German University in Cairo

Mechatronics Engineering (MCTR601)

Autonomous robotic fire detection and extinguishing system

Name	ID	Lab Number
Salma Essam	49-9180	t-26
Joseph Maher	49-1660	t-28

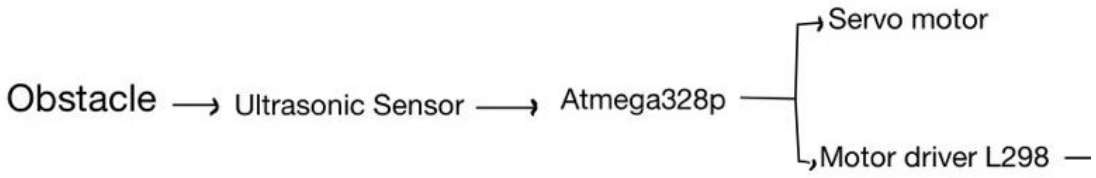
Table of Contents

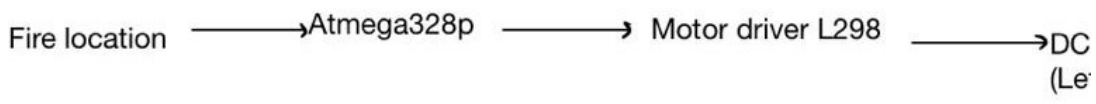
1. Project Description

The field of firefighting has long been a dangerous one, and there have been numerous and devastating losses because of a lack in technological advancement. Additionally, the current methods applied in firefighting are inadequate and inefficient relying heavily on humans who are prone to error, no matter how extensively they have been trained. A recent trend that has become popular is to use robots instead of humans to handle fire hazards. Moreover at the household level, it is observed that if the fire can be extinguished at an early stage, many major accidents can be averted. The aim here is to build a fire extinguishing robot that can help in-case fire breaks out. This fire extinguishing robot is a prototype of the actual one. It uses an ultrasonic sensor and simple dc motors for obstacle avoidance as it is headed towards the fire .Sensors used here for fire detection are simple infrared (IR) photodiodes that detect IR rays coming out of the fire . Voltage across the photodiode is given as input to ADC pins of ATmega328p . When IR waves fall on the IR photodiode, its resistance decreases from 650kΩ to 150kΩ, reducing the voltage across the photodiode, thus changing the input voltage at the ADC pin. By proper quantization , the presence and absence of the flame can be distinguished.. Six IR sensors are mounted on top of the robot's chassis in a circular in shape so that it gives the robot all-round detection view of 360°. Sensors are equally spaced at 60 ◦ each. In the actual robot, use of fire sensors or IR cameras is recommended. But these are too expensive and hence IR photodiodes have been used as a substitute in the prototype. The pump used here symbolises the fire extinguishing mechanism, and is used as a substitute in the prototype. Also, the body used is not fire-proof. The actual robot must use fire-proof material for proper and better functioning for efficient results.

The main sensors : Ultrasonic sensor and IR flame sensors

The main actuators : 2 DC motors , servo motor and dc pump





Methodology

2. Methodology

2.1 Mechanical design

1. **Output display** :
    - 6 LEDs are used to indicate if a fire is detected and are connected to the analog outputs of the flame sensors to indicate the fire intensity
2. **Data input :**
    - A switch
3. **Sensors** :
    - 6 IR Flame sensors are connected to the 6 ADCs to indicate the location and intensity of the fire .The flame sensor basically detects IR (Infrared) light wavelength between 760 nm – 1100 nm that is emitted from the fire flame or light source with a detection angle 0 – 60 degrees. The flame sensor comes with a YG1006 NPN Phototransistor, LM393 Comparators IC, Variable Resistor (Trim pot)and Power LED. Two types of IR Infrared Flame Sensor Module available in the market one having three pins (D0, Gnd, Vcc) and another one having four pins (A0, D0, Gnd, Vcc) . The latter type was used . When this sensor detects fire/flame ,then the resistance of the Phototransistor decreases. Then the maximum amount of voltage will be allocated across the Resistor(R2). So, a Low amount of voltage from the Phototransistor is given to the Inverting input (2) of the IC. Then the Comparator IC compares this voltage with the threshold voltage. In this condition, this input voltage is less than the threshold voltage, so the sensor output goes LOW (0).

      In contrast, when the Flame Sensor module does not detect fire/flame (light in a range of 760 nm – 1100 nm wavelength) then the resistance of the Phototransistor is High. Then the maximum amount of voltage will be allocated across the Phototransistor. So, a High amount of voltage from the Phototransistor is given to the Inverting input (2) of the IC. Then the Comparator IC compares this voltage with the threshold voltage. In this condition, this input voltage is greater than the threshold voltage, so the sensor output goes High (1).

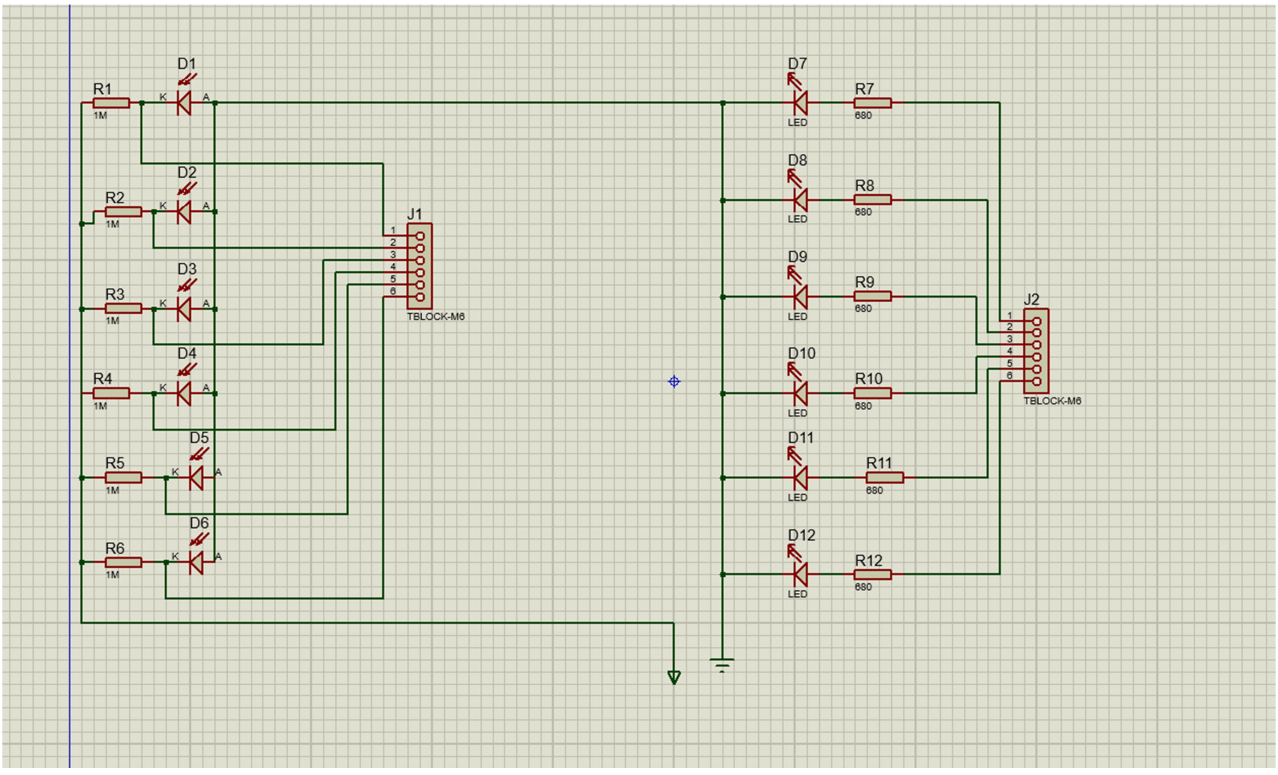
    - Ultrasonic sensor : Mounted on a servo motor to rotate 180 and detect if there is an obstacle in the way. An HC-SR04 ultrasonic distance sensor actually consists of two ultrasonic transducers. One acts as a transmitter that converts the electrical signal into 40 KHz ultrasonic sound pulses. The other acts as a receiver and listens for the transmitted pulses. When the receiver receives these pulses, it produces an output pulse whose width is

proportional to the distance of the object in front. This sensor provides excellent non-contact range detection between 2 cm to 400 cm (~13 feet) with an accuracy of 3 mm.

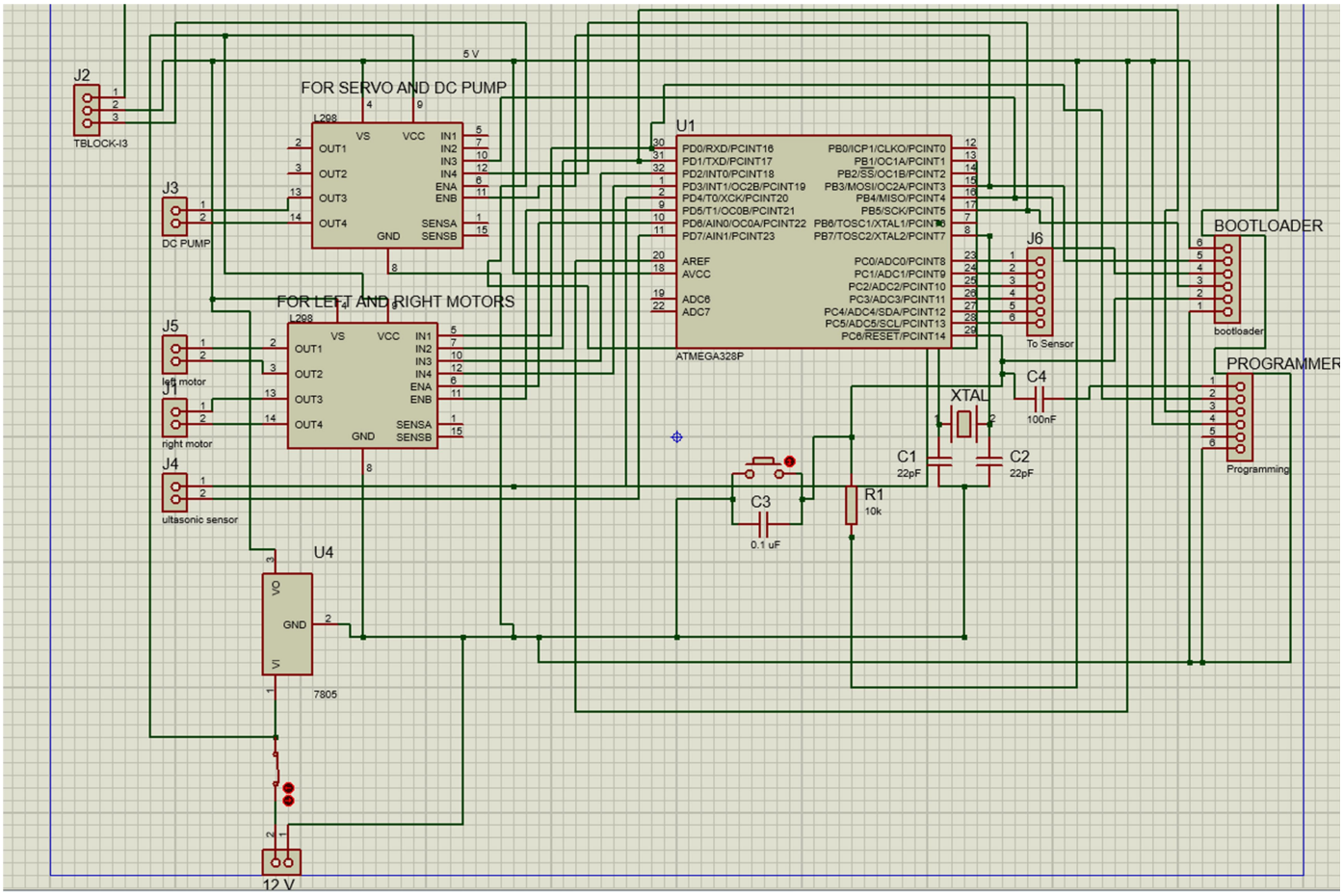
4. **Actuators** :
    - 2 DC motors : controlled by PWM signals OC0A and OCOB
    - DC pump : controlled by PWM signal OC2A
    - Servo motor
5. **Logic ,processing and control :**
    - 2 L298 driver circuits(H-bridge) , 1 voltage regulator , 16 MHz crystal , capacitors (0.1 uF , 22 pF) , resistor (10 k ohm) , ATmega328p microcontroller , 12V battery .

2.2 Electrical design

Sensor Module :



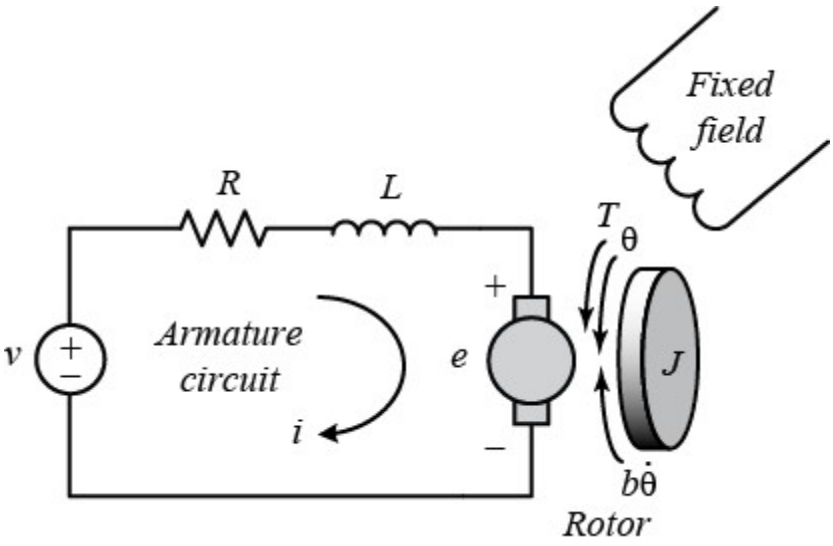
Main Module :



2.3 Control

2.3.1. Modeling

The main actuator in our system is the DC motor . It is used for the motion of the robot ,the rotation of the ultrasonic sensor and in powering the pump. Dc motors directly provide rotary motion and, coupled with wheels ,can provide translational motion. The electric equivalent circuit of the armature and the free-body diagram of the rotor are shown in the following figure.



In general ,the torque generated by a DC motor is proportional to the armature current. The constant of proportionality is called the torque constant and is given the symbol Kt.

$$Tm(t) = Kt * Ia(t) \qquad (1)$$

The back emf, e, is proportional to the angular velocity of the shaft by a constant factor Ke.

$$e(t) = Ke * d\Theta/dt \qquad (2)$$

In SI units, the motor torque and back emf constants are equal, that is, Kt=Ke, therefore, we will use K to represent both the motor torque constant and the back emf constant.

Based on Newton's 2nd law and Kirchhoff's voltage law :

$$J\ddot{\theta} + b\dot{\theta} = Ki \qquad \textit{(3) load equation}$$

$$L\frac{di}{dt} + Ri = V - K\dot{\theta} \qquad \textit{(4) circuit equation}$$

Applying the Laplace transform, the above modeling equations can be expressed in terms of the Laplace variable s.

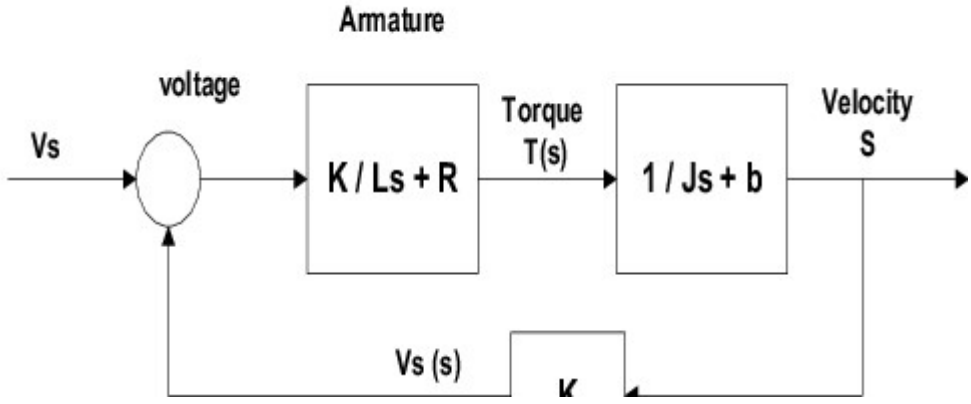
$$S(Js+b)\Theta(s) = KI(s)$$

$$(Ls+R)I(s) = V(s) - Ks\Theta(s)$$

We arrive at the following open-loop transfer function by eliminating I(s) between the two above equations, where the rotational speed is considered the output and the armature voltage is considered the input.

$$H(s) = \frac{\dot{\theta}}{V} = \frac{K}{(Js+b)(Ls+R)+K^2}$$

Block diagram representation :



2.3.2. Analysis

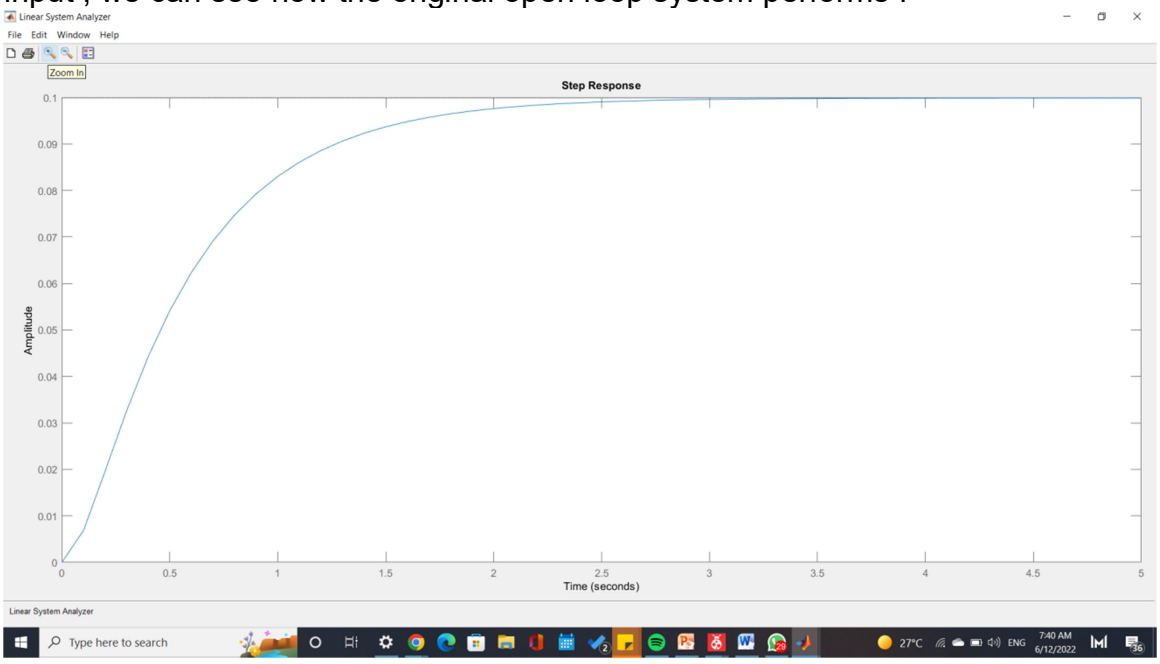
For a typical motor, the paramteres can be assumed as follows :

```
J = 0.01;

b = 0.1;

K = 0.01;

R = 1;

L = 0.5;
```

We can represent the above open-loop transfer function of the motor in MATLAB by defining the parameters and transfer function:

```
Command Window                                   Editor - inputHex.coe
>> J = 0.01;
b = 0.1;
K = 0.01;
R = 1;
L = 0.5;
s = tf('s');
P_motor = K/((J*s+b)*(L*s+R)+K^2)

P_motor =

             0.01
  ---------------------------
  0.005 s^2 + 0.06 s + 0.1001

Continuous-time transfer function.

fx >>
```

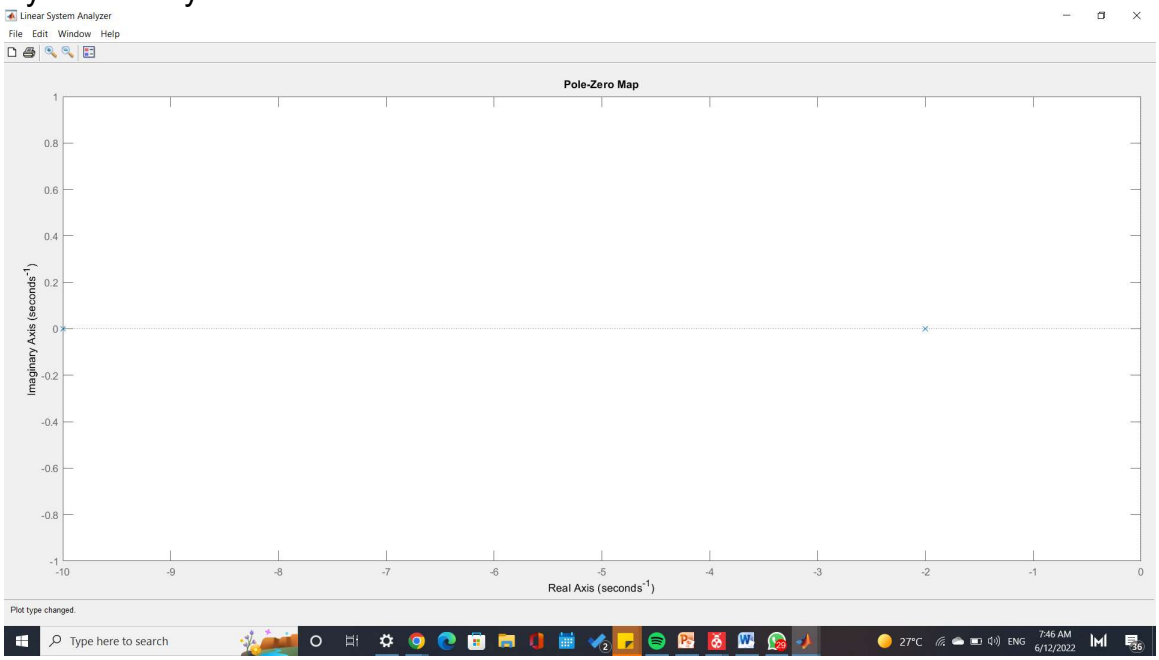
Using the **`linearSystemAnalyzer`** command in matlab and passing a step input , we can see how the original open loop system performs :



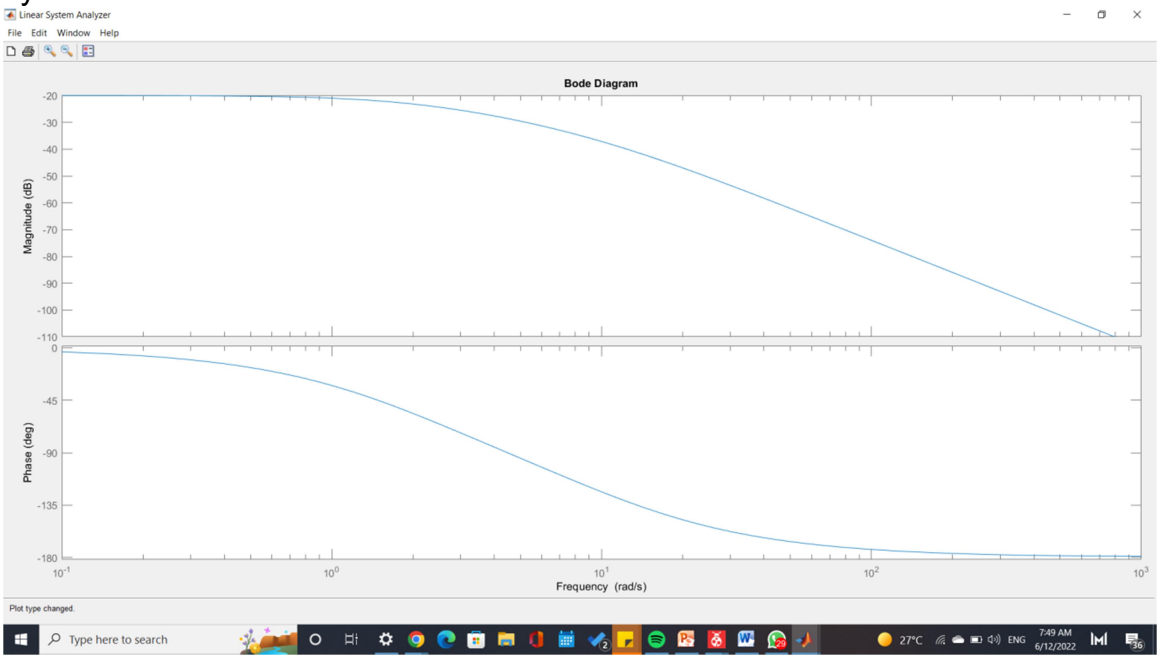
From the plot we see that when 1 Volt is applied to the system the motor can only achieve a maximum speed of 0.1 rad/sec. Also, it takes the motor 2.07 seconds to reach its steady-state speed

Since our open-loop transfer function is a second-order system, we should be able to accurately predict the step response characteristics observed above

based on the transfer function's pole locations. We can graphically see the location of the poles (and zeros) of the P_motor system from within the Linear System Analyzer :

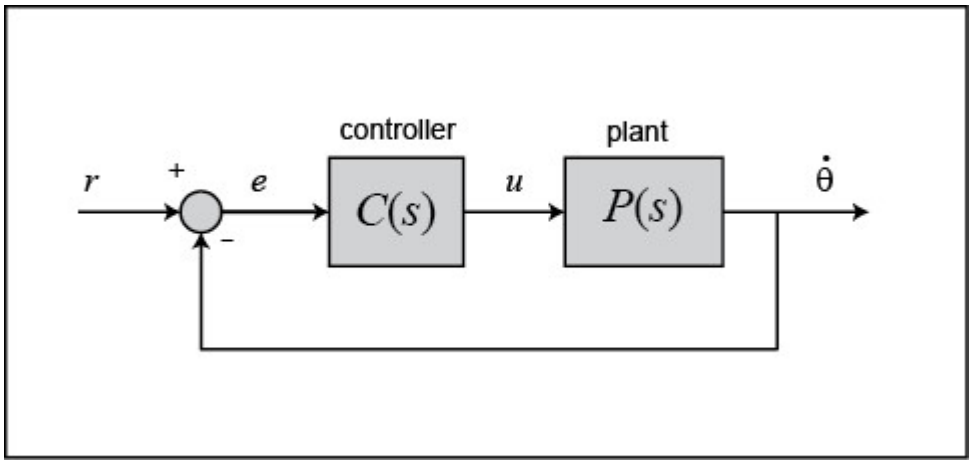


From the above you can see that the open-loop transfer function has two real poles, one at s = -2 and one at s = -10. Since both poles are real, there is no oscillation in the step response (or overshoot) as we have already seen and the system is stable.



2.3.3. Controller Design

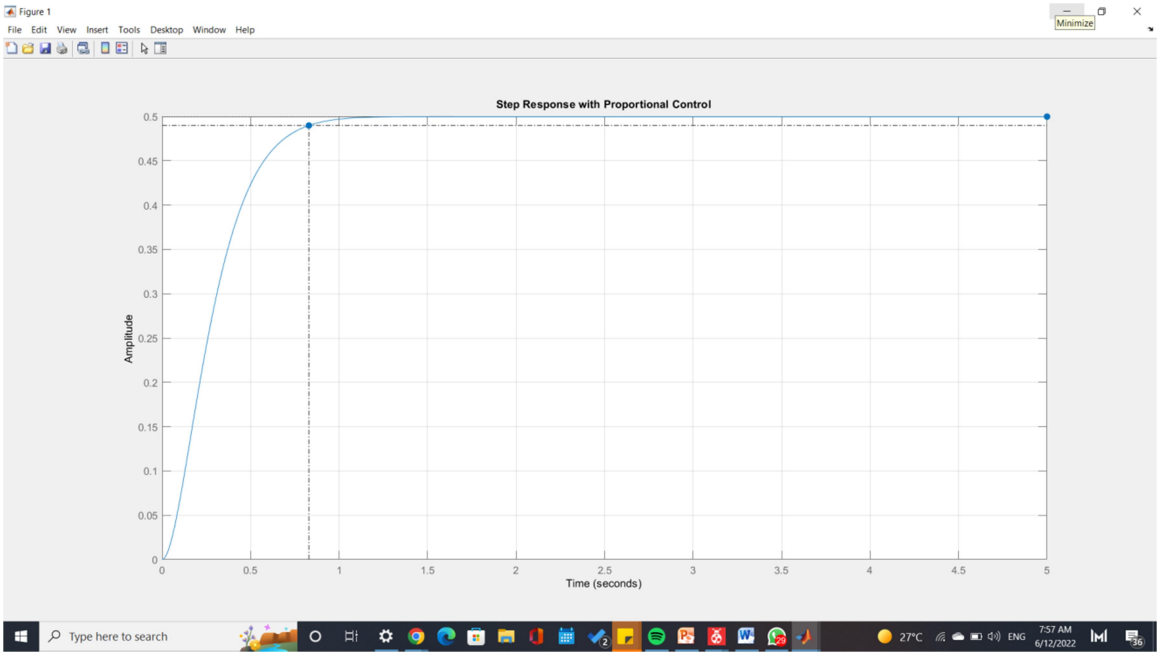
The structure of the control system has the form shown in the figure below.



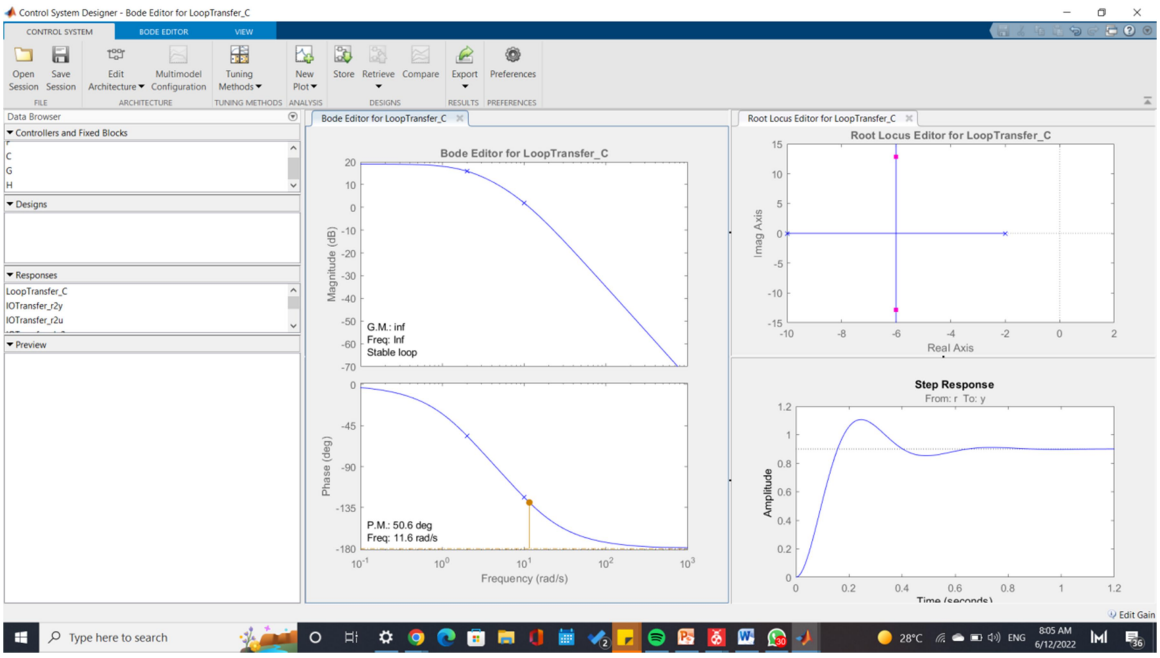
We assume that for a 1-rad/sec step reference, the design requirements are the following.

- Settling time less than 2 seconds
- Overshoot less than 5%
- Steady-state error less than 1%

We start by employing a proportional controller with a gain(Kp) of 10, that is, C(s) = 10. To determine the closed-loop transfer function, we use the feedback command.



By experimenting with different values of Kp using Control System Designer command, we can find the ideal value to meet our requirements

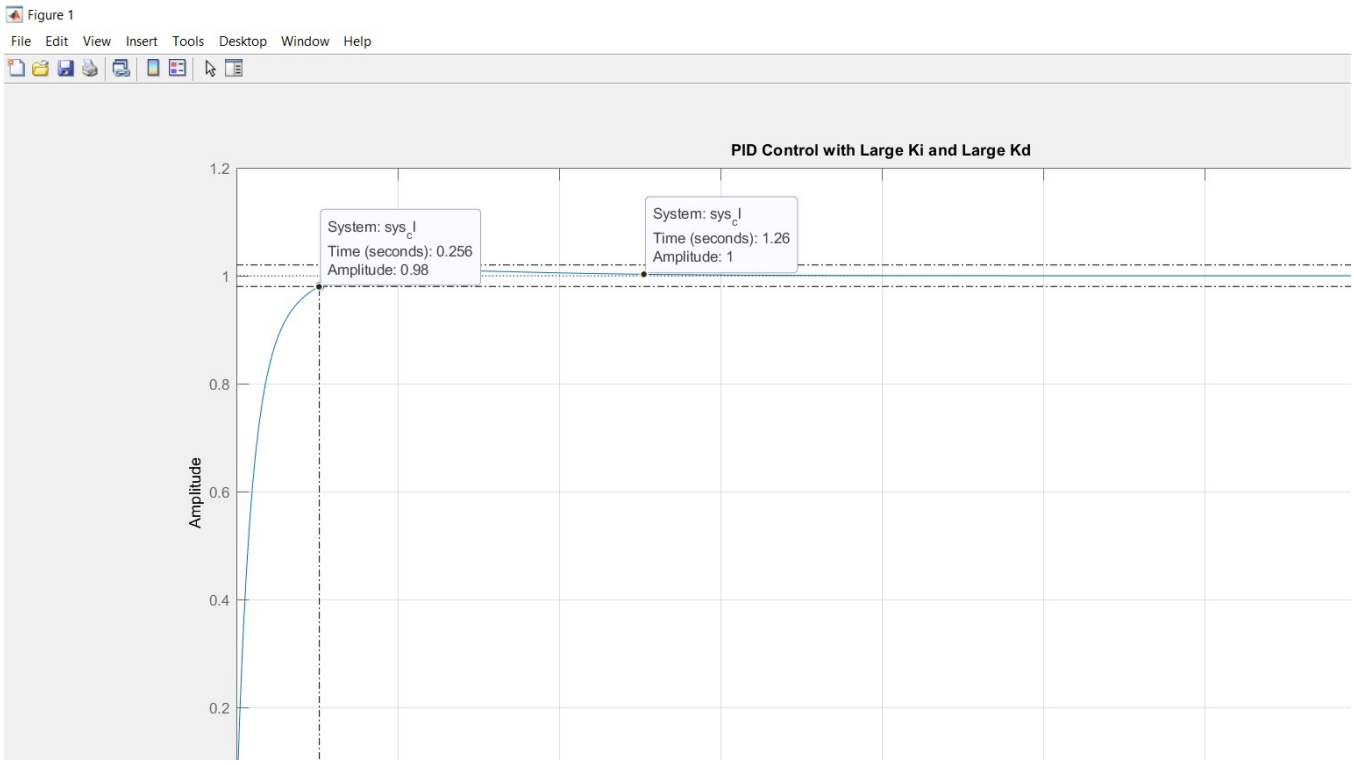


A little experimentation verifies what we anticipated, a proportional controller is insufficient for meeting the given design requirements; derivative and/or integral terms must be added to the controller.

```
Kp = 100;

Ki = 200;

Kd = 10;

C = pid(Kp,Ki,Kd);

sys_cl = feedback(C*P_motor,1);

step(sys_cl, 0:0.01:4)

grid

title('PID Control with Large Ki and Large Kd')
```

we end up with the following control law :

$$C(s) = 100 + \frac{200}{s} + 10s$$



2.3 Programming

```
#define F_CPU 16000000L

#include "FreeRTOS.h"
#include "task.h"
#include "FreeRTOSConfig.h"

#include <avr/io.h>

#include <util/delay.h>


#define US_PORT PORTD
#define  US_PIN   PIND
#define US_DDR    DDRD
#define US_POS PORTD4

#define US_ERROR 0xffff
#define  US_NO_OBSTACLE 0xfffe

int r;
int RightDistance , LeftDistance , FrontDistance;
volatile char x ;
TaskHandle_t xHandle ;

static void vADCRead(void* pvParameters);
static void ObjectAvoidance(void* pvParameters);
```

```
static void DCPump(void* pvParameters);

uint16_t getPulseWidth()
{
	DDRC |= (1<<DDC2);
	uint32_t i,result;

	//Wait for the rising edge
	for(i=0;i<600000;i++)
	{
		if(!(US_PIN & (1<<PORTD7))) continue; else break;
	}

	if(i==600000)
	return 0xffff; //Indicates time out

	//High Edge Found

	//Setup Timer1
	TCCR1A=0X00;
	TCCR1B=(1<<CS11); //Prescaler = Fcpu/8
	TCNT1=0x00;	//Init counter

	//Now wait for the falling edge
	for(i=0;i<600000;i++)
	{
		if(US_PIN & (1<<PORTD7))
		{
			if(TCNT1 > 60000) break; else continue;
		}
		else
		break;
	}

	if(i==600000)
	return 0xffff; //Indicates time out

	//Falling edge found
	result=TCNT1;
	//Stop Timer
	TCCR1B=0x00;
	if(result > 2500)
	{
		PORTC &= ~(1<<PORTC2);
		//Wait();
		//forward();
		return 0xfffe; //No obstacle
	}
	else
	{
		PORTC |= (1<<PORTC2);
		//Wait();
		//push_break();
		return (result);
	}
}
```

```
int scan ()
{

	//Set Ultra Sonic Port as out
	US_DDR|=(1<<US_POS);

	_delay_us(10);

	//Give the US pin a 15us High Pulse
	US_PORT|=(1<<US_POS);   //High

	_delay_us(15);

	US_PORT&=(~(1<<US_POS));//Low

	_delay_us(20);

	//Now make the pin input
	US_DDR&=(~(1<<DDD7));

	//Measure the width of pulse
	r=getPulseWidth();
	return r;
}


void init_Timer_for_Servo()
{
	DDRB |=(1<<1);
	TCCR1A|=(1<<COM1A1)|(1<<COM1B1)|(1<<WGM11);        //NON Inverted PWM
	TCCR1B|=(1<<WGM13)|(1<<WGM12)|(1<<CS11)|(1<<CS10); //PRESCALER=64 MODE
14(FAST PWM)
	ICR1=4999;  //fPWM=50Hz
	TCNT1=0;
}



void motorA_forward()
{
	PORTD |= (1<<1); // 1 is High
	PORTD &= ~(1<<0);// 0 is Low
}


void motorB_forward()
{
	PORTD |= (1<<3); // 3 is High
	PORTD &= ~(1<<2);// 2 is Low
}

void forward()
{
	motorB_forward();
	motorA_forward();

	//delay_ms(duration);
}
```

```
void motorA_backward()
{
	PORTD |= (1<<0); // 0 is High
	PORTD &= ~(1<<1);// 1 is Low
}


void motorB_backward()
{
	PORTD |= (1<<2); // 2 is High
	PORTD &= ~(1<<3);// 3 is Low
}

void backward()
{
	motorA_backward();
	motorB_backward();
	//delay_ms(duration);
}



void motorA_stop()
{
	PORTD &= ~(1<<0); //0 is LOW
	PORTD &= ~(1<<1);// 1 is LOW
}


void motorB_stop()
{
	PORTD &= ~(1<<2); //2 is LOW
	PORTD &= ~(1<<3);// 3 is LOW
}
void push_break()
{
	motorA_stop();
	motorB_stop();

}




void spin()
{
	motorA_forward();
	motorB_backward();
    _delay_ms(500);
}

void left()
{
	motorA_stop();
	motorB_forward();
	_delay_ms(500);
}
```

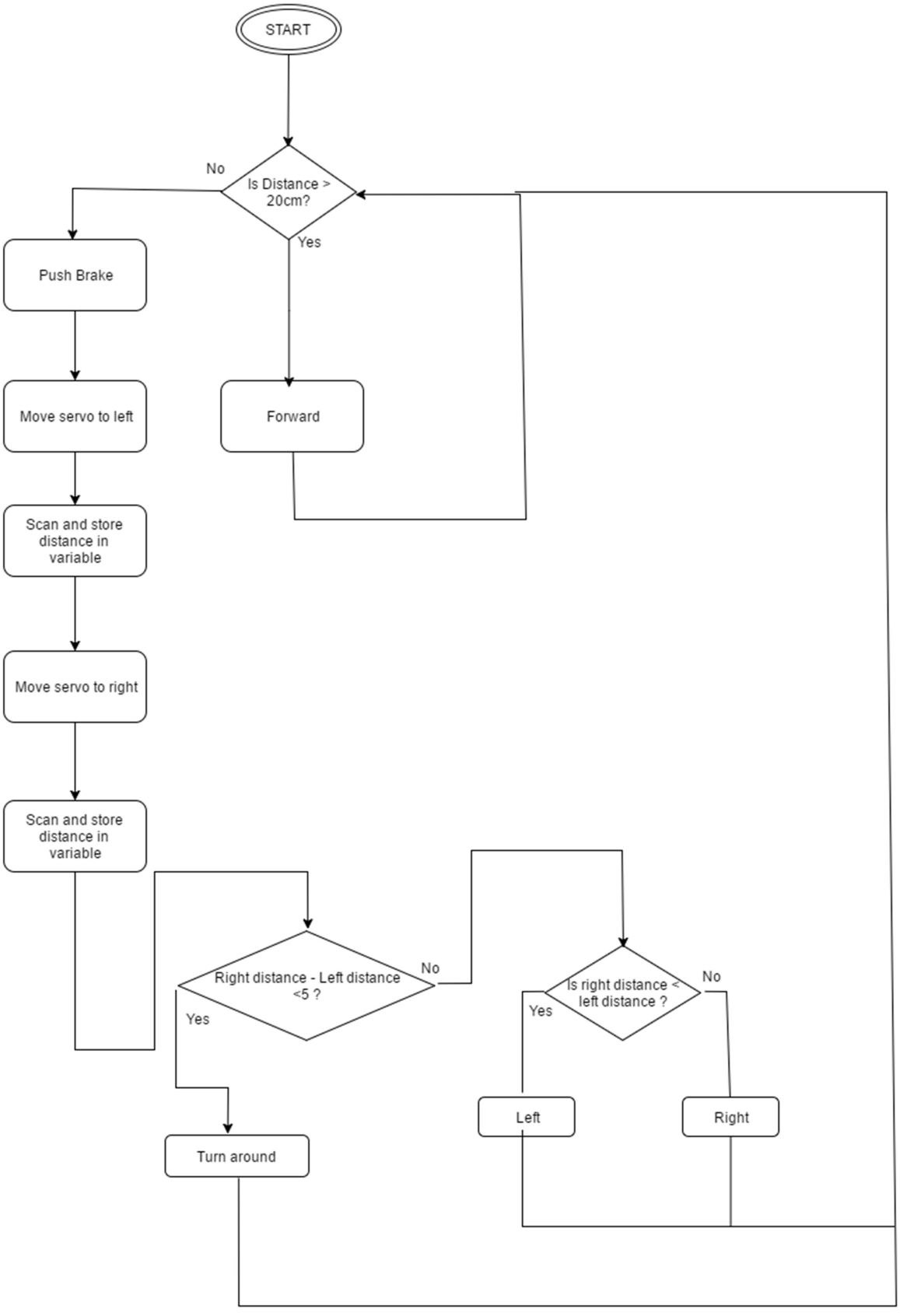
```
void right()
{
	motorA_forward();
	motorB_stop();
	_delay_ms(500);
}

void navigate (){
	init_Timer_for_Servo();
	OCR1A = 120; // right
	_delay_ms(500);
	RightDistance = scan();
	if (RightDistance == 0xfffe) // no obstacle found on the right
	{
		right();
	}
	else {
		init_Timer_for_Servo();
		OCR1A = 555; // left
		_delay_ms(500);
		LeftDistance = scan();
		if (LeftDistance == 0xfffe){ // no obstacle found on the left
			left();
		}
		else if (!(LeftDistance == 0xfffe)) // obstacle found on both sides
		{
			spin();

		}

	}
}



static void vADCRead (void* pvParamters)
{ // for motion
	 DDRD |= (1<<0)|(1<<1)|(1<<2)|(1<<3)|(1<<5)|(1<<6);
	TCCR0A = 0b10100001;
	TCCR0B = 0b00000010;
	OCR0A = 255;
	OCR0B = 255;
	//for ADC
	DDRC = 0b00000000;
	ADCSRA |= 0b00000111;
	ADCSRA |= 0b10000000;
	DIDR0 = 0xff;
	while(1)
	{
	PORTD=0;
	ADMUX= 0;
	ADCSRA|= 0b01000000;
	while (ADCSRA & 0b01000000);
	if(ADC<204){
		x=0;  // fire forward
```

```
            xTaskCreate(DCPump , "Pump", 128 , NULL , tskIDLE_PRIORITY+3,NULL);
// highest priority
       }
       else{
              ADMUX =0b00000001 ;
              ADCSRA|= 0b01000000;
              while (ADCSRA & 0b01000000);
              if(ADC<204){ // fire on left
                     x=1;
                     left();
                     //while(!(x==0));
                     _delay_ms(500);
              }
                           else {
            ADMUX =0b00000010;
                  ADCSRA|= 0b01000000;
              while (ADCSRA & 0b01000000);
                 if(ADC<204){  //fire left x2
                 x=2;
              left();
              //while(!(x==0));
              _delay_ms(500);
                     else{
                     ADMUX =0b00000011;
                     ADCSRA|= 0b01000000;
                     while (ADCSRA & 0b01000000);
                     if(ADC<204){  // fire in the back
                            x=3;
                            spin();
                            // while(!(x==0));
                            _delay_ms(700);
                     }
                     else{
                            ADMUX =0b00000100;
                            ADCSRA|= 0b01000000;
                            while (ADCSRA & 0b01000000);
                            if(ADC<204){  // fire on right
                                   x=4;
                                   right();
                                   _delay_ms(500);
                            }
                            else{
                                   ADMUX =0b00000101;
                                   ADCSRA|= 0b01000000;
                                   while (ADCSRA & 0b01000000);
                                   if(ADC<204){ // fire on right x2
                                          x=5;
                                          right();
                                          _delay_ms(500);
                                   }
                            }
                     }
              }
              }
              }
       //     _delay_ms(100);  // delay between reads
              vTaskDelay(70);
```

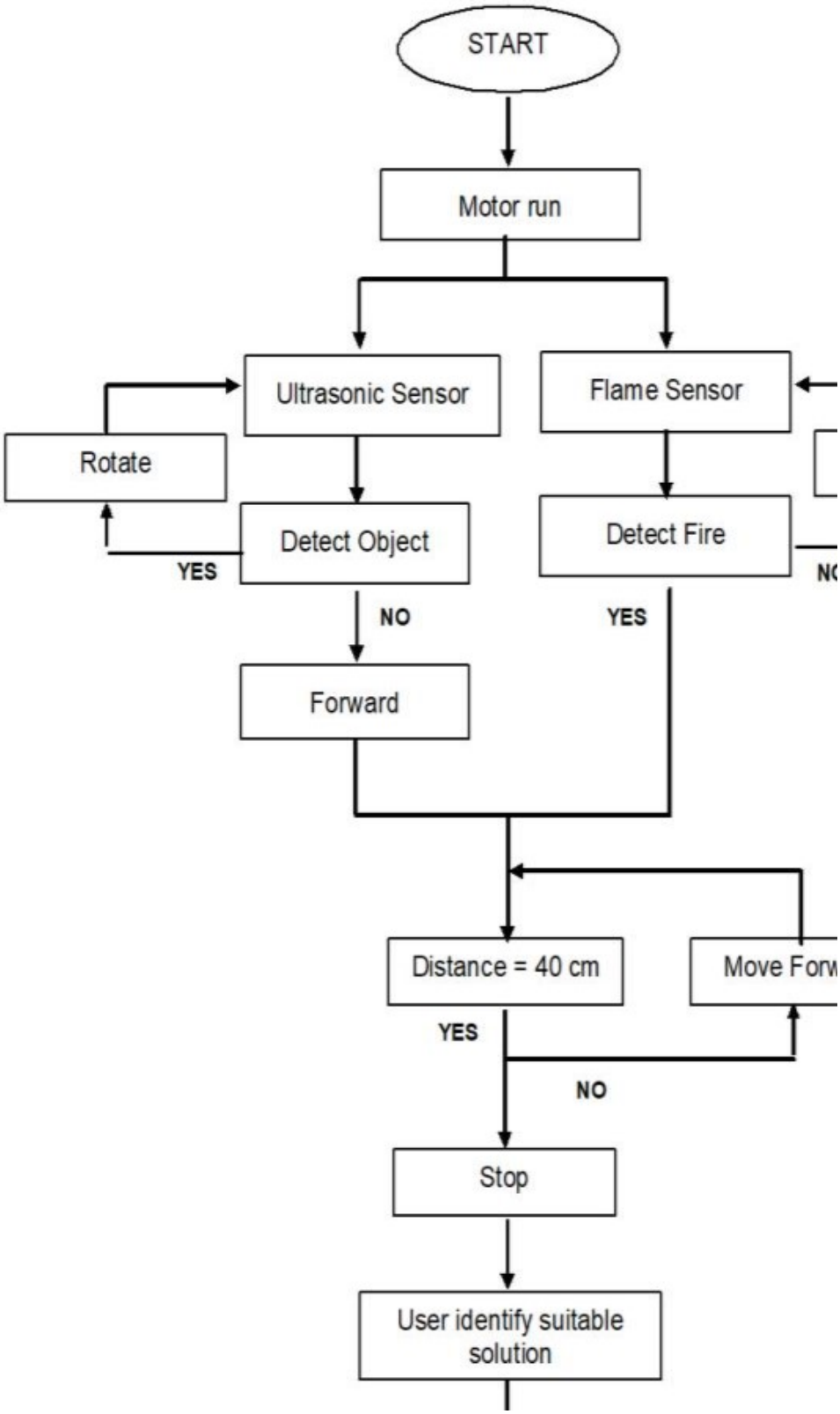
```
        }
}
}
static void ObjectAvoidance(void* pvParameters)
{   TickType_t lastWakeTime= xTaskGetTickCount();
        DDRD |= (1<<0)|(1<<1)|(1<<2)|(1<<3)|(1<<5)|(1<<6);
        TCCR0A = 0b10100001;
        TCCR0B = 0b00000010;
        OCR0A = 255;
        OCR0B = 255;
        forward();
        while (1)
        {
                init_Timer_for_Servo();
                OCR1A = 316; //~90 deg center
                _delay_ms(50);
                int FrontDistance = scan();
                if(FrontDistance ==(0xfffe)) // no obstacle in front
                {
                        forward();
                        //      _delay_ms(1000);
                }
                else        // obstacle detected ---> navigate
                {
                        push_break();
                        navigate();
                }

                 vTaskDelay(100);

        }

}

static void DCPump (void* pvParameters)
{  push_break();
        backward();
        _delay_ms(250);
        push_break();

         int x = 0 ;
        DDRB = 0b00111000;
        TCCR2A = 0b10000001;
        TCCR2B = 0b00000010;

        DDRD = 0b01101111;
        TCCR0A = 0b10100001;
        TCCR0B = 0b00000010;

        PORTB = 0b00010000;
        DDRC = 0b00000000;
        ADCSRA |= 0b00000111;
        ADCSRA |= 0b10000000;
        DIDR0 = 0xff;

        while(1){

                // to move the pump left and right to better target the fire
```

```
              if (x%2==0)
     {left();
       _delay_ms(500);
              x=x+1;
}
  else{
right();
_delay_ms(500);
       x=x+1;
}
//read the adc to see if the fire is getting weaker
                    ADMUX= 0;
               ADCSRA|= 0b01000000;
              while (ADCSRA & 0b01000000);
// PWM signal depends on the fire intensity
              OCR2A =(ADC-400)/4;  //400 is the maximum result for the flame
sensor in case no fire is detected , and as the fire increases the reading
decreases
              // for the motors
              OCR0A = 200;
              OCR0B = 200;
       }
}


int main(void)
{

xTaskCreate(ObjectAvoidance , "Avoid" , 128 , NULL , tskIDLE_PRIORITY + 1 , NULL);
xTaskCreate(vADCRead , "ADC" , 128 , NULL , tskIDLE_PRIORITY + 2 , &xHandle );

       vTaskStartScheduler();
}
```

Flowchart for obstacle Avoidance :



Flowchart of the whole system



:

3. Design Evaluation

Upon implementation , it was found that the actual range of the flame sensors is limited to about 30 cm rather than 100 cm . It is recommended to use another type to get more accurate results . Moreover , it was found that a simple P controller was enough to drive the DC pump and there was no need for the integral/derivative gain .

With that being said , the hardware implementation behaves as expected and according the model simulations obtained