```
/*
 * Project_Test.c
 *
 * Created: 14-04-2015 02:22:16
 *  Author: Team 389
 *    Vikram Bhosale - 14D070026 - 14d070026@iitb.ac.in
 *    Ashwin Wagh - 14D070025 - 14d070025@iitb.ac.in
 *    Aditya Jain - 140070005 - 140070005@iitb.ac.in
 *    Abhimanyu - 140070035 - 140070035@iitb.ac.in
 */

#define F_CPU 14745600
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
unsigned char ADC_Value;
unsigned char Left = 0;
unsigned char Center = 0;
unsigned char Right = 0;
unsigned char speed = 70;
int Integral=0;
int Differential=0;
int error=0;
int lastError=0;
double MotorSpeed_Double=0;
int MotorSpeed;
int SavedTurnSpeed=0;
unsigned char flag=0;
volatile unsigned long int pulse = 0; //to keep the track of the
number of pulses generated by the color sensor
volatile unsigned long int red;       // variable to store the pulse
count when read_red function is called
volatile unsigned long int blue;      // variable to store the pulse
count when read_blue function is called
volatile unsigned long int green;     // variable to store the pulse
count when read_green function is called
volatile unsigned long int ShaftCountLeft = 0; //to keep track of left
position encoder
volatile unsigned long int ShaftCountRight = 0; //to keep track of
right position encoder
volatile unsigned int Degrees; //to accept angle in degrees for
turning
void color_sensor_pin_config(void)
{
      DDRD  = DDRD | 0xFE; //set PD0 as input for color sensor output
      PORTD = PORTD | 0x01;//Enable internal pull-up for PORTD 0 pin
}
//ADC pin configuration
void adc_pin_config (void)
{
      DDRF = 0x00;
      PORTF = 0x00;
```

```
	DDRK = 0x00;
	PORTK = 0x00;
}
//Function to configure ports to enable robot's motion
void motion_pin_config (void)
{
	DDRA = DDRA | 0x0F;
	PORTA = PORTA & 0xF0;
	DDRL = DDRL | 0x18;   //Setting PL3 and PL4 pins as output for
PWM generation
	PORTL = PORTL | 0x18; //PL3 and PL4 pins are for velocity control
using PWM.
}
//Function to configure INT4 (PORTE 4) pin as input for the left
position encoder
void left_encoder_pin_config (void)
{
	DDRE  = DDRE & 0xEF;  //Set the direction of the PORTE 4 pin as
input
	PORTE = PORTE | 0x10; //Enable internal pull-up for PORTE 4 pin
}
//Function to configure INT5 (PORTE 5) pin as input for the right
position encoder
void right_encoder_pin_config (void)
{
	DDRE  = DDRE & 0xDF;  //Set the direction of the PORTE 4 pin as
input
	PORTE = PORTE | 0x20; //Enable internal pull-up for PORTE 4 pin
}
//Function to Initialize PORTS
void port_init()
{
	adc_pin_config();
	motion_pin_config();
	left_encoder_pin_config(); //left encoder pin config
	right_encoder_pin_config(); //right encoder pin config
	color_sensor_pin_config();//color sensor pin configuration
}
void left_position_encoder_interrupt_init (void) //Interrupt 4 enable
{
	cli(); //Clears the global interrupt
	EICRB = EICRB | 0x02; // INT4 is set to trigger with falling edge
	EIMSK = EIMSK | 0x10; // Enable Interrupt INT4 for left position
encoder
	sei();   // Enables the global interrupt
}
void right_position_encoder_interrupt_init (void) //Interrupt 5 enable
{
	cli(); //Clears the global interrupt
	EICRB = EICRB | 0x08; // INT5 is set to trigger with falling edge
	EIMSK = EIMSK | 0x20; // Enable Interrupt INT5 for right position
encoder
```

```
	sei();   // Enables the global interrupt
}
//ISR for right position encoder
ISR(INT5_vect)
{
	ShaftCountRight++;  //increment right shaft position count
}
//ISR for left position encoder
ISR(INT4_vect)
{
	ShaftCountLeft++;  //increment left shaft position count
}
void color_sensor_pin_interrupt_init(void) //Interrupt 0 enable
{
	cli(); //Clears the global interrupt
	EICRA = EICRA | 0x02; // INT0 is set to trigger with falling edge
	EIMSK = EIMSK | 0x01; // Enable Interrupt INT0 for color sensor
	sei(); // Enables the global interrupt
}
//ISR for color sensor
ISR(INT0_vect)
{
	pulse++; //increment on receiving pulse from the color sensor
}

// Timer 5 initialized in PWM mode for velocity control
// Prescale:256
// PWM 8bit fast, TOP=0x00FF
// Timer Frequency:225.000Hz
void timer5_init()
{
	TCCR5B = 0x00;   //Stop
	TCNT5H = 0xFF;   //Counter higher 8-bit value to which OCR5xH
value is compared with
	TCNT5L = 0x01;   //Counter lower 8-bit value to which OCR5xH value
is compared with
	OCR5AH = 0x00;   //Output compare register high value for Left
Motor
	OCR5AL = 0xFF;   //Output compare register low value for Left
Motor
	OCR5BH = 0x00;   //Output compare register high value for Right
Motor
	OCR5BL = 0xFF;   //Output compare register low value for Right
Motor
	OCR5CH = 0x00;   //Output compare register high value for Motor C1
	OCR5CL = 0xFF;   //Output compare register low value for Motor C1
	TCCR5A = 0xA9;   /*{COM5A1=1, COM5A0=0; COM5B1=1, COM5B0=0;
COM5C1=1 COM5C0=0}
                                For Overriding normal port functionality
to OCRnA outputs.
                                {WGM51=0, WGM50=1} Along With WGM52 in
TCCR5B for Selecting FAST PWM 8-bit Mode*/
```

```
	TCCR5B = 0x0B;   //WGM12=1; CS12=0, CS11=1, CS10=1 (Prescaler=64)
}
void adc_init()
{
	ADCSRA = 0x00;
	ADCSRB = 0x00;		//MUX5 = 0
	ADMUX = 0x20;		//Vref=5V external --- ADLAR=1 --- MUX4:0 =
0000
	ACSR = 0x80;
	ADCSRA = 0x86;		//ADEN=1 --- ADIE=1 --- ADPS2:0 = 1 1 0
}
//Function For ADC Conversion
unsigned char ADC_Conversion(unsigned char Ch)
{
	unsigned char a;
	if(Ch>7)
	{
		ADCSRB = 0x08;
	}
	Ch = Ch & 0x07;
	ADMUX= 0x20| Ch;
	ADCSRA = ADCSRA | 0x40;		//Set start conversion bit
	while((ADCSRA&0x10)==0);	//Wait for conversion to complete
	a=ADCH;
	ADCSRA = ADCSRA|0x10; //clear ADIF (ADC Interrupt Flag) by
writing 1 to it
	ADCSRB = 0x00;
	return a;
}
//Function for velocity control
void velocity (unsigned char left_motor, unsigned char right_motor)
{
	OCR5AL = (unsigned char)left_motor;
	OCR5BL = (unsigned char)right_motor;
}
//Function used for setting motor's direction
void motion_set (unsigned char Direction)
{
	unsigned char PortARestore = 0;
	Direction &= 0x0F;		// removing upper nibble for the
protection
	PortARestore = PORTA;		// reading the PORTA original
status
	PortARestore &= 0xF0;		// making lower direction nibble
to 0
	PortARestore |= Direction; // adding lower nibble for forward
command and restoring the PORTA status
	PORTA = PortARestore;		// executing the command
}
void forward (void)
{
	motion_set (0x06);
```

```
}
void stop (void)
{
     motion_set (0x00);
}
void left (void) //Left wheel backward, Right wheel forward
{
     motion_set(0x05);
}
void right (void) //Left wheel forward, Right wheel backward
{
     motion_set(0x0A);
}
//Function used for turning robot by specified degrees
void angle_rotate(unsigned int Degrees)
{
     float ReqdShaftCount = 0;
     unsigned long int ReqdShaftCountInt = 0;
     ReqdShaftCount = (float) Degrees/ 4.090; // division by
resolution to get shaft count
     ReqdShaftCountInt = (unsigned int) ReqdShaftCount;
     ShaftCountRight = 0;
     ShaftCountLeft = 0;
     while (1)
     {
          if((ShaftCountRight >= ReqdShaftCountInt) | (ShaftCountLeft
>= ReqdShaftCountInt))
          break;
     }
     stop(); //Stop robot
}
//Function used for moving robot forward by specified distance
void linear_distance_mm(unsigned int DistanceInMM)
{
     float ReqdShaftCount = 0;
     unsigned long int ReqdShaftCountInt = 0;
     ReqdShaftCount = DistanceInMM / 5.338; // division by resolution
to get shaft count
     ReqdShaftCountInt = (unsigned long int) ReqdShaftCount;
     ShaftCountRight = 0;
     while(1)
     {
          if(ShaftCountRight > ReqdShaftCountInt)
          {
               break;
          }
     }
     stop(); //Stop robot
}
void forward_mm(unsigned int DistanceInMM)
{
     forward();
```

```
      linear_distance_mm(DistanceInMM);
}
void left_degrees(unsigned int Degrees)
{
      // 88 pulses for 360 degrees rotation 4.090 degrees per count
      left(); //Turn left
      angle_rotate(Degrees);
}
void right_degrees(unsigned int Degrees)
{
      // 88 pulses for 360 degrees rotation 4.090 degrees per count
      right(); //Turn right
      angle_rotate(Degrees);
}
void init_devices (void)
{
      cli(); //Clears the global interrupts
      port_init();
      adc_init();
      timer5_init();
      left_position_encoder_interrupt_init();
      right_position_encoder_interrupt_init();
      color_sensor_pin_interrupt_init();
      sei();   //Enables the global interrupts
}
//Filter Selection
void filter_red(void)    //Used to select red filter
{
      //Filter Select - red filter
      PORTD = PORTD & 0xBF; //set S2 low
      PORTD = PORTD & 0x7F; //set S3 low
}
void filter_green(void)    //Used to select green filter
{
      //Filter Select - green filter
      PORTD = PORTD | 0x40; //set S2 High
      PORTD = PORTD | 0x80; //set S3 High
}
void filter_blue(void)     //Used to select blue filter
{
      //Filter Select - blue filter
      PORTD = PORTD & 0xBF; //set S2 low
      PORTD = PORTD | 0x80; //set S3 High
}
void filter_clear(void)    //select no filter
{
      //Filter Select - no filter
      PORTD = PORTD | 0x40; //set S2 High
      PORTD = PORTD & 0x7F; //set S3 Low
}
//Color Sensing Scaling
```

```
void color_sensor_scaling()      //This function is used to select the
scaled down version of the original frequency of the output generated
by the color sensor, generally 20% scaling is preferable, though you
can change the values as per your application by referring datasheet
{
      //Output Scaling 20% from datasheet
      //PORTD = PORTD & 0xEF;
      PORTD = PORTD | 0x10; //set S0 high
      //PORTD = PORTD & 0xDF; //set S1 low
      PORTD = PORTD | 0x20; //set S1 high
}
unsigned long int red_read(void) // function to select red filter and
display the count generated by the sensor on LCD. The count will be
more if the color is red. The count will be very less if its blue or
green.
{
      //Red
      filter_red(); //select red filter
      pulse=0; //reset the count to 0
      _delay_ms(100); //capture the pulses for 100 ms or 0.1 second
      red = pulse; //store the count in variable called red
      return red;
}
unsigned long int green_read(void) // function to select green filter
and display the count generated by the sensor on LCD. The count will
be more if the color is green. The count will be very less if its blue
or red.
{
      //Green
      filter_green(); //select green filter
      pulse=0; //reset the count to 0
      _delay_ms(100); //capture the pulses for 100 ms or 0.1 second
      green = pulse;  //store the count in variable called green
      return green;
}
unsigned long int blue_read(void) // function to select blue filter
and display the count generated by the sensor on LCD. The count will
be more if the color is blue. The count will be very less if its red
or green.
{
      //Blue
      filter_blue(); //select blue filter
      pulse=0; //reset the count to 0
      _delay_ms(100); //capture the pulses for 100 ms or 0.1 second
      blue = pulse;  //store the count in variable called blue
      return blue;
}
//Main Function
int main()
{
      unsigned long int r,b,black_threshold=0;
      int mode; // *comment something*
```

```
	init_devices();
	color_sensor_scaling();
	//Setting Black Threshold. mode variable as counter
	for(mode=5;mode>1; mode--)
	{
		r=red_read();
		b=blue_read();
		if(black_threshold<r)
		{
			black_threshold=r;
		}
		if(black_threshold<b)
		{
			black_threshold=b;
		}
	}
	black_threshold+=100;
	while(1)
	{
		flag = 0;
		Left = ADC_Conversion(3);  //Getting data of Left WL Sensor
		Center = ADC_Conversion(2);//Getting data of Center WL
Sensor
		Right = ADC_Conversion(1); //Getting data of Right WL
Sensor
		//Positive is right, negative is left
		error=(int)(Right-Left)/2;
		Differential=error-lastError;
		Integral+=error;

	MotorSpeed_Double=(double)(0.5*error+10*Differential+0.001*Integr
al);
		MotorSpeed=(int)(MotorSpeed_Double);
		if(MotorSpeed>205)
		{
			MotorSpeed=205;
		}
		if(MotorSpeed<-205)
		{
			MotorSpeed=-205;
		}
		if(Center<0x10&&Left<0x10&&Right<0x10)//junction
		{
			velocity(speed,speed);
			forward_mm(90);
			//code for checking color in front
			//checking for distinction only between blue and red
for branching
			while(1)
			{
				_delay_ms(1000);
				b=blue_read();
```

```
                        r=red_read();
                        if(b<=r&&b>black_threshold&&r>black_threshold)
                        {
                              if(mode==1)
                              {
                                    velocity(70,70);
                                    right_degrees(50);
                                    right();
                                    while (ADC_Conversion(2)>0x10){}
                                    velocity(0,0);
                                    mode=2;
                              }
                              else if(mode==2)
                              {
                                    velocity(70,70);
                                    left_degrees(50);
                                    left();
                                    while (ADC_Conversion(2)>0x10){}
                                    left_degrees(50);
                                    left();
                                    while (ADC_Conversion(2)>0x10){}
                                    velocity(0,0);
                                    mode=3;
                              }
                              else if(mode==3)
                              {
                                    velocity(70,70);
                                    right_degrees(50);
                                    right();
                                    while (ADC_Conversion(2)>0x10){}
                                    velocity(0,0);
                                    mode=1;
                              }
                        }
                        else
if(b>r&&b>black_threshold&&r>black_threshold)
                        {
                              velocity(speed,speed);
                              mode=1;
                              break;
                        }
                        else if(b<black_threshold&&r<black_threshold)
                        {
                              if(speed==100) speed=70;
                              else if(speed==70) speed=100;
                              velocity(speed,speed);
                              mode=1;
                              break;
                        }
                  }
            }
            if(Center<0x0A&&flag==0)
```

```
            {
                  forward();
                  velocity(speed,speed);
                  flag=1;
            }
            if(MotorSpeed>0&&flag==0)
            {
                  SavedTurnSpeed=MotorSpeed;
                  forward();
                  velocity(0,50+MotorSpeed);
            }
            if(MotorSpeed<0&&flag==0)
            {
                  SavedTurnSpeed=MotorSpeed;
                  MotorSpeed=-1*MotorSpeed;
                  forward();
                  velocity(50+MotorSpeed,0);
            }
            if(Center>0x0A&&Left>0x0A&&Right>0x0A)
            {
                  if(SavedTurnSpeed>0&&flag==0)
                  {
                        forward();
                        velocity(0,50+SavedTurnSpeed);
                  }
                  if(SavedTurnSpeed<0&&flag==0)
                  {
                        forward();
                        velocity(50-SavedTurnSpeed,0);
                  }
            }
            lastError=error;
      }
}
```