```
#define F_CPU 12000000
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
float Kp =0.8 ;
float Ki = 0;
float Kd =0;
float offset = 70;
float Tp = 200;
float integral1 = 0;
float lastError1 = 0;
float derivative1 = 0;
float integral2 = 0;
float lastError2 = 0;
float derivative2 = 0;
float error1,error2;
float Turn1,Turn2;
unsigned char ADC_Conversion(unsigned char);
unsigned char ADC_Value;

unsigned char a,b,c,d,e,f,value;

void init_devices (void) //use this function to
initialize all devices
{
 cli(); //disable all interrupts
 adc_init();
 timer1_init();
 sei(); //re-enable interrupts
}
// Timer 1 initialized in PWM mode for velocity
control
// Prescale:256
// PWM 8bit fast, TOP=0x00FF
// Timer Frequency:225.000Hz
void timer1_init()
{
 TCCR1B = 0x00;  //Stop
 TCNT1H = 0xFF;  //Counter higher 8-bit value to
which OCR5xH value is compared with
 TCNT1L = 0x01;  //Counter lower 8-bit value to
which OCR5xH value is compared with
 OCR1AH = 0x00;  //Output compare register high
value for Left Motor
 OCR1AL = 0xFF;  //Output compare register low
value for Left Motor
 OCR1BH = 0x00;  //Output compare register high
value for Right Motor
 OCR1BL = 0xFF;  //Output compare register low
value for Right Motor
 TCCR1A = 0xA1;  /*{COM5A1=1, COM5A0=0;
COM5B1=1, COM5B0=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*/

 TCCR1B = 0x0B;  //WGM12=1; CS12=0, CS11=1,
CS10=1 (Prescaler=64)
}

// Function for robot velocity control
void velocity (unsigned char left_motor,
unsigned   char right_motor)
{
 OCR1AL = (unsigned char)left_motor;
//SEPARATE OUTPUT COMPARE OR REQUIRED AT LEFT
MOTOR
```

```
 OCR1BL = (unsigned char)right_motor;
//SEPARATE OUTPUT COMPARE OR REQUIRED AT RIGHT
MOTOR
}

void adc_init()
{
 ADCSRA = 0x00;
 ADMUX = 0x20;   //Vref=5V external --- ADLAR=1
--- MUX4:0 = 0000
 ACSR = 0x80;
 ADCSRA = 0x86;   //ADEN=1 --- ADIE=1 ---
ADPS2:0 = 1 1 0
}

//This Function accepts the Channel Number and
returns the corresponding Analog Value
unsigned char ADC_Conversion(unsigned char Ch)
{
 unsigned char a;
 Ch = Ch & 0x07;
 ADMUX= 0x20| Ch;
 ADCSRA = ADCSRA | 0x40;   //Set start
conversion bit
 while((ADCSRA&0x10)==0);  //Wait for ADC
conversion to complete
 a=ADCH;
 ADCSRA = ADCSRA|0x10; //clear ADIF (ADC
Interrupt Flag) by writing 1 to it
 return a;
}
void forward (void) //both wheels forward
{
 PORTB=0b00000110;
```

```
}

void back (void) //both wheels backward
{
 PORTB=0b00001001;
}

void left (void) //Left wheel backward, Right
wheel forward
{
 PORTB=0b00000101;
}

void right (void) //Left wheel forward, Right
wheel backward
{
 PORTB=0b00001010;
}

void soft_left (void) //Left wheel stationary,
Right wheel forward
{
 PORTB=0b00000100;
}

void soft_right (void) //Left wheel forward,
Right wheel is stationary
{
 PORTB=0b00000010;
}

void stop (void) //hard stop
{
 PORTB=0b00000000;
```

```
}

//Main Function
int main()
{
 int p[100],s[100],i,j,z,x,y;
 char h,l;
 int g;
 int flag=1;
 h=65;l=5;

 DDRB = DDRB | 0x06;
 PORTB = PORTB | 0x06;
 DDRD=0b01111111;   //sensor 7
 PORTD=0b00000000;

 init_devices();


 for (i=0;i<100;i++)
 {
  p[i]=0;
 }

 i=0;
 int flag1=1,flag2=1,flag3=1,flag4=1;
 while(flag==1)
 {
  a=ADC_Conversion(0); //PC0
  b=ADC_Conversion(1); //PC1
  c=ADC_Conversion(2); //PC2
  d=ADC_Conversion(3); //PC3
  e=ADC_Conversion(4); //PC4
  f=ADC_Conversion(5); //PC5
  g=PIND&0b00000001;

  error1=b-offset;
  error2=d-offset;
  integral1 = integral1 + error1;
  derivative1 = error1 - lastError1;
  Turn1 = Kp*error1 + Ki*integral1 +
Kd*derivative1;
  integral2 = integral2 + error2;
  derivative2 = error2 - lastError2;
  Turn2 = Kp*error2 + Ki*integral2 +
Kd*derivative2;

  if (a>l && b<h && c<h && d<h && e>l)    //u
turn
  {
   flag1=1;
   flag2=1;
   flag3=1;

   while(flag4==1)
   {
    p[i]=4;
    flag4=0;
    i++;
   }
   left();
   velocity(0,255);

  }
  else if (a<l && b>h && c>h && d>h &&
e<l)   //t intersection
  {
```

```
 if (g==0b00000000 && f<l)
 {
  stop();
  velocity(0,0);
  flag=0;
  p[i]=0;

 }
 else
 {

  flag2=1;
  flag3=1;
  flag4=1;

  while(flag1==1)
  {
   p[i]=1;
   flag1=0;
   i++;
  }
  forward();
  velocity(255,255);
  _delay_ms(100);
  left();
  velocity(200,255);
  _delay_ms(400);

 }
}
else if(c>h && d>h && e<l)
{
 flag1=1;
 if (f>l)
```

```
{
 flag1=1;
 flag2=1;
 flag4=1;

 while(flag3==1)
 {
  p[i]=3;
  flag3=0;
  i++;
 }
 right();
 velocity(255,200);
 _delay_ms(300);

}
else
{
 flag1=1;
 flag3=1;
 flag4=1;

 while(flag2==1)
 {
  p[i]=2;
  flag2=0;
  i++;
 }
 forward();
 velocity(255,255);

 _delay_ms(300);

}
```

```
 }
 else if (a<l && b>h && c>h)
 {
  flag2=1;
  flag3=1;
  flag4=1;
  while(flag1==1)
  {
   p[i]=1;
   flag1=0;
   i++;
  }
  left();
  velocity(200,255);
  _delay_ms(300);

 }



 else if (c>h)
 {
  flag1=1;
  flag2=1;
  flag3=1;
  flag4=1;
  forward();
  velocity(Tp+Turn1,Tp+Turn2);
 }
  lastError1 = error1;
  lastError2 = error2;
}

_delay_ms(5000);
```

```
 //shorting the path
 for(x=0;x<10;x++)
 {

  // replace
  i=3;
  while(i<=100)
  {
   if(p[i-3]==1 && p[i-2]==4 &&
p[i-1]==1)  //LUL
   {
    p[i-3]=2;
    p[i-2]=0;
    p[i-1]=0;
   }
   if(p[i-3]==1 && p[i-2]==4 &&
p[i-1]==3)  //LUR
   {
    p[i-3]=4;
    p[i-2]=0;
    p[i-1]=0;
   }
   if(p[i-3]==1 && p[i-2]==4 &&
p[i-1]==2)  //LUS
   {
    p[i-3]=3;
    p[i-2]=0;
    p[i-1]=0;
   }
   if(p[i-3]==3 && p[i-2]==4 &&
p[i-1]==1)  //RUL
   {
    p[i-3]=4;
```

```
     p[i-2]=0;
     p[i-1]=0;
    }
    if(p[i-3]==3 && p[i-2]==4 &&
p[i-1]==3)  //RUR
    {
     p[i-3]=2;
     p[i-2]=0;
     p[i-1]=0;
    }
    if(p[i-3]==3 && p[i-2]==4 &&
p[i-1]==1)  //RUS
    {
     p[i-3]=1;
     p[i-2]=0;
     p[i-1]=0;
    }
    if(p[i-3]==2 && p[i-2]==4 &&
p[i-1]==1)  //SUL
    {
     p[i-3]=3;
     p[i-2]=0;
     p[i-1]=0;
    }
    if(p[i-3]==2 && p[i-2]==4 &&
p[i-1]==3)  //SUR
    {
     p[i-3]=1;
     p[i-2]=0;
     p[i-1]=0;
    }
    if(p[i-3]==2 && p[i-2]==4 &&
p[i-1]==2)  //SUS
    {
```

```
   p[i-3]=4;
   p[i-2]=0;
   p[i-1]=0;
  }
  i++;
}

//remove zero
j=0;
i=0;
while(i<100)
{
  if(p[i]!=0)
  {
   s[j]=p[i];
   j++;
  }
  i++;
}

//make p array all 0
i=0;
while(i<100)
{
  p[i]=0;
  i++;
}

//p contain non zero element
i=0;
while(i<100)
{
  p[i]=s[i];
  i++;
```

```
 }
 for(i=0;i<100;i++)
 {
  s[i]=0;
 }

}



 //following shortest path
 i=0;
 flag=1;
 while(flag==1)
 {
  PORTD=0b00000000;
  if (p[i]==1)
  {
   PORTD=0b01000000;
   _delay_ms(2000);
   i++;
   PORTD=0b00000000;
   _delay_ms(500);
  }
  else if (p[i]==2)
  {
   PORTD=0b00100000;
   _delay_ms(2000);
   i++;
   PORTD=0b00000000;
   _delay_ms(500);
  }
  else if (p[i]==3)
  {
     PORTD=0b00010000;
     _delay_ms(2000);
     i++;
     PORTD=0b00000000;
     _delay_ms(500);
    }
    else if (p[i]==4)
    {
     PORTD=0b01110000;
     _delay_ms(2000);
     i++;
     PORTD=0b00000000;
     _delay_ms(500);
    }
    else if (p[i]==0)
    {
     flag=0;
    }

  }
}
```