EIE

Escuela de
Ingeniería Eléctrica

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

UNIVERSIDAD DE COSTA RICA

`IE-0117` Programación bajo plataformas abiertas

Proyecto Final: AuVeTA, Autonomous Vehicle for Tracking and Control Aplications.

Jorge Isaac Fallas Mejía B62562
Esteban Rodríguez Quintana B66076
Fabio Villalobos Pacheco B78346

I-2019

Tabla de contenidos

1. Reseña del programa

El proyecto AuVeTA consiste en diseñar y poner en funcionamiento un vehículo que pueda ser utilizado como una base para prototipado de un vehículo que a futuro pueda utilizarse en aplicaciones de seguimiento de linea para algún tipo de aplicación industrial o inclusive evolucionar al mundo del reconocimiento de patrones y la visión por computador para generar una propuesta de vehículos realmente autónomos y con aplicación enfocada a la movilidad humana.

Este proyecto consiste en el desarrollo de un vehículo semi autónomo que, mediante la implementación de microcontroladores, sensores, un módulo de comunicación y programación, pueda cumplir con el objetivo de seguir una línea de color en el suelo y llegar a un punto donde culmine con la ejecución de una tarea determinada. Parte de las funciones que se desean incorporar a este prototipo, además del seguimiento de línea y remoción de obstáculos, es la comunicación del vehículo con un centro de control, desde el cual también se pueda conducir a discreción el mismo, con la finalidad de ser utilizado en otro tipo de aplicaciones mediante sus distintos modos de operación.

El proyecto consta de dos partes principales: *hardware* y *software*. El desarrollo e integración de estos dos subsistemas han de ir de la mano, puesto que un error en uno de ellos ha de limitar la acción correcta del otro.

2. Funcionamiento del programa

2.1. Funcionamiento de Software

El código se implementará en dos etapas, la primera consiste en una etapa de control y la segunda en una etapa de seguimiento, ambas son completamente independientes la una de la otra.

- **Seguimiento**: La etapa de seguimiento implementa el sensor de línea, la idea es regular el funcionamiento de los motores de acuerdo a la corrección que ha de ser necesario para mantener el vehículo encima de la línea a seguir, y garantizar de esta forma seguimiento del trazo deseado. Toda la implementación de este código se hará programada en el IDE de Arduino puesto que al ser la etapa semi autónoma del proyecto no requiere de ningún comando desde un centro de control para la toma de decisiones. Esta etapa de seguimiento implementa 3 sensores, uno para reconocer la línea a seguir, y dos sensores de obstáculo, uno de ellos encargado de detectar obstáculos frente al vehículo para su remoción, y el otro sensor para reconocer cuando se llega al final del recorrido.

- **Control**: La etapa de control se encarga de manejar el funcionamiento de los motores de manera remota mediante un módulo bluetooth, la idea es enviar datos desde un programa en C a través del puerto serial. Dependiendo de qué dato sea enviado, el vehículo llevará a cabo distintos movimientos. El programa en C cuenta con distintas funciones que permiten controlar el puerto serial, al cual se conecta el módulo bluetooth, son dos funciones en total. La primera se encarga de inicializar el puerto serial, una vez inicializado se puede utilizar la otra función, que se utiliza para escribir en el puerto serial, de esa forma se establece una comunicación unidireccional. Dichas funciones serán implementadas en una interfaz gráfica que simula un control remoto, la idea es controlar el vehículo con el teclado, y con la interfaz visualizar los movimientos que realiza el vehículo.

2.1.1. Interfaz

La interfaz utilizada en este proyecto utiliza una herramienta llamada GTK, más específicamente GTK+-2.0, la cual es una multiplataforma de herramientas que permite crear interfaces de gráficas de uso para el usuario. Este ofrece un set completo de widgets, estos siendo pequeñas aplicaciones con accesos directos permitiendo la sencilla visualización de las funciones principales utilizadas en determinado programa o documento. También es adaptable a proyectos que van desde pequeñas herramientas hasta aplicaciones muy completas permitiendo su funcionalidad. Se usa en diferentes sistemas operativos tales como Windows, GNU/Linux and Unix, OSX e inclusive dispositivos móviles. Escrito en lenguaje C, y permitido además en C++, Perl y Python. [9]

- **Función para crear las características del botón:** Esta es la función tipo Gtkwidget que recibe que la tabla o matriz interna se establecerá en la ventana de la interfaz que contendrá los botones, un dato char que es el tamaño de los botones, además dos datos de tipo int que son la posición en fila y columna de los botones, luego esta hace un gtk button new with label que es el botón, además del color, de la posicion en la interfaz y gtk widget show que hace que retorne todas esas características del botón.

```
1
2 GtkWidget *CreateButton(GtkWidget *table, char *szLabel, int row, int column)
3 {
4         GtkWidget *button;
5
```

```
6           button = gtk_button_new_with_label(szLabel);
7
8           gtk_signal_connect(GTK_OBJECT(button), "clicked",
9                              GTK_SIGNAL_FUNC(button_clicked), szLabel);
10
11          GdkColor color;
12
13          gdk_color_parse("red", &color);
14
15          gtk_widget_modify_bg(GTK_WIDGET(button), GTK_STATE_NORMAL, &color);
16
17          gtk_table_attach(GTK_TABLE(table), button,
18                           column, column + 1,
19                           row, row + 1,
20                           GTK_FILL | GTK_EXPAND,
21                           GTK_FILL | GTK_EXPAND,
22                           10, 15);
23
24          gtk_widget_show(button);
25
26          return (button);
27  }
```

- **Función que crea el botón en la interfaz:** Esta función es de tipo void que recibe la tabla o matriz interna, se establecerá en la ventana de la interfaz que contendrá los botones, luego se crea un dato que hace que el ciclo for recorra la lista de los botones y con ayuda de la función CreateButton se cree el botón y aparezca en la interfaz.

```
1   void CreateControlButtons(GtkWidget *table)
2   {
3           int nIndex;
4
5           for (nIndex = 0; nIndex < numbuttons; nIndex++)
6           {
7
8                   buttonList[nIndex].widget =
9                       CreateButton(table,
10                                   buttonList[nIndex].szLabel,
11                                   buttonList[nIndex].row,
12                                   buttonList[nIndex].col);
13          }
14  }
```

- **Función para poder usar los botones del teclado:** Esta función es de tipo void que recibe un widget, un evento y la información del teclado, hace uso de un ciclo for que da busqueda entre los botones, y una condición if que permite si la pulsación de la tecla es el primer carácter de un botón y la longitud de la etiqueta del botón es de 1, entonces permite el retorno que conecta el botón presionado en el teclado con la interfaz gráfica y la tarea asociada.

```
1   void key_press(GtkWidget *widget,
2                  GdkEventKey *event,
3                  gpointer data)
4   {
5           int npressed;
6
```

```
        for (npressed = 0; npressed < numbuttons; npressed++)
        {
                if (event->keyval == buttonList[npressed].szLabel[0] &&
                    buttonList[npressed].szLabel[1] == (char)0)
                {
                        gtk_widget_grab_focus(buttonList[npressed].widget);
                        gtk_button_clicked(GTK_BUTTON(buttonList[npressed].
                            widget));
                        return;
                }
        }
}
```

- **Función para clickear el botón y realizar una tarea:** Es la función de tipo void que recibe un widget y la información, este contiene dos datos tipo int los cuales son el baud rate y el puerto, dos punteros de tipo char, el str se encarga de agarrar el texto del botón y la información, además contiene la ventana que contiene la interfaz, luego varias condiciones if que permiten comparar el botón clickeado o presionado en el teclado que conecta con el puerto serial y retorna la tarea que realiza el vehículo.

```
void button_clicked(GtkWidget *widget, gpointer data)
{
        int baudrate = 115200;
        int port;
        port = serialport_init("/dev/rfcomm0", baudrate);

        char ch = *((char *)data);
        char *str;
        GtkWidget *window;
        window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

        str = (char *)data;

        if (strcmp(str, "w") == 0)
        {

                char a = 'w';
                const char *ptr = &a;
                serialport_write(port, ptr);

                return;
        }
        else if (strcmp(str, "a") == 0)
        {

                char a = 'a';
                const char *ptr = &a;
                serialport_write(port, ptr);

                return;
        }
        else if (strcmp(str, "d") == 0)
        {

                char a = 'd';
                const char *ptr = &a;
                serialport_write(port, ptr);
```

```
38
39                  return;
40          }
41          else if (strcmp(str, "s") == 0)
42          {
43
44                  char a = 's';
45                  const char *ptr = &a;
46                  serialport_write(port, ptr);
47
48                  return;
49          }
50          else if (strcmp(str, "1") == 0)
51          {
52
53                  char a = '1';
54                  const char *ptr = &a;
55                  serialport_write(port, ptr);
56
57                  gtk_label_set(GTK_LABEL(label), "Control Mode");
58                  return;
59          }
60          else if (strcmp(str, "2") == 0)
61          {
62
63                  char a = '2';
64                  const char *ptr = &a;
65                  serialport_write(port, ptr);
66
67                  gtk_label_set(GTK_LABEL(label), "Waiting mode");
68                  return;
69          }
70          else if (strcmp(str, "3") == 0)
71          {
72
73                  char a = '3';
74                  const char *ptr = &a;
75                  serialport_write(port, ptr);
76
77                  gtk_label_set(GTK_LABEL(label), "Tracking mode");
78                  return;
79          }
80          else if (strcmp(str, "y") == 0)
81          { //scalubur time
82
83                  char a = 'e';
84                  const char *ptr = &a;
85                  serialport_write(port, ptr);
86
87                  return;
88          }
89          else if (strcmp(str, "b") == 0)
90          { //buzzer
91
92                  char a = 'b';
93                  const char *ptr = &a;
94                  serialport_write(port, ptr);
95
96                  return;
```

```
97          }
98          else if (strcmp(str, "t") == 0)
99          { //trompo
100
101                 char a = 't';
102                 const char *ptr = &a;
103                 serialport_write(port, ptr);
104
105                 return;
106         }
107
108
109 }
```

- **Función para cerrar la ventana y finalizar el programa:** Es una función de tipo int que recibe un widget y la información, que utiliza un gtk main quit que retorna el cierre de la ventana y, por ende, del programa.

```
1 int CloseAppWindow(GtkWidget *widget, gpointer data)
2 {
3         gtk_main_quit();
4
5         return (FALSE);
6 }
```

2.1.2. Comunicación

La comunicación cuenta con varios elementos fundamentales.

- **Módulo bluetooth HC-05:** El módulo bluetooth permite establecer una comunicación inalámbrica para poder manipular los movimientos del vehículo por medio de una interfaz en el computador. El módulo se configura utilizando los comandos AT, los cuales permiten establecer el modo de operación del módulo, el nombre, la contraseña, el baud rate, etc.

- **Blueman:** Es una aplicación que permite gestionar dispositivos bluetooth, de manera que se puedan enlazar a un computador y además permite conectarlos a un puerto serial para poder enviar datos desde algún programa.

- **Programa en C:** En un programa en C se implementaron dos funciones, una para poder inicializar el puerto serial y la otra para poder escribir en el puerto serial. Dichas funciones se explican con más detalle a continuación.

- **Función para inicializar el puerto serial:** Esta función recibe de parámetros el nombre del puerto en el que está conectado el módulo bluetooth y el baud rate que éste utiliza. Luego, se implementa otra función que permita abrir el puerto para poder escribir y leer de él. En caso de que no se encuentre nada conectado al puerto, imprime un error y falla la inicialización. Si el puerto está conectado a algún dispositivo, entonces retorna la función que abre el puerto. La función que abre el puerto está implementada en las bibliotecas utilizadas.

```
1     int serialport_init(const char *serialport, int baud)
2
3 {
4
5         int fd;
```

```
6           fd = open(serialport, O_RDWR | O_NOCTTY);
7           if (fd == -1)
8           {
9
10                  perror("init_serialport: Unable to open port ");
11
12                  return -1;
13          }
14          return fd;
15  }
```

- **Función para escribir en el puerto serial:** Esta función recibe de parámetros la función anterior y el caracter que se desea escribir en el puerto serial. Luego, con el puerto ya inicializado y con el caracter que se desea enviar, entonces se escribe dicho caracter en el puerto. Ya en el arduino se implementan otras funciones que permitan leer estos datos, y el vehículo pueda realizar sus distintas acciones.

```
1       int serialport_write(int fd, const char *str)
2
3   {
4           int len = strlen(str);
5
6           int n = write(fd, str, len);
7
8           if (n != len)
9
10                  return -1;
11          return n;
12  }
```

2.1.3. Vehículo

En esta sección se presentan únicamente algunas de las funciones más importantes generadas en el código fuente AuveTA.ino"para el vehículo, además del loop de dicho código en arduino, esto debido a que el mismo tiene una extensión considerable y que se encuentra bien comentado y documentado en el *Doxygen*. Podrá encontrar el código completo y comentado en la sección de anexos al final del trabajo, específicamente en el apartado *6.1.1. AuVeTA.ino"*, además de poder consultar el *Doxygen* disponible en: *https://gitlab.com/UCR-EIE/IE-0117/i-2019/g0/tree/master/AuVeTAº %20Proyecto %20Plataformas.*

- **Setup general:** En la programación de arduino tenemos dos componentes estructurales esenciales, una de ellas es el setup. Acá es donde se inicializan por ejemplo, las comunicaciones seriales, asigna funcionamiento de entrada o salida a los pines digitales y analógicos, se declaran objetos a utilizar como myServo de la biblioteca servo.h. Esta es una estructura por la que se pasa solamente una vez cuando se reinicie el arduino.

```
1   void setup(){
2     //Seteo para comunicacin serial USB:
3     Serial.begin(9600);
4
5     ////Seteo para comunicacin serial Bluetooth:
6     mySerial.begin(115200);
7
8     //Set for digital pins:
```

```
  pinMode(R, OUTPUT);
  pinMode(G, OUTPUT);
  pinMode(B, OUTPUT);
  pinMode(servo_pin, OUTPUT);
  pinMode(obstacle_pin, INPUT);
  pinMode(track1_pin, INPUT);
  pinMode(final_sensor, INPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(motorL_forward, OUTPUT);
  pinMode(motorL_reverse, OUTPUT);
  pinMode(motorR_forward, OUTPUT);
  pinMode(motorR_reverse, OUTPUT);

  //Seteo para servo
  myServo.attach(servo_pin);
  myServo.write(servo_close);
  delay(500);
  myServo.detach();

  //Seteo inicial de motores en cero
  digitalWrite(motorR_forward, 0);
  digitalWrite(motorL_forward, 0);
  digitalWrite(motorR_reverse, 0);
  digitalWrite(motorL_reverse, 0);
}
```

- **Loop general** Como ya se mencionó antes, en la programación de arduino tenemos dos componentes estructurales esenciales, uno de ellos es el loop. Acá es donde se ejecutan y convocan las funciones principales del código para el vehículo. Esta es una estructura, que a diferencia del setup, se repite una y otra vez hasta que se reinicie el arduino.

```
void loop()
{
//luz roja en "Waiting Mode":
  digitalWrite(R, LOW);
  digitalWrite(G, LOW);
  digitalWrite(B, LOW);
//Revisi n de puerto serial disponible
  if (mySerial.available())
  {
//Lectura Serial para selecci n de modo: 3 -> "Trackig Mode" || 2 -> "Waiting Mode" || 3 -> "Control Mode"
    if (mySerial.read() == '3') //trackin mode
    { //Luz azul para "Tracking Mode":
      digitalWrite(R, LOW);
      digitalWrite(G, HIGH);
      digitalWrite(B, LOW);
      while (1) // "Tracking Mode" hasta cuando el puerto serial recibe un car cter '2' para luego ir a "Waiting Mode"
      {
        tracking_mode_func();
        if (mySerial.read() == '2') //Leer puerto serial para ir a "Waiting Mode"
        {
          break;
        };
      }
    }
```

```
25      if (mySerial.read() == '1') //Control mode
26      {
27        digitalWrite(R, LOW);
28        digitalWrite(G, LOW);
29        digitalWrite(B, HIGH);
30        control_mode_func();     //"Control Mode" hasta cuando el puerto serial
              recibe un car cter '2' para luego ir a "Waiting Mode"
31      }
32    }
33    else
34    {
35      while (mySerial.available() == false)
36      {
37        digitalWrite(R, HIGH);
38        delay(100);
39        digitalWrite(R, LOW);
40        delay(100);
41      };
42    };
43
44    no_move(); //detener movimiento cuando se est  en modo control y no se
          precionen teclas.
45  }
```

- **Función principal para el modo de control:** Esta es la función encargada de manipular las acciones en el *Control Mode*. En el interior de esta función se realiza una serie de comparaciones de los valores recibidos en el puerto serial que le permiten al código realizar acción sobre el servo, el *buzzer* y los motores físicos que mueven al vehículo. Además acá se incluye una comparación para que cuando el puerto serial reciba un carácter '2' se rompa el ciclo del *while* y el código se vaya al *Waiting Mode* que se tiene en el loop general del programa de arduino.

```
1   void control_mode_func()
2   {
3     char c;
4     while (1)
5     {
6       no_move();
7       c = mySerial.read();     //actualizaci n constante para valor recibido en
            puerto serial
8       if (c == 's') //Reverse
9       {
10        forward_control();
11      };
12      if (c == 'w') //forward
13      {
14        reverse_control();
15      };
16      if (c == 'a') //Turn left
17      {
18        goL_control();
19      };
20      if (c == 'd') //Turn rigth
21      {
22        goR_control();
23      };
24      if (c == 'b') //buzzer sound
```

```
    {
      digitalWrite(buzzer, HIGH);
      delay(1);
      digitalWrite(buzzer, LOW);
      delay(1);
      digitalWrite(buzzer, HIGH);
      delay(1);
    };
    if (c == 't') //trompo
    {
      trompo();
    };
    if (c == '2') //break cuando el puerto serial recibe un car cter '2' y
        luego se va a "Waiting Mode"
    {
      break;
    };
    if (c == 'y') //Obstacle remove
    {
      myServo.attach(servo_pin);
      myServo.write(servo_open);
      delay(700);
      myServo.write(servo_close);
      delay(700);
      myServo.detach();
    };
  }
};
```

- **Función principal para el modo de *tracking*:** Esta función es la encargada de ejecutar todas las acciones para el *Tracking Mode* y al igual que en la función anterior se convoca a otras funciones más básicas como las que realizan lectura de los sensores y las de activación de los actuadores físicos. Además acá se incluye una comparación para que cuando el puerto serial reciba un carácter '2' se rompa el ciclo del *while* y el código se vaya al *Waiting Mode* que se tiene en el *loop* general del programa de arduino.

```
   void tracking_mode_func()
{
  goL(235);
  goR(235);
  obstacle_detection(digitalRead(obstacle_pin));
  if (end_detection(digitalRead(track1_pin), digitalRead(final_pin)) == 1)
          // if structure to check final of track
  {
    delay(1000);
    while (1)
    {
      trompo_final(200);
      delay(300);
      trompo_final(0);
      delay(100);
      final_track_indicator();
      if (mySerial.read() == '2')    //break cuando el puerto serial recibe un
          car cter '2' y luego se va a "Waiting Mode"
      {
        break;
      };
```

```
20          }
21      };
22  };
```

2.2. Funcionamiento de Hardware

2.2.1. Microcontrolador ATmega328

El ATmega es un microcontrolador de baja potencia CMOS de 8 bits. Mediante la ejecución de instrucciones potentes en un solo ciclo, el microcontrolador permite que el usuario optimice el consumo versus la velocidad de procesos. Este microcontrolador forma parte del Arduino UNO, el cual será la placa de *hardware* sobre la cual se desarrollará el control del vehículo en sus dos modalidades, seguidor de linea y control mediante *bluetooth*.

La estructura del *core* combina un conjunto robusto de instrucciones con 32 registros de trabajo. Estos 32 registros se encuentran directamente conectados a la Unidad Lógica Aritmética (ALU), permitiendo el acceso independiente de dos registros ejecutados en un solo ciclo de reloj [2].

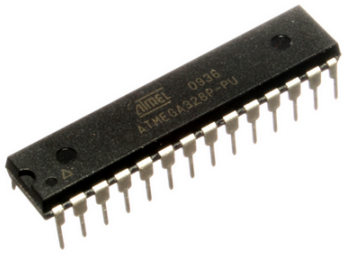

Figura 1: ATmega 328
[2]

2.2.2. Sensor de obstáculo

El sensor de obstáculo opera mediante una configuración emisor-receptor de luz infrarroja. Si la luz infrarroja choca contra un obstáculo, esta será reflejada y detectada por el fotodiodo. El rango de detección puede ser configurado mediante los dos controladores. Si no hay un obstáculo en frente la salida del sensor estará en alto, en el momento que se detecte un obstáculo la señal de salida se apagará.

El sensor permite activar o desactivar la detección de obstáculos mediante la manipulación del pin *enable*. Por defecto, el sensor se encuentra en modo activo [7].

Figura 2: Sensor de obstáculo
[7]

2.2.3. Sensor de *traking*

Es un módulo tipo LEGO, listo para ser conectado al microcontrolador. Se basa en un sensor óptico reflexivo con salida de transistor, el cual se encarga de recibir las señales infrarrojas enviadas para detectar la intensidad de la señal. Con cierto rango de altura, los sensores de pista son ampliamente utilizados para vehículos inteligentes o impresoras para detección de líneas en blanco y negro [4].

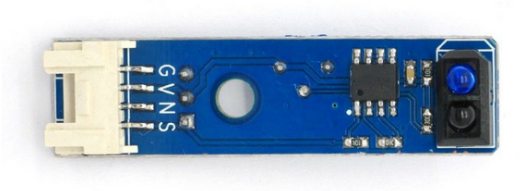

Figura 3: Sensor de pista
[4]

2.2.4. Motores DC 200 RPM a escala

Compuesto por un par de motores DC a escala, son usualmente recomendados como una opción barata y sencilla para poner ruedas en movimiento. Requieren una tensión de 4.5V y una corriente sin carga de 190 mA además posee una caja reductora 48:1 y una velocidad de giro de 200 RPM sin carga [8].

Figura 4: Hobby Gearmotor - 200 RPM
[8]

2.2.5. Módulo de control de motores DC 800 mA

El módulo de control L9110S 2-Canales es una tarjeta compacta que puede ser utilizada para controlar robots pequeños. Este módulo posee dos controladores independientes capaces de entregar hasta 800 mA en corriente continua. Pueden operar entre 2.5V y 12V lo cual permite que sean usados mediante microcontroladores.

Median de control PWM —*Pulse Width Modulation*— es posible controlar la velocidad del motor y una salida digital es usada para indicar la dirección [5].

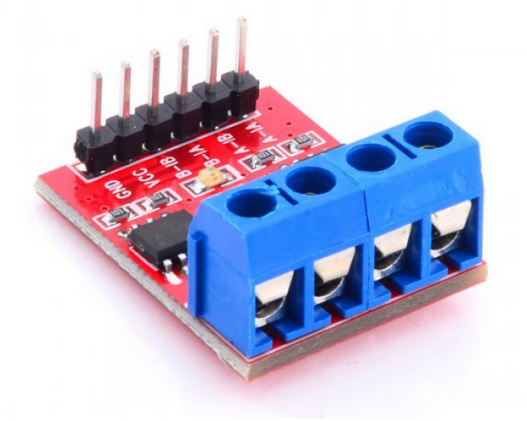

Figura 5: L9110S Drive Dual Motor DC 800mA
[5]

2.2.6. Servomotor

El servo es un tipo de motor que funciona paso-a-paso, es decir, su giro se determina en ángulos y no en RPM. Pequeño, liviano y con una alta potencia de salida; este tipo de servomotor es capaz de rotar hasta 180 grados [6].

Figura 6: Servomotor SG90
[6]

2.2.7. Piezo Speaker

Es un pequeño parlante redondo de 12mm el cual opera en todo el rango audible. Pueden ser utilizados para crear interfaces simples musicales. Requiere de una tensión entre los 3.5V y 5V con una corriente media de 35 mA además de una onda cuadrada para su correcto funcionamiento.

Figura 7: Magnetic buzzer
[3]

2.2.8. Módulo bluetooth HC-05

El módulo bluetooth HC-05 se utiliza en este proyecto para poder enviar datos desde un programa en C al arduino, la comunicación se realiza a través del puerto serial, mediante el cual se transmiten los datos. Dichos datos son interpretados por el arduino para llevar a cabo las funciones que el vehículo realiza.

Figura 8: Módulo bluetooth [1]

2.2.9. Esquemático de conexiones de vehículo

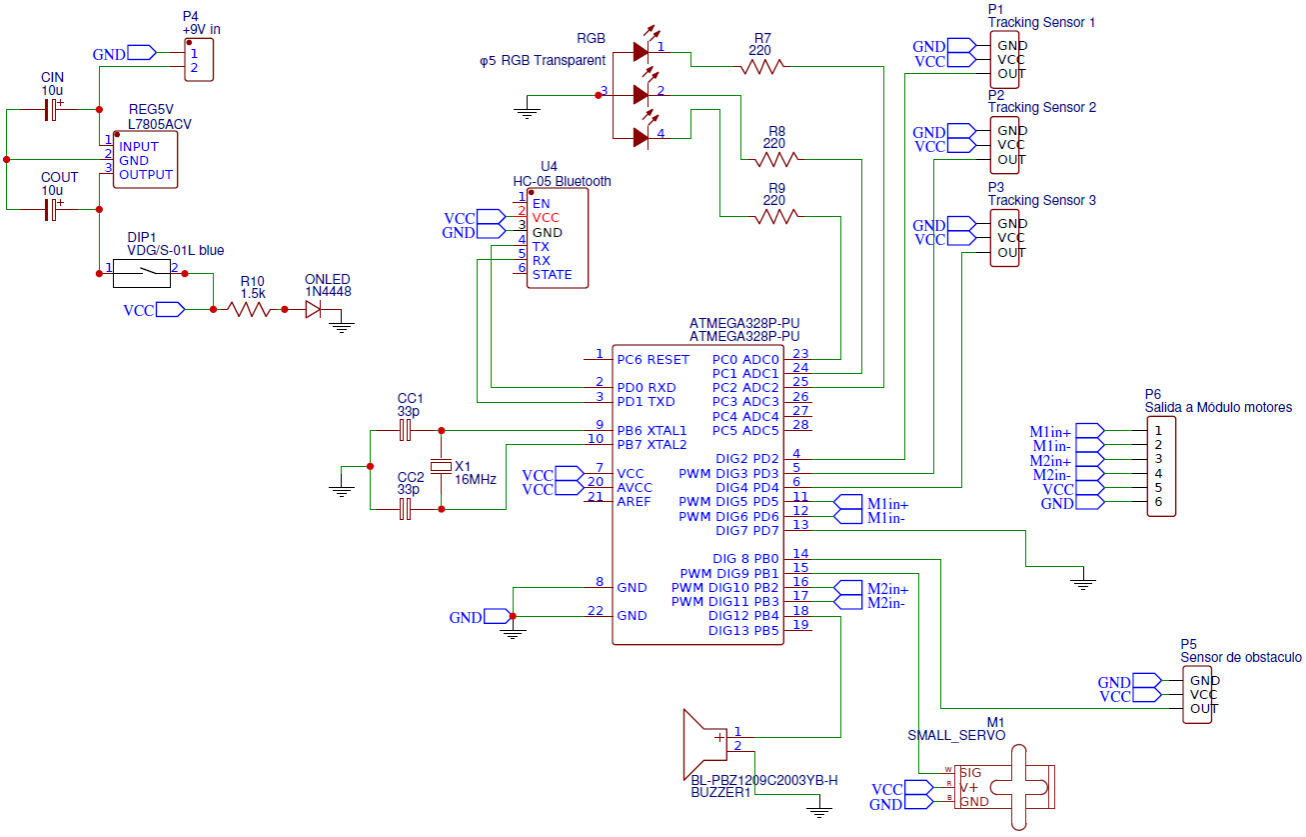

Figura 9: Esquemático del Vehículo Seguidor de Línea

3. Experimentos y pruebas realizadas

Es importante recalcar que las pruebas son parte fundamental durante el desarrollo y para el éxito de cualquier proyecto, puesto que en esta etapa es donde se puede determinar de manera crítica el funcionamiento de cada uno de los subsistemas que conforman cualquier proyecto. Además, la etapa de pruebas en muchas ocasiones revela posibles puntos de optimización tanto a nivel de software como *hardware* y en general permite descubrir inconvenientes que puedan existir en la integración e interacciones de dichos subsistemas. Este proyecto se conformó de dos partes estrechamente relacionadas, y puesto que el resultado final dependería tanto del *hardware* como el software se realizaron las siguientes pruebas a los subsistemas:

- Pruebas de *software*.
- Pruebas de *hardware*.
- Pruebas de integración.
- Pruebas de campo.

3.1. Pruebas de Software

- Revisión de interfaz y posible manejo de errores de comunicación u ocasionados por el usuario.
- Asignación correcta de pines en el código de acuerdo al tipo de entrada o salida.
- Verificación de correcta implementación de las bibliotecas en los códigos.
- Revisión de cantidad de memoria utilizada por la interfaz y en el arduino.

3.2. Pruebas de Hardware

- Chequeo de continuidad en cables y conexiones.
- Verificación de sujeción de componentes en su lugar.
- Prueba de cada módulo individualmente para verificar su correcto funcionamiento.

3.3. Pruebas de integración

- Pruebas con todos los módulos y sensores integrados.
- Conexión correcta de pines al Arduino en relación al código.
- Cálculo de tiempo de autonomía de fuente de poder con todos los subsistemas en funcionamiento.

3.4. Pruebas de campo

- Control de potencia de las ruedas dependiendo de donde se implemente el vehículo para evitar deslizamiento.
- Pruebas de rango de acción de sensores de acuerdo a las condiciones de luz del entorno donde se implemente el vehículo.

Todas las pruebas fueron importantes, y se prestó especial atención a la integración, puesto que la interacción *hardware-software* pudo representar una barrera importante a la hora de implementar los distintos modos de operación del vehículo y su accionar en un entorno físico.

A excepción del cálculo de tiempo de autonomía de fuente de poder, prueba la cual no se consideró puesto que el tiempo de operación iba a ser realmente corto, se realizaron las pruebas correspondientes a todos los subsistemas en funcionamiento. Se logra realizar todas las pruebas anteriores de manera exitosa, solo se debe considerar que el valor de potencia para los motores debe de ser siempre probado para adaptarlo a las condiciones de la superficie sobre la cual trabajará el vehículo.

4. Resultados obtenidos

4.1. Integración de las funciones en C para Interfaz

Como se explicó anteriormente, el código utilizado para la creación de la interfaz gráfica, implementa varias funciones que son parte del correcto funcionamiento de la misma. Este permite la combinación de las funciones de GTK y del puerto serial para dar como resultado el control del vehículo de manera inalámbrica, por medio de un módulo *bluetooth* que se puede conectar sin ningún problema con la computadora y así mismo con el código. Por esta razón, al correr el programa se obtiene como resultado la siguiente ventana que corresponde al control del Autonomous Vehicle for Tracking and Control Aplications (AuVeTA):

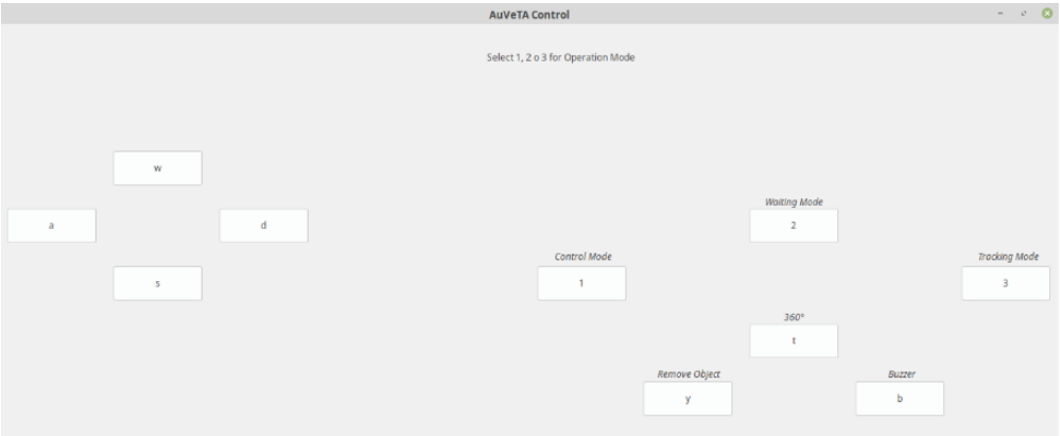

Figura 10: Interfaz Gráfica (Autoría propia)

4.2. Implementación del vehículo

Tras la integración de todos los subsistemas para la conformación del vehículo y la realización de las pruebas correspondientes se produce a efectuar las pruebas críticas finales con el sistema completamente integrado, que implican tanto el funcionamiento del *hardware* como el *software* y las interacciones que existen entre ambos subsistemas.

Como se ha abordado en la sección 3 de experimentos y pruebas, se obtienen resultados positivos para cada subsistema y además para la integración como un todo. El comportamiento de los sensores de obstáculo y de *tracking* fue el esperado, ya que tanto la detección de objetos para su respectiva remoción como el sensor encargado de enviar la señal indicando el final de la pista y el que detecta la pista en el suelo funcionaron de manera correcta durante la demostración.

Cabe destacar que para la demostración se esperaba contar con una superficie de trabajo clara y regular, pero en el edificio en el que se realizó la demostración el suelo no reunía las condiciones de uniformidad de color necesarias para el funcionamiento del sensor de tracking, por tal motivo se optó por utilizar hojas blancas en el suelo y encima de ellas una cinta de color negro para garantizar el funcionamiento del sensor; ahora bien, este cambio afectó el rodamiento de las llantas y del tercer apoyo por el cambio en la aspereza de la superficie y por tanto además la cantidad de potencia que se utiliza en los motores para el movimiento en el *Tracking Mode"* del vehículo, por lo cual fue necesario realizar una modificación de última hora en el código; el aspecto de modificación de potencia de motores se tenia previsto por lo cual solo fue necesario ajustar una de las variables del código para lograr la modificación de manera rápida y sencilla de la cantidad de potencia enviada a los motores.

5. Conclusiones

- Se logra comprobar con éxito el funcionamiento del vehículo tanto en el modo de control como en el modo de tracking.
- Se realizaron de manera exitosa las pruebas planteadas en la sección 3 de este documento, esto a pesar de que no se toma en consideración el cálculo de tiempo de autonomía de la fuente de poder con todos los subsistemas en funcionamiento, ya que el tiempo de operación es relativamente corto. No se descarta realizar un pruba de este punto en el futuro con todos los subsistemas trabajando al máximo ciclo de trabajo para determinar la duración de la batería ante condiciones críticas.
- Se logra la correcta implementación de la interfaz gráfica que permite la comunicación inalámbrica por conexión *bluetooth* con el vehículo.
- El valor asignado a la potencia del motor para el modo tracking debe ser ajustado dependiendo de la superficie en donde trabaje el vehículo, ya que mucha potencia propiciará que el vehículo se salga de la pista y muy poca hará que aumenten las posibilidades de atascamiento.
- Para los sensores ópticos se deben considerar las condiciones de luz en el entorno de trabajo y por ello también la calibración de los mismos para garantizar un funcionamiento adecuado y óptimo de los sistemas de control.
- Se concluye que se pueden mejorar aspectos constructivos y estéticos de la interfaz para alcanzar un nivel más depurado de experiencia para el usuario.

Referencias

[1] ArduinoVe. *Módulo Bluetooth Hc-05*. URL: http://www.arduinove.com/index.php?route=product/product&product_id=65. (consultado el: 20 July. 2019).

[2] Atmel. *8-bit AVR Microcontroller with 4/8/16/32K Bytes In-SystemProgrammable Flash*. URL: https://www.sparkfun.com/datasheets/Components/SMD/ATMega328.pdf. (consultado el: 25 May. 2019).

[3] CUI.INC. *Magnetic buzzer*. URL: http://files.microjpm.webnode.com/200001546-06a0708951/cem-1203-42-.pdf. (consultado el: 25 May. 2019).

[4] ITEAD. *Track Sensor*. URL: https://www.itead.cc/wiki/Track_Sensor. (consultado el: 25 May. 2019).

[5] Micro JMP. *L9110 2-CHANNEL MOTOR DRIVER*. URL: http://files.microjpm.webnode.com/200001268-23d5b24d30/L9110_2_CHANNEL_MOTOR_DRIVER.pdf. (consultado el: 25 May. 2019).

[6] Micro JMP. *SERVO MOTOR SG90*. URL: http://files.microjpm.webnode.com/200000416-ec3e6ed3b6/SG90%5C%20Datasheet.pdf. (consultado el: 25 May. 2019).

[7] SensorKit. *KY-032 Obstacle-detect module*. URL: http://sensorkit.en.joy-it.net/index.php?title=KY-032_Obstacle-detect_module. (consultado el: 25 May. 2019).

[8] SparkFun. *Hobby Gearmotor - 140 RPM*. URL: https://www.sparkfun.com/products/13302. (consultado el: 25 May. 2019).

[9] The GTK Team. *The GTK Project*. URL: https://www.gtk.org/. (consultado el: 19 July. 2019).

6. Anexos

6.1. Código fuente generado para el Vehículo

6.1.1. AuVeTA.ino

```
/**
 * @file AuVeTA.ino
 * @author Jorge Isaac Fallas Mej a B62562
 * @author Esteban Rodr guez Quintana B66076
 * @brief Archivo de c digo en arduino para el proyecto final de Programaci n Bajo
      Plataformas Abiertas.
 * @version 1.3
 * @date 2019-06-14
 */


/**********include de bibliotecas:************/
#include <Servo.h>          //Servo lib
#include <SoftwareSerial.h> //Bluetooh lib

/*****************************************/

/********** Definici n de variables: ************/
int RX_bluetooh = 0;
int TX_bluetooh = 1;
int servo_pin = 11;
int obstacle_pin = 8;
int track1_pin = 12;
int final_pin = 7;
int motorR_forward = 9;
int motorR_reverse = 5;
int motorL_forward = 10;
int motorL_reverse = 6;
int buzzer = 13;
int R = 2;
int G = 3;
int B = 4;
int servo_open = 125;  //revisar valores para los grados del servo
int servo_close = 10;  //revisar valores para los grados del servo
int adjustM_delay = 1; //valor cercano a cero
int stopD = 40;        //valor cercano 50
int reactionD = 15;    //entre 10 y 20
int motorPower = 254;

/*  Valores de PWM para variable motorPower:
     ---------------------------------
    |    Valor    |  Ciclo de trabajo |
    |=================================|
    |     0       |        0%         |
    |     63      |        25%        |
    |     127     |        50%        |
    |     190     |        75%        |
    |     255     |        100%       |
    |---------------------------------|
*/

//////////////////////////////////////////////////////
```

```
52  bool obstacle_presence = digitalRead(obstacle_pin); //detecci n de objeto frontal
                  // detect = 0 //  no detect = 1  //
53  bool track1 = digitalRead(track1_pin);              //detecci n de pista
                          // detect = 1 //  no detect = 0  //
54  bool final_sensor = digitalRead(final_pin);         //detecci n de objeto lateral
                  // detect = 0 //  no detect = 1  //
55
56
57  Servo myServo;
58  SoftwareSerial mySerial(RX_bluetooh, TX_bluetooh); // (RX,TX)
59
60  /*****************************************/
61  /************** SETUP: ******************/
62  /*****************************************/
63  void setup()
64  {
65    //Seteo para comunicaci n serial USB:
66    Serial.begin(9600);
67
68    ////Seteo para comunicaci n serial Bluetooth:
69    mySerial.begin(115200);
70
71    //Set for digital pins:
72    pinMode(R, OUTPUT);
73    pinMode(G, OUTPUT);
74    pinMode(B, OUTPUT);
75    pinMode(servo_pin, OUTPUT);
76    pinMode(obstacle_pin, INPUT);
77    pinMode(track1_pin, INPUT);
78    pinMode(final_sensor, INPUT);
79    pinMode(buzzer, OUTPUT);
80    pinMode(motorL_forward, OUTPUT);
81    pinMode(motorL_reverse, OUTPUT);
82    pinMode(motorR_forward, OUTPUT);
83    pinMode(motorR_reverse, OUTPUT);
84
85    //Seteo para servo
86    myServo.attach(servo_pin);
87    myServo.write(servo_close);
88    delay(500);
89    myServo.detach();
90
91    //Seteo inicial de motores en cero
92    digitalWrite(motorR_forward, 0);
93    digitalWrite(motorL_forward, 0);
94    digitalWrite(motorR_reverse, 0);
95    digitalWrite(motorL_reverse, 0);
96  }
97
98  /*****************************************/
99  /*************** LOOP: ******************/
100 /*****************************************/
101 void loop()
102 {
103 //luz roja en "Waiting Mode":
104   digitalWrite(R, LOW);
105   digitalWrite(G, LOW);
106   digitalWrite(B, LOW);
107 //Revisi n de puerto serial disponible
```

```
108   if (mySerial.available())
109   {
110 //Lectura Serial para selecci n de modo: 3 -> "Trackig Mode" || 2 -> "Waiting Mode"
      || 3 -> "Control Mode"
111     if (mySerial.read() == '3') //trackin mode
112     { //Luz azul para "Tracking Mode":
113       digitalWrite(R, LOW);
114       digitalWrite(G, HIGH);
115       digitalWrite(B, LOW);
116       while (1) // "Tracking Mode" hasta cuando el puerto serial recibe un car cter
              '2' para luego ir a "Waiting Mode"
117       {
118         tracking_mode_func();
119         if (mySerial.read() == '2') //Leer puerto serial para ir a "Waiting Mode"
120         {
121           break;
122         };
123       }
124     }
125     if (mySerial.read() == '1') //Control mode
126     {
127       digitalWrite(R, LOW);
128       digitalWrite(G, LOW);
129       digitalWrite(B, HIGH);
130       control_mode_func();    //"Control Mode" hasta cuando el puerto serial recibe
              un car cter '2' para luego ir a "Waiting Mode"
131     }
132   }
133   else
134   {
135     while (mySerial.available() == false)
136     {
137       digitalWrite(R, HIGH);
138       delay(100);
139       digitalWrite(R, LOW);
140       delay(100);
141     };
142   };
143
144   no_move(); //detener movimiento cuando se est  en modo control y no se precionen
        teclas.
145 }
146
147 //***************FUNCTIONS: **********************
148
149 /**
150  * @brief Esta funci n da movimiento de rueda izquierda y por tanto un giro hacia
        la derecha del veh culo para el modo de tracking.
151  * @param int para potencia deseada en el motor.
152  * @return no retorna valores, solo ejecuta acciones.
153 */
154 void goR(int PW)
155 {
156   while (track1 != 1)
157   {
158     track1 = digitalRead(track1_pin);
159     analogWrite(motorR_forward, 0);
160     analogWrite(motorL_forward, 0);
161     analogWrite(motorR_reverse, 0);
```

```
162       analogWrite(motorL_reverse, PW);
163       delay(adjustM_delay);
164       if (mySerial.read() == '2')
165       {
166         break;
167       }
168     };
169
170     analogWrite(motorR_forward, 0);
171     analogWrite(motorL_forward, 0);
172     analogWrite(motorR_reverse, 0);
173     analogWrite(motorL_reverse, 0);
174     delay(stopD);
175 };
176
177
178 /**
179  * @brief Esta funci n da movimiento de rueda derecha y por tanto un giro hacia la
        izquierda del veh culo para el modo de tracking.
180  * @param int para potencia deseada en el motor.
181  * @return no retorna valores, solo ejecuta acciones.
182 */
183 void goL(int PW)
184 {
185   while (track1 != 0)
186   {
187     track1 = digitalRead(track1_pin);
188     analogWrite(motorR_forward, 0);
189     analogWrite(motorL_forward, 0);
190     analogWrite(motorR_reverse, PW);
191     analogWrite(motorL_reverse, 0);
192     delay(adjustM_delay);
193     if (mySerial.read() == '2')
194     {
195       break;
196     }
197   };
198   analogWrite(motorR_forward, 0);
199   analogWrite(motorL_forward, 0);
200   analogWrite(motorR_reverse, 0);
201   analogWrite(motorL_reverse, 0);
202   delay(stopD);
203 }
204
205 /**
206  * @brief Esta funci n da movimiento de rueda izquierda y por tanto un giro hacia
        la derecha del veh culo para el modo de control.
207  * @param no recibe par metros
208  * @return no retorna valores, solo ejecuta acciones.
209 */
210 void goR_control()
211 {
212   digitalWrite(motorL_reverse, motorPower);
213   digitalWrite(motorL_forward, 0);
214   digitalWrite(motorR_reverse, 0);
215   digitalWrite(motorR_forward, 0);
216   delay(reactionD);
217 };
218
```

```
219 /**
220  * @brief Esta funci n da movimiento de rueda derecha y por tanto un giro hacia la
        izquierda del veh culo para el modo de control.
221  * @param no recibe par metros.
222  * @return no retorna valores, solo ejecuta acciones.
223 */
224 void goL_control()
225 {
226   digitalWrite(motorL_reverse, 0);
227   digitalWrite(motorL_forward, 0);
228   digitalWrite(motorR_reverse, motorPower);
229   digitalWrite(motorR_forward, 0);
230   delay(reactionD);
231 };
232
233 /**
234  * @brief Esta funci n da movimiento de ambas ruedas en direcci n de avance del
        veh culo para el modo de control.
235  * @param no recibe par metros.
236  * @return no retorna valores, solo ejecuta acciones.
237 */
238 void forward_control()
239 {
240   digitalWrite(motorL_reverse, 0);
241   digitalWrite(motorL_forward, motorPower);
242   digitalWrite(motorR_reverse, 0);
243   digitalWrite(motorR_forward, motorPower);
244   delay(reactionD);
245 };
246
247 /**
248  * @brief Esta funci n da movimiento de ambas ruedas en direcci n contraria al
        avance del veh culo para el modo de control.
249  * @param no recibe par metros.
250  * @return no retorna valores, solo ejecuta acciones.
251 */
252 void reverse_control()
253 {
254   digitalWrite(motorL_reverse, motorPower);
255   digitalWrite(motorL_forward, 0);
256   digitalWrite(motorR_reverse, motorPower);
257   digitalWrite(motorR_forward, 0);
258   delay(reactionD);
259 };
260
261 /**
262  * @brief Esta funci n apaga motores del veh culo para el modo de control.
263  * @param no recibe par metros.
264  * @return no retorna valores, solo ejecuta acciones.
265 */
266 void no_move()
267 {
268   digitalWrite(motorL_reverse, 0);
269   digitalWrite(motorL_forward, 0);
270   digitalWrite(motorR_reverse, 0);
271   digitalWrite(motorR_forward, 0);
272 };
273
274 /**
```

```
275  * @brief Esta funci n determina si se detecta el final del recorrido gracias a la
         condici n recibida de un sensor de tracking y otro de obst culo.
276  * @param Recibe el valor del sensor de tracking.
277  * @param Recibe el valor del sensor de obst culo lateral.
278  * @return Retorna un bool que indica si el final de la pista ha sido alcanzado.
279 */
280 bool end_detection(int _track1, int _track2)
281 {
282   bool FINAL_DE_PISTA = false;
283   if ((_track1 == 1) && (_track2 == 0))
284   {
285     FINAL_DE_PISTA = true;
286   }
287   return FINAL_DE_PISTA;
288 };
289
290
291
292 /**
293  * @brief Esta funci n determina si se detecta un obst culo frontal gracias al
         sensor de obst culo, en caso de presencia de obst culo activa el servomotor
         para removerlo..
294  * @param Recibe el valor de un sensor de obst culo frontal.
295  * @return No retorna valores, solo activa el servomotor.
296 */
297 void obstacle_detection(bool _obstacle_presence)
298 {
299   if (_obstacle_presence == 0)
300   {
301     myServo.attach(servo_pin);
302     myServo.write(servo_open);
303     delay(700);
304     myServo.write(servo_close);
305     delay(700);
306   };
307   myServo.detach();
308 }
309
310
311
312 /**
313  * @brief Esta funci n solo enciende las luces en una determinada secuencia para
         indicar el final del recorrido.
314  * @param No recibe par metros.
315  * @return No retorna, solo acciona LED RGB.
316 */
317 void final_track_indicator()
318 {
319   digitalWrite(R, HIGH);
320   delay(100);
321   digitalWrite(R, HIGH);
322   delay(50);
323   digitalWrite(G, HIGH);
324   delay(100);
325   digitalWrite(G, LOW);
326   delay(50);
327   digitalWrite(B, HIGH);
328   delay(100);
329   digitalWrite(B, LOW);
```

```
330   delay(50);
331   digitalWrite(R, HIGH);
332   delay(100);
333   digitalWrite(R, HIGH);
334   delay(50);
335   digitalWrite(G, LOW);
336   delay(100);
337   digitalWrite(G, LOW);
338   delay(50);
339   digitalWrite(B, HIGH);
340   delay(100);
341   digitalWrite(B, LOW);
342   digitalWrite(R, LOW);
343   digitalWrite(G, LOW);
344   delay(100);
345 };
346
347
348 /**
349  * @brief Esta funci n hace que se activen las ruedas de manera que el veh ulo
        gire sobre su propio eje horizontal para el modo tracking.
350  * @param No recibe par metros.
351  * @return No retorna.
352 */
353 void trompo()
354 {
355   digitalWrite(motorL_reverse, motorPower);
356   digitalWrite(motorL_forward, 0);
357   digitalWrite(motorR_reverse, 0);
358   digitalWrite(motorR_forward, motorPower);
359   delay(reactionD);
360 };
361
362 /**
363  * @brief Esta funci n hace que se activen las ruedas de manera que el veh ulo
        gire sobre su propio eje horizontal para el modo control.
364  * @param Recibe la potencia que se desea usar para los motores.
365  * @return No retorna.
366 */
367 void trompo_final(int power)
368 {
369   digitalWrite(motorL_reverse, power);
370   digitalWrite(motorL_forward, 0);
371   digitalWrite(motorR_reverse, 0);
372   digitalWrite(motorR_forward, power);
373 };
374
375 /**
376  * @brief Esta funci n implementa y ejecuta las acciones y funciones necesarias
        para el funcionamiento durante el modo tracking.
377  * @param No recibe par metros.
378  * @return No retorna.
379 */
380 void tracking_mode_func()
381 {
382   goL(235);
383   goR(235);
384   obstacle_detection(digitalRead(obstacle_pin));
```

```
385   if (end_detection(digitalRead(track1_pin), digitalRead(final_pin)) == 1)       //
          if structure to check final of track
386   {
387     delay(1000);
388     while (1)
389     {
390       trompo_final(200);
391       delay(300);
392       trompo_final(0);
393       delay(100);
394       final_track_indicator();
395       if (mySerial.read() == '2')    //break cuando el puerto serial recibe un
              car cter '2' y luego se va a "Waiting Mode"
396       {
397         break;
398       };
399     }
400   };
401 };
402
403
404 /**
405  * @brief Esta funci n implementa y ejecuta las acciones y funciones necesarias
          para el funcionamiento durante el modo de control.
406  * @param No recibe par metros.
407  * @return No retorna.
408 */
409 void control_mode_func()
410 {
411   char c;
412   while (1)
413   {
414     no_move();
415     c = mySerial.read();     //actualizaci n constante para valor recibido en puerto
              serial
416     if (c == 's') //Reverse
417     {
418       forward_control();
419     };
420     if (c == 'w') //forward
421     {
422       reverse_control();
423     };
424     if (c == 'a') //Turn left
425     {
426       goL_control();
427     };
428     if (c == 'd') //Turn rigth
429     {
430       goR_control();
431     };
432     if (c == 'b') //buzzer sound
433     {
434       digitalWrite(buzzer, HIGH);
435       delay(1);
436       digitalWrite(buzzer, LOW);
437       delay(1);
438       digitalWrite(buzzer, HIGH);
439       delay(1);
```

```
440       };
441       if (c == 't') //trompo
442       {
443         trompo();
444       };
445       if (c == '2') //break cuando el puerto serial recibe un car cter '2' y luego se
              va a "Waiting Mode"
446       {
447         break;
448       };
449       if (c == 'y') //Obstacle remove
450       {
451         myServo.attach(servo_pin);
452         myServo.write(servo_open);
453         delay(700);
454         myServo.write(servo_close);
455         delay(700);
456         myServo.detach();
457       };
458     }
459 };
```

6.2. Código fuente generado para Interfaz y Comunicación Serial

6.2.1. h_comunicación_interfaz.h

```
1  /**
2   * @file h_comunicaci n_interfaz.h
3   * @author Jorge Isaac Fallas Mej a B62562
4   * @author Esteban Rodr guez Quintana B66076
5   * @author Fabio Villalobos Pacheco B78346
6   * @brief Archivo de encabezado para la interfaz y la comunicaci n
7   * @version 1
8   * @date 2019-07-18
9   *
10  *
11  */
12 #ifndef A
13 #define A
14
15 #include <stdio.h> /* Standard input/output definitions */
16 #include <stdlib.h>
17 #include <stdint.h>  /* Standard types */
18 #include <string.h>  /* String function definitions */
19 #include <unistd.h>  /* UNIX standard function definitions */
20 #include <fcntl.h>   /* File control definitions */
21 #include <errno.h>   /* Error number definitions */
22 #include <termios.h> /* POSIX terminal control definitions */
23 #include <sys/ioctl.h>
24 #include <getopt.h>
25 #include <gtk/gtk.h>
26
27 #define BUF_SIZE 88
28 static float num1 = 0;
29 static char lastChar = (char)0;
30 static char prevCmd = (char)0;
31 /**
32  * @brief Estructura de datos para realizar un seguimiento de los botones de la
        interfaz.
33  */
34
35 typedef struct {
36
37         char *szLabel;
38         int row;
39         int col;
40         GtkWidget *widget;
41
42 } MoButton;
43
44 /**
45  * @brief Lista MoButton: Crea una lista que contendr  los botones que aparecer n
        en la interfaz gr fica.
46  */
47
48 MoButton * buttonList;
49
50 /**
51  * @brief int numbuttons: Es la variable de tipo entero que contiene el n mero de
        botones que estar n en
52  * la lista MoButton.
```

```
53  */
54
55  int numbuttons;
56
57  /**
58   * @brief Widget label: Este es el panel en el que se "imprime" la funci n de los
        botones.
59   *
60   */
61
62  GtkWidget *label;
63
64  /*Funciones comunicaci n serial*/
65  /**
66   *@brief Funci n serialport_init: Inicializa el puerto serial.
67   *@param serialport: El nombre del puerto al que se conecta el m dulo bluetooth.
68   *@param baud: El baud rate que utiliza el m dulo bluetooth.
69   *@return int: Si el puerto pudo ser inicializado.
70  */
71  int serialport_init(const char *serialport, int baud);
72  /**
73   *@brief Funci n serialport_write: Escribe en el puerto serial.
74   *@param fd: La funci n serialport_init.
75   *@param str: El caracter que se env a en el puerto.
76   *@return int: El dato enviado por el puerto.
77  */
78  int serialport_write(int fd, const char *str);
79
80
81  /* Funciones GTK */
82  /**
83   * @brief Funci n CloseAppWindow: Cierra la ventana creada de la interfaz.
84   * @param widget: La "caja" que contiene el texto del bot n.
85   * @param data: El puntero del texto contenido en el "widget".
86   * @return FALSE: Retorna el cierre de la ventana del programa y finalizaci n de
        operaci n.
87   */
88  gint CloseAppWindow (GtkWidget *widget, gpointer data);
89
90  /**
91   * @brief Funci n key_press: Se encarga de la lectura del bot n presionado en el
        teclado.
92   * @param widget: La "caja" que contiene el texto del bot n.
93   * @param event: El evento que lee el bot n presionado.
94   * @param data: El puntero del texto contenido en el "widget".
95   * @return Retorna la tarea del bot n presionado.
96   */
97  void key_press(GtkWidget *widget, GdkEventKey *event, gpointer data);
98
99  /**
100  * @brief Funci n button_clicked: Se encarga de leer que el bot n en la interfaz
       se haya clickeado.
101  * @param widget: La "caja" que contiene el texto del bot n.
102  * @param data: El puntero del texto contenido en el "widget".
103  * @return Retorna la tarea del bot n clickeado.
104  */
105 void button_clicked (GtkWidget *widget, gpointer data);
106
107 /**
```

```
108  * @brief Funci n CreateButton: Se encarga de crear las caracter sticas de cada
         bot n.
109  * @param table: La tabla que contiene los botones.
110  * @param szLabel: El tama o del texto que contiene el bot n.
111  * @param row: La fila donde se ubicar  el bot n.
112  * @param column: La columna donde se ubicar  el bot n.
113  * @return button: Retorna el bot n creado.
114  */
115 GtkWidget *CreateButton (GtkWidget *table, char *szLabel, int row, int column);
116
117 /**
118  * @brief Funci n CreateControlButtons: Se encarga de poner en la interfaz los
         botones que se crearon.
119  * @param table: La tabla que contiene los botones.
120  * @return Retorna los botones en la interfaz.
121  */
122 void CreateControlButtons (GtkWidget *table);
123 #endif
```

6.2.2. comunicación_interfaz.c

```c
/**
 * @file comunicaci n_interfaz.c
 * @author Jorge Isaac Fallas Mej a B62562
 * @author Esteban Rodr guez Quintana B66076
 * @author Fabio Villalobos Pacheco B78346
 * @brief Archivo de funciones para la interfaz y la comunicaci n
 * @version 1
 * @date 2019-07-18
 *
 *
 */
#include "h_comunicaci n_interfaz.h"

/**
 * @brief int CloseAppWindow: Es una funci n de tipo int que
 * recibe un widget y la informaci n, que utiliza un
 * gtk_main_quit que retorna el cierre de la ventana y, por
 * ende, del programa.
 */
int CloseAppWindow(GtkWidget *widget, gpointer data)
{
        gtk_main_quit();

        return (FALSE);
}

/**
 * @brief void key_press: Es una funci n de tipo void que recibe
 * un widget, un evento y la informaci n, hace uso de un ciclo
 * for que busca entre los botones, y una condici n if que si la
 * pulsaci n de tecla es el primer car cter de un bot n y la longitud
 * de la etiqueta del bot n es uno, entonces permite el retorno que
 * conecta el bot n presionado en el teclado con la interfaz gr fica
 * tarea asociada.
 */

void key_press(GtkWidget *widget,
               GdkEventKey *event,
               gpointer data)
{
        int npressed;

        for (npressed = 0; npressed < numbuttons; npressed++)
        {
                if (event->keyval == buttonList[npressed].szLabel[0] &&
                    buttonList[npressed].szLabel[1] == (char)0)
                {
                        gtk_widget_grab_focus(buttonList[npressed].widget);

                        gtk_button_clicked(GTK_BUTTON(buttonList[npressed].widget));
                        return;
                }
        }
}

/**
 * void button_clicked: Es una funci n de tipo void que recibe un widget
```

```
58  * y la informaci n , este contiene instancias de las funciones que trabajan
59  * con el puerto serial , dos punteros de tipo char , el str se encarga
60  * de agarrar el texto del bot n y la informaci n , adem s contiene
61  * la ventana que contiene la interfaz , luego varias condiciones if
62  * que permiten comparar el bot n clickeado o presinado en el teclado
63  * que conecta con el puerto serial y retorna la tarea que realiza
64  * el veh culo .
65  */
66
67 void button_clicked(GtkWidget *widget, gpointer data)
68 {
69         int baudrate = 115200;
70         int port;
71         port = serialport_init("/dev/rfcomm0", baudrate);
72
73         char ch = *((char *)data);
74         char *str;
75         GtkWidget *window;
76         window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
77
78         str = (char *)data;
79
80         if (strcmp(str, "w") == 0)
81         {
82
83                 char a = 'w';
84                 const char *ptr = &a;
85                 serialport_write(port, ptr);
86
87                 return;
88         }
89         else if (strcmp(str, "a") == 0)
90         {
91
92                 char a = 'a';
93                 const char *ptr = &a;
94                 serialport_write(port, ptr);
95
96                 return;
97         }
98         else if (strcmp(str, "d") == 0)
99         {
100
101                 char a = 'd';
102                 const char *ptr = &a;
103                 serialport_write(port, ptr);
104
105                 return;
106         }
107         else if (strcmp(str, "s") == 0)
108         {
109
110                 char a = 's';
111                 const char *ptr = &a;
112                 serialport_write(port, ptr);
113
114                 return;
115         }
116         else if (strcmp(str, "1") == 0)
```

```
117         {
118
119                 char a = '1';
120                 const char *ptr = &a;
121                 serialport_write(port, ptr);
122
123                 gtk_label_set(GTK_LABEL(label), "Control Mode");
124                 return;
125         }
126         else if (strcmp(str, "2") == 0)
127         {
128
129                 char a = '2';
130                 const char *ptr = &a;
131                 serialport_write(port, ptr);
132
133                 gtk_label_set(GTK_LABEL(label), "Waiting mode");
134                 return;
135         }
136         else if (strcmp(str, "3") == 0)
137         {
138
139                 char a = '3';
140                 const char *ptr = &a;
141                 serialport_write(port, ptr);
142
143                 gtk_label_set(GTK_LABEL(label), "Tracking mode");
144                 return;
145         }
146         else if (strcmp(str, "y") == 0)
147         { //scalubur time
148
149                 char a = 'e';
150                 const char *ptr = &a;
151                 serialport_write(port, ptr);
152
153                 return;
154         }
155         else if (strcmp(str, "b") == 0)
156         { //buzzer
157
158                 char a = 'b';
159                 const char *ptr = &a;
160                 serialport_write(port, ptr);
161
162                 return;
163         }
164         else if (strcmp(str, "t") == 0)
165         { //trompo
166
167                 char a = 't';
168                 const char *ptr = &a;
169                 serialport_write(port, ptr);
170
171                 return;
172         }
173
174
175 }
```

```
176
177 /**
178  * @brief GtkWidget *CreateButton: Es una funci n tipo Gtkwidget
179  * que recibe la tabla que contiene los botones, un dato
180  * char que es el tama o de los botones, adem s dos datos de
181  * tipo int que son la posici n en fila y columna de los botones,
182  * luego esta hace un gtk_button_new_with_label que es el bot n,
183  * adem s del color, de la posicion en la interfaz y gtk_widget_show
184  * que hace que retorne todas esas caracter sticas del bot n.
185  */
186
187 GtkWidget *CreateButton(GtkWidget *table, char *szLabel, int row, int column)
188 {
189         GtkWidget *button;
190
191         button = gtk_button_new_with_label(szLabel);
192
193         gtk_signal_connect(GTK_OBJECT(button), "clicked",
194                            GTK_SIGNAL_FUNC(button_clicked), szLabel);
195
196         GdkColor color;
197
198         gdk_color_parse("red", &color);
199
200         gtk_widget_modify_bg(GTK_WIDGET(button), GTK_STATE_NORMAL, &color);
201
202         gtk_table_attach(GTK_TABLE(table), button,
203                          column, column + 1,
204                          row, row + 1,
205                          GTK_FILL | GTK_EXPAND,
206                          GTK_FILL | GTK_EXPAND,
207                          10, 15);
208
209         gtk_widget_show(button);
210
211         return (button);
212 }
213
214 /**
215  * @brief void CreateControlButtons: Es una funci n tipo void
216  * que recibe la tabla que contiene los botones, luego se crea
217  * un dato que hace que el ciclo for recorra la lista de los
218  * botones y con ayuda de la funci n CreateButton se cree el
219  * bot n y aparezca en la interfaz.
220  */
221
222 void CreateControlButtons(GtkWidget *table)
223 {
224         int nIndex;
225
226         for (nIndex = 0; nIndex < numbuttons; nIndex++)
227         {
228
229                 buttonList[nIndex].widget =
230                     CreateButton(table,
231                                  buttonList[nIndex].szLabel,
232                                  buttonList[nIndex].row,
233                                  buttonList[nIndex].col);
234         }
```

```
235 }
236
237 ////////////////////////////////////////////////////
238 /**
239  *@brief Funci n serialport_write: Esta funci n recibe de par metros el puerto
240  *inicializado y el caracter que se desea enviar. Luego, el caracter se escribe
241  *en el puerto inicializado para ser recibido en el arduino.
242 */
243
244 int serialport_write(int fd, const char *str)
245
246 {
247
248         int len = strlen(str);
249
250         int n = write(fd, str, len);
251
252         if (n != len)
253
254                 return -1;
255
256         return n;
257 }
258 /**
259  *@brief Funci n serialport_init: Esta funci n recibe de par metros el nombre
260  *del puerto que se desea inicializar y el baud rate que utiliza el m dulo
        bluetooth.
261  *Seguidamente, el puerto es abierto para poder escribir o leer datos.
262 */
263
264 int serialport_init(const char *serialport, int baud)
265
266 {
267
268         int fd;
269         fd = open(serialport, O_RDWR | O_NOCTTY);
270         if (fd == -1)
271         {
272
273                 perror("init_serialport: Unable to open port ");
274
275                 return -1;
276         }
277         return fd;
278 }
```

6.2.3. main.c

```
/**
 * @file main.c
 * @author Jorge Isaac Fallas Mej a B62562
 * @author Esteban Rodr guez Quintana B66076
 * @author Fabio Villalobos Pacheco B78346
 * @brief Archivo de c digo main para el proyecto AuVeTA
 * @version 1
 * @date 2019-07-18
 *
 *
 */

#include "h_comunicaci n_interfaz.h"

int main(int argc, char *argv[])
{

 numbuttons = 10;
 buttonList = malloc(sizeof(MoButton) * 11);

 int i = 0;
 buttonList[i++] = (MoButton){"w", 2, 1, NULL};
 buttonList[i++] = (MoButton){"a", 3, 0, NULL};
 buttonList[i++] = (MoButton){"d", 3, 2, NULL};
 buttonList[i++] = (MoButton){"s", 4, 1, NULL};
 buttonList[i++] = (MoButton){"1", 4, 5, NULL};
 buttonList[i++] = (MoButton){"2", 3, 7, NULL};
 buttonList[i++] = (MoButton){"3", 4, 9, NULL};
 buttonList[i++] = (MoButton){"y", 6, 6, NULL};
 buttonList[i++] = (MoButton){"t", 5, 7, NULL};
 buttonList[i++] = (MoButton){"b", 6, 8, NULL};

 GtkWidget *window;
 GtkWidget *table;

 /* --- GTK initialization --- */
 gtk_init(&argc, &argv);

 /* --- Create the AuVeTA Control window --- */
 window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

 /* --- Give the window a title. --- */
 gtk_window_set_title(GTK_WINDOW(window), "AuVeTA Control");

 /* --- Set the window size. --- */
 gtk_widget_set_usize(window, 1000, 700);

 /* --- We care if a key is pressed --- */
 gtk_signal_connect(GTK_OBJECT(window), "key_press_event",
        GTK_SIGNAL_FUNC(key_press), NULL);

 /* --- You should always remember to connect the delete event
         *      to the main window. --- */
 gtk_signal_connect(GTK_OBJECT(window), "delete_event",
        GTK_SIGNAL_FUNC(CloseAppWindow), NULL);

 /* --- Create a 10x10 table for the items in the control. --- */
```

```
58  table = gtk_table_new(10, 10, TRUE);
59
60  /* --- Create the control buttons. --- */
61  CreateControlButtons(table);
62
63  /* --- Create the label --- */
64
65
66
67  label = gtk_label_new (NULL);
68  const char *str1 = "Control Mode";
69  const char *format1 = "<span style=\"italic\">%s</span>" ;
70  char *markup1;
71
72  markup1 = g_markup_printf_escaped (format1, str1);
73  gtk_label_set_markup (GTK_LABEL (label), markup1);
74
75  gtk_misc_set_alignment(GTK_MISC(label), 0.96, .58);
76  /* --- Add label to the table --- */
77  gtk_table_attach_defaults(GTK_TABLE(table), label, 0, 6, 0, 7);
78
79  gtk_widget_show(label);
80
81  g_free (markup1);
82
83  /* --- Create the label --- */
84  label = gtk_label_new (NULL);
85  const char *str2 = "Waiting Mode";
86  const char *format2 = "<span style=\"italic\">%s</span>" ;
87  char *markup2;
88
89  markup2 = g_markup_printf_escaped (format2, str2);
90  gtk_label_set_markup (GTK_LABEL (label), markup2);
91
92  gtk_misc_set_alignment(GTK_MISC(label), 0.97, .44);
93
94
95  /* --- Add label to the table --- */
96  gtk_table_attach_defaults(GTK_TABLE(table), label, 0, 8, 0, 7);
97
98  gtk_widget_show(label);
99
100 g_free (markup2);
101
102
103 /* --- Create the label --- */
104 label = gtk_label_new (NULL);
105 const char *str3 = "Tracking Mode";
106 const char *format3 = "<span style=\"italic\">%s</span>" ;
107 char *markup3;
108
109 markup3 = g_markup_printf_escaped (format3, str3);
110 gtk_label_set_markup (GTK_LABEL (label), markup3);
111
112 gtk_misc_set_alignment(GTK_MISC(label), 0.98, .58);
113 /* --- Add label to the table --- */
114 gtk_table_attach_defaults(GTK_TABLE(table), label, 0, 10, 0, 7);
115
116 gtk_widget_show(label);
```

```
117
118   g_free (markup3);
119
120   /* --- Create the label --- */
121   label = gtk_label_new (NULL);
122   const char *str4 = "Remove Object";
123   const char *format4 = "<span style=\"italic\">%s</span>" ;
124   char *markup4;
125
126   markup4 = g_markup_printf_escaped (format4, str4);
127   gtk_label_set_markup (GTK_LABEL (label), markup4);
128
129   gtk_misc_set_alignment(GTK_MISC(label), 0.97, .61);
130   /* --- Add label to the table --- */
131   gtk_table_attach_defaults(GTK_TABLE(table), label, 0, 7, 0, 10);
132
133   gtk_widget_show(label);
134
135   g_free (markup4);
136
137   /* --- Create the label --- */
138   label = gtk_label_new (NULL);
139   const char *str5 = "360  ";
140   const char *format5 = "<span style=\"italic\">%s</span>" ;
141   char *markup5;
142
143   markup5 = g_markup_printf_escaped (format5, str5);
144   gtk_label_set_markup (GTK_LABEL (label), markup5);
145
146   gtk_misc_set_alignment(GTK_MISC(label), 0.949, .51);
147   /* --- Add label to the table --- */
148   gtk_table_attach_defaults(GTK_TABLE(table), label, 0, 8, 0, 10);
149
150   gtk_widget_show(label);
151
152   g_free (markup5);
153
154   /* --- Create the label --- */
155   label = gtk_label_new (NULL);
156   const char *str6 = "Buzzer";
157   const char *format6 = "<span style=\"italic\">%s</span>" ;
158   char *markup6;
159
160   markup6 = g_markup_printf_escaped (format6, str6);
161   gtk_label_set_markup (GTK_LABEL (label), markup6);
162
163   gtk_misc_set_alignment(GTK_MISC(label), 0.96, .61);
164   /* --- Add label to the table --- */
165   gtk_table_attach_defaults(GTK_TABLE(table), label, 0, 9, 0, 10);
166
167   gtk_widget_show(label);
168
169   g_free (markup6);
170
171   /* --- Create the label --- */
172   label = gtk_label_new("Select 1, 2 o 3 for Operation Mode");
173   gtk_misc_set_alignment(GTK_MISC(label), 2, .6);
174
175   /* --- Add label to the table --- */
```

```
176   gtk_table_attach_defaults(GTK_TABLE(table), label, 0, 6, 0, 1);
177   gtk_widget_show(label);
178
179   /* --- Make LABELS visible --- */
180   gtk_container_add(GTK_CONTAINER(window), table);
181   gtk_widget_show(table);
182
183   gtk_widget_show(window);
184
185   /* --- Grab focus for the keystrokes --- */
186   //gtk_widget_grab_focus (buttonList[0].widget);
187
188   gtk_main();
189   free(buttonList);
190   return (0);
191 }
```

6.2.4. Makefile

```
1  SRC_DIR = ./src
2  INCLUDE_DIR = ./incl
3  BIN_DIR = ./bin
4  DOC_DIR = ./doc
5  CFLAGS =
6
7  BIN_OUT = Carrito.out
8
9  CC = gcc
10
11 build:
12   mkdir --parents $(BIN_DIR)
13   $(CC) -I $(INCLUDE_DIR) $(SRC_DIR)/comunicaci n_interfaz.c main.c  -o $(BIN_DIR)/$
       (BIN_OUT) `pkg-config --cflags --libs gtk+-2.0` -lm
14
15 run:
16   ./$(BIN_DIR)/$(BIN_OUT)
17
18 clean:
19   rm -rf $(BIN_DIR)/$(BIN)
20   rm -rf doc
21
22 doc:
23   doxygen ./Doxyfile
24   cd doc/latex && make
```