1. Codigo fuente

main.c

```
1
2  #include "main.h"
3
4  //Delacaracion de variables globales
5  volatile uint8_t    value, flag_piso_blanco;
6
7  //Variables encoders
8  volatile uint32_t   cuenta_encoder_izquierda, cuenta_encoder_derecha;
9
10 //Variables sensores pared
11 volatile uint8_t    sensor_active, status_flag;
12 volatile uint32_t   interrupciones_timer_1, distancia;
13
14
15 int main(void)
16 {
17     usart_init();   //Inicializacion de UART para enviar/recibir datos por
           serie.
18     sei();      //Activacion de las interrupciones.
19
20
21     inicializar_puertos_sensores_pared();
22     inicializar_sensores_piso();
23     inicializar_encoders();     //Al inicializar los encoders lo que se
          hace es configurar el TIMER 0 y el TIMER 2
24     inicializar_puertos_motores();
25     inicializar_PWM();    //Al inicializar el PWM lo que se hace es
          configurar el TIMER 1
26
27     cuenta_encoder_derecha = 0;
28     cuenta_encoder_izquierda = 0;
29     _delay_ms(500);
30
31
32   for (;;){
33
34         //Se corrige el rumbo, midiendo la distancia a la pared derecha.
35         motores_corregir_rumbo();
36
37         //Se avanza una distancia muy pequena.
38         motores_avanzar(200,200, 40);
39
40         //Se mide a la derecha para saber si hay pared o no.
41         distancia = medicion_distancia_pared(SENSOR_PARED_DER);
42
43         //Si se entra aca es que no hay pared a la derecha, entonces, se
              giro a la derecha
44         if(distancia > DISTANCIA_GRANDE){
45
```

```
46              //Se revisa si el piso es de color blanco, para saber si llego
                    al final del laberinto
47              //Para esto, primero se encienden los sensores del piso y se
                   mide. Si hay suelo blanco,
48              //se avanza y se mide de nuevo. Si hay doble confirmacion,
                   entonces estoy en la casilla final.
49              encender_sensores_piso();
50
51              _delay_us(100);
52
53              if((PINA & (1 << PA2)) == (1 << PA2)){
54
55                  motores_avanzar(200,200,200);
56
57                  motores_detener();
58
59                  _delay_ms(10000);
60
61              }
62
63              //Si se llega a este punto, el suelo es color negro.
64              apagar_sensores_piso();
65
66              motores_avanzar(200,200, 200);
67
68              motores_rotar_der_90_grados();
69
70              motores_avanzar(200,200, 450);
71
72          }
73
74          //Si se entra aca, es que hay una pared a la derecha
75          else{
76
77              distancia = medicion_distancia_pared(SENSOR_PARED_CEN);
78
79              //Si se entra acÃ¡, es que hay pared a la derecha, y al frente
80              if(distancia < DISTANCIA_CHICA){
81
82                  motores_rotar_izq_90_grados();
83
84                  _delay_ms(10);
85
86                  distancia = medicion_distancia_pared(SENSOR_PARED_CEN);
87
88                  _delay_ms(10);
89
90                  //Si se entra aca, es que hay pared a la derecha, al
                       frente y a la izquierda.
91                  //Entonces rota 180 grados.
92                  //La distancia tiene que se mayor a distancia chica,
                       porque a veces no queda centrado en la casilla
93                  if(distancia < 0x00001800){
94
```

```
95                          motores_rotar_izq_90_grados();
96
97                      }
98
99                      motores_avanzar(200,200, 450);
100                 }
101             }
102         }
103
104     return 0;
105 }
106
107
108 //Inicializacion puerto serie
109 void usart_init()
110 {
111   UCSRB |= (1<<RXCIE) | (1 << RXEN) | (1 << TXEN);  //Encender recepcion
        transmicion
112
113   UCSRC |= (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);  //Configuro para
        usar 8 caracteres
114
115   UBRRL = BAUD_PRESCALE;                    // Cargo la parte baja del
        registro
116
117   UBRRH = (BAUD_PRESCALE >> 8);                  // Cargo la parte alta del
        registro
118 }
119
120 //Interrupcion puerto serie
121 ISR (USART_RXC_vect){
122
123     //Echo: Esta funcion recibe un dato por el puerto serie y lo reenvia.
          Sirve para probar que la conexion esta OK
124   value = UDR;     // Tomo el valor recibido, y lo cargo en la variable
        value
125   UDR = value;  // Cargo el buffer con lo almacenado en la variable value
126 }
127
128
129 //Interrupcion Timer 1
130 ISR (TIMER1_OVF_vect){
131
132     if (sensor_active)
133         interrupciones_timer_1++;
134
135   if(interrupciones_timer_1 > MAX_INTERRUPCIONES_SENSOR_DISTANCIA)
136         status_flag = 1;
137 }
138
139 //Interrupcion INT 0
140 ISR (INT0_vect)
141 {
142     cuenta_encoder_derecha++;
```

```
143
144 }
145
146 //Interrupcion INT 1
147 ISR (INT1_vect)
148 {
149     cuenta_encoder_izquierda++;
150
151 }
152
153
154 //interrupcion sensores de piso
155
156 ISR (INT2_vect){
157
158     if((PINA & (1 << PA2)) == (1 << PA2))
159
160         flag_piso_blanco = TRUE;
161
162     //Finalmente hacemos un clear al registro GIFR de interrupcion como se
          pide en hoja de datos [1].
163     GIFR &= ~(1<<INTF2);
164
165 }
```

main.h

```
1  #ifndef _MAIN_H_
2  #define _MAIN_H_
3
4  #include <avr/io.h>
5  #include <avr/interrupt.h>
6  #include <util/delay.h>
7  #include "pwm.h"
8  #include "motores.h"
9  #include "sensores_pared.h"
10 #include "sensores_piso.h"
11 #include "encoders.h"
12
13 #define USART_BAUDRATE  9600
14 #define BAUD_PRESCALE   (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)
15
16 #define MAX_STRING  25
17
18 #define DISTANCIA_CHICA    0x00000E00
19 #define DISTANCIA_GRANDE   0x00003000//0x8C
20
21 #define MAX_TIME     100000
22
23 #define MAX_INTERRUPCIONES_SENSOR_DISTANCIA 1000
24
25
```

```
26 #endif /* _MAIN_H_ */
27
28 //Definicion de prototipos
29 void usart_init(void);
```

motores.c

```
1  #include "motores.h"
2
3  extern volatile uint32_t      cuenta_encoder_derecha,
       cuenta_encoder_izquierda;
4
5
6  void inicializar_puertos_motores(void){
7
8    //Configuro pines como salida
9    DDRD |= MOTOR_DER_DIRECTION;
10   DDRB |= MOTOR_DER_BRAKE;
11
12   DDRB |= MOTOR_IZQ_BRAKE;
13        DDRD |= MOTOR_IZQ_DIRECTION;
14
15        //Detengo los motores una vez que inicializo los puertos.
16        motores_detener();
17 }
18
19
20
21 void motores_detener(void){
22
23   //Freno: poner brake en alto
24   PORTB |= MOTOR_DER_BRAKE; //Pongo en alto el pin BRAKE del motor 1
25   PORTB |= MOTOR_IZQ_BRAKE; //Pongo en alto el pin BRAKE del motor 2
26 }
27
28
29
30 void motores_retroceder(uint8_t velocidad_izquierda, uint8_t
       velocidad_derecha){
31
32   //Avance sentido 1: PWM H - Direction H - Brake L
33
34   PORTD |= MOTOR_DER_DIRECTION;    //Pongo en alto el pin DIRECTION del
         motor 1
35   PORTB &= ~MOTOR_DER_BRAKE;       //Pongo en bajo el pin BRAKE del motor 1
36
37   PORTD &= ~MOTOR_IZQ_DIRECTION;   //Pongo en bajo el pin DIRECTION del
         motor 2
38   PORTB &= ~MOTOR_IZQ_BRAKE;       //Pongo en bajo el pin BRAKE del motor 2
39
40     variar_PWM(velocidad_izquierda, velocidad_derecha);
41
```

```
42 }
43
44
45 void motores_avanzar(uint8_t velocidad_izquierda, uint8_t
       velocidad_derecha, uint32_t cantidad_cuentas){
46
47   uint8_t  flag_fin_avance = FALSE;
48     uint32_t cuenta_encoder_derecha_anterior;
49
50     cuenta_encoder_derecha = 0;
51     cuenta_encoder_izquierda = 0;
52     cuenta_encoder_derecha_anterior = 0;
53
54     variar_PWM(velocidad_izquierda, velocidad_derecha);
55
56     PORTD &= ~MOTOR_DER_DIRECTION;  //Pongo en bajo el pin DIRECTION del
           motor 1
57   PORTB &= ~MOTOR_DER_BRAKE;       //Pongo en bajo el pin BRAKE del motor 1
58
59   PORTD |= MOTOR_IZQ_DIRECTION;    //Pongo en alto el pin DIRECTION del
         motor 2
60   PORTB &= ~MOTOR_IZQ_BRAKE;       //Pongo en bajo el pin BRAKE del motor 2
61
62
63     while(flag_fin_avance == FALSE){
64
65         if((cuenta_encoder_derecha_anterior + 35) >=
               cuenta_encoder_derecha){
66
67             cuenta_encoder_derecha_anterior = cuenta_encoder_derecha;
68
69             if(cuenta_encoder_derecha < cuenta_encoder_izquierda){
70
71                 if(velocidad_derecha < 240)
72                     variar_PWM(velocidad_izquierda, velocidad_derecha +=
                           5);
73
74                 }
75
76             if(cuenta_encoder_derecha > cuenta_encoder_izquierda){
77
78                 if(velocidad_derecha > 130)
79                     variar_PWM(velocidad_izquierda, velocidad_derecha -=
                           5);
80
81             }
82         }
83
84         if(cantidad_cuentas < cuenta_encoder_derecha){
85
86             PORTB |= MOTOR_DER_BRAKE; //Pongo en alto el pin BRAKE del
                   motor 1
87             PORTB |= MOTOR_IZQ_BRAKE; //Pongo en alto el pin BRAKE del
                   motor 2
```

```
88
89
90              flag_fin_avance = TRUE;
91
92          }
93      }
94  }
95
96
97
98  void motores_rotar_der_90_grados(void){
99
100     uint8_t  flag_fin_rotacion = FALSE;
101     uint32_t cuenta_inicial_encoder_derecho     =   cuenta_encoder_derecha
            ;
102     uint32_t cuenta_inicial_encoder_izquierdo   =
            cuenta_encoder_izquierda;
103
104     variar_PWM(210, 210);
105
106     PORTD &= ~MOTOR_DER_DIRECTION;  //Pongo en bajo el pin DIRECTION del
            motor 1
107     PORTB &= ~MOTOR_DER_BRAKE;      //Pongo en bajo el pin BRAKE del motor
             1
108
109     PORTD &= ~MOTOR_IZQ_DIRECTION;  //Pongo en bajo el pin DIRECTION del
            motor 2
110     PORTB &= ~MOTOR_IZQ_BRAKE;      //Pongo en bajo el pin BRAKE del motor
             2
111
112     while(flag_fin_rotacion == FALSE){
113
114         if(cuenta_encoder_derecha >= (cuenta_inicial_encoder_derecho +
                360)){
115
116             if(cuenta_encoder_izquierda >= (
                    cuenta_inicial_encoder_izquierdo + 360)){
117
118                 PORTB |= MOTOR_DER_BRAKE; //Pongo en alto el pin BRAKE del
                        motor 1
119                 PORTB |= MOTOR_IZQ_BRAKE; //Pongo en alto el pin BRAKE del
                        motor 2
120
121                 flag_fin_rotacion = TRUE;
122
123             }
124         }
125     }
126 }
127
128
129 void motores_rotar_izq_90_grados(void){
130
131     uint8_t  flag_fin_rotacion = FALSE;
```

```
132     uint32_t cuenta_inicial_encoder_derecho     =   cuenta_encoder_derecha
            ;
133     uint32_t cuenta_inicial_encoder_izquierdo   =
            cuenta_encoder_izquierda;
134
135     variar_PWM(210, 210);
136
137     PORTD |= MOTOR_DER_DIRECTION;    //Pongo en alto el pin DIRECTION del
            motor 1
138     PORTB &= ~MOTOR_DER_BRAKE;       //Pongo en bajo el pin BRAKE del motor
             1
139
140     PORTD |= MOTOR_IZQ_DIRECTION;    //Pongo en alto el pin DIRECTION del
            motor 2
141     PORTB &= ~MOTOR_IZQ_BRAKE;       //Pongo en bajo el pin BRAKE del motor
             2
142
143     while(flag_fin_rotacion == FALSE){
144
145         if(cuenta_encoder_derecha >= (cuenta_inicial_encoder_derecho +
                370)){
146
147             if(cuenta_encoder_izquierda >= (
                    cuenta_inicial_encoder_izquierdo + 370)){
148
149                 //Antes tenia 300
150
151                 PORTB |= MOTOR_DER_BRAKE; //Pongo en alto el pin BRAKE del
                        motor 1
152                 PORTB |= MOTOR_IZQ_BRAKE; //Pongo en alto el pin BRAKE del
                        motor 2
153
154                 flag_fin_rotacion = TRUE;
155
156             }
157         }
158     }
159 }
160
161
162 void motores_corregir_rumbo (void){
163
164     uint32_t distancia;
165     uint8_t  flag_fin_rotacion = FALSE;
166
167     cuenta_encoder_derecha = 0;
168     cuenta_encoder_izquierda = 0;
169
170     _delay_ms(10);
171
172     distancia = medicion_distancia_pared(SENSOR_PARED_DER);
173
174     _delay_ms(10);
175
```

```
176     if(distancia < 0x00000650){
177
178         variar_PWM(200, 200);
179
180         PORTD |= MOTOR_DER_DIRECTION;    //Pongo en alto el pin DIRECTION
                del motor 1
181         PORTB &= ~MOTOR_DER_BRAKE;       //Pongo en bajo el pin BRAKE del
                motor 1
182
183         PORTD |= MOTOR_IZQ_DIRECTION;    //Pongo en alto el pin DIRECTION
                del motor 2
184         PORTB &= ~MOTOR_IZQ_BRAKE;       //Pongo en bajo el pin BRAKE del
                motor 2
185
186         while(flag_fin_rotacion == FALSE){
187
188             if(cuenta_encoder_derecha >= 15){
189
190                 if(cuenta_encoder_izquierda >= 15){
191
192                     flag_fin_rotacion = TRUE;
193
194                 }
195             }
196         }
197     }
198
199     if(distancia > 0x00000750){
200
201         variar_PWM(200, 200);
202
203         PORTD &= ~MOTOR_DER_DIRECTION;   //Pongo en bajo el pin DIRECTION
                del motor 1
204         PORTB &= ~MOTOR_DER_BRAKE;       //Pongo en bajo el pin BRAKE del
                motor 1
205
206         PORTD &= ~MOTOR_IZQ_DIRECTION;   //Pongo en bajo el pin DIRECTION
                del motor 2
207         PORTB &= ~MOTOR_IZQ_BRAKE;       //Pongo en bajo el pin BRAKE del
                motor 2
208
209         while(flag_fin_rotacion == FALSE){
210
211             if(cuenta_encoder_derecha >= 15){
212
213                 if(cuenta_encoder_izquierda >= 15){
214
215                     flag_fin_rotacion = TRUE;
216
217                 }
218             }
219         }
220     }
221 }
```

motores.h

```
1  #ifndef MOTORES_H_
2  #define MOTORES_H_
3  #include <util/delay.h>
4
5  #include <avr/io.h>
6  #include <avr/interrupt.h>
7
8  #include "main.h"
9  #include "pwm.h"
10 #include "sensores_pared.h"
11
12 #define MOTOR_DER_BRAKE    (1<<PB3)
13 #define MOTOR_IZQ_BRAKE    (1<<PB4)
14 #define MOTOR_DER_DIRECTION (1<<PD7)
15 #define MOTOR_IZQ_DIRECTION (1<<PD6)
16
17 #define TRUE     1
18 #define FALSE    0
19
20 #endif /* MOTORES_H_ */
21
22
23 //Definicion de prototipos
24 void inicializar_puertos_motores  (void);
25 void motores_avanzar       (uint8_t, uint8_t, uint32_t);
26 void motores_retroceder      (uint8_t, uint8_t);
27 void motores_detener       (void);
28 void motores_rotar_der_90_grados       (void);
29 void motores_rotar_izq_90_grados  (void);
30 void motores_corregir_rumbo            (void);
31 extern void inicializar_PWM            (void);
32 extern void variar_PWM                 (uint8_t, uint8_t);
```

encoders.c

```
1
2  #include "encoders.h"
3
4  void inicializar_encoders(void){
5
6      //Configuro el encoder usando el pin como entrada
7      DDRD    &= ~(1<<PD3);    //Configuro como entrada
8      PORTD   &= ~(1<<PD3);    //Desactivo el pull up
9
10     //Configuro el encoder usando el pin como entrada
11     DDRD    &= ~(1<<PD2);    //Configuro como entrada
```

```
12      PORTD   &= ~(1<<PD2);    //Desactivo el pull up
13
14      GICR    |= (1<<INT0);    //Activo INT0
15      GICR    |= (1<<INT1);    //Activo INT1
16     // GICR &= ~(1<<INT2);    //Desactivo INT2
17
18      MCUCR   |= ((1<<ISC01) | (1<<ISC00));      //Configuro la INT0 por
            flanco ascendente
19      MCUCR   |= ((1<<ISC11) | (1<<ISC10));      //Configuro la INT1 por
            flanco ascendente
20
21  }
```

encoders.h

```
1  #ifndef _ENCODER_H_
2  #define _ENCODER_H_
3
4
5  #include <avr/io.h>
6  #include <util/delay.h>
7  #include <avr/interrupt.h>
8
9
10 #endif /* _ENCODER_H_ */
11
12 //Definicion de prototipos
13 void inicializar_encoders(void);
```

pwm.c

```
1
2  #include "pwm.h"
3
4  void inicializar_PWM(void){
5
6    TCCR1A = ((1 << COM1A1)|(1 << COM1A0)|(1 << COM1B1)|(1 << COM1B0)|(0 <<
         WGM11)|(1 << WGM10));
7    //COM1A1=1, COM1A0=1, COM1B1=1, COM1B0=1, WGM11=0, WGM10=1
8    //Configuracion del PWM como fast PWM de 8 bits
9
10   TCCR1B = ((0 << WGM13)|(1 << WGM12));
11   //Timer Apagado WGM13=0, WGM12=1, CS12=0, CS11=0, CS10=0
12
13   // Configuracion de direccion de puertos I/O
14   // DDRA=0x00; // Puerto A como entrada
15   //DDRC=0xFF; // Puerto C como salida
16   //DDRD=0xFF;// Pin 3 y 4 del puerto B como salida
17   DDRD |= (1<<4)|(1<<5);
18   //PORTA=0xFF; // Habilitacion de pull-ups del puerto A
```

```
19    //PORTC=0x00; // Inicializacion de salida del puerto C
20    //OCR1AH = 0;//velocidad_motor_izq;
21    //OCR1BH = 0;//velocidad_motor_der;
22
23      TIMSK |= (1<<TOIE1);
24    // Configuracion del Timer 1:
25    TCCR1B |= ((0<<CS12)|(0<<CS11)|(1<<CS10)); //Enciendo el Timer sin
          prescaler (001)
26  }
27
28  void variar_PWM(uint8_t valor_izq, uint8_t valor_der){
29
30    OCR1AL = valor_izq;
31    OCR1BL = valor_der;
32    //TCNT1 = 0;
33  }
```

pwm.h

```
1
2  #ifndef _PWM_H_
3  #define _PWM_H_
4
5  #include <avr/io.h>
6  #include <util/delay.h>
7
8  #endif
9
10
11 //Definicion de prototipos
12 void inicializar_PWM     (void);
13 void variar_PWM          (uint8_t valor_izq, uint8_t valor_der);
```

sensoresPared.c

```
1
2  #include "sensores_pared.h"
3
4  extern volatile uint8_t      status_flag, sensor_active;
5  extern volatile uint32_t     interrupciones_timer_1;
6
7  void inicializar_puertos_sensores_pared(void){
8
9    //Configuro los puertos de TRIGGER como salida.
10   DDRA |= SENSOR_PARED_DER;
11   DDRA |= SENSOR_PARED_CEN;
12   DDRA |= SENSOR_PARED_IZQ;
13
14   //Configuro el puerto de ECHO como entrada.
15   DDRB &= ~SENSOR_PARED_ECHO;
```

```
16
17    //Configuro los puertos encendido como salida.
18    DDRA |= (1<<PA0);
19
20 }
21
22
23 uint32_t medicion_distancia_pared(uint8_t sensor){
24
25     uint32_t     cuentas = 0, cuentas_inicial, cuentas_final;
26
27     //Desactivo INT 2 al medir, para evitar problemas.
28     GICR &= ~(1<<INT2);
29
30     while(cuentas == 0){
31
32         //pongo en 1 el trigger
33         PORTA |= sensor;
34
35         //Espero 10 us
36         _delay_us(10);
37
38         //pongo en 0 el trigger
39         PORTA &= ~sensor;
40
41         //Espero a que el echo sea 1
42         while((PINB & SENSOR_PARED_ECHO) != SENSOR_PARED_ECHO);
43
44         interrupciones_timer_1 = 0;
45         cuentas_inicial = TCNT1;
46         status_flag = 0;
47         sensor_active = 1;
48
49         //Espero a que el echo sea 0
50         while( ((PINB & SENSOR_PARED_ECHO) == SENSOR_PARED_ECHO) && (
               status_flag == 0) );
51         sensor_active = 0;
52
53         cuentas_final = TCNT1;
54
55         cuentas = (256 - cuentas_inicial) + 256 * interrupciones_timer_1 +
               cuentas_final;
56
57     }
58
59     //Vuelvo a activar la interrupcion INT2
60     GICR |= (1<<INT2);
61
62
63     if (status_flag == 1)
64         return 0xFFFFFFFF;
65
66
67     else
```

```
68          return cuentas;
69  }
```

sensoresPared.h

```
1
2   #ifndef _SENSORES_PARED_H_
3   #define _SENSORES_PARED_H_
4
5   #include <avr/io.h>
6   #include <util/delay.h>
7
8   #define SENSOR_PARED_IZQ  (1 << PA3)
9   #define SENSOR_PARED_DER  (1 << PA4)
10  #define SENSOR_PARED_CEN  (1 << PA5)
11  #define SENSOR_PARED_ECHO (1 << PB2)
12
13  #define CANTIDAD      10
14
15  #endif
16
17  //Definicion de prototipos
18  inline void apagar_timer(void);
19  inline void encender_timer(void);
20  void        inicializar_timer(void);
21  void        inicializar_puertos_sensores_pared(void);
22  uint32_t    medicion_distancia_pared(uint8_t);
```

sensoresPiso.c

```
1
2   #include "sensores_piso.h"
3
4
5   /*los tres sensores del piso estan conectados a la misma interrupcion
        externa INT2
6   y a su vez a PA0=izq PA1=der y PA2=centro
7   La idea con este codigo es primero inicializar las interrupciones por
        flanco ascendente (usando algun tipo de debouncer  para evitar efecto
        rebote)
8   y poder identificar que interrupcion fue activada para luego poder usarla
        en el codigo del robot.
9   */
10
11  inline void encender_sensores_piso(void){
12
13      PORTC |= (1<<1);
14
15  }
16
```

```
17
18 inline void apagar_sensores_piso(void){
19
20     PORTC &= ~(1<<1);
21
22 }
23
24
25 void inicializar_sensores_piso(void)
26 {
27
28     //Configuro el pin de los CNY70 como salida.
29     DDRC     |= (1<<1);
30
31     //configuro el pin de interrupcion como entrada (no se si es necesario
       )
32     DDRB     &= ~(1<<2);
33     PORTB    &= ~(1<<2);
34
35     // activamos la interrupcion INT2 por flanco ascendente ISC2=1
36     MCUCSR |= (1<<ISC2);
37
38     //configuramos General Interrupt Control Register
39     //activamos la interrupcion INT2=1
40     GICR |= (1<<INT2);
41
42 }
```

sensoresPiso.h

```
1 #ifndef _SENSORES_PISO_H
2 #define _SENSORES_PISO_H
3
4
5 #include <avr/io.h>
6 #include <avr/interrupt.h>
7
8 #endif
9
10
11 //prototipos de funcion.
12 inline void encender_sensores_piso(void);
13 inline void apagar_sensores_piso(void);
14 void         inicializar_sensores_piso(void);
```