
**Home Project #9**

**PID control of DC motor**

Victor Mendoza

ID: 6348908

EML480 Introduction to Mechatronics

Date of Submission: 07-12-2025

Fall, 2025

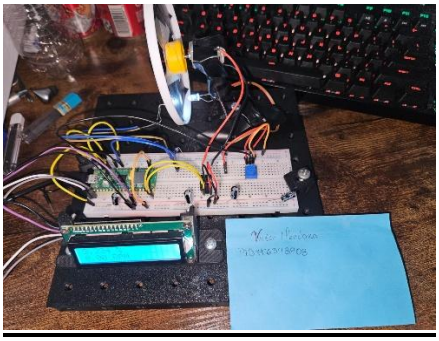
**Introduction:**

This project extends the previous DC motor speed-measurement system by integrating a PID control algorithm to regulate motor RPM using the Raspberry Pi Pico. The potentiometer provides the setpoint within a 0–1200 RPM range, while the PID controller adjusts the PWM output to maintain the desired speed. The system displays both the setpoint and process variable on an LCD, and the serial output is used to visualize tuning results. By tuning the proportional, integral, and derivative coefficients, the project aims to achieve smooth and stable RPM regulation with minimal overshoot and oscillation.

**Materials used:**

1. PICO board
2. Breadboard
3. Small DC motor
4. LED
5. 2x 220 Ohm resistor
6. 10k Ohm's resistor
7. 3x 10µF capacitors
8. TA6586 – Motor Driver Chip
9. LCD
10. Improvised Mechanical Assembly
11. Wires

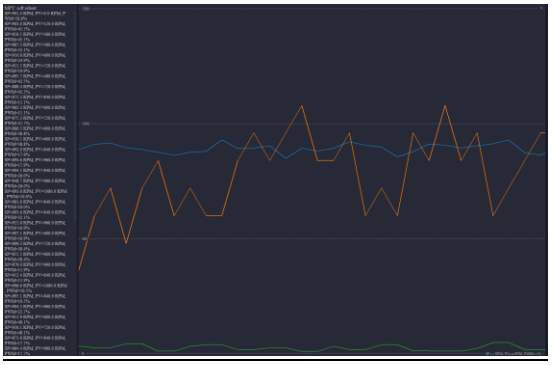
**Picture:**



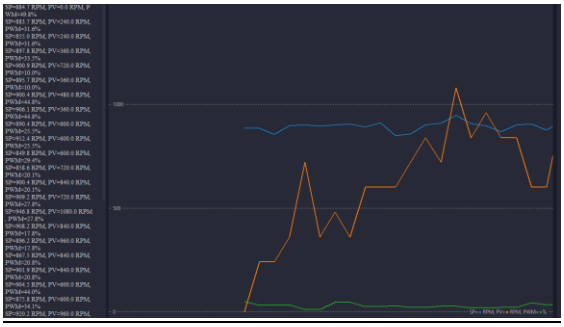
P = 1, I = 0, D = 0

```
MPY: soft reboot
SP=517.8 RPM, PV=30.0 RPM, PWM=100.0%
SP=548.1 RPM, PV=720.0 RPM, PWM=0.0%
SP=534.5 RPM, PV=600.0 RPM, PWM=0.0%
SP=522.0 RPM, PV=450.0 RPM, PWM=72.0%
SP=498.5 RPM, PV=780.0 RPM, PWM=0.0%
SP=540.3 RPM, PV=660.0 RPM, PWM=0.0%
SP=542.3 RPM, PV=330.0 RPM, PWM=100.0%
SP=512.6 RPM, PV=960.0 RPM, PWM=0.0%
SP=518.3 RPM, PV=840.0 RPM, PWM=0.0%
SP=545.0 RPM, PV=690.0 RPM, PWM=0.0%
SP=518.3 RPM, PV=420.0 RPM, PWM=98.3%
SP=566.9 RPM, PV=690.0 RPM, PWM=0.0%
SP=518.3 RPM, PV=780.0 RPM, PWM=0.0%
```

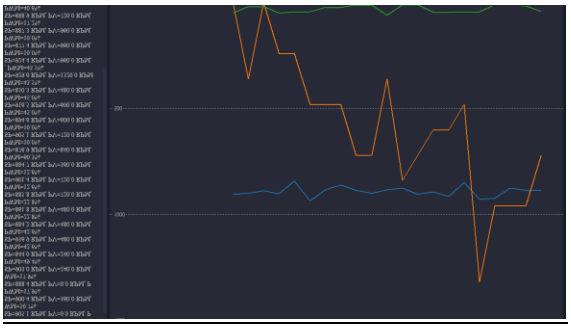
P = 0.045, I = 0.005, D = 0.03



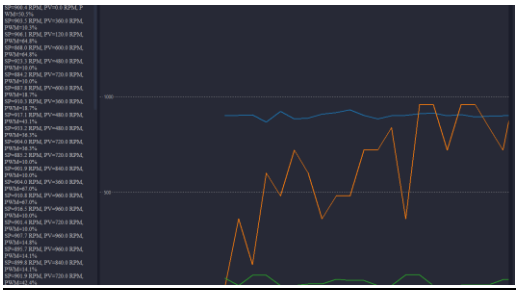
P = 0.045, I = 0.0035, D = 0.04



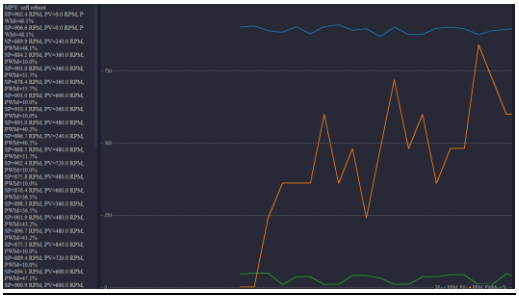
P = 0.045, I = 0.003, D = 0.05



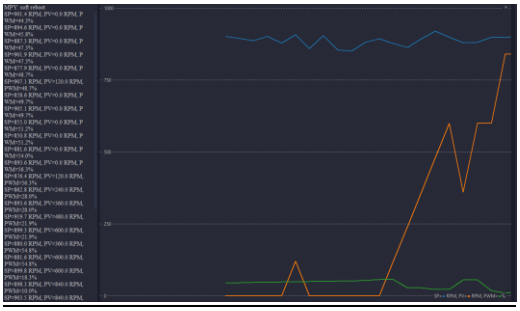
P = 0.045, I = 0.002, D = 0.07



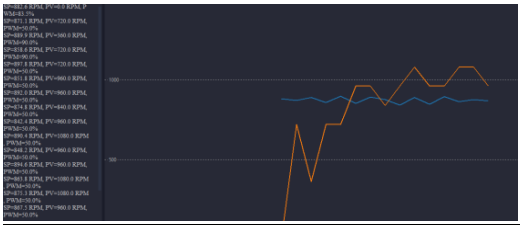
P = 0.040, I = 0.002, D = 0.07



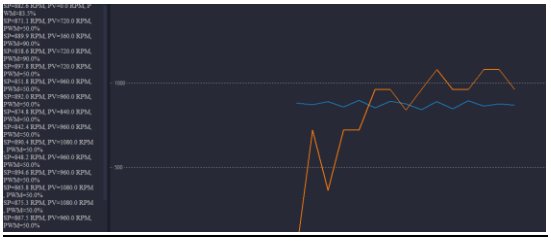
P = 0.038, I = 0.002, D = 0.08



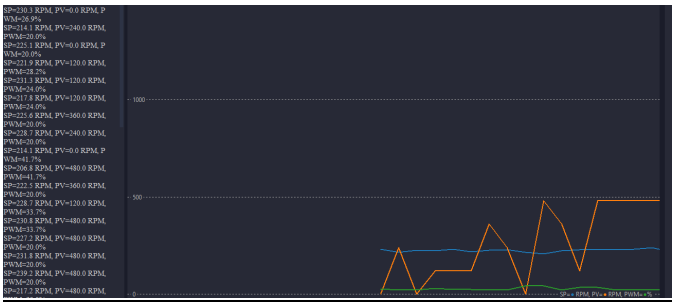
P = 0.045, I = 0.005, D = 0.03



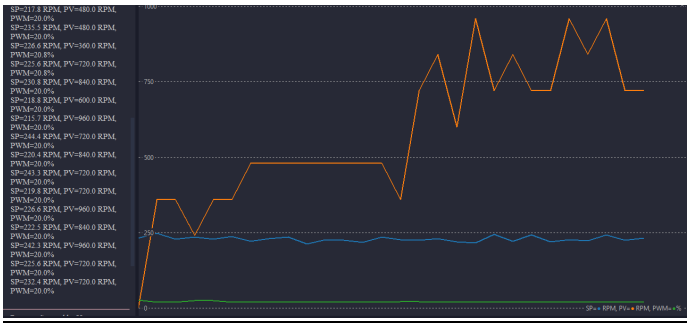
P = 0.038, I = 0.002, D = 0.08



P = 0.03, I = 0.003, D = 0.04



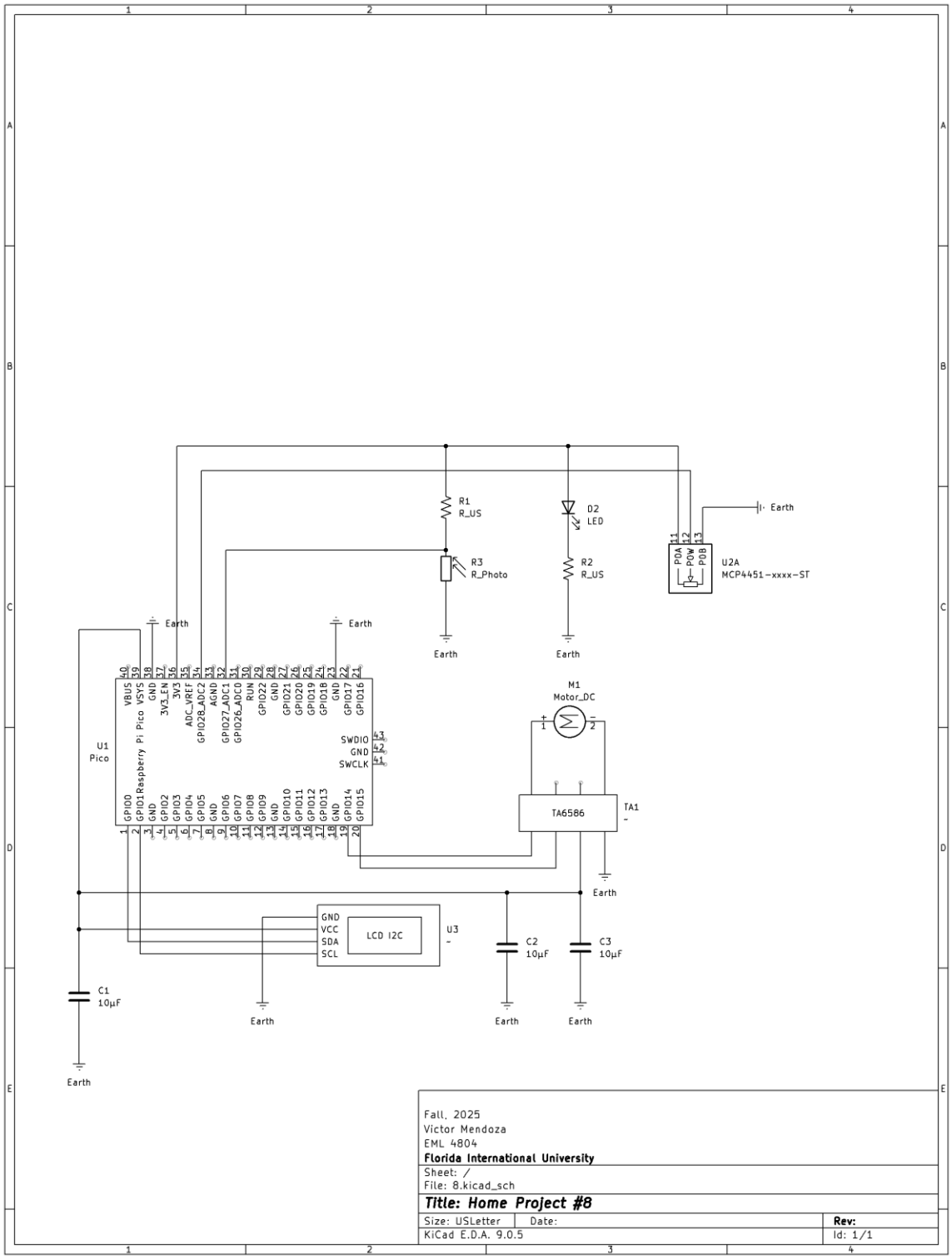
P = 0.03, I = 0.002, D = 0.04



**Video link:**

https://youtube.com/shorts/RkDiSv1iGQ8

**Diagram:**


R1
R_US
D2
LED
Earth
R3
R_Photo
R2
R_US
U2A
MCP4451-xxxx-ST
P0A
P0W
P0B
Earth
Earth
Earth
U1
Pico
Raspberry Pi Pico
VBUS
VSYS
GND
3V3_EN
3V3
ADC_VREF
GPIO28_ADC2
AGND
GPIO27_ADC1
GPIO26_ADC0
RUN
GPIO22
GND
GPIO21
GPIO20
GPIO19
GPIO18
GND
GPIO17
GPIO16
SWDIO
GND
SWCLK
GPIO0
GPIO1
GND
GPIO2
GPIO3
GPIO4
GPIO5
GND
GPIO6
GPIO7
GPIO8
GPIO9
GND
GPIO10
GPIO11
GPIO12
GPIO13
GND
GPIO14
GPIO15
M1
Motor_DC
TA6586
TA1
Earth
GND
VCC
SDA
SCL
LCD I2C
U3
C1
10µF
C2
10µF
C3
10µF
Earth
Earth
Earth
Earth
Fall, 2025
Victor Mendoza
EML 4804
Florida International University
Sheet: /
File: 8.kicad_sch
Title: Home Project #8
Size: USLetter
Date:
Rev:
KiCad E.D.A. 9.0.5
Id: 1/1

**Code:**

```
from machine import Pin, PWM
import machine
import time
import utime
from lcd1602 import LCD
from pid import PID

# ---------- Hardware setup ----------
icPin1 = Pin(14, Pin.OUT)   # Forward PWM pin
icPin2 = Pin(15, Pin.OUT)   # Reverse PWM pin

fPWM = PWM(icPin1)
rPWM = PWM(icPin2)
fPWM.freq(1000)
rPWM.freq(1000)

MAX_DUTY = 65535

# Potentiometer and sensor for RPM
analog_pot = machine.ADC(28)   # Potentiometer
analog_sensor = machine.ADC(27)  # Photosensor

# LCD
lcd = LCD()
lcd.clear()

# ---------- PID setup ----------
# Start with only P, then add I and D following your tuning procedure
P = 0.05
I = 0.005
D = 0.03

pid = PID(
    Kp=P,
    Ki=I,
    Kd=D,
    setpoint=0.0,
    sample_time=0.25,
    output_limits=(10, 100)  # PID output = 0–100 % duty
```

```
)

# Pot range 1200–3500 (12-bit) scaled for read_u16 (0–65535)
POT_MIN = 19200      # 19200
POT_MAX = 56000    # 56000
SP_MAX_RPM = 1200

def get_setpoint_rpm():
    raw = analog_pot.read_u16()

    # Clamp potentiometer reading
    if raw < POT_MIN:
        raw = POT_MIN
    elif raw > POT_MAX:
        raw = POT_MAX

    span = POT_MIN if POT_MAX == POT_MIN else (POT_MAX - POT_MIN)
    sp = (raw - POT_MIN) * SP_MAX_RPM / span
    return sp, raw

# ---------- RPM measurement variables ----------
pulse_count = 0
prev_v = 0.0
rpm = 0.0

# Number of slots / pulses per revolution in your sensor disc
SLOTS_PER_REV = 2

# Time window for RPM computation (seconds)
WINDOW = 0.25
t_acc = 0.0        # accumulated time for RPM window

try:
    while True:
        # ----- Read setpoint from potentiometer -----
        setpoint_rpm, pot_raw = get_setpoint_rpm()
        pid.setpoint = setpoint_rpm

        # ----- Read sensor and detect pulses -----
        z = analog_sensor.read_u16()
```

```
v = (3.3 / 65535) * z   # Convert to voltage

# Simple falling-edge detection with threshold
if (prev_v - v) > 0.10:
    pulse_count += 1

prev_v = v

# ----- Timing / RPM window -----
dt = 0.01
t_acc += dt
time.sleep(dt)

if t_acc >= WINDOW:
    # Compute RPM from pulses in the window
    rpm = pulse_count * 60.0 / (SLOTS_PER_REV * WINDOW)

    # Reset counters for next window
    pulse_count = 0
    t_acc = 0.0

    # ----- PID control -----
    output = pid(rpm)  # process variable = measured RPM
    if output is None:
        output = 0.0

    pwm_percent = output              # 0–100 %
    duty = int(pwm_percent / 100 * MAX_DUTY)

    # Drive motor forward with PID output
    rPWM.duty_u16(0)
    fPWM.duty_u16(duty)

    # ----- LCD display: SP and PV -----
    lcd.write(0, 0, " " * 16)
    lcd.write(0, 1, " " * 16)

    line0 = "SP:{:5.1f} RPM".format(setpoint_rpm)
    line1 = "PV:{:5.1f} RPM".format(rpm)
```

```
            lcd.write(0, 0, line0[:16])
            lcd.write(0, 1, line1[:16])

            # ----- Serial monitor: SP, PV, PWM -----
            print("SP={:.1f} RPM, PV={:.1f} RPM, PWM={:.1f}%".format(
                setpoint_rpm, rpm, pwm_percent
            ))

except KeyboardInterrupt:
    print("Program Stopped by User")
    rPWM.duty_u16(0)
    fPWM.duty_u16(0)
    time.sleep(0.15)
    rPWM.deinit()
    fPWM.deinit()
    icPin1.off()
    icPin2.off()
```

**Conclusions:**

The PID controller was successfully implemented to regulate the DC motor's speed based on the potentiometer-defined setpoint. The serial plots and LCD readings confirmed that the system accurately tracked changes in the setpoint, and the PWM output adjusted in real time to compensate for speed variations. Through systematic tuning of the P, I, and D coefficients, the motor achieved stable operation with reduced overshoot and improved response time. The final configuration provided smooth RPM stabilization across the full operating range, demonstrating effective closed-loop control and reliable integration of the PID library with the motor driver system.

**References:**


- Raspberri Pi Pico Library: https://github.com/ncarandini/KiCad-RP-Pico/tree/main
- How to use a breadboard https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/all
- Raspberry Pi Pico usage https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html
- LCD wiring instructions https://docs.sunfounder.com/projects/thales-kit/en/latest/micropython/liquid_crystal_display.html

- Using I/O with MicroPython on the Pi Pico
  https://www.upesy.com/blogs/tutorials/micropython-raspberry-pi-pico-gpio-pins-usage#google_vignette
- Measure an analog voltage with the Pi Pico ADC in MicroPython https://www.upesy.com/blogs/tutorials/micropython-raspberry-pi-pico-adc-usage-measure-voltage
- TA6586 Motor Driver Chip https://docs.sunfounder.com/projects/pico-2w-kit/en/latest/component/component_ta6585.html
- Control DC Motor with Raspberry Pi Pico W and TA6586 https://toptechboy.com/control-dc-motor-with-raspberry-pi-pico-w-and-ta6586/