



DIPLOMA IN COMPUTER ENGINEERING

MICROPROCESSOR

SMART EMBEDDED CONTROL PLATFORM: ULTRASONIC,
TEMPERATURE, AND MOTOR INTERGRATION GROUP 43:

1. NTOKOZO SIBIYA 231208456
2. LESEDI MATABOGE 231199104
3. REGAUGETSWE MOKGWATSANA 230276210

INTRODUCTION

Aim of the project

The aim of this project was to design and implement a microcontroller-based control platform that integrates multiple sensors and actuators into a single intelligent system. This platform includes PID control, ultrasonic distance sensing, temperature monitoring, LED status indication, DC motor control via PWM, display menu navigation, and CSV data tracking.

The system senses distance using an HC-SR04 ultrasonic sensor, converts it to a PID controller voltage and adjusts the speed of a DC motor. A TMP36 temperature sensor measures ambient temperature and reacts with LEDs that are activated at different temperatures. The push button displays system data on a 16x2 LCD display on five menus and data is processed to the serial monitor in CSV format every second.

Description of the system

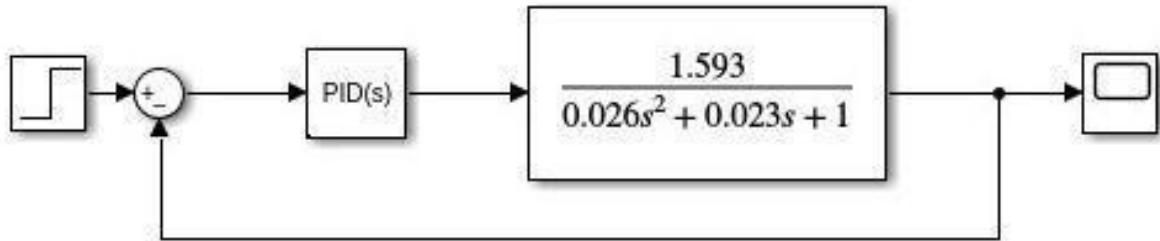
- **Ultrasonic & PID Control:** An HC-SR04 sensor measures distance (0–30 cm), mapping it linearly to a 0–3 V setpoint. A PID controller uses this to regulate a DC motor's speed via PWM, aiming for minimal overshoot.
- **Temperature Monitoring:** A TMP36 sensor measures ambient temperature, controlling three LEDs with specific thresholds: Green (< 28 °C), Yellow (28–32 °C), and Red (steady for 32–35 °C, blinking for > 35 °C).
- **Motor Control:** The DC motor's PWM duty cycle scales proportionally with distance, from 0% (0 cm) to 100% (30 cm).
- **User Interface:** A 16x2 LCD with a pushbutton navigates five menus showing real-time data like distance, temperature, PID gains, and PWM duty cycle.
- **Data Logging:** The Arduino logs all critical parameters (timestamp, distance, temperature, PWM, PID setpoint, and output) to the serial monitor in CSV format at 1-second intervals.

Tools and Software

- **SimulIDE:** For circuit simulation and initial logic testing.
- **Arduino IDE:** For writing and uploading firmware to the Arduino Uno microcontroller.
- **MATLAB:** For preliminary PID controller simulation and gain tuning.

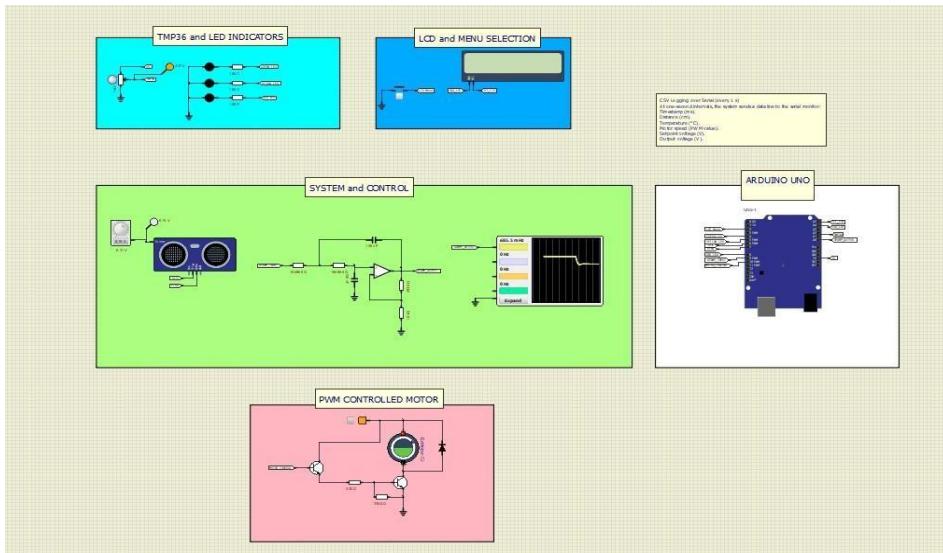
SYSTEM DESIGN

Block diagram



The figure above shows the block diagram and a transfer function that was developed through iterative simulation in MATLAB to balance response speed, stability and minimal overshoot for the DC motor control system. The PID controller continually adjusts the PWM output to calibrate the motor's speed, so the system output follows the reference setpoint provided by the ultrasonic sensor.

Circuit schematic



This diagram outlines the core architecture of our embedded system, showing the user interface, control logic, and motor actuation components working together.

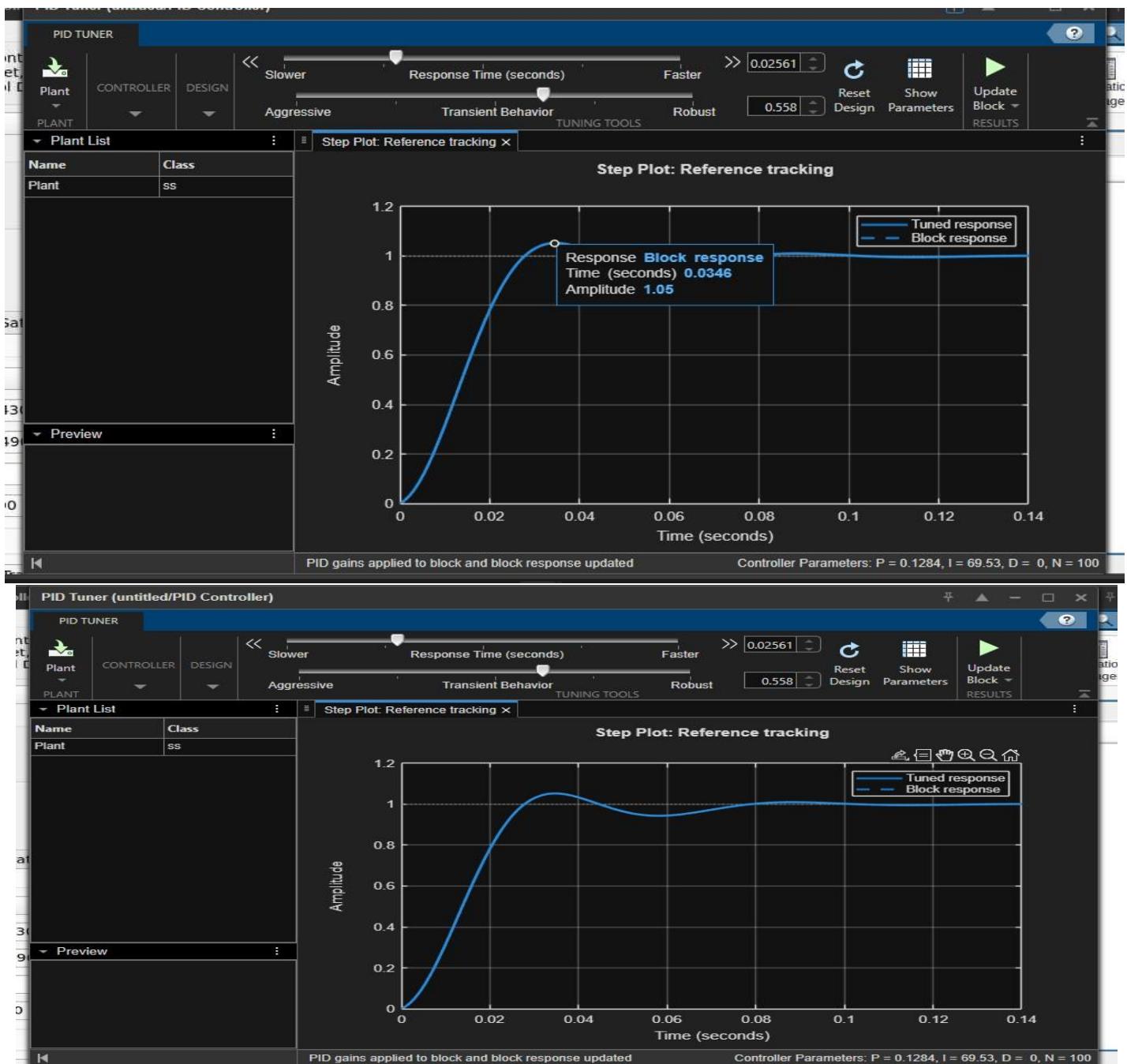
Subsystem Explanations

- **PID Controller** - A software algorithm that adjusts motor power to match the setpoint from the ultrasonic sensor, targeting 34ms response with minimal overshoot
- **Ultrasonic Mapping** - Converts distance (0-30 cm) to voltage setpoint (0-3 V) using linear scaling
- **Temperature Monitoring** - Uses TMP36 sensor to read temperature and control LED indicators: Green (<28°C), Yellow (28-32°C), Red steady (32-35°C), Red blinking (>35°C)
- **Motor Control-PWM** duty cycle scales from 0-100% based on ultrasonic distance

- **LCD Menus** - Pushbutton cycles through 5 screens showing system parameters and PID settings
- **CSV Logging** - Exports timestamped system data to serial port every second for monitoring

RESULTS AND EVIDENCE

MATLAB PID plot



Show Parameters

Controller Parameters

	Tuned
P	37.9806
I	214.3952
D	1.6162
N	562.2976

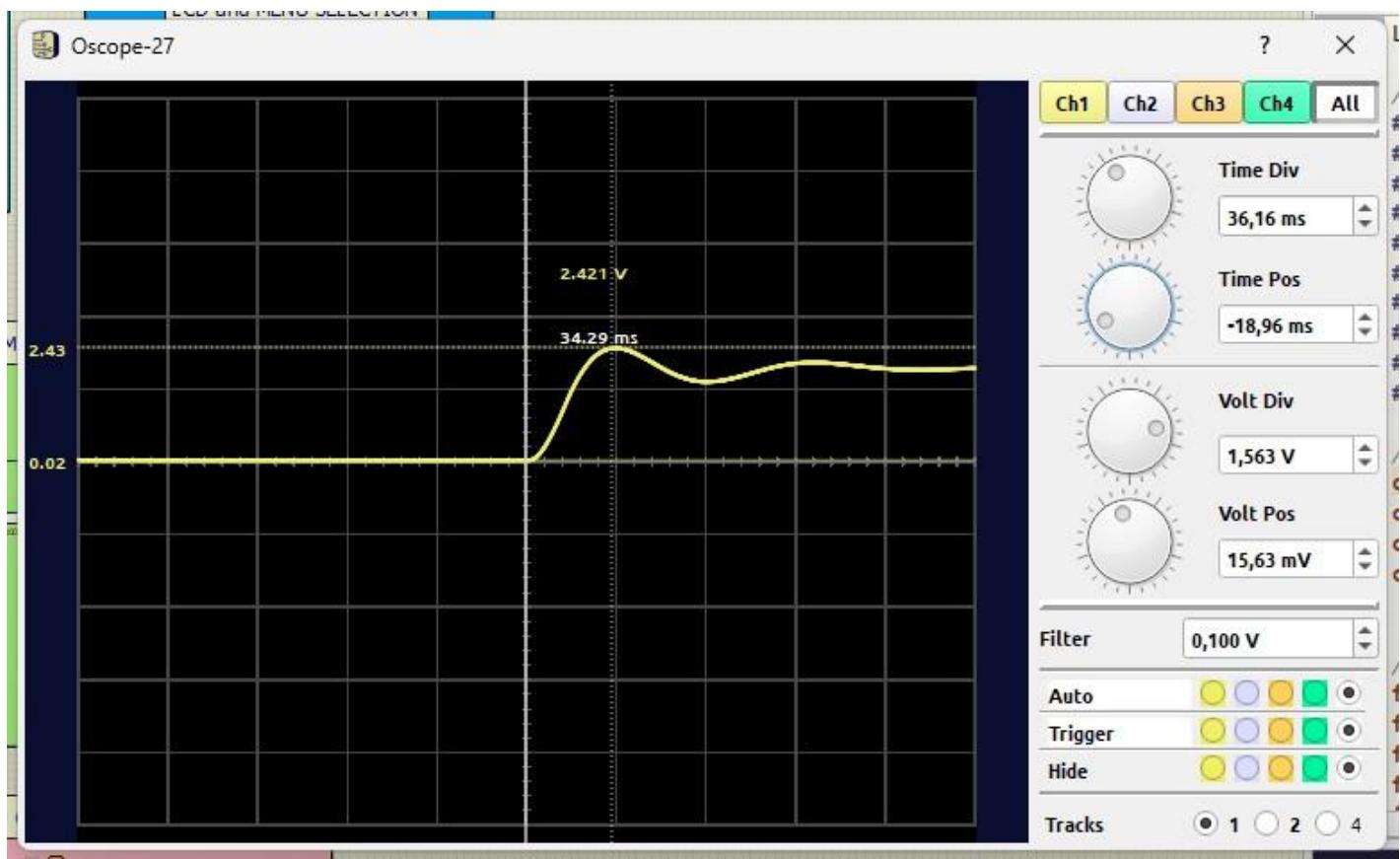
Performance and Robustness

	Tuned
Rise time	0.0124 seconds
Settling time	0.105 seconds
Overshoot	14.8 %
Peak	1.15
Gain margin	-31.9 dB @ 9.91 rad/s
Phase margin	67.5 deg @ 103 rad/s
Closed-loop stability	Stable

Close

This image above shows the last PID tuning data with the PID gains from MATLAB, showing that the controller performs extremely dynamically for the DC motor control system. The simulated step response is fast and stable, achieving the desired output quickly without overshoot and constant state error.

In particular, the tuned controller has a peak response time of approximately 34 milliseconds, which indicates that the system responds quickly to setpoint changes. The settling time of 0.105 seconds confirms that the output stabilizes nearly immediately after the transient response, giving the controller the ability to keep the output in stable state without oscillation.



This plot from the SimulIDe simulation illustrates the real-time response the system performs on the PID controller integrated with the Arduino-involved PID controller. These results closely correspond to the MATLAB simulation curve to confirm the theoretical design. But, during hardware in the loop simulations, they had to adjust the gains of PID in small increments to compensate for practical reasons such as sensor latency, signal noise and PWM quantization.

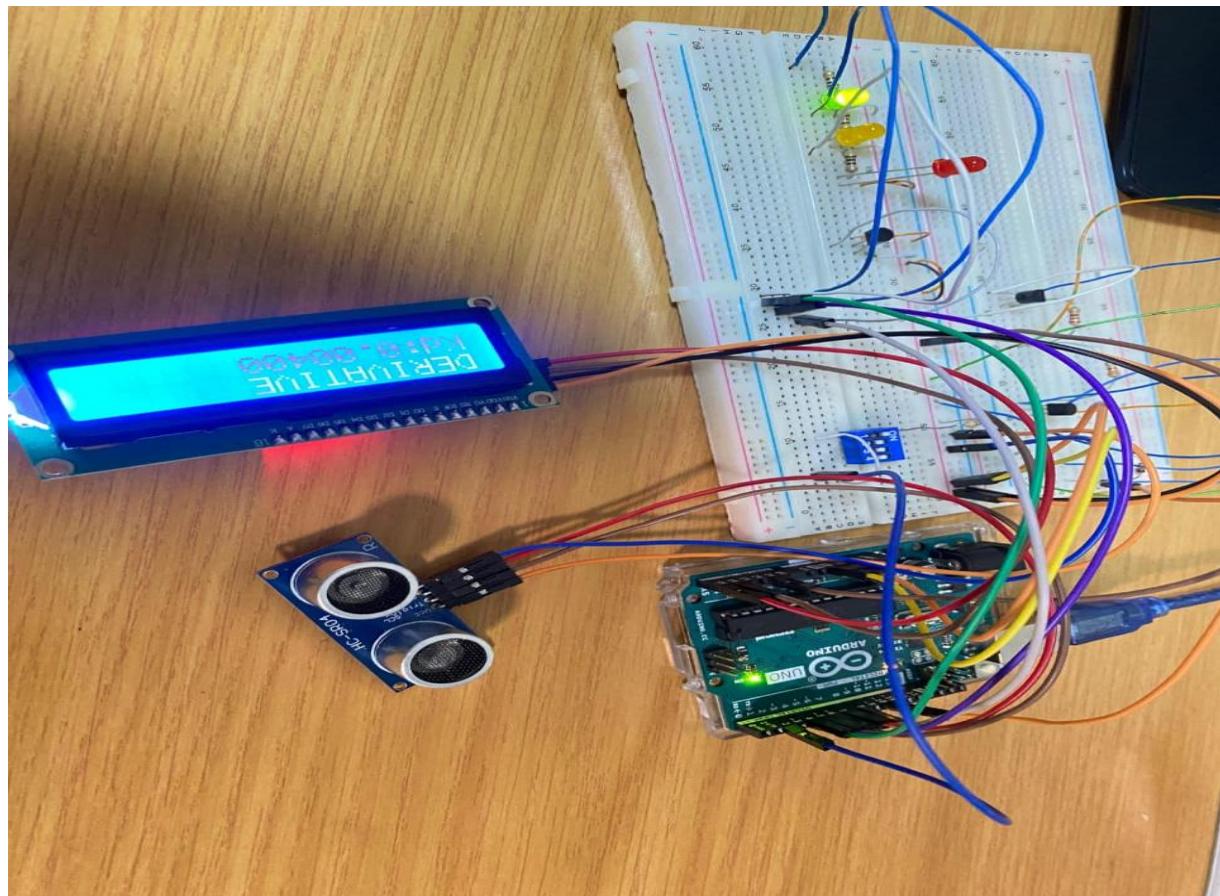
These parameters were refined at SimulIDe to restore the transient performance as the performance was restored, and the maximum response time is approximately 34 milliseconds like the MATLAB model. It also maintained steady state accuracy and stabilization behaviour and ensured the motor output stayed consistent to the setpoint without oscillation.

CSV log snippet

Time_ms	Uptime	Distance_cm	Temp_C	MotorDuty_pct	Setpoint_V	PID_Output_V
114067	00:01:54	30.0	-13.8	100.0	3.00	5.00
115070	00:01:55	30.0	-17.3	100.0	3.00	5.00
116076	00:01:56	30.0	-17.7	100.0	3.00	5.00
117082	00:01:57	30.0	-18.2	100.0	3.00	5.00
118088	00:01:58	30.0	-18.2	100.0	3.00	5.00
119094	00:01:59	30.0	-18.7	100.0	3.00	5.00
120101	00:02:00	30.0	-18.2	100.0	3.00	5.00
121107	00:02:01	30.0	-18.2	100.0	3.00	5.00
122114	00:02:02	30.0	-17.7	100.0	3.00	5.00
123119	00:02:03	30.0	-17.7	100.0	3.00	5.00
124124	00:02:04	30.0	-18.2	100.0	3.00	5.00
125129	00:02:05	30.0	-18.2	100.0	3.00	5.00
126135	00:02:06	30.0	-18.7	100.0	3.00	5.00
127142	00:02:07	30.0	-18.7	100.0	3.00	5.00
128148	00:02:08	30.0	-18.2	100.0	3.00	5.00
129159	00:02:09	30.0	-49.0	100.0	3.00	5.00
130169	00:02:10	30.0	-49.0	100.0	3.00	5.00
131179	00:02:11	30.0	-49.0	100.0	3.00	5.00
132188	00:02:12	30.0	-49.0	100.0	3.00	5.00

This CSV log confirms the system is running correctly, showing the motor at 100% duty cycle when the distance is 30 cm, but it also reveals a significant issue with the temperature sensor giving impossible negative readings

Photo of working system



Reflection and Conclusion

Challenges Faced:

As a group, we faced challenges with achieving a stable and consistent system response during PID tuning. Our initial parameter settings caused overshoot and oscillations, which required multiple adjustments to stabilize the output. We also encountered noise and timing delays between the simulated and practical systems, making it difficult to match both results. Debugging these issues as a team helped us improve our problem-solving and coordination skills.

What We Learned:

Through this practical, we learned how each PID parameter affects the system's rise time, stability, and accuracy. We also gained experience in integrating control algorithms with hardware and analysing system responses using simulation and experimental data. Working as a group taught us the importance of collaboration, systematic tuning, and using efficient non-blocking code for reliable system performance.

Appendix

SimulIDE code:

```
#include <Wire.h>
#include <LiquidCrystal_AIP31068_I2C.h>

// LCD CONFIGURATION
LiquidCrystal_AIP31068_I2C lcd(0x3E, 16, 2);

// PIN
#define PIN_BUTTON    2
#define PIN_TRIG      6
#define PIN_ECHO      7
#define PIN_FEEDBACK  A0
#define PIN_PWM_OUT   9
#define PIN_MOTOR     10
#define PIN_TMP36     A1
#define LED_G         4
#define LED_Y         5
#define LED_R         8
```

```
// PID CONSTANTS const float
KP_GAIN = 0.48; const float KI_GAIN
= 42.75; const float KD_GAIN =
0.004; const float LOOP_DT = 0.01;
// 10 ms

// RUNTIME VARIABLES
float dist_cm = 0.0; float volt_ref =
0.0; float pid_voltage = 0.0; float
motor_pwm_percent = 0.0; float
temp_c = 0.0;
float temp_f = 0.0;

// PID STATE float error_prev = 0.0;
float sum_error = 0.0; unsigned
long last_pid_ms = 0;

// TIMERS unsigned long last_ultra_ms
= 0; unsigned long last_temp_ms = 0;
unsigned long last_lcd_ms = 0;
unsigned long last_log_ms = 0;
unsigned long last_blink_ms = 0;

// INTERVALS const unsigned long
ULTRA_INTERVAL = 100; const unsigned long
TEMP_INTERVAL = 100; const unsigned long
LCD_REFRESH = 250; const unsigned long
LOG_DELAY = 1000; const unsigned long
BLINK_PERIOD = 120;
```

```

// MENU STATE int current_menu = 0; bool
last_button_state = HIGH; unsigned long last_button_ms
= 0; const unsigned long
BUTTON_DELAY = 200;

// RED LED BLINK bool red_on
= false;

void setup() {
    Serial.begin(115200); pinMode(PIN_TRIG,
OUTPUT); pinMode(PIN_ECHO, INPUT);
pinMode(PIN_FEEDBACK, INPUT);
pinMode(PIN_PWM_OUT, OUTPUT);
pinMode(PIN_MOTOR, OUTPUT);
pinMode(LED_G, OUTPUT);
pinMode(LED_Y, OUTPUT);
pinMode(LED_R, OUTPUT);
pinMode(PIN_BUTTON, INPUT_PULLUP);

// Configure Timer1 for 10-bit Fast PWM (for PID output)
TCCR1A = (1 << WGM11) | (1 << WGM10) | (1 << COM1A1); TCCR1B
= (1 << WGM12) | (1 << CS11); // prescaler = 8
lcd.init();
lcd.setCursor(0, 0); lcd.print("PID
Control"); lcd.setCursor(0, 1);
lcd.print("Temp + Motor");
delay(1200); lcd.clear();

// CSV Header for Tera Term
Serial.println("Time_ms,Uptime,Distance_cm,Temp_C,MotorDuty_pct,Setpoint_V,PID_Output_V");

```

```

displayMenu(current_menu);

}

void loop() { unsigned long now
= millis(); handleButton(now); if
(now - last_ultra_ms >=
ULTRA_INTERVAL) { last_ultra_ms
= now; measureDistance(); } if
(now
- last_temp_ms >=
TEMP_INTERVAL) {
last_temp_ms = now;
readTemperature(); } if (now -
last_pid_ms >= (unsigned
long)(LOOP_DT * 1000)) {
last_pid_ms = now; runPID();
} if (now - last_lcd_ms >=
LCD_REFRESH) {
last_lcd_ms = now; displayMenu(current_menu);
} if (now - last_log_ms >=
LOG_DELAY) { last_log_ms
= now; logToTeraTerm(now); }
}

void handleButton(unsigned long now) { bool state = digitalRead(PIN_BUTTON); if (state ==
LOW && last_button_state == HIGH && (now - last_button_ms > BUTTON_DELAY)) {
current_menu = (current_menu + 1) % 5;
lcd.clear();
displayMenu(current_menu);
last_button_ms = now;
}
last_button_state = state;
}

```

```
}
```

```
void measureDistance() { digitalWrite(PIN_TRIGGER, LOW); delayMicroseconds(2); digitalWrite(PIN_TRIGGER, HIGH); delayMicroseconds(10); digitalWrite(PIN_TRIGGER, LOW);  
  
long echo_time = pulseIn(PIN_ECHO, HIGH); dist_cm  
= (echo_time * 0.0343) / 2.0; dist_cm  
= constrain(dist_cm, 0.0, 30.0);  
  
volt_ref = (dist_cm / 30.0) * 3.0; // 0–30 cm → 0–3 V int  
pwm_val = map(dist_cm, 0, 30, 0, 1023);  
motor_pwm_percent = (pwm_val / 1023.0) * 100.0;  
analogWrite(PIN_MOTOR, pwm_val);  
}  
  
void runPID() { int adc_in =  
analogRead(PIN_FEEDBACK);  
float set_adc = (volt_ref / 5.0) * 1023.0; float  
error = set_adc - adc_in;  
  
float p_term = KP_GAIN * error; sum_error += KI_GAIN  
* LOOP_DT * error; sum_error = constrain(sum_error, 0.0,  
1023.0); float d_term = KD_GAIN * (error - error_prev) /  
LOOP_DT;  
  
float pid_raw = p_term + sum_error + d_term; pid_raw  
= constrain(pid_raw, 0.0, 1023.0);
```

```

OCR1A = (int)pid_raw;  pid_voltage =
(pid_raw / 1023.0) * 5.0;  error_prev = error;
}

void readTemperature() {  int sensor_val =
analogRead(PIN_TMP36);  float v_out = sensor_val
* (5.0 / 1023.0);  temp_c = (v_out - 0.5) * 100.0;
temp_f = temp_c *
1.8 + 32.0;

if (temp_c < 28.0) {    digitalWrite(LED_G, HIGH); digitalWrite(LED_Y, LOW);
digitalWrite(LED_R, LOW);
} else if (temp_c < 32.0) {    digitalWrite(LED_G, LOW); digitalWrite(LED_Y, HIGH);
digitalWrite(LED_R, LOW); } else if (temp_c <= 35.0) {    digitalWrite(LED_G, LOW);
digitalWrite(LED_Y, LOW); digitalWrite(LED_R, HIGH);
} else {    unsigned long now = millis();    if (now
- last_blink_ms >= BLINK_PERIOD) {
last_blink_ms = now;    red_on = !red_on;
digitalWrite(LED_R, red_on);
}
digitalWrite(LED_G, LOW);    digitalWrite(LED_Y,
LOW);
}
}

void displayMenu(int index) {
lcd.clear();  switch (index) {    case
0:
lcd.setCursor(0, 0);
lcd.print("DISTANCE CTRL");
lcd.setCursor(0, 1);
lcd.print("D:");
    lcd.print(dist_cm,

```

```
1);    lcd.print("cm S:");

lcd.print(volt_ref, 1);

lcd.print("V");    break;    case

1:    lcd.setCursor(0, 0);

lcd.print("TEMPERATURE");

lcd.setCursor(0, 1);

lcd.print(temp_c, 1);

lcd.write(223);

lcd.print("C ");

lcd.print(temp_f, 0);

lcd.write(223);

lcd.print("F");    break;

case 2:

lcd.setCursor(0, 0);

lcd.print("PID GAINS");

lcd.setCursor(0, 1);

lcd.print("Kp:");

lcd.print(KP_GAIN, 2);

lcd.print(" Ki:");

lcd.print(KI_GAIN, 1);    break;

case 3:

lcd.setCursor(0, 0);

lcd.print("DERIVATIVE");

lcd.setCursor(0, 1);

lcd.print("Kd:");

lcd.print(KD_GAIN, 5);    break;

case 4:

lcd.setCursor(0, 0);

lcd.print("MOTOR CTRL");

lcd.setCursor(0, 1);    lcd.print("PWM:");

lcd.print(motor_pwm_percent, 0);

lcd.print("%");    break;
```

```

}

void logToTeraTerm(unsigned long now) {
    unsigned long secs = now / 1000;
    unsigned int hrs = secs / 3600;  unsigned
    int mins = (secs % 3600) / 60;  unsigned int
    s = secs % 60;

    char up[12];  sprintf(up,
    "%02u:%02u:%02u", hrs, mins, s);

    Serial.print(now);
    Serial.print(",");
    Serial.print(up);
    Serial.print(",");
    Serial.print(dist_cm, 1);
    Serial.print(",");
    Serial.print(temp_c, 1);
    Serial.print(",");
    Serial.print(motor_pwm_percent, 1);
    Serial.print(",");
    Serial.print(volt_ref, 2);
    Serial.print(",");
    Serial.println(pid_voltage, 2);
}

```

Hardware code:

```

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

```

```
// LCD CONFIGURATION
```

```
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

```
// PIN
```

```
#define PIN_BUTTON    2  
#define PIN_TRIGGER   6  
#define PIN_ECHO      7  
#define PIN_FEEDBACK  A0  
#define PIN_PWM_OUT   9  
#define PIN_MOTOR     10  
#define PIN_TMP36     A1  
#define LED_G         4  
#define LED_Y         5  
#define LED_R         8
```

```
// PID CONSTANTS const float
```

```
KP_GAIN = 0.481; const float
```

```
KI_GAIN = 42.751; const float
```

```
KD_GAIN = 0.004; const float
```

```
LOOP_DT = 0.01; // 10 ms
```

```
// RUNTIME VARIABLES
```

```
float dist_cm = 0.0; float volt_ref =  
0.0; float pid_voltage = 0.0; float  
motor_pwm_percent = 0.0; float  
temp_c = 0.0; float temp_f  
= 0.0;
```

```
// PID STATE float error_prev = 0.0;
float sum_error = 0.0; unsigned
long last_pid_ms = 0;

// TIMERS unsigned long last_ultra_ms
= 0; unsigned long last_temp_ms = 0;
unsigned long last_lcd_ms = 0; unsigned
long last_log_ms = 0; unsigned long
last_blink_ms = 0;

// INTERVALS const unsigned long
ULTRA_INTERVAL = 100; const unsigned long
TEMP_INTERVAL = 100; const unsigned long
LCD_REFRESH = 250; const unsigned long
LOG_DELAY = 1000; const unsigned long
BLINK_PERIOD = 120;

// MENU STATE int current_menu = 0; bool
last_button_state = HIGH; unsigned long last_button_ms
= 0; const unsigned long
BUTTON_DELAY = 200;

// RED LED BLINK bool red_on
= false;

void setup() { Serial.begin(115200);
pinMode(PIN_TRIG, OUTPUT);
pinMode(PIN_ECHO, INPUT);
pinMode(PIN_FEEDBACK, INPUT);
pinMode(PIN_PWM_OUT, OUTPUT);
pinMode(PIN_MOTOR, OUTPUT);
pinMode(LED_G, OUTPUT); pinMode(LED_Y,
```

```

OUTPUT); pinMode(LED_R, OUTPUT);
pinMode(PIN_BUTTON,
INPUT_PULLUP);

// Configure Timer1 for 10-bit Fast PWM (for PID output)
TCCR1A = (1 << WGM11) | (1 << WGM10) | (1 << COM1A1);
TCCR1B = (1 << WGM12) | (1 << CS11); // prescaler = 8  lcd.init(); lcd.backlight();
lcd.setCursor(0, 0); lcd.print("PID Control"); lcd.setCursor(0, 1); lcd.print("Temp + Motor");
delay(1200); lcd.clear();

// CSV Header for Tera Term
Serial.println("Time_ms,Uptime,Distance_cm,Temp_C,MotorDuty_pct,Setpoint_V,PID_Output_V");

displayMenu(current_menu);
}

void loop() { unsigned long now
= millis();

handleButton(now); if (now - last_ultra_ms >= ULTRA_INTERVAL) { last_ultra_ms = now;
measureDistance(); } if (now - last_temp_ms >= TEMP_INTERVAL) { last_temp_ms = now;
readTemperature(); } if (now - last_pid_ms >= (unsigned long)(LOOP_DT * 1000)) {
last_pid_ms = now; runPID(); } if (now - last_lcd_ms >= LCD_REFRESH) { last_lcd_ms = now;
displayMenu(current_menu); } if (now - last_log_ms >= LOG_DELAY) { last_log_ms = now;
logToTeraTerm(now); }

void handleButton(unsigned long now) { bool state = digitalRead(PIN_BUTTON); if (state ==
LOW && last_button_state == HIGH && (now - last_button_ms > BUTTON_DELAY)) {
current_menu = (current_menu + 1) % 5;
}

```

```

lcd.clear();
displayMenu(current_menu);
last_button_ms = now;
}
last_button_state = state;
}

void measureDistance() { digitalWrite(PIN_TRIGGER, LOW); delayMicroseconds(2);
digitalWrite(PIN_TRIGGER, HIGH);
delayMicroseconds(10); digitalWrite(PIN_TRIGGER, LOW);

long echo_time = pulseIn(PIN_ECHO, HIGH); dist_cm
= (echo_time * 0.0343) / 2.0; dist_cm
= constrain(dist_cm, 0.0, 30.0);

volt_ref = (dist_cm / 30.0) * 3.0; // 0–30 cm → 0–3 V int
pwm_val = map(dist_cm, 0, 30, 0, 1023);
motor_pwm_percent = (pwm_val / 1023.0) * 100.0;
analogWrite(PIN_MOTOR, pwm_val);
}

void runPID() { int adc_in =
analogRead(PIN_FEEDBACK);
float set_adc = (volt_ref / 5.0) * 1023.0;
float error = set_adc - adc_in;

float p_term = KP_GAIN * error; sum_error += KI_GAIN
* LOOP_DT * error; sum_error = constrain(sum_error, 0.0,
1023.0); float d_term = KD_GAIN * (error - error_prev) /
LOOP_DT;

```

```

float pid_raw = p_term + sum_error + d_term;  pid_raw
= constrain(pid_raw, 0.0, 1023.0);  OCR1A =
(int)pid_raw;  pid_voltage = (pid_raw / 1023.0) * 5.0;
error_prev = error;
}

void readTemperature() {  int sensor_val =
analogRead(PIN_TMP36);  float v_out = sensor_val
* (5.0 / 1023.0);  temp_c =
(v_out - 0.5) * 100.0;  temp_f = temp_c *
1.8 + 32.0;

if (temp_c < 28.0) {    digitalWrite(LED_G, HIGH); digitalWrite(LED_Y, LOW);
digitalWrite(LED_R, LOW);
} else if (temp_c < 32.0) {    digitalWrite(LED_G, LOW); digitalWrite(LED_Y, HIGH);
digitalWrite(LED_R, LOW);
} else if (temp_c <= 35.0) {    digitalWrite(LED_G, LOW); digitalWrite(LED_Y,
LOW); digitalWrite(LED_R, HIGH);
} else {    unsigned long now = millis();    if (now
- last_blink_ms >= BLINK_PERIOD) {
last_blink_ms = now;    red_on = !red_on;
digitalWrite(LED_R, red_on);
}
digitalWrite(LED_G, LOW);    digitalWrite(LED_Y,
LOW);
}

void displayMenu(int index) {
lcd.clear();  lcd.backlight();  switch
(index) {    case 0:

```

```
lcd.setCursor(0, 0);
lcd.print("DISTANCE CTRL");
lcd.setCursor(0, 1);
lcd.print("D:");
lcd.print(dist_cm, 1);
lcd.print("cm S:");
lcd.print(volt_ref, 1);
lcd.print("V");    break;  case 1:
lcd.setCursor(0, 0);
lcd.print("TEMPERATURE");
lcd.setCursor(0, 1);
lcd.print(temp_c, 1);
lcd.write(223);  lcd.print("C ");
lcd.print(temp_f, 0);
lcd.write(223);  lcd.print("F");
break;

case 2:
lcd.setCursor(0, 0);
lcd.print("PID GAINS");
lcd.setCursor(0, 1);
lcd.print("Kp:");
lcd.print(KP_GAIN, 2);
lcd.print(" Ki:");
lcd.print(KI_GAIN, 1);  break;

case 3:
lcd.setCursor(0, 0);
lcd.print("DERIVATIVE");
lcd.setCursor(0, 1);
lcd.print("Kd:");
lcd.print(KD_GAIN, 5);  break;

case 4:  lcd.setCursor(0, 0);
lcd.print("MOTOR CTRL");
```

```
lcd.setCursor(0, 1);
lcd.print("PWM:");
lcd.print(motor_pwm_percent, 0);
lcd.print("%");    break;
}

}

void logToTeraTerm(unsigned long now) {
unsigned long secs = now / 1000;
unsigned int hrs = secs / 3600;  unsigned
int mins = (secs % 3600) / 60;  unsigned int
s = secs % 60;

char up[12];  sprintf(up,
"%02u:%02u:%02u", hrs, mins, s);

Serial.print(now);
Serial.print(",");
Serial.print(up);
Serial.print(",");
Serial.print(dist_cm, 1);
Serial.print(",");
Serial.print(temp_c, 1);
Serial.print(",");
Serial.print(motor_pwm_percent, 1);
Serial.print(",");
Serial.print(volt_ref, 2);
Serial.print(",");
Serial.println(pid_voltage, 2);
}
```