Escuela Superior de Ingeniería

DISEÑO DE COMPUTADORES EMPOTRADOS

Robot sigue líneas

AUTOR:
Juan Jesús Zamorano Butrón

Puerto Real.

Índice

Índice de figuras

1. Introducción

En este trabajo se lleva a cabo un avance secuencial que consta de ir introduciendo diferentes elementos para acabar teniendo un robot que se mueva siguiendo líneas negras.

Se comenzó utilizando el robot para producir diferentes sonidos a través del zumbador. Se continúo haciendo que esos pitidos variaran según fuese el valor leído por los sensores infrarrojos. Se diferenciaron tres tipos de sonidos que correspondían con los colores negro, blanco u otro. Avanzamos en nuestro diseño del robot haciendo mover las ruedas en las dos direcciones de giro, y desplazando el robot en los cuatro posibles movimientos: hacia delante, hacia atrás, girar a la izquierda y girar a la derecha. Finalmente, se combinó todo esto para crear un robot que siga líneas de color negro.

2. Tecnologías a utilizar

2.1. Descripción MCU

Para este trabajo se utiliza un Arduino Leonardo de 8bit. El microcontrolador cuenta con 135 instrucciones, 32x8 registros de propósito general y una frecuencia de reloj de 16MHz entre otras características. Además, dispone de una memoria flash de 16/32KB, una SRAM interna de 1.25/2.5KB y una EEPROM interna de 512Bytes/1KB. Podemos consultar más información en su datasheet [**1**].

2.2. Descripción robot/placa electrónica: recursos globales

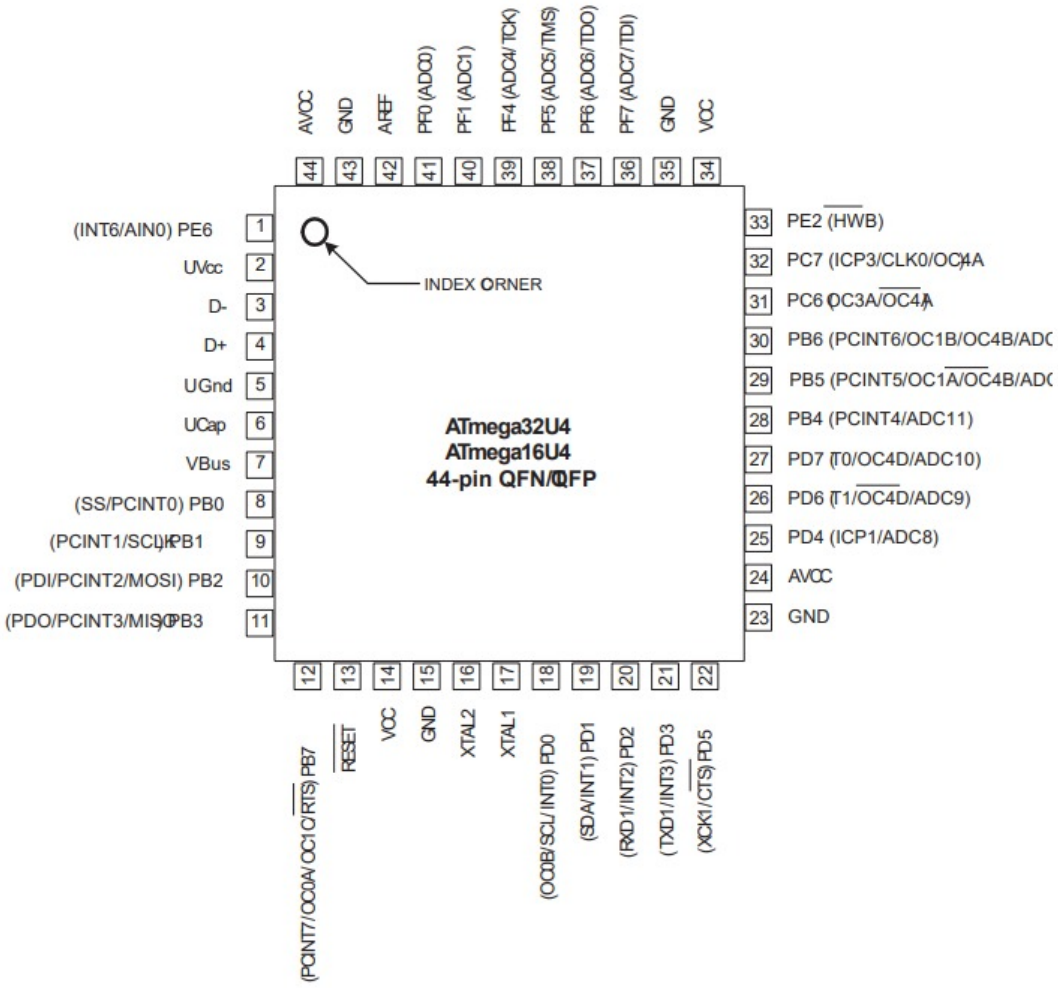

Figura 1: Pinout

2.2.1. Tabla de pines

Número de pin MCU	Funciones del pin MCU	Función usada del pin MCU	Dispositivo del robot conectado	Observaciones
10	PB2 (PDI/PCIND2/MOSI)	PB2	Zumbador	Se debe configurar el puerto como salida
36	PF7 (ADC7/TDI)	ADC7	IR0 - Sensor	Sensor derecho mirando el robot de cara
37	PF6 (ADC6/TDO)	ADC6	IR1 - Sensor	Sensor entre el derecho y central mirando el robot de cara
38	PF5 (ADC5/TMS)	ADC5	IR2 - Sensor	Sensor central mirando el robot de cara
39	PF4 (ADC4/TCK)	ADC4	IR3 - Sensor	Sensor entre el izquierdo y central mirando el robot de cara
40	PF1 (ADC1)	ADC1	IR4 - Sensor	Sensor izquierdo mirando el robot de cara
26	PD6 (T1/OC4D/ADC9)	PD6	EN1 - Motor	Enable de la rueda derecha
1	PE6 (INT6/AIN0)	PE6	EN2 - Motor	Enable de la rueda izquierda
31	PC6 (OC3A/OC4A)	OC4A	PWM1 - Motor	Timer pwm1
27	PD7 (T0/OC4D/ADC10)	OC4D	PWM2 - Motor	Timer pwm2

2.2.2. Dispositivos incluidos

- Zumbador
  Es un elemento que produce sonidos. Su construcción consta de dos elementos, un electroimán o disco piezoeléctrico y una lámina metálica de acero o metal.

- Sensor infrarrojo ITR20001/T [**4**]
  Costa de un diodo emisor de infrarrojos y un fototransistor de silicio NPN. El diodo y el fototransistor se encuentran encerrados uno al lado del otro en eje óptico convergente en una carcasa de termoplástico negro.

- Motor
  El robot incluye un puente H para cada motor construido mediante transistores en lugar de un circuito integrado como el L292D. El circuito se controla utilizando las salidas del MCU EN1, PWM1 (motor izquierdo) y EN2, PWM2 (motor derecho). EN1 y EN2 permiten modificar el sentido del giro encendiendo y apagando los transistores adecuados en cada puente H. Un ejemplo sería:

MOTOR LEFT	EN1 = 0	Transistors Q7 and Q8 $\rightarrow ON$ Transistors Q6 and Q9 $\rightarrow OFF$	Wheel turn FWD
MOTOR LEFT	EN1 = 1	Transistors Q7 and Q8 $\rightarrow OFF$ Transistors Q6 and Q9 $\rightarrow ON$	Wheel turn BCK

2.3. Lenguaje/s de desarrollo

El lenguaje utilizado en el desarrollo es C embebido.

2.4. Entorno de desarrollo: Atmel Studio 7

Atmel Studio [**3**] es un IDE de desarrollo integrado para los microcontroladores AVR y ARM de Atmel, donde se puede empezar a explorar los proyectos incluidos como ejemplos y ejecutar su solución en un kit básico o de evaluación. Atmel Studio se construye sobre la plataforma Microsoft Visual Studio.

2.5. Entorno de programación: Avrdude.exe

El entorno de programación se integra en Atmel Studio accediendo a: Tools → Externals tools → Add. Introduciremos en el apartado Command la ruta del archivo avrdude.exe y en arguments se le indicaran los diferentes parámetros.

Las líneas command y arguments para este trabajo son:

```
C:\Program Files (x86)\Arduino\hardware\tools\avr\bin\avrdude.exe

-C"C:\Program Files (x86)\Arduino\hardware\tools\avr\etc\avrdude.conf"
     -v -patmega32u4 -cavr109 -PCOM4 -b57600 -D -Uflash:w:"$(ProjectDir)
     Debug\$(TargetName).hex":i
```

3. Requisitos del diseño

3.1. Señal acústica de aviso

Para generar una señal acústica se ha utilizado el elemento zumbador del robot. En este trabajo se ha utilizado señales acústicas correspondientes a las notas musicales:

Nota	**Frecuencia en Hz**	**Tiempo en us**
DO	261.6	3822
RE	294	3401
MI	330	3030
SOL	391.99	2551

3.2. Detección de blanco o negro

Se han utilizado los cinco sensores del robot para hacer sonar el zumbador según sea la señal detectada: un color blanco, negro o cualquier otro.

Nota	Frecuencia en Hz	Tiempo en us
SONIDO_BLANCO	500	2000
SONIDO_NEGRO	1000	1000
SONIDO_OTRO_COLOR	10000	100

3.3. Movimiento del robot

Para el movimiento del robot se ha utilizado una señal acústica de inicio que indica que el robot va a comenzar a moverse. Otra cuando el robot se detiene, que suena antes de iniciar el movimiento hacia atrás. Y otra de final que indica que ya no se moverá más. Esta señal será reproducida por el zumbador.

Nota	Frecuencia en Hz	Tiempo en us
START_SOUND	1000	1000
STOP_SOUND	10000	100
END_SOUND	666,66	1500

Además, para conseguir el movimiento del robot hemos hecho uso de los dos motores del robot, uno para cada rueda. A estos motores les podemos cambiar el sentido de giro para así conseguir mover el robot hacia delante o hacia atrás. También, gracias al timer (en PWM) conseguimos modificar la velocidad de giro de las ruedas.

4. Test de los recursos del robot

4.1. Descripción de los tests

4.1.1. Test del zumbador

Para llevar a cabo la prueba del zumbador se ha realizado un programa que consta de la emisión de sonidos a través del zumbador, en mi caso, de la nota musical DO. La emisión del sonido la consigo gracias a la conmutación del pin al que está conectado el zumbador a una frecuencia determinada. Código incluido en el anexo.

4.1.2. Test de un único sensor

Para llevar a cabo la prueba de un sensor se ha realizado un pequeño programa que consta en leer el valor detectado por el sensor y según sea este, hacer sonar un sonido u otro. En mi caso, tenemos tres tipos de sonidos que son sonido blanco, negro u otro color. A su vez, se ha utilizado el programa externo putty [**5**] para poder ver el valor que está leyendo el sensor. Además, este test nos sirve para calibrar el valor que obtenemos para un color negro y para uno blanco. Código incluido en el anexo.

4.1.3. Test de todos los sensores

Para llevar a cabo la prueba de los sensores se ha realizado un programa que consta en que el usuario pueda seleccionar el sensor de donde quiere leer el valor.

Una vez seleccionado el sensor se aplicaría lo explicado en el punto Test de un único sensor. Para mostrar el menú e interaccionar se ha utilizado el programa externo putty. Código incluido en el anexo.

4.1.4. Test de sentido de giro de las ruedas

Para llevar a cabo la prueba del giro de las ruedas se ha creado un pequeño programa que consiste en ir cambiando la dirección de giro de las ruedas cada dos segundos. Esto lo conseguimos modificando el valor del enable del motor. Este enable está por defecto a 0, que significa el movimiento hacia delante por lo que, si lo ponemos a 1, nos moveremos a la inversa. Código incluido en el anexo.

4.1.5. Test de movimientos del robot

Para llevar a cabo la prueba de los diferentes movimientos del robot se ha realizado un pequeño programa que consiste en ir probando cada dos segundo un movimiento diferente. La secuencia de la prueba es: Parado → Hacia atrás → Giro a la izquierda → Hacia delante → Giro a la derecha. Estos movimientos como los explicados en el anterior punto, lo conseguimos modificando el valor del enable del motor. Por ejemplo, para girar hacia un lado indicaremos que cada rueda tenga un sentido de giro, es decir, una vaya hacia delante y la otra hacia atrás. Código incluido en el anexo.

```
inicio
    inicialización de ruedas y timer
    mientras 1 hacer
        parar()
        esperar(2000) // Esperar recibe el tiempo en ms
        atrás()
        esperar(2000)
        giro_izquierda()
        esperar(2000)
        delante()
        esperar(2000)
        giro_derecha()
        esperar(2000)
    fin_mientras
fin_algoritmo
```

Código 1: Código explicado con más claridad en pseudocódigo.

4.2. Resultados de los test

4.2.1. Resultados del test del zumbador

El test funcionó correctamente.

4.2.2. Resultados del test del sensor

El test funcionó correctamente.

4.2.3. Resultados del test de los sensores

El test funcionó correctamente.

4.2.4. Resultados del test de sentido de giro de las ruedas

El test funcionó correctamente.

4.2.5. Resultados del test de movimientos del robot

El test funcionó correctamente.

5. Metodología de diseño

5.1. Señal acústica de aviso

5.1.1. Dispositivos electrónicos a utilizar

Zumbador

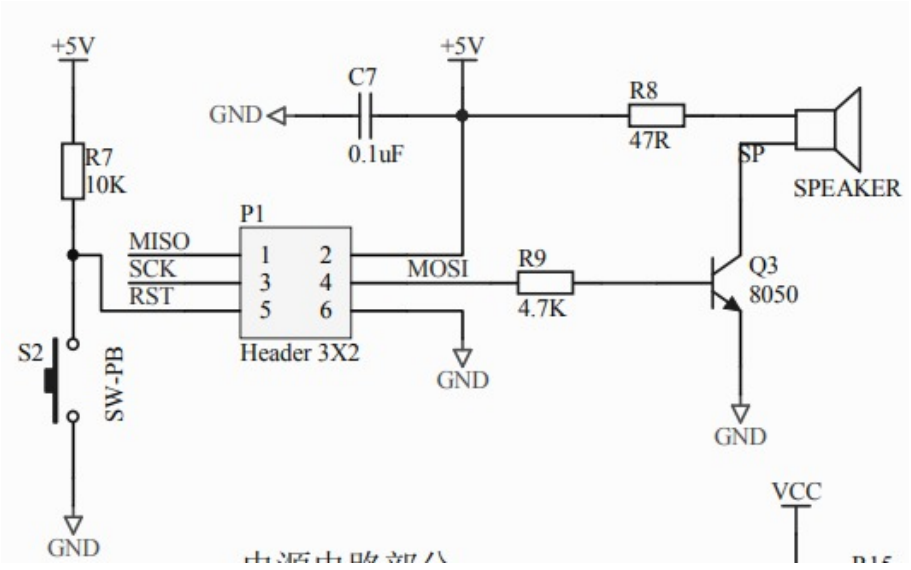

Figura 2: Esquema del Zumbador

5.1.2. Configuración pines

Se configura el pin correspondiente como salida. En este caso será el pin MOSI que como vemos corresponde con el PB2, por tanto, pondremos este como salida para asegurarnos de que suene la señal acústica.

5.1.3. Periféricos de MCU

No se utiliza ningún periférico.

5.1.4. Diseño del software

Se aplican diferentes frecuencias al zumbador, para ver la variación del sonido. Esto lo conseguimos aplicando una forma de onda alterna a las frecuencias indicadas anteriormente para conseguir diferentes tonos. Esta variación del sonido se consigue gracias a ir variando el quantum de tiempo de la rutina de retardo software.

5.2. Detectar blanco y negro

5.2.1. Dispositivos electrónicos a utilizar

Sensores

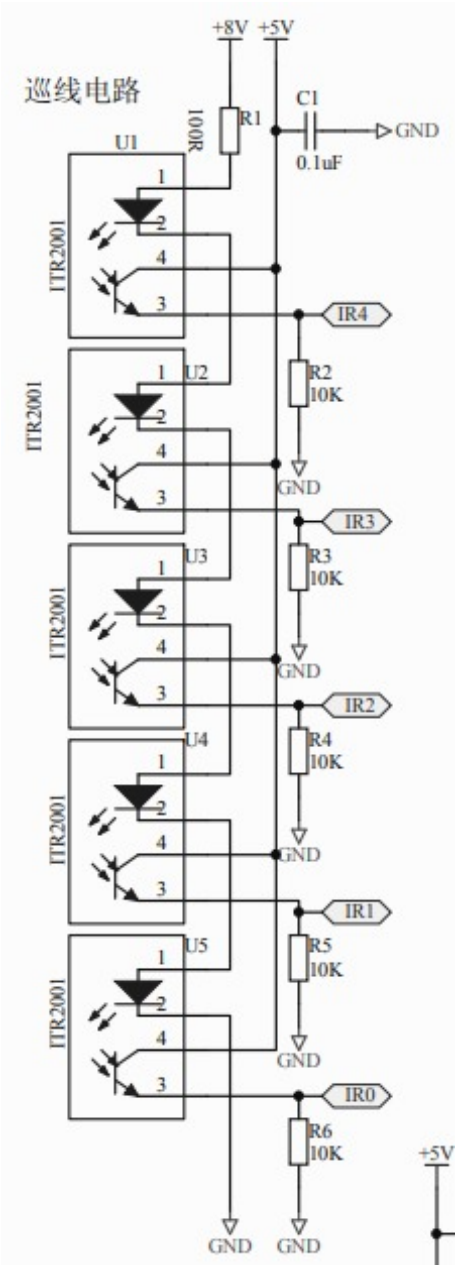

Figura 3: Esquema de los sensores

5.2.2. Configuración pines

Como se puede ver en la tabla de pines, se utilizan los pines 36, 37, 38, 39 y 40 del microcontrolador. Estos pines son los que se encuentran conectados al convertidor analógico digital.

Se configuran los pines correspondiente del ADC en la función initADC(). En esta función, como primer paso ponemos el ADMUX y el ADCRSA todo a 0. Una vez hecho esto configuramos el ADMUX poniendo a uno el REFS0, y, configuramos también el ADCSRA poniendo a 1 el bit ADEN para iniciar una conversión con un

factor de división de 128. Esto lo conseguimos poniendo a 1 el ADPS0, ADPS1 y ADPS2 en el ADCSRA. Con esto, tendremos configurado el ADC en modo polling.

5.2.3. Periféricos de MCU

Convertidor analógico-digital.

5.2.4. Diseño del software

Según sea el valor leído por el sensor aplicaremos una frecuencia o otra al zumbador. En este punto, diferenciaremos entre los valores leído si corresponde al blanco, negro o a un valor entre medio de ambos. Aclarar que los valores leídos están en el rango de valores del ADC, que al ser un registro de 10-bit, irá de de 0 a 1023.

5.3. Movimiento del robot

5.3.1. Dispositivos electrónicos a utilizar

Motores

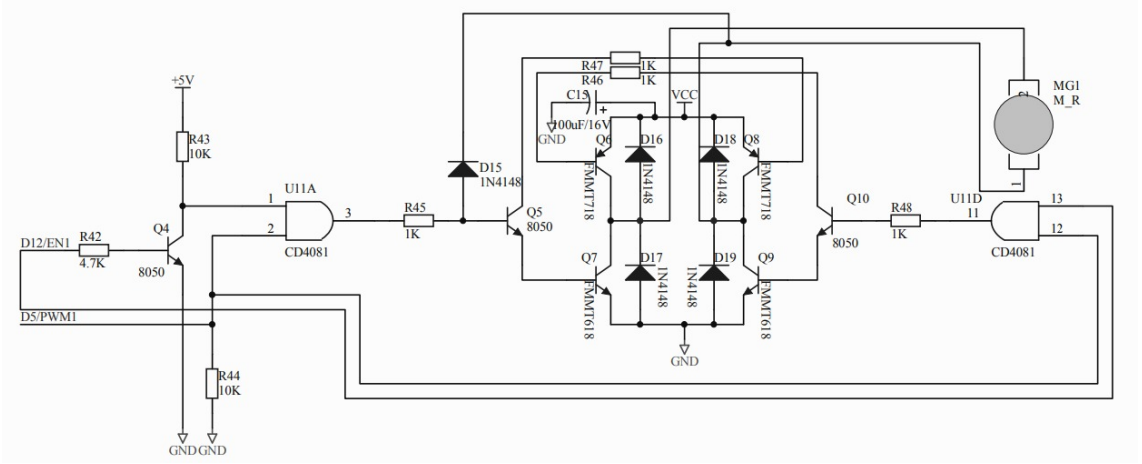

Figura 4: Esquema de uno de los dos motores

5.3.2. Configuración pines

Se configuran los pines correspondientes como salida. En este caso habrá que poner como salida tanto los pines enable de los motores como los pines del PWM. Por tanto, pondremos como salida los pines PD6 y PE6 para los enables y PC6 y PD7 para los PWM.

Además, habrá que configurar el timer 4 para que funcione correctamente en PWM de fase correcta, ya que es el que está conectado. Esto lo conseguimos poniendo a 1 la flag WGM40 en TCCR4D, las flags COM4D1 y PWM4D en TDDR4C y las flags COM4A0 y PWM4A en TCCR4A. Finalmente aplicaremos un divisor de frecuencia de 64 bit poniendo a 1 las flag CS42, CS41 y CS40 en TCCR4B.

5.3.3. Periféricos de MCU

Timers, en concreto como ya se ha dicho se utilizara el timer 4.

5.3.4. Diseño del software

El programa tendrá una máquina de estados. La máquina se caracteriza por tener tres estados INIT, TEST y STOP. Los estados descritos corresponden respectivamente con hacer sonar un beep, ejecutar el test del robot y pararse y hacer sonar tres beeps. Además, tendremos un sonido diferente para ese beep inicial y para los tres beeps finales.

5.4. Robot sigue líneas

5.4.1. Dispositivos electrónicos a utilizar

Motores, sensores infrarrojos y zumbador.

5.4.2. Configuración pines

La configuración de pines para el robot sigue líneas, se encuentra explicada en detalle en los tres puntos anteriores.

5.4.3. Periféricos de MCU

Como también hemos mencionado anteriormente, los periféricos utilizados son el convertidor analógico digital y los timers, en concreto el timer 4.

5.4.4. Diseño del software

Constará de una máquina de estados. En esta máquina distinguiremos cinco estados diferentes que se explicaran a continuación. El movimiento del robot se consigue haciendo uso de los tres sensores centrales del robot. El programa iniciará con un beep (estado INIT). Luego irá comprobando el sensor central, mientras este sea negro avanzará hacia delante, si dejará de detectar negro, comprobará los sensores a su izquierda y derecha para ver donde se encuentra la línea de color negro e iniciará el giro en esa dirección (estado MOVE). Si por el contrario ninguno de los sensores leyera color negro, pararíamos el robot y pitaría (estado STOP). A continuación, se desplazaría un poco hacia atrás en busca de la línea negra (estado MOVE_BACK). Si al tercer intento de ir hacia atrás no consigue encontrar línea negra, el robot emitirá un pitido diferente y se quedará parado para siempre (estado END).

6. Implementación

6.1. Señal acústica de aviso

6.1.1. Grafo de la FMS, diagrama de flujo de las funciones o diagramas UML

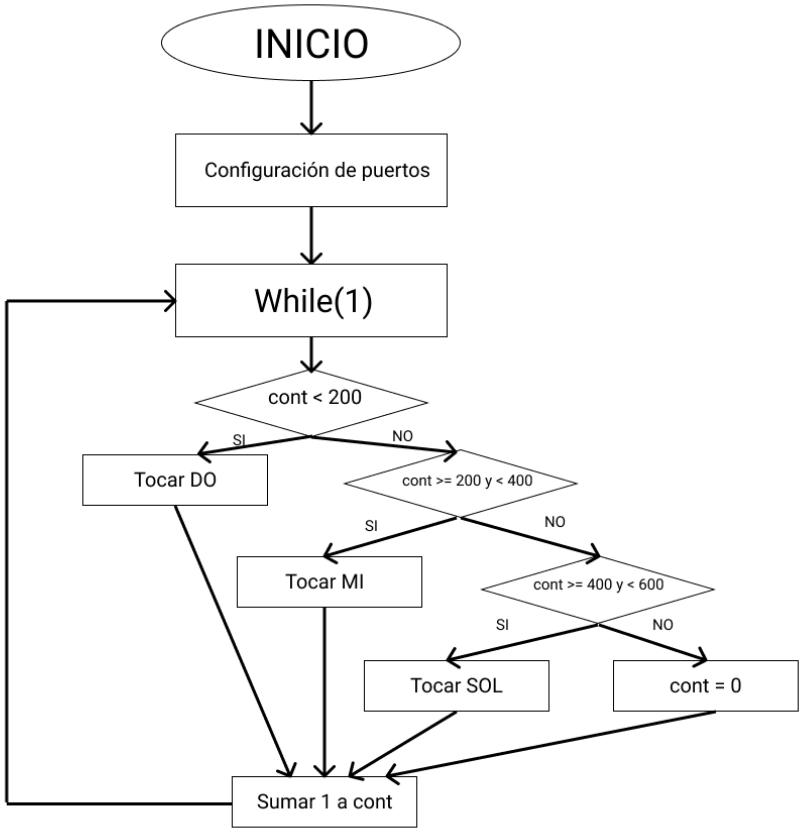

Figura 5: Diagrama de flujo del programa

El código que se utiliza para esto se encuentra introducido en el anexo.

6.1.2. Librerías

Se utilizan las librerías de C: *util/delay.h.* y *avr/io.h.*

6.2. Detectar blanco y negro

6.2.1. Grafo de la FMS, diagrama de flujo de las funciones o diagramas UML

```
1 inicio
2     esperar_conexión_usb
```

```
3       mientras 1 hacer
4           valor <- leer_valor_ADC
5           si (valor <= 300) entonces
6               pitar_Negro
7           sino si (valor >= 900) entonces
8               pitar_Blanco
9           sino
10              pitar_otro
11          fin_si
12      fin_mientras
13  fin_algoritmo
```

Código 2: Código de la aplicación en pseudocódigo.

6.2.2. Librerías

Se utilizan las librerías de C: *util/delay.h.* y *avr/io.h.*

Se incluye una librería propia con las funciones correspondientes del convertidor analógico digital (ADC). Código correspondiente incluido en el anexo.

Se utilizan las librerías externas de comunicación USB.[**2**]

6.3. Movimiento del robot

6.3.1. Grafo de la FMS, diagrama de flujo de las funciones o diagramas UML

Diseño de la máquina de estados:

- **Estados**
  - INIT
  - TEST
  - STOP
- **Transiciones**
  - INIT → TEST
  - TEST → STOP
  - STOP → INIT
  - Si fallara o pasara cualquier cosa, iría a INIT.

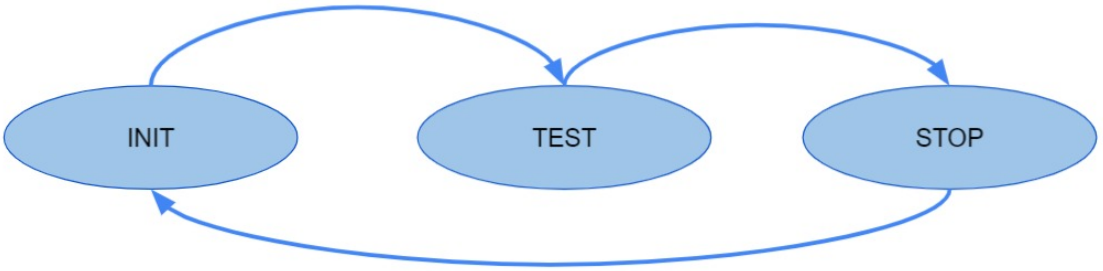

Figura 6: Esquema de la máquina de estados del movimiento del robot

6.3.2. Librerías

Se utilizan las librerías de C: *util/delay.h.* y *avr/io.h.*

Además, se utilizan librerías propias. Estas librerías son BUZZER y WHEELS, la primera se encarga de manejar el zumbador y la segunda con ayuda de otras librerías de manejar los motores de las ruedas. La librería WHEELS se apoya en las librerías TIMERS y OPTICAL, también propias. La librería TIMERS nos configura el timer 4 en pwm de fase correcta y la OPTICAL los distintos sensores infrarrojos. Código correspondiente incluido en el anexo.

6.4. Robot sigue líneas

6.4.1. Grafo de la FMS, diagrama de flujo de las funciones o diagramas UML

Diseño de la máquina de estados:

- **Estados**
  - INIT
  - MOVE
  - STOP
  - MOVE_BACK
  - END
- **Transiciones**
  - INIT → MOVE
  - MOVE → MOVE
  - MOVE → STOP
  - MOVE → END
  - STOP → MOVE_BACK

- MOVE_BACK → MOVE
- END → END
- Si fallara o pasara cualquier cosa, iría a INIT.

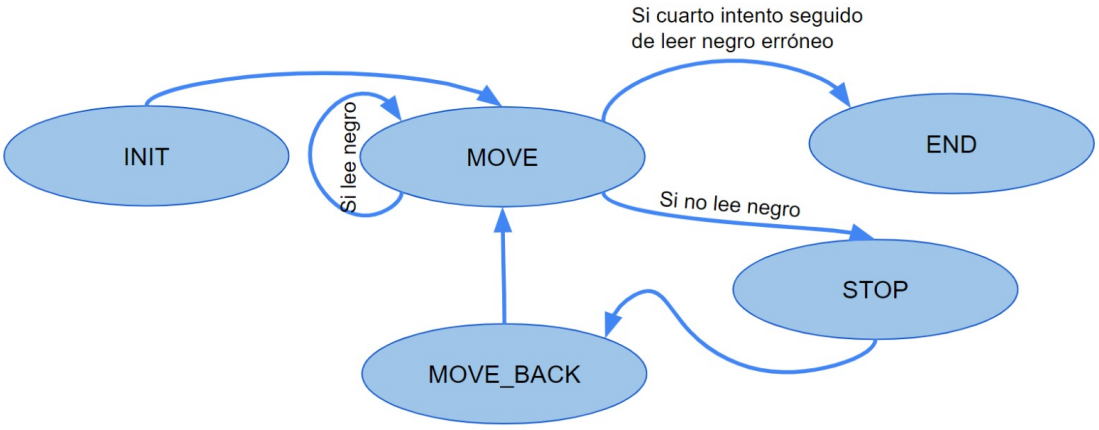

Figura 7: Esquema de la máquina de estados del sigue líneas

6.4.2. Librerías

Se utilizan las librerías de C: *util/delay.h.* y *avr/io.h.*

Además, se utilizan librerías propias. Estas librerías son BUZZER y WHEELS, la primera se encarga de manejar el zumbador y la segunda con ayuda de otras librerías de manejar los motores de las ruedas. La librería WHEELS se apoya en las librerías TIMERS y OPTICAL, también propias. La librería TIMERS nos configura el timer 4 en pwm de fase correcta y la OPTICAL los distintos sensores infrarrojos. Código correspondiente incluido en el anexo.

6.5. Integración del software

Como podemos observar, en el código del robot sigue líneas, tenemos incluidas todas las librerías y códigos directa o indirectamente. Vemos que para hacer sonar los diferentes sonidos hacemos uso de la función playSound de la primera entrega. También, hacemos uso de las funciones del ADC para leer los sensores que se realizaron en la segunda entrega. Además, utilizamos las diferentes funciones de movimiento del robot realizadas en la tercera práctica. Finalmente, podemos decir que gracias a la integración de las distintas entregas, hemos conseguido obtener el proyecto final del robot que se mueve siguiendo una línea negra.

6.6. Descarga del fichero de configuración en MCU

Para descargar el archivo de configuración en el MCU.

1. Compilamos el código (F7).

2. Pulsamos el botón reset en el robot para que este sea programado.

3. Tools → Robot. Donde robot es la herramienta configurada en el punto 2.5.

Referencias

[1] https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf. Última visita: 13/05/2021

[2] http://medesign.seas.upenn.edu/index.php/Guides/MaEvArM-usb. Última visita: 19/05/2021

[3] https://atmel-studio.software.informer.com/Descargar-gratis/. Última visita: 13/05/2021

[4] https://eu.mouser.com/datasheet/2/143/20140706162748548l-597080.pdf. Última visita: 21/05/2021

[5] https://www.putty.org/. Última visita: 21/05/2021

Anexos

Anexo A: Códigos

Códigos: Test

```
#define F_CPU 16000000UL
#include <util/delay.h>
#include  <avr/io.h>

#define BUZZER 2
#define DO 3822

static inline void configPorts();
static inline void toggle(uint8_t LED, uint16_t son);

int main(void)
{
    //--- Setup ----------------------
    configPorts();

    //--- Loop -----------------------
    while (1)
    {
        toggle(BUZZER,DO);
    }
}

static inline void configPorts()
{
    DDRB |= (1<<BUZZER);
}

static inline void toggle(uint8_t LED, uint16_t son)
{
    PORTB ^= (1<<LED);
    _delay_us(son);
}
```

Código 3: Test señal acústica.

```
#define F_CPU 16000000UL
#include <util/delay.h>
#include  <avr/io.h>
```

```
4  #include "m_general.h"
5  #include "m_usb.h"
6  #include "ADC.h"
7
8  #define BUZZER 2
9
10 #define BLANCO 800
11 #define NEGRO 300
12
13 #define SONIDO_BLANCO 2000
14 #define SONIDO_NEGRO 1000
15 #define SONIDO_OTRO_COLOR 100
16
17 #define sensor1 7
18 #define sensor2 6
19 #define sensor3 5
20 #define sensor4 4
21 #define sensor5 1
22
23
24 static inline void configPorts();
25 static inline void toggle_LED(uint8_t LED, uint16_t son);
26
27 int main(void)
28 {
29     unsigned int value;
30     void initADC();
31
32     m_usb_init(); // USB peripheral setup
33     while(!m_usb_isconnected()); // Wait for a USB connection
34     m_usb_tx_string("Hello world\n\r");
35
36
37     while (1)
38     {
39         value = func_ADC(sensor1);
40         m_usb_tx_char(value); // Print the ASCII character
41         m_usb_tx_char('\t');
42         m_usb_tx_uint(value); // Print the decimal equivalent
43         m_usb_tx_char('\n');
44         m_usb_tx_char('\r');
45
46         if(value<NEGRO)
47             toggle_LED(BUZZER,SONIDO_NEGRO);
48         else if(value>BLANCO)
49             toggle_LED(BUZZER,SONIDO_BLANCO);
```

```
50          else
51              toggle_LED(BUZZER,SONIDO_OTRO_COLOR);
52      }
53  }
54
55
56  static inline void configPorts()
57  {
58      DDRB |= (1<<BUZZER);
59  }
60
61
62  static inline void toggle_LED(uint8_t LED, uint16_t son)
63  {
64      for(int i = 0; i<300; i++)
65      {
66          PORTB ^= (1<<LED);
67          _delay_us(son);
68      }
69  }
```

Código 4: Test de prueba de un único sensor.

```
1   #define F_CPU 16000000UL
2   #include<util/delay.h>
3   #include <avr/io.h>
4   #include "m_general.h"
5   #include "m_usb.h"
6   #include "ADC.h"
7
8
9   #define BUZZER 2
10  #define SONIDO_BLANCO 2000
11  #define SONIDO_NEGRO 1000
12  #define SONIDO_OTRO_COLOR 100
13
14  #define BLANCO 800
15  #define NEGRO 400
16
17  #define sensor1 7
18  #define sensor2 6
19  #define sensor3 5
20  #define sensor4 4
```

```
21  #define sensor5 1
22
23
24  static inline void configPorts();
25  static inline void toggle_LED(uint8_t LED, uint16_t son);
26  static inline void mainMenu (void);
27  static inline void sensor_sonar_sonido (unsigned char input);
28
29  int main(void)
30  {
31      unsigned int value = 0;
32      //unsigned char input;
33
34
35      void initADC();
36
37      m_usb_init(); // USB peripheral setup
38      while(!m_usb_isconnected()); // Wait for a USB connection
39      m_usb_tx_char(12);
40      mainMenu();
41
42      while (1)
43      {
44          value = 0;
45          if(m_usb_rx_available())
46          {
47              sensor_sonar_sonido(m_usb_rx_char());
48          }
49      }
50  }
51
52
53  static inline void configPorts()
54  {
55      DDRB |= (1<<BUZZER);
56  }
57
58
59  static inline void toggle_LED(uint8_t LED, uint16_t son)
60  {
61      for(int i = 0; i<300; i++)
62      {
63          PORTB ^= (1<<LED);
64          _delay_us(son);
65      }
66  }
```

```
67
68  static inline void mainMenu (void){
69      m_usb_tx_string(" ----------------------------------
70      "----------------------------------\r\n"
71      " Selecciona el sensor que quieres utilizar:\r\n"
72      " ---------------------------------------------------
73      "-------------------\r\n"
74      " Presiona 'a' para el sensor 1 \r\n"
75      " Presiona 'b' para el sensor 2 \r\n"
76      " Presiona 'c' para el sensor 3 \r\n"
77      " Presiona 'd' para el sensor 4 \r\n"
78      " Presiona 'e' para el sensor 5 \r\n"
79      " ---------------------------------------------------
80      "------------------\r\n"
81      " Press 'h' in anytime to show the MENU\r\n"
82      "---------------------------------------------------
83      "--------------------\r\n");
84  }
85
86
87  static inline void sensor_sonar_sonido (unsigned char input)
88  {
89      switch (input)
90      {
91          case 'A':
92          case 'a':
93              value = func_ADC(sensor1);
94
95              m_usb_tx_string("Valor leido del sensor 1: ");
96              m_usb_tx_char('\t');
97              m_usb_tx_uint(value);
98              m_usb_tx_char('\n');
99              m_usb_tx_char('\r');
100
101             if(value<NEGRO)
102                 toggle_LED(BUZZER,SONIDO_NEGRO);
103             else if(value>BLANCO)
104                 toggle_LED(BUZZER,SONIDO_BLANCO);
105             else
106                 toggle_LED(BUZZER,SONIDO_OTRO_COLOR);
107
108             break;
109
110         case 'B':
111         case 'b':
112             value = func_ADC(sensor2);
```

```
113
114             m_usb_tx_string("Valor leido del sensor 2: ");
115             m_usb_tx_char('\t');
116             m_usb_tx_uint(value);
117             m_usb_tx_char('\n');
118             m_usb_tx_char('\r');
119
120             if(value<NEGRO)
121                 toggle_LED(BUZZER,SONIDO_NEGRO);
122             else if(value>BLANCO)
123                 toggle_LED(BUZZER,SONIDO_BLANCO);
124             else
125                 toggle_LED(BUZZER,SONIDO_OTRO_COLOR);
126
127             break;
128
129         case 'C':
130         case 'c':
131             value = func_ADC(sensor3);
132
133             m_usb_tx_string("Valor leido del sensor 3: ");
134             m_usb_tx_char('\t');
135             m_usb_tx_uint(value);
136             m_usb_tx_char('\n');
137             m_usb_tx_char('\r');
138
139             if(value<NEGRO)
140                 toggle_LED(BUZZER,SONIDO_NEGRO);
141             else if(value>BLANCO)
142                 toggle_LED(BUZZER,SONIDO_BLANCO);
143             else
144                 toggle_LED(BUZZER,SONIDO_OTRO_COLOR);
145
146             break;
147
148         case 'D':
149         case 'd':
150             value = func_ADC(sensor4);
151
152             m_usb_tx_string("Valor leido del sensor 4: ");
153             m_usb_tx_char('\t');
154             m_usb_tx_uint(value);
155             m_usb_tx_char('\n');
156             m_usb_tx_char('\r');
157
158             if(value<NEGRO)
```

```
                toggle_LED(BUZZER,SONIDO_NEGRO);
            else if(value>BLANCO)
                toggle_LED(BUZZER,SONIDO_BLANCO);
            else
                toggle_LED(BUZZER,SONIDO_OTRO_COLOR);

            break;

        case 'E':
        case 'e':
            value = func_ADC(sensor5);

            m_usb_tx_string("Valor leido del sensor 5: ");
            m_usb_tx_char('\t');
            m_usb_tx_uint(value);
            m_usb_tx_char('\n');
            m_usb_tx_char('\r');

            if(value<NEGRO)
                toggle_LED(BUZZER,SONIDO_NEGRO);
            else if(value>BLANCO)
                toggle_LED(BUZZER,SONIDO_BLANCO);
            else
                toggle_LED(BUZZER,SONIDO_OTRO_COLOR);

            break;

        case 'H':    // Call the main menu in anytime
        case 'h':
            m_usb_tx_char(12);    //Limpia la pantalla
            mainMenu ();
            break;

        default:
            m_usb_tx_string
                ("I don't know what to do with that key.\n\r");
            m_usb_tx_string("\r\n");
            break;
    }
}
```

Código 5: Test de prueba de los sensores.

```
#include <avr/io.h>
```

```
#define F_CPU 16000000UL
#include <util/delay.h>
#include "WHEELS.h"


int main(void)
{
    //--- Setup -------------------------
    initWHEELS();            // Configuracion de las ruedas

    // Configuracion del timer 4
    TMR4_PWM_Phase_Init();
    TMR4_PWM_Phase_Duty(255-178,178);
    TMR4_PWM_Phase_Start();

    //--- Loop --------------------------
    while (1)
    {
        _delay_ms(2000);
        PORTD ^= (1<<PORTD6);
        PORTE ^= (1<<PORTE6);
    }
}
```

Código 6: Test de sentido de giro de las ruedas.

```
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>
#include "WHEELS.h"


int main(void)
{
    //--- Setup -------------------------
    initWHEELS();            // Configuracion de las ruedas

    // Configuracion del timer 4
    TMR4_PWM_Phase_Init();
    TMR4_PWM_Phase_Duty(255-178,178);
    TMR4_PWM_Phase_Start();

    //--- Loop --------------------------
```

```
18      while (1)
19      {
20          stop();
21          _delay_ms(2000);
22          backward(178,178);
23          _delay_ms(2000);
24          turn_left(178,178);
25          _delay_ms(2000);
26          forward(178,178);
27          _delay_ms(2000);
28          turn_right(178,178);
29          _delay_ms(2000);
30      }
31  }
```

Código 7: Test de movimientos del robot.

Códigos: Finales

```
1   #define F_CPU 16000000UL
2   #include<util/delay.h>
3   #include <avr/io.h>
4
5   #define BUZZER 2
6   #define DO 3822
7   #define RE 3401
8   #define MI 3030
9   #define SOL 2551
10
11  static inline void configPorts();
12  static inline void toggle(uint8_t LED, uint16_t son);
13
14  int main(void)
15  {
16      //--- Setup ----------------------
17      configPorts();
18      int cont = 0;
19
20      //--- Loop -----------------------
21      while (1)
22      {
23          if(cont < 200)
24              toggle(BUZZER,DO);
25          else if(cont >= 200 && cont < 400)
26              toggle(BUZZER,MI);
```

```
27          else if (cont >= 400 && cont < 600 )
28              toggle(BUZZER,SOL);
29          else
30              cont = 0;
31
32          cont++;
33      }
34  }
35
36  static inline void configPorts()
37  {
38      DDRB |= (1<<BUZZER);
39  }
40
41  static inline void toggle(uint8_t LED, uint16_t son)
42  {
43      PORTB ^= (1<<LED);
44      _delay_us(son);
45  }
```

Código 8: Señales acústicas.

```
1   #include <avr/io.h>
2   #define F_CPU 16000000UL
3   #include<util/delay.h>
4   #include "WHEELS.h"
5   #include "BUZZER.h"
6
7
8   typedef enum
9   {
10      INIT,
11      TEST,
12      STOP
13  } State_t;
14
15  int main(void)
16  {
17      State_t current_State = INIT;
18      State_t next_State = INIT;
19
20      //--- Setup -------------------------
21      initBuzzer();        // Configuracion del zumbador
```

```
22      initWHEELS();         // Configuracion de las ruedas
23
24      // Configuracion del timer 4
25      TMR4_PWM_Phase_Init();
26      TMR4_PWM_Phase_Start();
27
28      stop(); //Comenzamos con el robot parado
29      _delay_ms(1000);
30
31      //--- Loop -------------------------
32      while (1)
33      {
34          switch(current_State)
35          {
36              case INIT:
37                  playSOUND(BUZZER,START_SOUND);
38                  _delay_ms(1000);
39                  next_State = TEST;
40                  break;
41
42              case TEST:
43                  testRobot();
44                  next_State = STOP;
45                  break;
46
47              case STOP:
48                  stop();
49                  _delay_ms(3000);
50                  for(int i = 0; i<3; i++)
51                  {
52                      playSOUND(BUZZER,END_SOUND);
53                      _delay_ms(1000);
54                  }
55                  next_State = INIT;
56                  break;
57
58              default:
59                  next_State = INIT;
60          }
61
62          current_State = next_State;
63      }
64  }
```

Código 9: Movimiento del robot.

```c
#define F_CPU 16000000UL
#include <util/delay.h>
#include <avr/io.h>


#include "BUZZER.h"
#include "WHEELS.h"


typedef enum
{
    INIT,
    MOVE,
    STOP,
    MOVE_BACK,
    END
} State_t;



int main(void)
{
    State_t current_State = INIT;
    State_t next_State = INIT;
    uint8_t value = 0;
    uint8_t flag = 0;

    //--- Setup -------------------------
    initBuzzer();        // Configuracion del zumbador
    initADC();            // Configuracion del convertidor
                         // analogico digital
    initWHEELS();        // Configuracion de las ruedas

    // Configuracion del timer 4
    TMR4_PWM_Phase_Init();
    TMR4_PWM_Phase_Start();

    stop();    //Comenzamos con el robot parado
    _delay_ms(1000);


    //--- Loop -------------------------
    while (1)
    {
```

```
45          switch(current_State)
46          {
47              case INIT:
48                  playSOUND(BUZZER,START_SOUND);
49                  _delay_ms(1000);
50                  next_State = MOVE;
51                  break;
52
53              case MOVE:
54                  flag = moveRobot(func_ADC(sensor3),
55                      func_ADC(sensor2), func_ADC(sensor4));
56                  if(flag == 1)
57                      value++;
58                  else
59                      value = 0;
60
61                  if(value != 0 && value%50 == 0 && value != 200)
62                      next_State = STOP;
63                  else if (value >= 200)
64                  {
65                      next_State = END;
66                      for(int i = 0; i<5; i++)
67                      {
68                          playSOUND(BUZZER,END_SOUND);
69                      }
70                  }
71
72                  break;
73
74              case STOP:
75                  stop();
76                  _delay_ms(2000);
77                  for(int i = 0; i<3; i++)
78                  {
79                      playSOUND(BUZZER,STOP_SOUND);
80                      _delay_ms(500);
81                  }
82                  _delay_ms(2000);
83                  next_State = MOVE_BACK;
84                  break;
85
86              case MOVE_BACK:
87                  backward(SPEEP_WHEEL_LEFT,SPEEP_WHEEL_RIGHT);
88                  _delay_ms(200);
89                  stop();
90                  next_State = MOVE;
```

```
                break;

            case END:

            break;

            default:
                next_State = INIT;
        }

        current_State = next_State;


    }
}
```

Código 10: Robot sigue líneas.

Códigos: Librerías

```
#ifndef ADC_H
#define ADC_H

#include <avr/interrupt.h>

/*
 * Inicializo el convertidor analogico digital para su uso
 * Ponemos a 0 los registros ADMUX y ADCSRA.
 * Configuramos los diferentes flag destacando que ponemos
 * el factor de division en 128.
 */
void initADC();

/*
 * Leemos el valor del ADC de el canal indicado.
 * En esta funcion leeremos el valor del ADC, descartando
 * el primer valor y haciendo una media aritmetica de los
 * dieciseis valores siguientes.
 */
uint16_t func_ADC(uint8_t ADC_channel);

#endif
```

Código 11: Cabecera de la librería ADC.

```
#include "ADC.h"

void initADC()
{
    ADMUX = 0x00;
    ADCSRA= 0x00;

    ADMUX |= (1<<REFS0);
    ADCSRA |= (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
}

uint16_t func_ADC(uint8_t ADC_channel)
{
    uint16_t ADC_accum=0;
    //----- Delete a previous setup ------
    ADMUX=0x00;
    ADCSRA=0x00;
    //----- Initialize the ADC -----------
    ADMUX |= (ADC_channel << MUX0);    // Channel is selected
    ADMUX |= (1 << REFS0);          //
    ADCSRA |= (7 << ADPS0);          // Set pre-scaler
    ADCSRA |= (1<<ADEN);          // Enable ADC (ADEN = '1')
    //----- The first readout is throw away
    ADCSRA |= (1 << ADSC);              //Start a conversion
    while ((ADCSRA & (1 << ADIF))==0);
                // Wait for conversion  completes
    ADCSRA |= (1 << ADIF);          // Clear interrupt flag.
    //----N Samples are acquired to average----------
    for (uint8_t i=16;i>0;i--)
    {
        ADCSRA |= (1 << ADSC);    // Start a conversion
        while ((ADCSRA & (1 << ADIF))==0);
                    // Wait for conversion  completes
        ADCSRA |= (1 << ADIF);    // Clear interrupt flag.
        ADC_accum +=ADC;
    }
    ADCSRA &= ~(1 << ADEN);
    return (ADC_accum >> 4);
}
```

Código 12: Código de la librería ADC.

```c
#ifndef BUZZER_H
#define BUZZER_H

#define BUZZER 2

#define START_SOUND 1000
#define STOP_SOUND 100
#define END_SOUND 1500

/*
 * Funcion de inicializacion del zumbador
 */
static inline void initBuzzer()
{
    DDRB |= (1<<BUZZER);
}

/*
 *    Funcion que hace sonar un sonido. Este sonido se
 *    emitira a traves del zumbador con un periodo de son.
 */
static inline void playSOUND(uint8_t LED, uint16_t son)
{
    for(int i = 0; i<300; i++)
    {
        PORTB ^= (1<<LED);
        _delay_us(son);
    }
}

#endif
```

Código 13: Cabecera de la librería BUZZER.

```c
#ifndef OPTICAL_H
#define OPTICAL_H

#include "m_general.h"
#include "m_usb.h"
#include "ADC.h"
#include "BUZZER.h"

```

```
9  #define BUZZER 2
10
11 #define sensor1 7
12 #define sensor2 6
13 #define sensor3 5
14 #define sensor4 4
15 #define sensor5 1
16
17 #define BLANCO 800
18 #define NEGRO 880
19
20 #define SONIDO_BLANCO 2000
21 #define SONIDO_NEGRO 1000
22 #define SONIDO_OTRO_COLOR 100
23
24 /*
25  * Funcion que muestra un menu por pantalla
26  */
27 static inline void mainMenu (void)
28 {
29     m_usb_tx_string(" ------------------------------\r\n"
30     " Selecciona el sensor que quieres utilizar:\r\n"
31     " ----------------------------------------------\r\n"
32     " Presiona 'a' para el sensor 1 \r\n"
33     " Presiona 'b' para el sensor 2 \r\n"
34     " Presiona 'c' para el sensor 3 \r\n"
35     " Presiona 'd' para el sensor 4 \r\n"
36     " Presiona 'e' para el sensor 5 \r\n"
37     " ----------------------------------------------\r\n"
38     " Press 'h' in anytime to show the MENU\r\n"
39     "----------------------------------------------\r\n");
40 }
41
42 /*
43  * Funcion realizada para probar los diferentes sensores.
44  * El valor leido sera mostrado por pantalla y segun sea este
45  * sonara un sonido u otro diferenciando entre blanco, negro y
46  * otro color
47  */
48 void sensor_sonar_sonido (unsigned char input);
49
50 #endif
```

Código 14: Cabecera de la librería OPTICAL.

```c
#include "OPTICAL.h"

void sensor_sonar_sonido (unsigned char input)
{
    unsigned int value = 0;
    switch (input)
    {
        case 'A':
        case 'a':
            value = func_ADC(sensor1);

            m_usb_tx_string("Valor leido del sensor 1: ");
            m_usb_tx_char('\t');
            m_usb_tx_uint(value);
            m_usb_tx_char('\n');
            m_usb_tx_char('\r');

            if(value<NEGRO)
                playSOUND(BUZZER,SONIDO_NEGRO);
            else if(value>BLANCO)
                playSOUND(BUZZER,SONIDO_BLANCO);
            else
                playSOUND(BUZZER,SONIDO_OTRO_COLOR);

            break;

        case 'B':
        case 'b':
            value = func_ADC(sensor2);

            m_usb_tx_string("Valor leido del sensor 2: ");
            m_usb_tx_char('\t');
            m_usb_tx_uint(value);
            m_usb_tx_char('\n');
            m_usb_tx_char('\r');

            if(value<NEGRO)
                playSOUND(BUZZER,SONIDO_NEGRO);
            else if(value>BLANCO)
                playSOUND(BUZZER,SONIDO_BLANCO);
            else
                playSOUND(BUZZER,SONIDO_OTRO_COLOR);

            break;
```

```
45
46          case 'C':
47          case 'c':
48              value = func_ADC(sensor3);
49
50              m_usb_tx_string("Valor leido del sensor 3: ");
51              m_usb_tx_char('\t');
52              m_usb_tx_uint(value);
53              m_usb_tx_char('\n');
54              m_usb_tx_char('\r');
55
56              if(value<NEGRO)
57                  playSOUND(BUZZER,SONIDO_NEGRO);
58              else if(value>BLANCO)
59                  playSOUND(BUZZER,SONIDO_BLANCO);
60              else
61                  playSOUND(BUZZER,SONIDO_OTRO_COLOR);
62
63              break;
64
65          case 'D':
66          case 'd':
67              value = func_ADC(sensor4);
68
69              m_usb_tx_string("Valor leido del sensor 4: ");
70              m_usb_tx_char('\t');
71              m_usb_tx_uint(value);
72              m_usb_tx_char('\n');
73              m_usb_tx_char('\r');
74
75              if(value<NEGRO)
76                  playSOUND(BUZZER,SONIDO_NEGRO);
77              else if(value>BLANCO)
78                  playSOUND(BUZZER,SONIDO_BLANCO);
79              else
80                  playSOUND(BUZZER,SONIDO_OTRO_COLOR);
81
82              break;
83
84          case 'E':
85          case 'e':
86              value = func_ADC(sensor5);
87
88              m_usb_tx_string("Valor leido del sensor 5: ");
89              m_usb_tx_char('\t');
90              m_usb_tx_uint(value);
```

```
91              m_usb_tx_char('\n');
92              m_usb_tx_char('\r');
93
94              if(value<NEGRO)
95                  playSOUND(BUZZER,SONIDO_NEGRO);
96              else if(value>BLANCO)
97                  playSOUND(BUZZER,SONIDO_BLANCO);
98              else
99                  playSOUND(BUZZER,SONIDO_OTRO_COLOR);
100
101             break;
102
103         case 'H':    // Call the main menu in anytime
104         case 'h':
105             m_usb_tx_char(12);    //Limpia la pantalla
106             mainMenu();
107             break;
108
109         default:
110             m_usb_tx_string("I don't know what to do
111                         "with that key.\n\r");
112             m_usb_tx_string("\r\n");
113             break;
114     }
115 }
```

Código 15: Código de la librería OPTICAL.

```
1  #ifndef TIMERS_H
2  #define TIMERS_H
3
4  #include <avr/io.h>
5
6  /*
7   * Funcion de inicializacion y configuracion de las flag
8   * correspondientes para que el timer funcione correctamente.
9   */
10 static inline void TMR4_PWM_Phase_Init(void)
11 {
12     TCNT4=0;
13
14     //Configuracion del timer en fase correcta
15     TCCR4D |= (1<<WGM40);
```

```
16      TCCR4C |= (1<<COM4D1) | (1<<PWM4D);
17      TCCR4A |= (1<< COM4A0) | (1<<PWM4A);
18  }
19
20  /*
21   * Funcion que configura el ciclo de trabajo del timer.
22   * Esta funcion recibe dos parametros que son los que
23   * se asignaran a la rueda izquierda y derecha.
24   */
25  static inline void TMR4_PWM_Phase_Duty(uint8_t valueOCR4A,
26                                         uint8_t valueOCR4D)
27  {
28      OCR4A = valueOCR4A;
29      OCR4D = valueOCR4D;
30  }
31
32  /*
33   * Funcion que inicializa el timer y configura el
34   * divisor de frecuencia de 64 bit.
35   */
36  static inline void TMR4_PWM_Phase_Start(void)
37  {
38      TCCR4B |= (1<<CS42) | (1<<CS41) | (1<<CS40); //Divisor de 64
39  }
40
41  #endif
```

Código 16: Cabecera de la librería TIMERS.

```
1   #ifndef WHEELS_H
2   #define WHEELS_H
3
4   #include "TIMERS.h"
5   #include "OPTICAL.h"
6   #include "m_general.h"
7   #include "m_usb.h"
8   #include "ADC.h"
9
10  #define SPEEP_WHEEL_LEFT 50
11  #define SPEEP_WHEEL_RIGHT 50
12
13
14  /*
```

```
15  * Funcion de inicializacion de las ruedas
16  */
17 static inline void initWHEELS()
18 {
19     DDRD |= (1<<DDD7) | (1<<DDD6);
20     DDRC |= (1<<PORTC6);
21     DDRE |= (1<<PORTE6);
22 }
23
24
25 /*
26  * Funcion que se encarga de parar las ruedas
27  */
28 static inline void stop()
29 {
30     TMR4_PWM_Phase_Duty(255,0);
31 }
32
33 /*
34  * Funcion que hace que el robot vaya hacia delante
35  */
36 static inline void forward(uint8_t speedLeft,
37                                 uint8_t speedRight)
38 {
39     PORTD &= (~(1<<PORTD6));
40     PORTE &= (~(1<<PORTE6));
41
42     TMR4_PWM_Phase_Duty(255-speedLeft,speedRight);
43 }
44
45 /*
46  * Funcion que hace que el robot vaya hacia atras
47  */
48 static inline void backward(uint8_t speedLeft,
49                                 uint8_t speedRight)
50 {
51     PORTD |= ((1<<PORTD6));
52     PORTE |= ((1<<PORTE6));
53
54     TMR4_PWM_Phase_Duty(255-speedLeft,speedRight);
55 }
56
57 /*
58  * Funcion que hace que el robot gire a la derecha
59  */
60 static inline void turn_right(uint8_t speedLeft,
```

```
61                                      uint8_t speedRight)
62  {
63      PORTD &= (~(1<<PORTD6));
64      PORTE |= ((1<<PORTE6));
65
66      TMR4_PWM_Phase_Duty(255-speedLeft,speedRight);
67  }
68
69  /*
70   * Funcion que hace que el robot gire a la izquierda
71   */
72  static inline void turn_left(uint8_t speedLeft,
73                                          uint8_t speedRight)
74  {
75      PORTD |= ((1<<PORTD6));
76      PORTE &= (~(1<<PORTE6));
77
78      TMR4_PWM_Phase_Duty(255-speedLeft,speedRight);
79  }
80
81  /*
82   * Funcion de test de movimientos del robot
83   */
84  void testRobot();
85
86
87  /*
88   * Funcion que controla el movimiento del robot.
89   * Mientras que el sensor central lee negro el robot ira
90   * hacia delante. Cuando este sensor no lea negro,  se
91   * comprobaran el izquierdo y derecho para hacerlo girar al
92   * lado que le corresponda.
93   */
94  uint8_t moveRobot(uint16_t central, uint16_t left,
95                                          uint16_t rigth);
96
97  #endif
```

Código 17: Cabecera de la librería WHEELS.

```
1  #include "WHEELS.h"
2
3  void testRobot()
```

```
{
    forward(SPEEP_WHEEL_LEFT,SPEEP_WHEEL_RIGHT);
    _delay_ms(2000);
    turn_left(SPEEP_WHEEL_LEFT,SPEEP_WHEEL_RIGHT);
    _delay_ms(2000);
    backward(SPEEP_WHEEL_LEFT,SPEEP_WHEEL_RIGHT);
    _delay_ms(2000);
    turn_right(SPEEP_WHEEL_LEFT,SPEEP_WHEEL_RIGHT);
    _delay_ms(2000);
    stop();
    _delay_ms(2000);
}


uint8_t moveRobot(uint16_t central, uint16_t left,
                                    uint16_t right)
{
    if(central < NEGRO)
    {
        forward(SPEEP_WHEEL_LEFT,SPEEP_WHEEL_RIGHT);
    }
    else if (left < NEGRO)
    {
        turn_left(SPEEP_WHEEL_LEFT,SPEEP_WHEEL_RIGHT);
    }
    else if (right < NEGRO)
    {
        turn_right(SPEEP_WHEEL_LEFT,SPEEP_WHEEL_RIGHT);
    }
    else
    {
        return 1;
    }
    return 0;
}
```

Código 18: Código de la librería WHEELS.

Anexo B: Hoja de características

Enlace al datasheet del Arduino Leonardo: [**1**] Enlace al sensor infrarrojo: [**4**]

Anexo C: Esquema

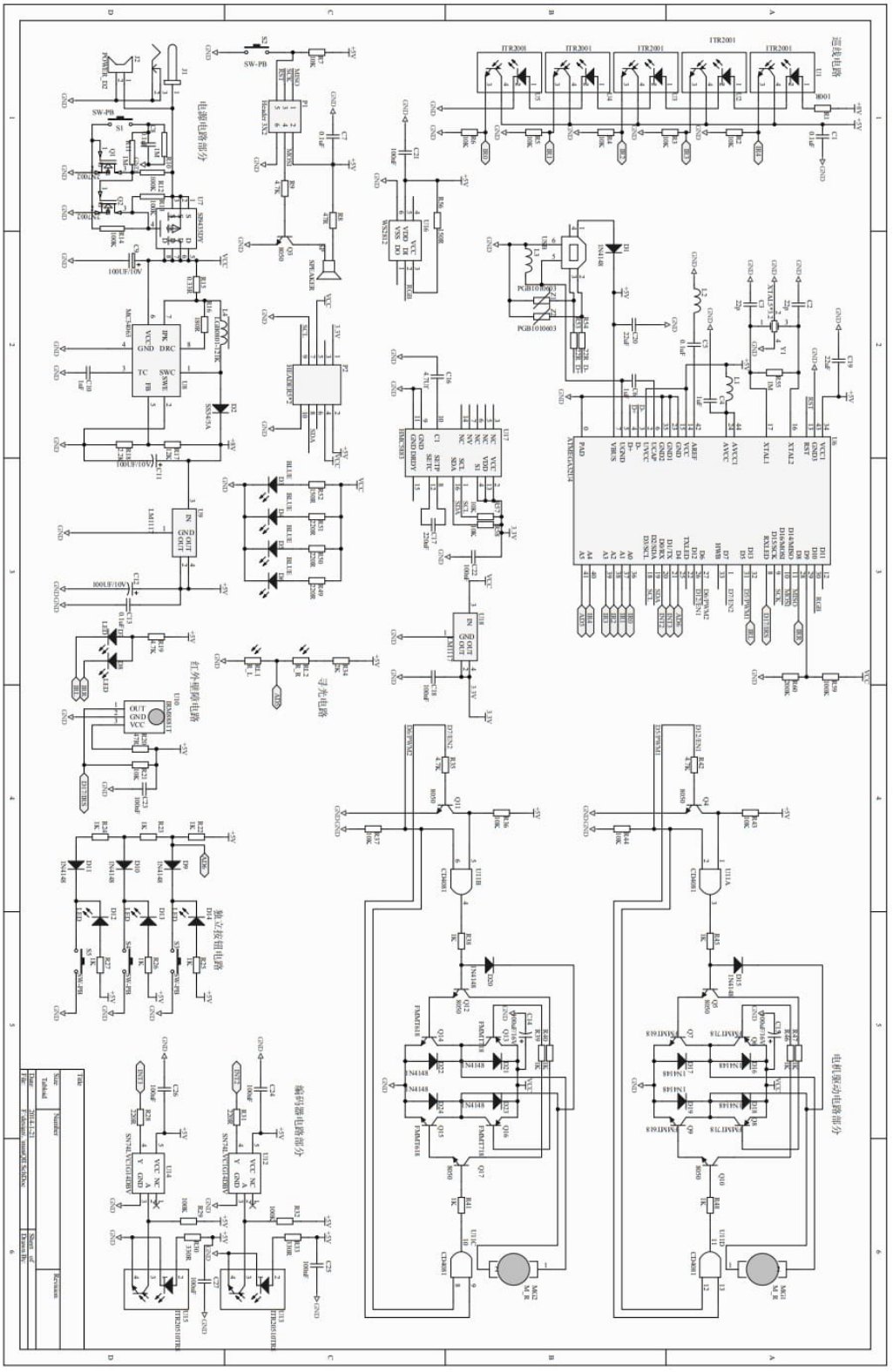

Figura 8: Esquema del robot en vertical

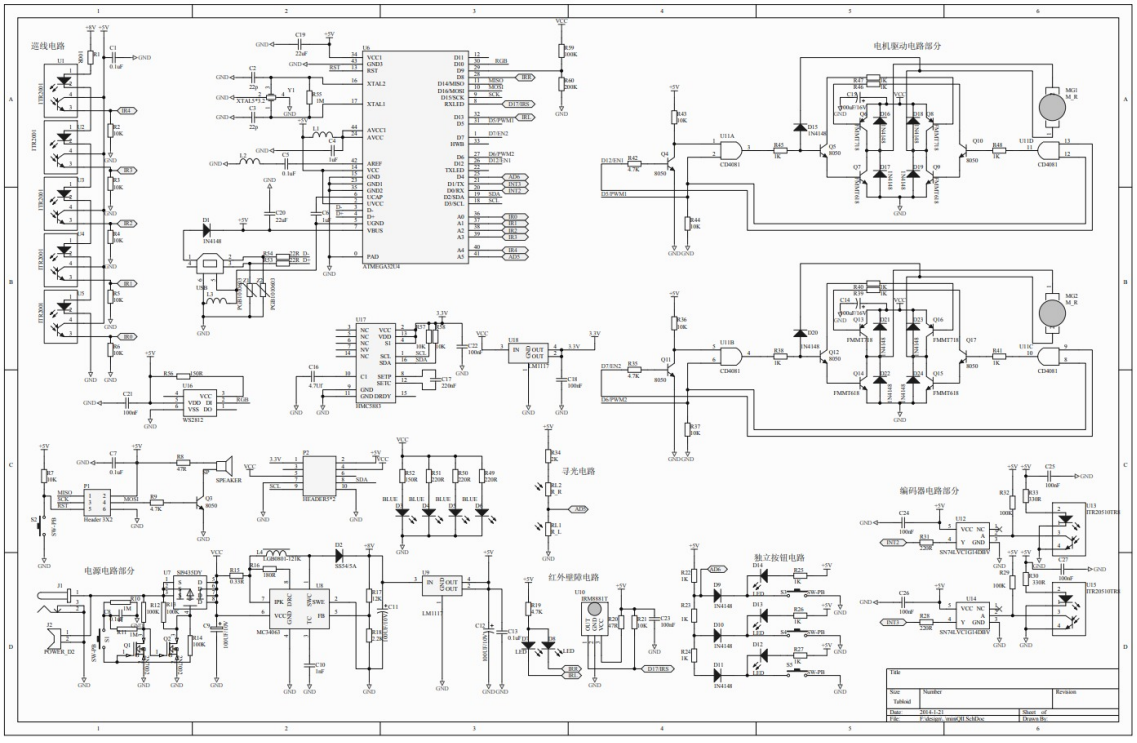

Figura 9: Esquema del robot en horizontal