```
//April 2 2025
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <IRremote.h> // infra red receiver
LiquidCrystal_I2C lcd(0x27, 20, 4); // I2C address 0x27, 20 , 4

String command;
int n;
int trigPin = 9;    // TRIG pin ultrasonic sensor
int echoPin = 8;    // ECHO pin ultrasonic sensor
int fan = 10;
int disturb = 6;
int potA0;
int potA1;

int receiver = 7; // Nano Signal Pin of IR receiver

float disturbanceFan;

float propBand=100;
float integralTime=20;
float derivativeTime=0.0;
float sampleTime=100;

bool enableDisturbance=false;// square wave disabled
bool component=false; // display  of PID components (not tuning constants) disabled
bool tuning=true;// display PID Tuning Constants
bool Auto=true;
bool Manual=false;

float Setpoint_Potentiometer();
float Ultrasonic_filterFunction(float timeConstant, float processGain,float blockIn, float intervalTime);
float normalizesDistance();
float PID_output(float process, float setpoint, float Prop, float Integ, float deriv, int Interval);
float DerivativefilterFunction(float timeConstant, float processGain,float blockIn, float intervalTime);

void Disturbance_Potentiometer();
void printText();// Print static text to display

IRrecv irrecv(receiver);    // create instance of 'irrecv'
decode_results results;     // create instance of 'decode_results'
void setup()
{
  lcd.init(); //initialize the lcd
```

```
 lcd.backlight(); //open the backlight
 irrecv.enableIRIn(); // Start the receiver
 // begin serial port
 Serial.begin (9600);

 // configure the trigger pin to output mode
 pinMode(trigPin, OUTPUT);

 // configure the echo pin to input mode
 pinMode(echoPin, INPUT);
 // power to fan
 pinMode(fan, OUTPUT);
 pinMode(disturb, OUTPUT);
 pinMode(7, INPUT);
 irrecv.enableIRIn(); // Start the receiver
 printText();
}

void loop()
{
 float controlledVariable;
 float contOutNorm;
 float setPoint;
//************Infra Red Remote Control******************
if (irrecv.decode(&results) )// have we received an IR signal?
    {
    translateIR();
    irrecv.resume(); // receive the next value
    }

 if(enableDisturbance==false)//enable/disble disturbace potentiometer
 {
  Disturbance_Potentiometer();
 }
 else if(enableDisturbance==true)//square wave disturbance
  {
   n++;

   if(n<=500)
    {
     analogWrite(disturb,255);
     disturbanceFan=10.0;
    }
   if(n>500)
```

```
  {
    analogWrite(disturb,0);
    disturbanceFan=0.0;
  }
  if(n==1000)
    n=0;
 }


 controlledVariable=normalizesDistance();
 setPoint=-Setpoint_Potentiometer() + 1.0;// reveresing pot
 if(Auto==true )
  {
  contOutNorm=PID_output(controlledVariable, setPoint,propBand, integralTime, derivativeTime,
sampleTime);
  analogWrite( fan,((float)contOutNorm)*35+170);
  //analogWrite( fan,((float)contOutNorm)*255);
  lcd.setCursor(0, 3);
  lcd.print("           ");
  lcd.setCursor(0, 3);
  lcd.print("Auto");

  }
 if (Manual==true )
   {
   potA0=analogRead(A0);
   analogWrite(fan,(float)potA0/1023*255);
   Serial.print(-31*controlledVariable+31);
   Serial.print(" = Process");
   Serial.print(",");
   Serial.print(-31*setPoint+31);
   Serial.print(" = Set Point");
   Serial.print(",");
   Serial.print(31*(float)potA0/1023);
   Serial.println(" = manual Fan");
   Serial.print(",");

   lcd.setCursor(0, 3);
   lcd.print("          ");
   lcd.setCursor(0, 3);
   lcd.print("Manual % ");
   lcd.setCursor(9, 3);
   lcd.print("       ");
   lcd.setCursor(9, 3);
```

```
    lcd.print((int)((float)potA0/1023*100));
    lcd.setCursor(9 , 2);
    lcd.print("   ");
    lcd.setCursor(9 , 2);
    lcd.print((int)(-31*setPoint+31));

    lcd.setCursor(9, 0);
    lcd.print("    ");
    lcd.setCursor(9, 0);
    lcd.print((int)(-31*controlledVariable+31));
    }
  delay(sampleTime);
}
// *************Ultrasonic sensor***************
float normalizesDistance()
{
float duration_us;
float distance_cm;
float filtered_distance;
// generate 10-microsecond pulse to TRIG pin
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // measure duration of pulse from ECHO pin
  duration_us = pulseIn(echoPin, HIGH);

  // calculate the distance
  distance_cm = 0.017 * duration_us;
  filtered_distance = Ultrasonic_filterFunction(0.2, 1.0,distance_cm, sampleTime);// first argument is
filter time constant
  if (filtered_distance>31)
    filtered_distance=0;
  return filtered_distance/31;  // range 0 to 31 cm
}

float Ultrasonic_filterFunction(float timeConstant, float processGain,float blockIn, float intervalTime)
{
float static blockOut;
blockOut=blockOut+(intervalTime/5000/(timeConstant+intervalTime/5000))*(processGain*blockIn-
blockOut);
return blockOut;
}
```

```
float Setpoint_Potentiometer()//0 - 500
{

potA0=analogRead(A0);
return (float)potA0/1023;
}

void Disturbance_Potentiometer()
{

potA1=analogRead(A1);
analogWrite(disturb,(float)potA1/1023*255);// temporary for test
disturbanceFan=(float)potA1/1023*10;
}

float PID_output(float process, float setpoint, float Prop, float Integ, float deriv, int Interval)
{
float Er;
static float Olderror, Cont;
static int Limiter_Switch;
static float Integral=0.82;
float derivative;
float proportional;
float deltaT;
float filteredDerivative;
deltaT=float(Interval)/1000;
Limiter_Switch = 1;
//delay(Interval);  // Interval in msec is delta t in the integral and derivative  calculations
 Er=(process-setpoint); //forward acting

//Limiter switch turns integration OFF if controller is already at 100% output or 0% output
//Prevents integral windup, where controller keeps integrating when controller output can no longer
//affect the process.

if ((Cont >= 1 && Er > 0) || (Cont <= 0 && Er < 0) || (Integ >= 3600))
      Limiter_Switch = 0;
else
      Limiter_Switch = 1;

Integral = Integral + 100 / Prop / Integ * Er *deltaT * Limiter_Switch;// Integral calculator
derivative = 100 / Prop * deriv * (Er - Olderror) / deltaT;// Derivative calculator
//filteredDerivative=DerivativefilterFunction(0.1, 1.0,derivative, sampleTime);
proportional = 100 / Prop * Er;// Proportional calculator
```

```
Cont = proportional + Integral + derivative;
Olderror = Er;// remember previous error for deriative calculator

if (Cont > 1) // limit controller output between 0.0 and 1.0 a normalized value
   Cont = 1;

if (Cont < 0)
   Cont = 0;

 // next Serial.print statements are for serial plotter
 Serial.print(-31*process+31);
 Serial.print(" = Process");
 Serial.print(",");
 Serial.print(-31*setpoint+31);
 Serial.print(" = Set Point");
 Serial.print(",");
 Serial.print(disturbanceFan);
 Serial.println(" = disturbance Fan");
 Serial.print(",");
// print to LCD screen*********
lcd.setCursor(9 , 1);
lcd.print("    ");
lcd.setCursor(9 , 1);
lcd.print((int)(Cont*100.0));
if(component==true)
{
  lcd.setCursor(15 , 0);
  lcd.print("     ");
  lcd.setCursor(15 , 0);
  lcd.print((int)(proportional*100.0));

  lcd.setCursor(15 , 1);
  lcd.print("    ");
  lcd.setCursor(15 , 1);
  lcd.print((int)(Integral*100.0));

  lcd.setCursor(15 , 2);
  lcd.print("     ");
  lcd.setCursor(15 , 2);
  lcd.print((int)(derivative*100.0));
}
lcd.setCursor(9 , 2);
lcd.print("   ");
lcd.setCursor(9 , 2);
```

```
lcd.print((int)(-31*setpoint+31));

lcd.setCursor(9, 0);
lcd.print("    ");
lcd.setCursor(9, 0);
lcd.print((int)(-31*process+31));
return  Cont;
}

float DerivativefilterFunction(float timeConstant, float processGain,float blockIn, float intervalTime)
{
float static blockOut;
blockOut=blockOut+(intervalTime/1000/(timeConstant+intervalTime/1000))*(processGain*blockIn-
blockOut);
return blockOut;
}

void printText()// static text never updated
 {

 lcd.setCursor(0, 0);
 lcd.print("Posit cm ");
 lcd.setCursor(0, 1);
 lcd.print("C Out  % ");
 lcd.setCursor(0, 2);
 lcd.print("SetPt cm  ");

 lcd.setCursor(13, 0);
 lcd.print("P ");
 lcd.setCursor(13, 1);
 lcd.print("I ");
 lcd.setCursor(13, 2);
 lcd.print("D ");
 }

 //********************IR Control*****************************************
void translateIR() // takes action based on IR code received
{
 switch(results.value)
{

 case 0xFFA25D: //POWER
 lcd.setCursor(15 , 0);
 lcd.print("     ");
```

```
lcd.setCursor(15 , 0);
lcd.print((int)(propBand));
lcd.setCursor(15 , 1);
lcd.print("     ");
lcd.setCursor(15 , 1);
lcd.print((int)(integralTime));
lcd.setCursor(15 , 2);
lcd.print("     ");
lcd.setCursor(15 , 2);
lcd.print((int)(derivativeTime));
component=false;
tuning=true;
break;

case 0xFFE21D: //FUNC/STOP
lcd.setCursor(15 , 0);
lcd.print("     ");
lcd.setCursor(15 , 1);
lcd.print("     ");
 lcd.setCursor(15 , 2);
lcd.print("     ");
component=true;
tuning=false;
break;

case 0xFF629D: //VOL+
enableDisturbance=false;
break;

case 0xFF22DD: //FAST BACK - Auto
Auto=true;
Manual=false;
break;

case 0xFFC23D: //FAST FORWARD - Manual
Auto=false;
Manual=true;
break;


case 0xFFA857: //VOL-
enableDisturbance=true;
break;
```

```
case 0xFF30CF: //1- Prop Band - 10
if(tuning==true)
{
propBand=propBand-10;
if (propBand<=0)
 {
   propBand=1;
 }
lcd.setCursor(15 , 0);
lcd.print("     ");
lcd.setCursor(15 , 0);
lcd.print((int)(propBand));
}
break;

case 0xFF7A85: //3 -Prop Band + 10
if(tuning==true)
{
propBand=propBand+10;
lcd.setCursor(15 , 0);
lcd.print("     ");
lcd.setCursor(15 , 0);
lcd.print((int)(propBand));
}
break;

case 0xFF10EF: //4 Integ Time - 2
if(tuning==true)
 integralTime=integralTime - 2;
{
 if (integralTime<2)
 {
   integralTime=1;
 }

lcd.setCursor(15 , 1);
lcd.print("    ");
lcd.setCursor(15 , 1);
lcd.print((int)(integralTime));
}
break;

case 0xFF5AA5: //6 -Integ Time + 2
if(tuning==true)
```

```
 {
 integralTime=integralTime + 2;
 lcd.setCursor(15 , 1);
 lcd.print("    ");
 lcd.setCursor(15 , 1);
 lcd.print((int)(integralTime));
 }
 break;

 case 0xFF42BD: //7 - Deriv Time
 if(tuning==true)
 {
 derivativeTime=derivativeTime-1;
 if(derivativeTime<0)
   derivativeTime=0;
 lcd.setCursor(15 , 2);
 lcd.print("     ");
 lcd.setCursor(15 , 2);
 lcd.print((int)(derivativeTime));
 }
 break;

 case 0xFF52AD: //9 - DerivTime+1
 if(tuning==true)
 {
 derivativeTime=derivativeTime+1;
 lcd.setCursor(15 , 2);
 lcd.print("     ");
 lcd.setCursor(15 , 2);
 lcd.print((int)(derivativeTime));
 }
 Break;
 }// End Case
}
```