node using it's address.
431         BluetoothDevice device = bluetooth.getRemoteDevice(address);
432
433         // Two things are needed to make a connection:
434         //   A MAC address, which we got above.
435         //   A Service ID or UUID. In this case we are using the
436         //       UUID for SPP.
437
438         try {
439             btSocket = createBluetoothSocket(device);
440         } catch (IOException e1) {
441             errorExit("Fatal Error", "In onResume() and socket create failed: "
442                 + e1.getMessage() + ".");
443         }
444
445         // Discovery is resource intensive. Make sure it isn't going on
446         // when you attempt to connect and pass your message.
447         bluetooth.cancelDiscovery();
448
449         // Establish the connection. This will block until it connects.
450         Log.d(TAG, "...Connecting...");
451         try {
452             btSocket.connect();
453             Log.d(TAG, "...Connection ok...");
454         } catch (IOException e) {
455             try {
456                 btSocket.close();
457             } catch (IOException e2) {
458                 errorExit("Fatal Error", "In onResume() and unable to close socket
459                     during connection failure" + e2.getMessage() + ".");
460             }
461         }
462
463         // Create a data stream so we can talk to server.
464         Log.d(TAG, "...Create Socket...");
465
466         try {
467             outputStream = btSocket.getOutputStream();
468         } catch (IOException e) {
469

```

```

466         errorExit("Fatal Error", "In onResume() and output stream creation
467             failed:" + e.getMessage() + ".");
468     }
469 }

```

B.0.2. CompassView

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

```

1  package arthur.robotics.compass;
2
3  import android.content.Context;
4  import android.graphics.Bitmap;
5  import android.graphics.BitmapFactory;
6  import android.graphics.Canvas;
7  import android.graphics.Matrix;
8  import android.graphics.Paint;
9  import android.util.AttributeSet;
10 import android.view.View;
11
12 /*
13  * CompassView.
14  * */
15
16 public class CompassView extends View{
17     private static final Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG);
18     private int width = 0;
19     private int height = 0;
20     private Matrix matrix; //Rotacion del compas.
21     private Bitmap bitmap;
22     private float bearing; //Angulo de rotacion al norte magnetico.
23
24     public CompassView(Context context){
25         super(context);
26         initialize();
27     }
28     public CompassView(Context context, AttributeSet attr){
29         super(context, attr);
30         initialize();
31     }
32     private void initialize(){
33         matrix = new Matrix();
34         //Crear bitmap para icono del compas.
35         bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.
36             icon);
37     }
38     public void setBearing(float b){
39         bearing = b;
40     }
41     @Override
42     protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec){
43         super.onMeasure(widthMeasureSpec, heightMeasureSpec);
44         width = MeasureSpec.getSize(widthMeasureSpec);
45         height = MeasureSpec.getSize(heightMeasureSpec);
46         setMeasuredDimension(width, height);
47     }

```



```

47     @Override
48     protected void onDraw(Canvas canvas){
49         int bitmapWidth = bitmap.getWidth();
50         int bitmapHeight = bitmap.getHeight();
51         int canvasWidth = canvas.getWidth();
52         int canvasHeight = canvas.getHeight();
53
54         if(bitmapWidth > canvasWidth || bitmapHeight > canvasHeight){
55             //Cambiar tamaño de icono para adecuarlo a canvas.
56             bitmap = Bitmap.createScaledBitmap(bitmap, (int) (
57                 bitmapWidth*0.85),
58                 (int) (bitmapHeight*0.85), true);
59         }
60         //Centrar.
61         int bitmapX = bitmap.getWidth() / 2;
62         int bitmapY = bitmap.getHeight() / 2;
63         int parentX = width / 2;
64         int parentY = height / 2;
65         int centerX = parentX - bitmapX;
66         int centerY = parentY - bitmapY;
67         //Calcular ángulo de rotación.
68         int rotation = (int) (360 - bearing);
69         //Resetear matriz.
70         matrix.reset();
71         matrix.setRotate(rotation, bitmapX, bitmapY);
72         //Centrar bitmap en canvas.
73         matrix.postTranslate(centerX, centerY);
74         //Dibujar bitmap.
75         canvas.drawBitmap(bitmap, matrix, paint);
76     }
  
```

B.0.3. 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.

```

1  package arthur.robotics.compass;
2
3  public class LowPassFilter {
4      private static final float alpha = 0.2f;
5
6      private LowPassFilter(){
7
8      }
9      public static float[] filter(float[] input, float[] output){
10         if(output == null)
11             return input;
12
13         for(int i=0; i<input.length; i++){
14             output[i] = output[i] + alpha * (input[i] - output[i]);
15         }
16         return output;
17     }
  
```

18 }

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

```

1 //Base robot de usos multiples.
2 /*
3 Autor: Angel Arturo Ramirez Suarez
4 Asignatura: Robotica
5 Carrera: Maestria en Ingenieria
6 Universidad Politecnica de Victoria
7 */
8 #include <SoftwareSerial.h>
9
10 //Control de tarjeta de motores pololu.
11 #include "Motor.h"
12 Motor *motor;
13 #define motor_rx 6
14 #define motor_tx 7
15 //Sensor ultrasonico.
16 #include "Ultrasonic.h"
17 Ultrasonic *ultra;
18 #define trigPin 8
19 #define echoPin 9
20 //Codificadores opticos.
21 #include <Encoder.h>
22 #include "NavControl.h"
23 NavControl *leftE;
24 NavControl *rightE;
25 #define lftA 2
26 #define lftB 4
27 #define rgtA 3
28 #define rgtB 5
29 float dR = 0;
30 float dDeseada = 1500;
31 //Acelerometro MPU 6050.
32 #include "Accel.h"
33 #include <Wire.h>
34 Accel *ax;
35 //Serial pins.
36 int ledPin = 13; // use the built in LED on pin 13 of the Uno
37 String state;
38 int flag = 0; // make sure that you return the state on
39 float bearing = 0;
40 float Ang = 0;
41 //PID Library.
42 #include <PID_v1.h>
43
44 // *****
45 int i=0;
46
47 void setup() {
48   //Inicializacion de comunicacion serial.
49   Serial.begin(9600);
50   delay(10);
51
52   //Inicializacion de tarjeta de potencia.
53   motor = new Motor(motor_rx,motor_tx);
  
```

```

54  motor->startMotor();
55
56  //Inicializacion de sensor ultrasonico.
57  ultra = new Ultrasonic(trigPin, echoPin);
58
59  //Inicializacion de codificadores opticos.
60  leftE = new NavControl(lftA, lftB);
61  rightE = new NavControl(rgtA, rgtB);
62
63  //Inicializacion de acelerometro.
64  ax = new Accel();
65
66  //Alerta de inicio de operaciones.
67  Serial.println("E.L.F.E.O. - Iniciando.");
68  delay(1000);
69  }
70
71  void loop() {
72    //Serial.println("En movimiento.");
73    dR = leftE->calcDistance(leftE->getEncoder(), rightE->getEncoder());
74    //Serial.print("Ultrasonico: "); Serial.println(ultra->getDistance(trigPin,
75    echoPin));
76    //Serial.print("Distancia recorrida: "); Serial.print(dR); Serial.println(" cm");
77    if(i <= 0){
78      if(Serial.available() > 0){
79        state = Serial.readString();
80        delay(1000);
81        state = Serial.readString();
82        state = Serial.readString();
83        char carray[state.length() + 1]; //determine size of the array
84        state.toCharArray(carray, sizeof(carray)); //put readStringinto an array
85        bearing = atof(carray);
86        Serial.print("Angulo inicial: "); Serial.println(bearing);
87        i = i+1;
88      }
89    }
90    if(i > 0){
91      if(dR < dDeseada){
92        float AngAccel = ax->getAccel();
93
94        if(Serial.available() > 0){
95          state = Serial.readString();
96          char carray[state.length() + 1]; //determine size of the array
97          state.toCharArray(carray, sizeof(carray)); //put readStringinto an array
98          Ang = atof(carray);
99          Serial.print("Angulo inicial: "); Serial.println(bearing);
100          Serial.print("Angulo actual: "); Serial.println(Ang);
101        }
102        float Control = leftE->calcPID(bearing, Ang) / 100.0;
103        float error = getError(abs(bearing), abs(Ang));
104        Serial.print("Error: "); Serial.println(error);
105        Serial.print("PID: ");Serial.println(Control, 4);
106        Serial.print("Acelerometro: "); Serial.println(Ang);
107        if(Control > 50){
108          Control = 30;
109        }
110        if(Control < 5){
111          Control = 15;
112        }
113      }
114    }
115  }

```

```

112     if(error >= 7){
113         Serial.println("Derecha");
114         motor->setSpeed(0, true, 8);
115         motor->setSpeed(1, false, 8);
116     }
117     else if(error <= -7){
118         Serial.println("Izquierda");
119         motor->setSpeed(0, false, 8);
120         motor->setSpeed(1, true, 8);
121     }
122     else{
123         Serial.println("Adelante");
124         motor->setSpeed(0, true, Control);
125         motor->setSpeed(1, true, Control);
126     }
127 }
128
129 if(dR > dDeseada){
130     motor->setSpeed(0, true, 0);
131     motor->setSpeed(1, true, 0);
132     delay(100);
133     motor->endMotor();
134 }
135 i = i+1;
136 }
137 }
138
139 float getError(float val1, float val2){
140     float error = val1 - val2;
141     return error;
142 }
  
```

B.2. Accel.cpp

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

```

1  #include "Arduino.h"
2  #include "Accel.h"
3  #include<Wire.h>
4  const int MPU=0x68;  // I2C address of the MPU-6050
5  int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
6
7  Accel::Accel(){
8      Wire.begin();
9      Wire.beginTransmission(MPU);
10     Wire.write(0x6B);  // PWR_MGMT_1 register
11     Wire.write(0);      // set to zero (wakes up the MPU-6050)
12     Wire.endTransmission(true);
13 }
14
15 float Accel::getAccel(){
16     Wire.beginTransmission(MPU);
17     Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
18     Wire.endTransmission(false);
19     Wire.requestFrom(MPU,14,true);  // request a total of 14 registers
20     AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
21     AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  
```

```

22   AcZ=Wire.read()<<8|Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
23   Tmp=Wire.read()<<8|Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
24   GyX=Wire.read()<<8|Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
25   GyY=Wire.read()<<8|Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
26   GyZ=Wire.read()<<8|Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
27   /*Serial.print("AcX = "); Serial.print(AcX);
28   Serial.print(" | AcY = "); Serial.print(AcY);
29   Serial.print(" | AcZ = "); Serial.print(AcZ);
30   Serial.print(" | Tmp = "); Serial.print(Tmp/340.00+36.53); //equation for
      temperature in degrees C from datasheet
31   Serial.print(" | GyX = "); Serial.print(GyX);
32   Serial.print(" | GyY = "); Serial.print(GyY);
33   Serial.print(" | GyZ = "); Serial.println(GyZ);*/
34   delay(333);
35   return GyX;
36 }

```

B.3. Accel.h

Declaración de la librería de acelerómetro.

```

1  #ifndef Accel_h
2  #define Accel_h
3  #include "Arduino.h"
4  #include<Wire.h>
5
6  class Accel{
7  public:
8      Accel();
9      float getAccel();
10 };
11
12 #endif

```

B.4. Motor.cpp

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

```

1  #include "Arduino.h"
2  #include "Motor.h"
3
4  SoftwareSerial *pololu;
5
6  Motor::Motor(int RXPIN, int TXPIN){
7      pololu = new SoftwareSerial(RXPIN, TXPIN);
8      pinMode(RXPIN, INPUT);
9      pinMode(TXPIN, OUTPUT);
10     delay(10);
11     _RXPIN = RXPIN;
12     _TXPIN = TXPIN;
13 }
14
15 void Motor::endMotor(){
16     pololu->end();
17 }
18

```

```

19 void Motor::startMotor(){
20     pololu->begin(19200);
21     delay(10);
22 }
23
24 void Motor::setSpeed(int MotorIndex, boolean Forward, int Speed){
25     // Validate motor index
26     if(MotorIndex < 0 || MotorIndex > 2)
27         return;
28
29     // Validate speed
30     if(Speed < 0)
31         Speed = 0;
32     else if(Speed > 127)
33         Speed = 127;
34
35     // Send the "set" command based on the motor
36     // Note that we do not accelerate to the
37     // speed, we just instantly set it
38     unsigned char SendByte = 0;
39     if(MotorIndex == 0)
40         SendByte = 0xC2;
41     else if(MotorIndex == 1)
42         SendByte = 0xCA;
43     else if(MotorIndex == 2)
44         SendByte = 0xF0;
45
46     // If we go backwards, the commands are the same
47     // but minus one
48     if(!Forward)
49         SendByte--;
50
51     // Send the set speed command byte
52     //pololu.write(SendByte);
53     pololu->write(SendByte);
54
55
56     // Send the speed data byte
57     pololu->write(Speed);
58 }
  
```

B.5. Motor.h

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

```

1  #ifndef Motor_h
2  #define Motor_h
3
4  #include "Arduino.h"
5  #include <SoftwareSerial.h>
6
7  class Motor{
8  public:
9      Motor(int RXPIN, int TXPIN);
10     void setSpeed(int MotorIndex, boolean Forward, int Speed);
11     void endMotor();
12     void startMotor();
  
```

```

13     private:
14         int _RXPIN;
15         int _TXPIN;
16     };
17 #endif

```

B.6. Navcontrol.cpp

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

```

1  #include "Arduino.h"
2  #include "NavControl.h"
3  #include <Encoder.h>
4
5  Encoder *enc;
6
7  long positionLeft = 0;
8  long positionRight = 0;
9
10 float distanciaRecorrida = 0;
11 float wheelDiam = 6.5 / 100.0;
12 float countsPRev = 145.0;
13
14 unsigned long initTime;
15 float tiempo;
16
17
18 float prev_error = 0, int_error = 0;
19 float proportional_gain = 10;
20 float integral_gain = 0.1;
21 float derivative_gain = 0.2;
22 float error = 0;
23 float dt = 1, diff;
24 int ddeseada = 20;
25 int P, I, D, PID;
26 int counter = 0;
27
28 NavControl::NavControl(int pinA, int pinB){
29     enc = new Encoder(pinA, pinB);
30 }
31
32 long NavControl::getEncoder(){
33     long Pulses = enc->read();
34     return Pulses;
35 }
36
37 float NavControl::calcDistance(long pulsesLeft, long pulsesRight){
38     float averageCounts = (pulsesLeft + pulsesRight) / 2.0;
39     distanciaRecorrida = (wheelDiam) * (averageCounts / countsPRev);
40     return distanciaRecorrida;
41 }
42
43 float NavControl::calcPID(int val1, int val2){
44     error = (val1 - val2); // / 300;
45     if(error < 0){
46         diff = (prev_error - error) / dt;
47         P = proportional_gain * error;

```

```

48     I = 0;
49     D = derivative_gain * diff;
50     prev_error = error;
51     PID = P + I + D;
52     return abs(PID);
53   }
54   else if(error > 0){
55     diff = (prev_error - error) / dt;
56     P = proportional_gain * error;
57     I = 0;
58     D = derivative_gain * diff;
59     prev_error = error;
60     PID = P + I + D;
61     return abs(PID);
62   }
63 }

```

B.7. NavControl.h

Contiene las funciones de navegación del módulo NavControl.cpp.

```

1  #ifndef NavControl_h
2  #define NavControl_h
3
4  #include "Arduino.h"
5  #include <Encoder.h>
6
7  class NavControl{
8  public:
9     NavControl(int pinA, int pinB);
10    long getEncoder();
11    float calcDistance(long pulsesLeft, long pulsesRight);
12    float calcPID(int val1, int val2);
13  private:
14  };
15
16  #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.

```

1  #include "Arduino.h"
2  #include "Ultrasonic.h"
3
4  Ultrasonic::Ultrasonic(int trigPin, int echoPin){
5     pinMode(trigPin, OUTPUT);
6     pinMode(echoPin, INPUT);
7     delay(10);
8     _trigPin = trigPin;
9     _echoPin = echoPin;
10 }
11
12 float Ultrasonic::getDistance(int trigPin, int echoPin){
13     long duration, distance;

```



```
14 //Realiza conteo de distancia.
15 digitalWrite(trigPin, LOW); // Added this line
16 delayMicroseconds(2); // Added this line
17 digitalWrite(trigPin, HIGH);
18 // delayMicroseconds(1000); - Removed this line
19 delayMicroseconds(10); // Added this line
20 digitalWrite(trigPin, LOW);
21 duration = pulseIn(echoPin, HIGH);
22 distance = (duration/2) / 29.1;
23 return distance;
24 }
```

B.9. Ultrasonic.h

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

```
1 #ifndef Ultrasonic_h
2 #define Ultrasonic_h
3
4 #include "Arduino.h"
5
6 class Ultrasonic{
7 public:
8     Ultrasonic(int trigPin, int echoPin);
9     float getDistance(int trigPin, int echoPin);
10 private:
11     int _trigPin, _echoPin;
12 };
13 #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.
- [6] A. Allan, *Basic Sensors in IOS: Programming the Accelerometer, Gyroscope, and More*. Real Time Bks, O'Reilly Media, 2011.
- [7] "Mpu-6050 accelerometer + gyro." <http://playground.arduino.cc/Main/MPU-6050>. Visto: 2015-06-20.