Middlesex University Dubai

PID Calculation using Arduino

Dr. Judhi Prasetyo, Middlesex University Dubai

The three control terms, Proportional (P), Integral (I), and Derivative (D), are calculated based on the current error and the time change (Δt)

- Proportional Term (P): Calculated by multiplying the proportional gain (K_p) by the current error. This provides an immediate, initial control response.

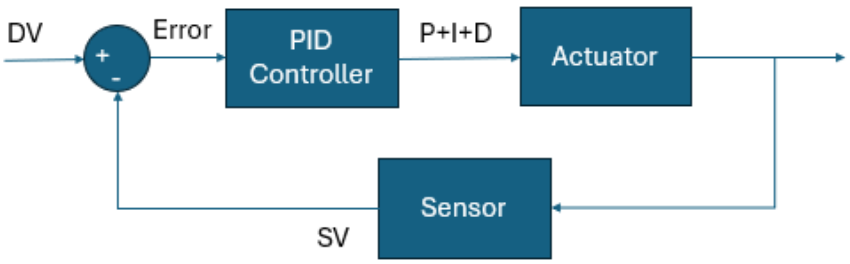
$$P = K_p \, . \, Error$$

- Integral Term (I): The accumulated error is updated by adding K_i multiplied by the current Error and the time interval (Δt). This term eliminates persistent offset (steady-state error).

$$I = I_{previous} + (K_i \, . \, Error \, . \, \Delta t)$$

- Derivative Term (D): Calculated by multiplying the derivative gain (K_d) by the rate of change of the error (change in error divided by Δt). This dampens oscillations and reacts to sudden temperature changes

$$D = Kd \, . \, (Error - Error_{previous}) / \Delta t$$



Arduino implementation:

At a certain period of time (dt) we need to read the sensor value (SV) and compare it with the desired value (DV). The Error is simply the difference between DV and SV:

```
Error = SV - DV;
```

Proportial Term:

```
P = Kp * Error;
```

Integral Term:

```
I += Ki * Error * dt;
```

Derivative Term:

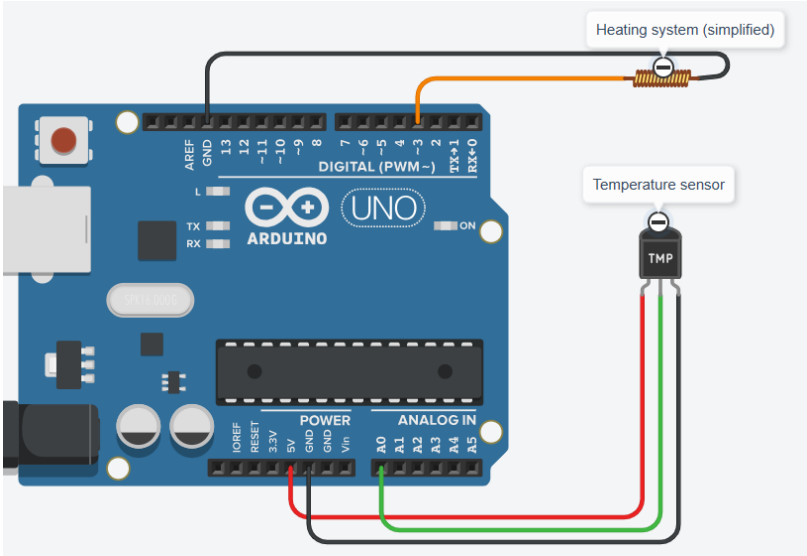
```
D = Kd * (Error - lastError) / dt;
```

Controller output:

```
out = P + I + D;
```

Middlesex University Dubai

Sample code for controlling temperature:




```
const int PIN_INPUT = A0;
const int PIN_OUTPUT = 3;

double Kp = 2.0;
double Ki = 0.005;
double Kd = 1.0;

double Setpoint = 100.0;
double Input;
double Output;

double propError;
double lastPropError = 0;
double integralError = 0;
double diffError;
const double outputLimitMin = 0;
const double outputLimitMax = 255;

unsigned long currentTime;
unsigned long previousTime = 0;
const float SAMPLE_TIME_SEC = 0.1;

double computePID();

void setup() {
    Serial.begin(9600);
    pinMode(PIN_OUTPUT, OUTPUT);

    Input = analogRead(PIN_INPUT);
    previousTime = millis();
    Serial.println("PID Initialized.");
}
```

```
void loop() {
    currentTime = millis();
    if ((currentTime - previousTime) >= (SAMPLE_TIME_SEC * 1000)) {

        Input = analogRead(PIN_INPUT);
        Output = computePID();

        analogWrite(PIN_OUTPUT, (int)Output);

        previousTime = currentTime;

        Serial.print("T:"); Serial.print(Input);
        Serial.print("|E:"); Serial.print(propError);
        Serial.print("|PWM:"); Serial.println(Output);
    }
}


double computePID() {
    double deltaTime = (double)(currentTime - previousTime) / 1000.0;

    propError = Setpoint - Input;

    double pTerm = Kp * propError;

    integralError += propError * deltaTime;

    // Anti-Windup
    if (integralError > outputLimitMax) {
        integralError = outputLimitMax;
    } else if (integralError < outputLimitMin) {
        integralError = outputLimitMin;
    }
    double iTerm = Ki * integralError;

    diffError = (propError - lastPropError) / deltaTime;
    double dTerm = Kd * diffError;

    double output = pTerm + iTerm + dTerm;

    // Clamp Output
    if (output > outputLimitMax) {
        output = outputLimitMax;
    } else if (output < outputLimitMin) {
        output = outputLimitMin;
    }

    lastPropError = propError;

    return output;
}
```