CAPÍTULO 3

3 Diseño Del Proyecto

En este capítulo se ponen de manifiesto las etapas de diseño, implementación para la elaboración de este proyecto, sus diagramas de bloques, algoritmos y los códigos que serán cargados en los micro controladores a usarse.

Para comenzar a trabajar con el Robot Pololu y el AVR Butterfly se realizaron algunas pruebas antes de la programación de nuestro proyecto.

3.1 Prueba Inicial

Para la realización de este proyecto se seleccionó el Robot Pololu 3pi que trae incorporado un microcontrolador ATMEGA 328 que es el que se programa para que ejecute el programa a realizarse.

Como primer pasó antes de comenzar a trabajar con el pololu se realizo las pruebas demo que vienen incorporadas a este robot, como es el seguir una línea usando las librerías respectivas, de igual manera se realizó con el AVR butterfly, y esto nos permitió familiarizarnos con el programa AVR.

Al cargar el programa demo en el Pololu este hace que el robot muestre en la pantalla LCD que este posee, el nombre de tal robot como es Pololu 3 π Robot. Además se puede escuchar una música que es atraves del buzzer del robot.

Se pudo realizar con el butterfly las pruebas como es prender la pantalla LCD y ver los comandos que tiene el Joystick.

3.2 Descripción del proyecto final

3.2.1 Diagrama de bloques

El diagrama de bloques del sistema se encuentra establecido por la etapa del transmisor y la etapa del receptor. Ver fig 3.4.1

La etapa de transmisión está constituida por el Kit AVR Butterfly que envía los datos correspondientes mediante el Joystick que es por el cual se seleccionan los comandos .La señal ingresa al módulo de transmisión de RF, que se encarga de modular la señal para ser transmitida.

En la etapa de recepción se encuentra el módulo receptor que demodula la señal para ser receptadas por medio del Robot Pololu que luego comenzara a moverse dependiendo de los comandos que sean presionados por el Joystick.

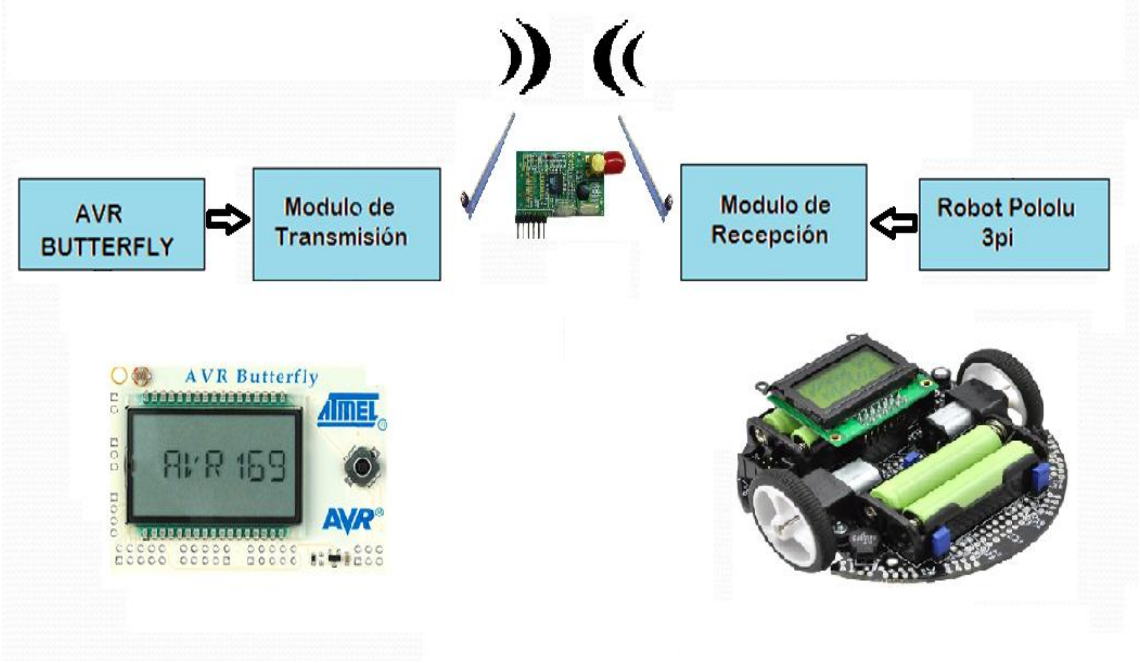


Fig 3.2.1 Diagrama de Bloques del proyecto

3.2.2 Diagrama de Flujo del Transmisor

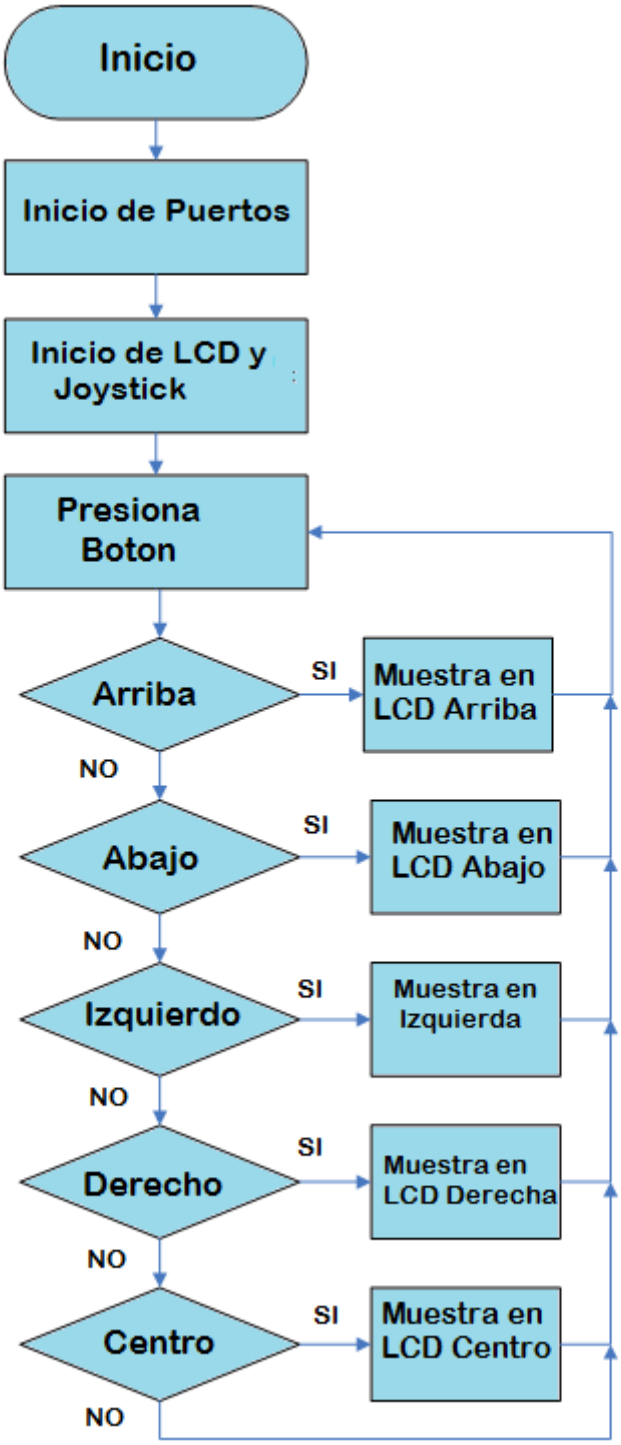


Figura 3.2.2 Diagrama de Transmisor

Con este algoritmo se muestra lo que realiza el transmisor primero se inicializa, luego se setean los puertos después de esto se da inicio de la LCD y del joystick, el programa espera a que presionen botón, si este se presiona, entonces se envía trama esta trama contiene un carácter que da una instrucción para que el robot pololu lo recepte.

Estos caracteres son los siguientes

u arriba entonces enciende motores del pololu hacia adelante

d abajo entonces enciende motores del pololu hacia atrás

R derecha , enciende los motores del pololu a la derecha

L izquierda, enciende los motores hacia la izquierda

S para, pone los motores en stop

3.2.3 Código AVR Butterfly para el Transmisor

```
//Declaración de constantes
#define  Centro    0
#define  Arriba    1
#define  Abajo     2
#define  Izquierda 3
#define  Derecha   4
#define  Otros     5

//Librerias a usar
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
```

```
#include <avr/delay.h>
#include <inttypes.h>

//Librerias a usar que no pertenecen al entorno
#include "mydefs.h"
#include "LCD_functions.h"
#include "LCD_driver.h"
#include "button.h"
#include "usart.h"


//Prototipo de funciones
int Obtener_Boton(void);
//Programa principal
int main(void)
{
                    //Declaración de variable
          int input;


//Se habilita las interrupciones globales
sei();
//Se muestra un mensaje a través del LCD
   PGM_P statetext = PSTR("AVR BUTTERFLY");
// Disable Analog Comparator (power save)
 ACSR = (1<<ACD);
 // Disable Digital input on PF0-2 (power save)
   DIDR0 = (7<<ADC0D);
```

```
 // Enable pullups
PORTB = (15<<PB0);
PORTE = (15<<PE4);

// Initialize pin change interrupt on joystick
Button_Init();
// initialize the LCD
LCD_Init();
// set Clock Prescaler Change Enable
CLKPR = (1<<CLKPCE);
 // set prescaler = 8, Inter RC 8Mhz / 8 = 1Mhz
CLKPR = (0<<CLKPS1) | (1<<CLKPS0);

//Configuración del USART a 207= 9600 baudios
USART_Init(207);
if (statetext)
{
  LCD_puts_f(statetext, 1);
   LCD_Colon(0);
 statetext = NULL;
}
//Lazo infinito
while (1)
 {
if (statetext)
```

```
        {
        LCD_puts_f(statetext, 1);
        LCD_Colon(0);
        statetext = NULL;
        }
        //Se espera a que sea presionado un botón
         input = Obtener_Boton();
         //Usart_Tx('1');
//Se realiza una determinada acción según el botón presionado
        switch (input)
        {
        case Centro:
         //Usart_Tx('0');
                statetext = PSTR("CENTRO");
            Usart_Tx('U');
          break;

        case Derecha:
  //Usart_Tx('0');
                statetext = PSTR("DERECHA");
     Usart_Tx('R');
   break;

        case Izquierda:
 //Usart_Tx('0');
```

```
                    //Usart_Tx('0');
                    statetext = PSTR("IZQUIERDA");
                Usart_Tx('I');
                    break;

            case Arriba:
 //Usart_Tx('0');
                    statetext = PSTR("ARRIBA");
    Usart_Tx('C');
    break;

  case Abajo:
          //Usart_Tx('0');
                    statetext = PSTR("ABAJO");
  Usart_Tx('a');
                    break;

 default:
  break;
          }
   }
   return 0;
}

/*Función que retorna un valor entero correspondiente al botón
```

```
presionado en el JoyStick */
int Obtener_Boton(void)
{
int Temp1;
        //PB4-->O  Centro
        //PB6-->A  Arriba
        //PB7-->B  Abajo
        //PE2-->C  Izquierda
        //PE3-->D  Derecha
        //Centro
Temp1=(PINB) & 0b00010000;
if(Temp1==0b00000000)
{
        sei();
        return Centro;
}
        //Arriba
Temp1=PINB & 0b01000000;
if(Temp1==0b00000000)
{
        sei();
        return Arriba;
}
        //Abajo
Temp1=PINB & 0b10000000;
```

```
        if(Temp1==0b00000000)
        {
                sei();
                return Abajo;
        }
                //Izquierda
        Temp1=PINE & 0b00000100;
        if(Temp1==0b00000000)
        {
                sei();
                return Izquierda;
        }
                //Derecha
        Temp1=PINE & 0b00001000;
        if(Temp1==0b00000000)
        {
                sei();
                return Derecha;
        }
        sei();
        return Otros;
}
        //Codigo del  usart
```

```
        #include <avr/io.h>
        #include "usart.h"
/* 25. May 2005 - adapted to avrlibc iom168.h version 1.17 */
void USART_Init(unsigned int baudrate) // UDRR Value not baudrate
        {
   UBRR0H = (unsigned char)(baudrate>>8);
        UBRR0L = (unsigned char)baudrate;
   UCSR0A = (1<<U2X0);
UCSR0B=(1<<RXEN0)|(1<<TXEN0)|(0<<RXCIE0)|(0<<UDRIE0;
UCSR0C=(0<<UMSEL0)|(0<<UPM00)|(0<<USBS0)|(3<<UCSZ00)|(0<<UCPOL0);
        }
void Usart_Tx(unsigned char data)
{
   while (!(UCSR0A & (1<<UDRE0)));
   UDR0 = data;
}
char Usart_Rx(void)
{
  while (!(UCSR0A & (1<<RXC0)));
  return UDR0;
}
        //Driver LCD
        #define REDUCED
        // Include files.
```

```
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <inttypes.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#ifndef REDUCED
// mt - only for AUTO:
#include "main.h"
#else
#include "mydefs.h"
#endif
// mt - for gButtonTimeout
#include "button.h"
#include "LCD_driver.h"
#ifndef BOOL
#define BOOL    char
#define FALSE   0
#define TRUE    (!FALSE)
#endif

// Variable from "button.c" to prevent button-bouncing
extern unsigned char gButtonTimeout;
volatile char gAutoPressJoystick = FALSE;
// Used to indicate when the LCD interrupt handler should update the LCD
// mt jw char gLCD_Update_Required = FALSE;
volatile char gLCD_Update_Required = FALSE;
```

```
// LCD display buffer (for double buffering).
volatile char LCD_Data[LCD_REGISTER_COUNT];

// Buffer that contains the text to be displayed
volatile char gTextBuffer[TEXTBUFFER_SIZE];

// Only six letters can be shown on the LCD.
volatile signed char gScroll;
volatile char gScrollMode;
////Start-up delay before scrolling a string over the LCD
char gLCD_Start_Scroll_Timer = 0;

// The gFlashTimer is used to determine the on/off
volatile char gFlashTimer = 0;
// Turns on/off the colons on the LCD
char gColon = 0;
// Look-up table used when converting ASCII to
// LCD display data (segment control)
unsigned int LCD_character_table[] PROGMEM =

/*************************************************************/
*   Function name : LCD_Init
*   Returns :       None
*   Parameters :    None
*   Purpose :       Initialize LCD_displayData buffer.
```

```
*                Set up the LCD (timing, contrast, etc.)
************************************************************/
void LCD_Init (void)
{
   LCD_AllSegments(FALSE);                  // Clear segment buffer.
   LCD_CONTRAST_LEVEL(LCD_INITIAL_CONTRAST);  //Set the LCD contrast
   // Select asynchronous clock source, enable all COM pins and enable all
   LCDCRB = (1<<LCDCS) | (3<<LCDMUX0) | (7<<LCDPM0);
   // Set LCD prescaler to give a framerate of 32,0 Hz
   LCDFRR = (0<<LCDPS0) | (7<<LCDCD0);
   LCDCRA = (1<<LCDEN) | (1<<LCDAB);          // Enable LCD and set low
power waveform
   //Enable LCD start of frame interrupt
   LCDCRA |= (1<<LCDIE);
   gLCD_Update_Required = FALSE;
}
/*****************************************************
*   Function name : LCD_WriteDigit(char c, char digit)
*   Returns :       None
*   Parameters :    Inputs
*   Purpose :       Stores LCD control data in the LCD_displayData buffer.
*                (The LCD_displayData is latched in the LCD_SOF interrupt.)
*****************************************************/
void LCD_WriteDigit(char c, char digit)
{
   unsigned int seg = 0x0000;               // Holds the segment pattern
```

```
char mask, nibble;
volatile char *ptr;
char i;
if (digit > 5)                          // Skip if digit is illegal
   return;
//Lookup character table for segmet data
if ((c >= '*') && (c <= 'z'))  // c is a letter
{
   if (c >= 'a')                        // Convert to upper case
      c &= ~0x20;                       // if necessarry
   c -= '*';       //mt seg = LCD_character_table[c];
seg = (unsigned int) pgm_read_word(&LCD_character_table[(uint8_t)c]);
      }
// Adjust mask according to LCD segment mapping
if (digit & 0x01)
   mask = 0x0F;            // Digit 1, 3, 5
else
   mask = 0xF0;            // Digit 0, 2, 4
ptr = LCD_Data + (digit >> 1);  // digit = {0,0,1,1,2,2}

for (i = 0; i < 4; i++)
{
   nibble = seg & 0x000F;
   seg >>= 4;
   if (digit & 0x01)
      nibble <<= 4;
```

```
        *ptr = (*ptr & mask) | nibble;
        ptr += 5;
    }
}
        /*************************************************
        *   Function name : LCD_AllSegments(unsigned char input)
        *   Returns :       None
        *   Parameters :    show -  [TRUE;FALSE]
        *   Purpose :       shows or hide all all LCD segments on the LCD
        *************************************************/
        void LCD_AllSegments(char show)
        {
           unsigned char i;
    if (show)
        show = 0xFF;
            // Set/clear all bits in all LCD registers
    for (i=0; i < LCD_REGISTER_COUNT; i++)
        *(LCD_Data + i) = show;
}
        /*************************************************
        *   LCD Interrupt Routine
        *   Returns :       None
        *   Parameters :    None
        *   Purpose: Latch the LCD_displayData and Set
        LCD_status.updateComplete
        *************************************************/
```

```
SIGNAL(SIG_LCD)
{
    static char LCD_timer = LCD_TIMER_SEED;
    char c;
  char c_flash;
   char flash;
   char EOL;
   unsigned char i;
    static char timeout_count;
  static char auto_joystick_count;
c_flash=0; // mt
/******** Button timeout for the button.c, START **********/
    if(!gButtonTimeout)
    {
timeout_count++;
  if(timeout_count > 3)
     {
         gButtonTimeout = TRUE;
timeout_count = 0;
   }
 }
/********** Button timeout for the button.c, END **********/
    if(gAutoPressJoystick == AUTO)
    {
       auto_joystick_count++;
```

```
    if(auto_joystick_count > 16)
    {
gAutoPressJoystick = TRUE;
auto_joystick_count = 15;
  }
}
    else
        auto_joystick_count = 0;
    LCD_timer--;
// Decreased every LCD frame
  if (gScrollMode)
    {
   // If we are in scroll mode, and the timer has expired,
   // we will update the LCD
   if (LCD_timer == 0)
   {
      if (gLCD_Start_Scroll_Timer == 0)
      {
        gLCD_Update_Required = TRUE;
}
else
    gLCD_Start_Scroll_Timer--;
     }
    }
    else
```

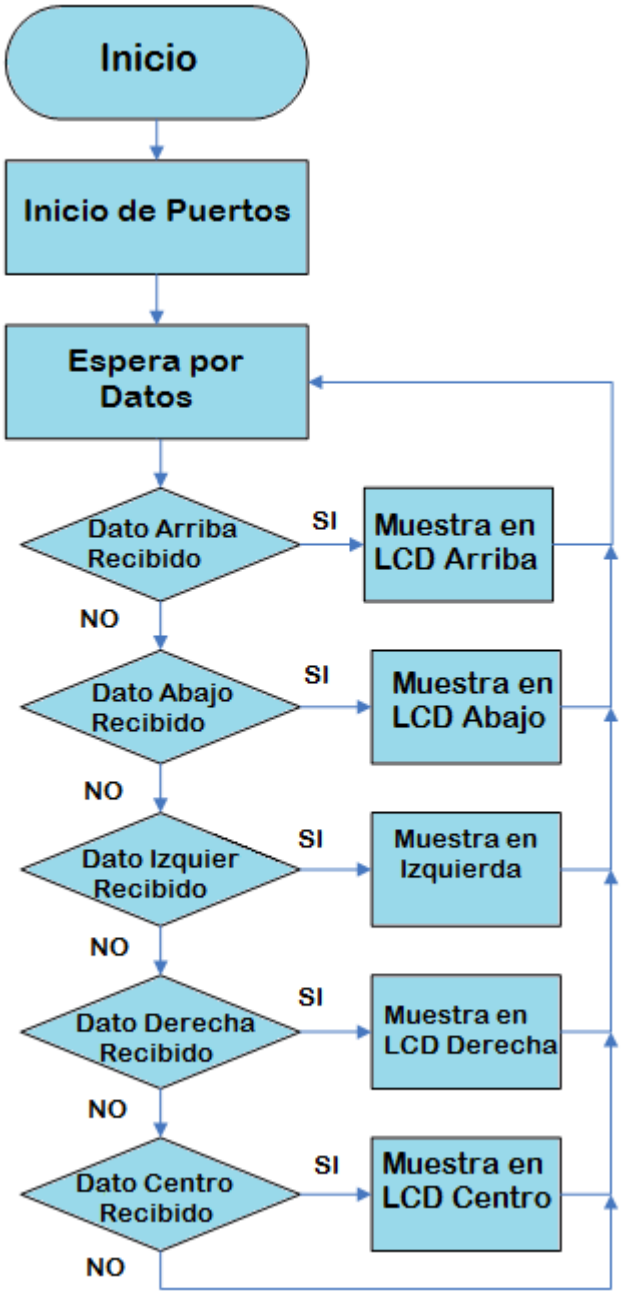
```
        {   // if not scrolling,
            // disble LCD start of frame interrupt
    //        cbi(LCDCRA, LCDIE);   //DEBUG
        gScroll = 0;
        }
        EOL = FALSE;
        if (gLCD_Update_Required == TRUE)
        {
     // Duty cycle of flashing characters
            if (gFlashTimer < (LCD_FLASH_SEED >> 1))
    flash = 0;
else
    flash = 1;
     // Repeat for the six LCD characters
for (i = 0; i < 6; i++)
{
    if ((gScroll+i) >= 0 && (!EOL))
    {

      // We have some visible characters
        c = gTextBuffer[i + gScroll];
        c_flash = c & 0x80 ? 1 : 0;
        c = c & 0x7F;
        if (c == '\0')
            EOL = i+1;      // End of character data
```

```
    }
    else
        c = ' ';
    // Check if this character is flashing
    if (c_flash && flash)
        LCD_WriteDigit(' ', i);
    else
        LCD_WriteDigit(c, i);
}
// Copy the segment buffer to the real segments
for (i = 0; i < LCD_REGISTER_COUNT; i++)
    *(pLCDREG + i) = *(LCD_Data+i);
// Handle colon
if (gColon)
    *(pLCDREG + 8) = 0x01;
else
    *(pLCDREG + 8) = 0x00;
// If the text scrolled off the display,
// we have to start over again.
if (EOL == 1)
    gScroll = -6;
else
    gScroll++;
// No need to update anymore
gLCD_Update_Required = FALSE;
```

```
    }
    // LCD_timer is used when scrolling text
    if (LCD_timer == 0)
    {
/*      if ((gScroll <= 0) || EOL)
          LCD_timer = LCD_TIMER_SEED/2;
       else*/
          LCD_timer = LCD_TIMER_SEED;
    }
    // gFlashTimer is used when flashing characters
    if (gFlashTimer == LCD_FLASH_SEED)
       gFlashTimer= 0;
    else
       gFlashTimer++;
}
```

3.2.4 Diagrama de Flujo del Receptor



3.7 Diagrama de Flujo Receptor

3.2.5 Código Receptor

```
//Declaración de constantes
#define F_CPU 20000000UL
#define BAUD 4800
#define MYUBRR F_CPU/16/BAUD-1
#define PIND_MASK ((1<<PIND0)|(1<<PIND1))

//Librerias a usar
#include <avr/pgmspace.h>
#include <pololu/orangutan.h>

//Prototipo de procedimientos
void USART_Init( unsigned int );
unsigned char ReceiveByte (void);

//Programa principal
int main ()
{
//Declaración de variable
int i, j=0;

//Seteo de puertos
DDRD = 0xFE;
PORTD |= PIND_MASK;
DDRB = 0x08; // set PORTD for output
```

```
PORTB = 0x00; // set LEDs off
//Configuración de USART a 2400 baudios
USART_Init(521);
//Lazo infinito en espera de caracter recibido
        while(1)
        {
        //Se muestra en pantalla un mensaje que indica que
        //se está a la espera de recibir un caracter
        clear();
        print("ESPERO");
        lcd_goto_xy(0, 1);
        print("TU ORDEN");
        delay_ms(500);

        //Se toma la decisión en función del caracter
        //recibido. El arranque del motor se lo realiza
        //en dos tiempos para no realizar una demanda súbita
        //de potencia

        //ADELANTE
        i = ReceiveByte();
        if(i!=j)
        {
                if(i == 0X5E)
                {
```

```
            clear();
            print("FORWARD");
            set_motors(50,50);
            delay_ms(1000);
            delay_ms(1000);
            set_motors(100,100);
}

            //ATRÁS
else if (i == 0X4F )// 'a'
{
            clear();
            print("BACK");
            set_motors(-50,-50);
            delay_ms(1000);
            set_motors(-100,-100);
            delay_ms(1000);
}
//IZQUIERDA
else if (i == 0X5B)//'I'
{
            clear();
            print("LEFT");
            set_motors(50,-50);
            delay_ms(500);
```

```
            set_motors(100,-100);
            delay_ms(500);
}
//DERECHA
else if (i == 0XAB)
{
            clear();
            print("RIGHT");
            set_motors(-50, 50);
            delay_ms(500);
            set_motors(-100, 100);
            delay_ms(500);
}

//ALTO
else if (i == 'U')
{
            clear();
            print("Stop");
            set_motors(0,0);
            delay_ms(250);
}
j=i;
}
```

```
                }
        }

//Implementación de procedimiento/*
Se configuran los registros para la transmisión por USART
de manera que el dato recibido indica el BAUDRATE al cual
se va a trabajar
*/
void USART_Init(unsigned int baudrate)
{
  // Set baud rate
 UBRR0H = (unsigned char)(baudrate>>8);
 UBRR0L = (unsigned char)baudrate;
        //UCSR0A = (0<<U2X0);
 // Enable receiver and transmitter
  UCSR0B = (1<<RXEN0)|(1<<TXEN0);
  // Async. mode, 8N1
  UCSR0C = (1<<USBS0)|(3<<UCSZ00);
}

//Implementación de función
/*
Función que espera a la recepción de un caracter y luego
retorna el dato recibido al programa principal
*/
```

```
unsigned char ReceiveByte (void)
{
 /* Wait for incomming data */
 while (!(UCSR0A & (1 << RXC0)));

  /* Return the data */
 return UDR0;
}
```

```
//Codigo del usart
void USART_Init(unsigned int baudrate);

void Usart_Tx(char);
char Usart_Rx(void);
```