Noto Serif CJK SC

: PID

July 16, 2024

Abstract

ArduinoPIDPythonGUIArduino

1

- **Arduino**
- H
-
-
- **LED**

2

2.1 Arduino

Arduino

-
- **PID**
- Python GUIGUI

```
#define PLOTTER 1
#define REVERSE 1

#define LED 12
#define STBY 7

#if REVERSE
#define N 8
#define P 9
```

```
#else
#define N 9
#define P 8
#endif

#define PWM 10
#define SetPoint A1
#define Sensor A0

#define delay_time 20

float kp = 3;
float ki = 0.5;
float kd = 0.1;
float e_sum = 0;
float e_last = 0;
unsigned long last_time;

const int MIN_PWM = 68;
const int POSITION_THRESHOLD = 5;
const int CONTROL_THRESHOLD = 5;

int set_point = 0;
int prev_set_point = 0;
int prev_pos = 0;
bool serial_control = false;

float pid(float pos_error)
{
  unsigned long now = millis();
  float dt = (now - last_time) / 1000.0;
  last_time = now;
  e_sum += pos_error * dt;
  float e_diff = (pos_error - e_last) / dt;
  float output = kp * pos_error + ki * e_sum + kd * e_diff;
  e_last = pos_error;
  return output;
}

void setup()
{
  Serial.begin(115200);
  while (!Serial);
  pinMode(LED, OUTPUT);
  pinMode(STBY, OUTPUT);
  pinMode(N, OUTPUT);
  pinMode(P, OUTPUT);
  pinMode(PWM, OUTPUT);
  pinMode(SetPoint, INPUT);
  pinMode(Sensor, INPUT);

  last_time = millis();
}

void loop()
{
  if (Serial.available() > 0)
  {
    String input = Serial.readStringUntil(’\n’);
    parseInput(input);
  }

  int curr_pos = analogRead(Sensor);

  if (!serial_control)
  {
    set_point = analogRead(SetPoint);
  }

  int ready = 0;

  if (abs(set_point - curr_pos) < 20)
  {
    ready = 1;
    digitalWrite(LED, 1);
  }
  else
  {
    ready = 0;
    digitalWrite(LED, 0);
  }

  if (PLOTTER)
  {
    Serial.print(float(set_point) / 10.24);
    Serial.print(",");
    Serial.print(float(curr_pos) / 10.24);
    Serial.print(",");
    Serial.print(kp);
    Serial.print(",");
    Serial.print(ki);
    Serial.print(",");
    Serial.print(kd);
    Serial.print(",");
    Serial.print(ready);
    Serial.print("\n");
  }
  else
  {
    Serial.print("Set Point: ");
    Serial.print(set_point / 10.24);
    Serial.print(" Current Position: ");
    Serial.print(curr_pos / 10.24);
    Serial.print(" P: ");
    Serial.print(kp);
    Serial.print(" I: ");
    Serial.print(ki);
    Serial.print(" D: ");
    Serial.print(kd);
    Serial.print(" Ready: ");
    Serial.print(ready);
    Serial.print("\n");
  }

  float pos_error = set_point - curr_pos;
  float control_error = set_point - prev_set_point;
  float control_signal = pid(pos_error);

  control_signal = constrain(control_signal, -255, 255);

  if (control_signal > 40)
  {
    control_signal = max(control_signal, MIN_PWM);
  }
  else if (control_signal < -40)
  {
    control_signal = min(control_signal, -MIN_PWM);
  }

  if (abs(pos_error) < POSITION_THRESHOLD && abs(control_error) < CONTROL_THRESHOLD)
  {
    digitalWrite(STBY, 0);
  }
  else
  {
    digitalWrite(STBY, 1);

    if (control_signal > 0)
```

```
    {
      digitalWrite(N, 0);
      digitalWrite(P, 1);
      analogWrite(PWM, control_signal);
    }
    else
    {
      digitalWrite(N, 1);
      digitalWrite(P, 0);
      analogWrite(PWM, -control_signal);
    }

    prev_set_point = set_point;
    prev_pos = curr_pos;

    delay(delay_time);
  }
}

void parseInput(String input)
{
  input.trim();

  if (input.startsWith("p="))
  {
    kp = input.substring(2).toFloat();
    Serial.print("Updated kp to ");
    Serial.println(kp);
  }
  else if (input.startsWith("i="))
  {
    ki = input.substring(2).toFloat();
    Serial.print("Updated ki to ");
    Serial.println(ki);
  }
  else if (input.startsWith("d="))
  {
    kd = input.substring(2).toFloat();
    Serial.print("Updated kd to ");
    Serial.println(kd);
  }
  else if (input.startsWith("s="))
  {
    int sp = input.substring(2).toInt();
    if (sp == -1)
    {
      serial_control = false;
      Serial.println("Switched to analog control");
    }
    else if (sp >= 0 && sp <= 100)
    {
      set_point = sp * 10.24;
      serial_control = true;
      Serial.print("Updated set point to ");
      Serial.println(set_point);
    }
    else
    {
      Serial.println("Invalid set point value");
    }
  }
  else
  {
    Serial.println("Invalid input");
  }
}
```

2.2 Python GUI

Python GUITkinterPID

```
import tkinter as tk
from tkinter import ttk, scrolledtext
import serial
import threading
import time
import serial.tools.list_ports

class PIDControllerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("PID Controller")

        self.serial_port_manager = SerialPortManager(self)
        self.running = False

        self.create_widgets()
        self.refresh_ports()

    def create_widgets(self):
        self.port_label = tk.Label(self.root, text="Select Port:")
        self.port_label.grid(row=0, column=0)
        self.port_combobox = ttk.Combobox(self.root)
        self.port_combobox.grid(row=0, column=1)

        self.connect_button = tk.Button(self.root, text="Connect", command=self.connect_seri
        self.connect_button.grid(row=0, column=2)

        self.connection_status = tk.Label(self.root, text="Not Connected", bg="grey")
        self.connection_status.grid(row=0, column=3)

        self.kp_label = tk.Label(self.root, text="Kp")
        self.kp_label.grid(row=1, column=0)
        self.kp_scale = tk.Scale(self.root, from_=0, to=10, resolution=0.1, orient=tk.HORIZO
        self.kp_scale.grid(row=1, column=1)
        self.kp_button = tk.Button(self.root, text="Set Kp", command=self.update_kp)
        self.kp_button.grid(row=1, column=2)

        self.ki_label = tk.Label(self.root, text="Ki")
        self.ki_label.grid(row=2, column=0)
        self.ki_scale = tk.Scale(self.root, from_=0, to=10, resolution=0.1, orient=tk.HORIZO
        self.ki_scale.grid(row=2, column=1)
        self.ki_button = tk.Button(self.root, text="Set Ki", command=self.update_ki)
        self.ki_button.grid(row=2, column=2)

        self.kd_label = tk.Label(self.root, text="Kd")
        self.kd_label.grid(row=3, column=0)
        self.kd_scale = tk.Scale(self.root, from_=0, to=10, resolution=0.1, orient=tk.HORIZO
        self.kd_scale.grid(row=3, column=1)
        self.kd_button = tk.Button(self.root, text="Set Kd", command=self.update_kd)
        self.kd_button.grid(row=3, column=2)

        self.setpoint_label = tk.Label(self.root, text="Set Point")
        self.setpoint_label.grid(row=4, column=0)
        self.setpoint_entry = tk.Entry(self.root)
        self.setpoint_entry.grid(row=4, column=1)
        self.setpoint_button = tk.Button(self.root, text="Set Set Point", command=self.updat
        self.setpoint_button.grid(row=4, column=2)

        self.current_position_label = tk.Label(self.root, text="Current Position:")
        self.current_position_label.grid(row=5, column=0)
        self.current_position_value = tk.Label(self.root, text="0%")
        self.current_position_value.grid(row=5, column=1)

        self.setpoint_display_label = tk.Label(self.root, text="Set Point Display:")
        self.setpoint_display_label.grid(row=6, column=0)
        self.setpoint_display_value = tk.Label(self.root, text="0%")
```

```
        self.setpoint_display_value.grid(row=6, column=1)

        self.kp_display_label = tk.Label(self.root, text="Kp Display:")
        self.kp_display_label.grid(row=7, column=0)
        self.kp_display_value = tk.Label(self.root, text="0")
        self.kp_display_value.grid(row=7, column=1)

        self.ki_display_label = tk.Label(self.root, text="Ki Display:")
        self.ki_display_label.grid(row=8, column=0)
        self.ki_display_value = tk.Label(self.root, text="0")
        self.ki_display_value.grid(row=8, column=1)

        self.kd_display_label = tk.Label(self.root, text="Kd Display:")
        self.kd_display_label.grid(row=9, column=0)
        self.kd_display_value = tk.Label(self.root, text="0")
        self.kd_display_value.grid(row=9, column=1)

        self.led_label = tk.Label(self.root, text="LED Status:")
        self.led_label.grid(row=10, column=0)
        self.led_status = tk.Label(self.root, text="TUNING", bg="grey")
        self.led_status.grid(row=10, column=1)

        self.canvas = tk.Canvas(self.root, width=200, height=200, bg="white")
        self.canvas.grid(row=0, column=4, rowspan=11)
        self.arc = self.canvas.create_arc(50, 50, 150, 150, start=90, extent=0, outline="blu
        self.set_point_arc = self.canvas.create_arc(50, 50, 150, 150, start=90, extent=0, ou

        # Add scrolled text box for serial output
        self.text_box = scrolledtext.ScrolledText(self.root, width=50, height=10, state='dis
        self.text_box.grid(row=11, column=0, columnspan=5, padx=10, pady=10)

    def refresh_ports(self):
        ports = self.get_serial_ports()
        self.port_combobox['values'] = ports
        self.root.after(1000, self.refresh_ports)  # 1

    def get_serial_ports(self):
        ports = serial.tools.list_ports.comports()
        return [port.device for port in ports]

    def connect_serial(self):
        if self.serial_port_manager.is_running:
            self.disconnect_serial()
        else:
            selected_port = self.port_combobox.get()
            if selected_port:
```

```
            try:
                self.serial_port_manager.set_name(selected_port)
                self.serial_port_manager.set_baud(115200)
                self.serial_port_manager.start()
                self.connect_button.config(text="Disconnect")
                self.connection_status.config(text="Connected", bg="green")
                self.running = True
                self.recursive_update_textbox()
                self.read_initial_data()
            except serial.SerialException:
                self.connection_status.config(text="Connection Failed", bg="red")
        else:
            self.connection_status.config(text="No Port Selected", bg="red")

def disconnect_serial(self):
    self.serial_port_manager.stop()
    self.connect_button.config(text="Connect")
    self.connection_status.config(text="Not Connected", bg="grey")
    self.running = False

def update_kp(self):
    if self.serial_port_manager.is_running:
        value = self.kp_scale.get()
        command = f"p={value}\n"
        print(f"Sending command: {command}")
        self.serial_port_manager.write(command.encode())

def update_ki(self):
    if self.serial_port_manager.is_running:
        value = self.ki_scale.get()
        command = f"i={value}\n"
        print(f"Sending command: {command}")
        self.serial_port_manager.write(command.encode())

def update_kd(self):
    if self.serial_port_manager.is_running:
        value = self.kd_scale.get()
        command = f"d={value}\n"
        print(f"Sending command: {command}")
        self.serial_port_manager.write(command.encode())

def update_setpoint(self):
    if self.serial_port_manager.is_running:
        value = self.setpoint_entry.get()
        try:
            setpoint = float(value)
```

```
                if 0 <= setpoint <= 100:
                    scaled_value = int(setpoint)
                    command = f"s={scaled_value}\n"
                    print(f"Sending command: {command}")
                    self.serial_port_manager.write(command.encode())
                else:
                    print("Setpoint out of range (0-100)")
            except ValueError:
                print("Invalid setpoint value")

    def read_initial_data(self):
        line = self.serial_port_manager.read_line().decode('utf-8').strip()
        if line:
            print(f"Initial data: {line}")  # Debug print
            data = line.split(",")
            if len(data) == 6:
                set_point, curr_pos, kp, ki, kd, ready = data
                self.setpoint_display_value.config(text=f"{set_point}%")
                self.current_position_value.config(text=f"{curr_pos}%")
                self.kp_display_value.config(text=kp)
                self.ki_display_value.config(text=ki)
                self.kd_display_value.config(text=kd)
                if ready == "1":
                    self.led_status.config(text="READY", bg="green")
                else:
                    self.led_status.config(text="TUNING", bg="yellow")
                self.update_arc(set_point, curr_pos)

    def update_arc(self, set_point, curr_pos):
        set_point_extent = (float(set_point) / 100) * 360
        self.canvas.itemconfig(self.set_point_arc, extent=set_point_extent)

        current_pos_extent = (float(curr_pos) / 100) * 360
        self.canvas.itemconfig(self.arc, extent=current_pos_extent)

    def recursive_update_textbox(self):
        serial_port_buffer = self.serial_port_manager.read_buffer()
        if serial_port_buffer:
            self.text_box.config(state='normal')
            self.text_box.insert(tk.END, serial_port_buffer.decode("ascii"))
            self.text_box.see(tk.END)
            self.text_box.config(state='disabled')
        if self.serial_port_manager.is_running:
            self.root.after(100, self.recursive_update_textbox)

    def on_closing(self):
        self.running = False
        if self.serial_port_manager.is_running:
            self.serial_port_manager.stop()
        self.root.destroy()


class SerialPortManager:
    def __init__(self, app):
        self.is_running = False
        self.serial_port_name = None
        self.serial_port_baud = 9600
        self.serial_port = serial.Serial()
        self.serial_port_buffer = bytearray()
        self.line_buffer = ""
        self.app = app

    def set_name(self, serial_port_name):
        self.serial_port_name = serial_port_name

    def set_baud(self, serial_port_baud):
        self.serial_port_baud = serial_port_baud

    def start(self):
        self.is_running = True
        self.serial_port_thread = threading.Thread(target=self.thread_handler)
        self.serial_port_thread.start()

    def stop(self):
        self.is_running = False

    def thread_handler(self):
        while self.is_running:
            try:
                if not self.serial_port.isOpen():
                    self.serial_port = serial.Serial(
                        port=self.serial_port_name,
                        baudrate=self.serial_port_baud,
                        bytesize=8,
                        timeout=2,
                        stopbits=serial.STOPBITS_ONE,
                    )
                else:
                    while self.serial_port.in_waiting > 0:
                        serial_port_byte = self.serial_port.read(1)
                        self.serial_port_buffer.append(int.from_bytes(serial_port_byte, byte
                        self.line_buffer += serial_port_byte.decode('utf-8')
                        if '\n' in self.line_buffer:
                            lines = self.line_buffer.split('\n')
                            for line in lines[:-1]:
                                self.update_app(line.strip())
                            self.line_buffer = lines[-1]
            except serial.SerialException as e:
                print(f"Serial error: {e}")
                self.app.disconnect_serial()

        if self.serial_port.isOpen():
            self.serial_port.close()

    def update_app(self, data_line):
        if data_line:
            print(f"Update app with data: {data_line}")  # Debug print
            data = data_line.split(",")
            if len(data) == 6:
                set_point, curr_pos, kp, ki, kd, ready = data
                self.app.setpoint_display_value.config(text=f"{set_point}%")
                self.app.current_position_value.config(text=f"{curr_pos}%")
                self.app.kp_display_value.config(text=kp)
                self.app.ki_display_value.config(text=ki)
                self.app.kd_display_value.config(text=kd)
                if ready == "1":
                    self.app.led_status.config(text="READY", bg="green")
                else:
                    self.app.led_status.config(text="TUNING", bg="yellow")
                self.app.update_arc(set_point, curr_pos)

    def read_buffer(self):
        buffer = self.serial_port_buffer
        self.serial_port_buffer = bytearray()
        return buffer

    def write(self, data):
        if self.serial_port.isOpen():
            self.serial_port.write(data)

    def read_line(self):
        if self.serial_port.isOpen():
            return self.serial_port.readline()
        return b''

    def main_process(self, input_byte):
        try:
            character = input_byte.decode("ascii")
        except UnicodeDecodeError:
            pass
        else:
            print(character, end="")


if __name__ == "__main__":
    root = tk.Tk()
    app = PIDControllerApp(root)
    root.protocol("WM_DELETE_WINDOW", app.on_closing)
    root.mainloop()
```

3

Python GUIArduinoPID

4

ArduinoPython GUIPIDPID

5

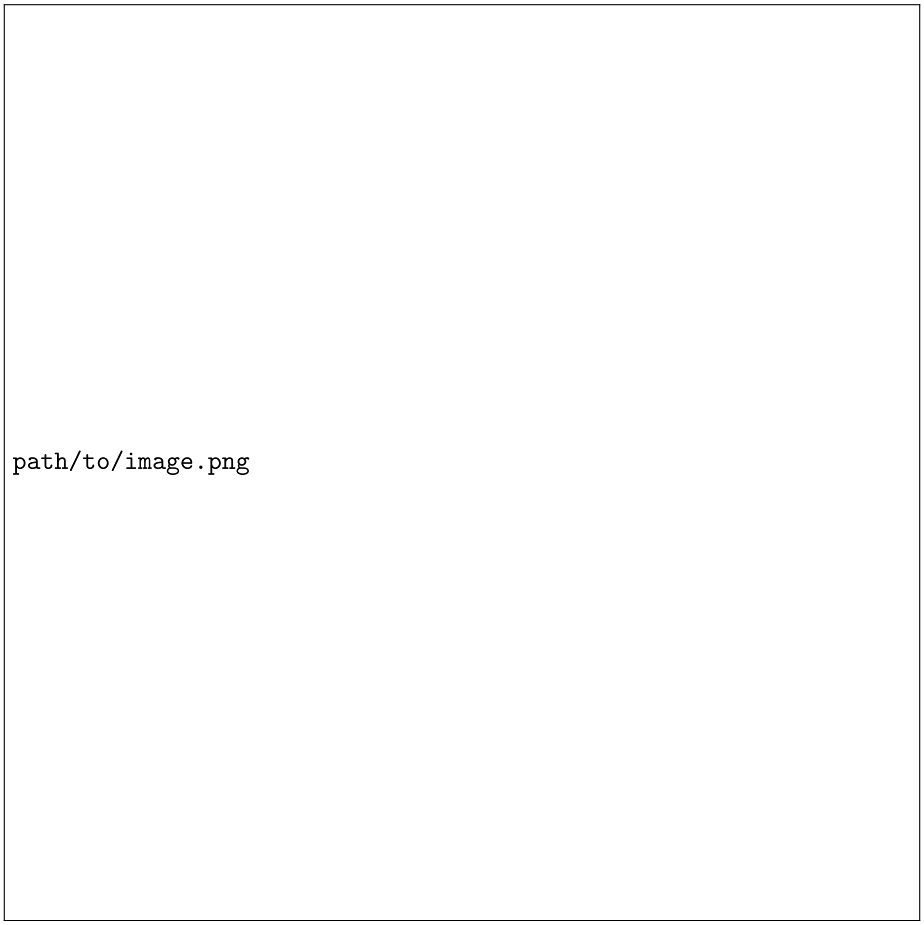

Figure 1:

path/to/image.png

Figure 2: