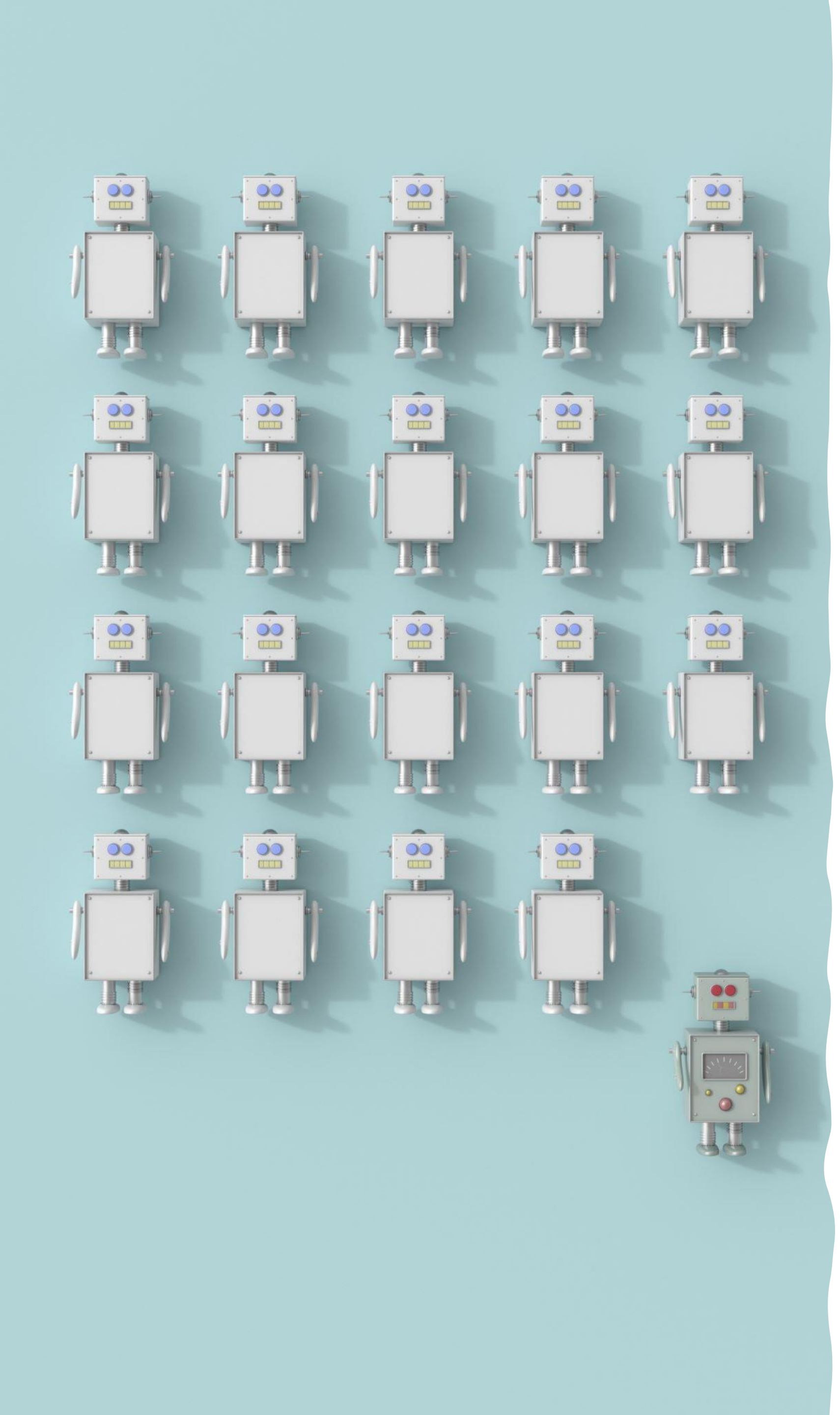
Wall Following robot

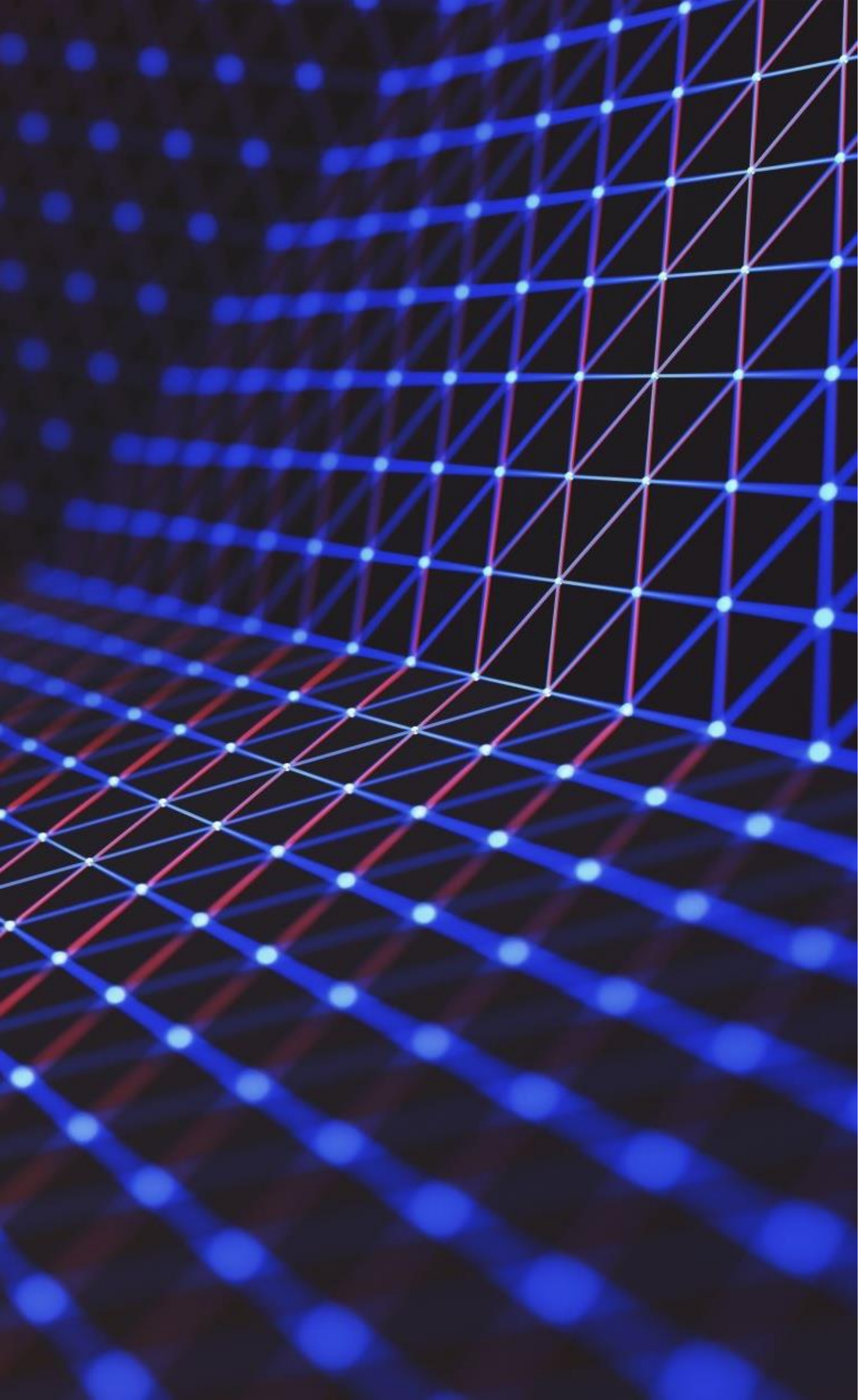
Aodan Hardy

Intelligent robotics



The task

- Implement a wall-following robot using an E-puck robot in Webots
- Robot must follow wall with max distance of 0.2m
- It must navigate corners and complete one full lap of the maze
- Uses proximity sensors to detect and follow the wall



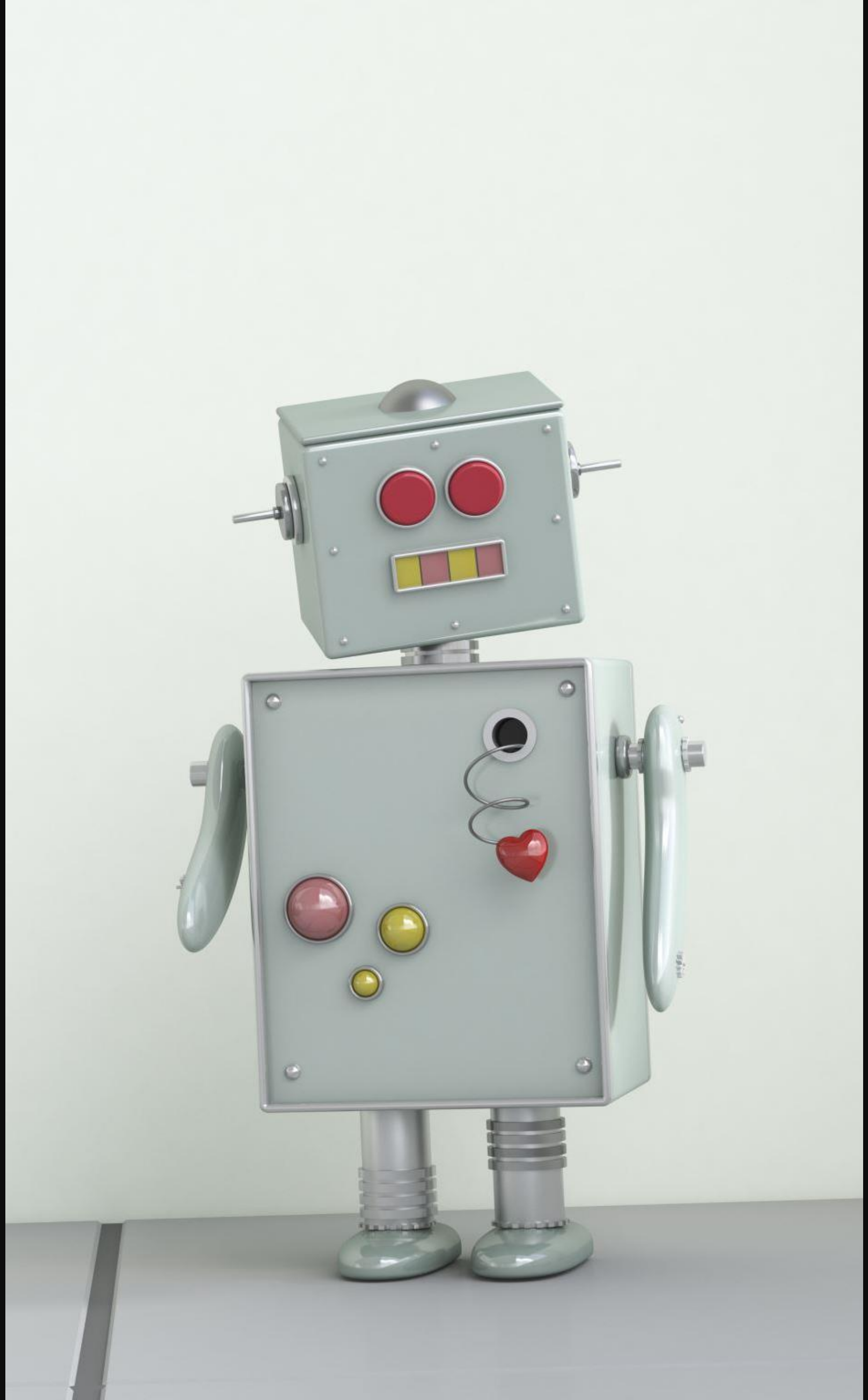
My Approach

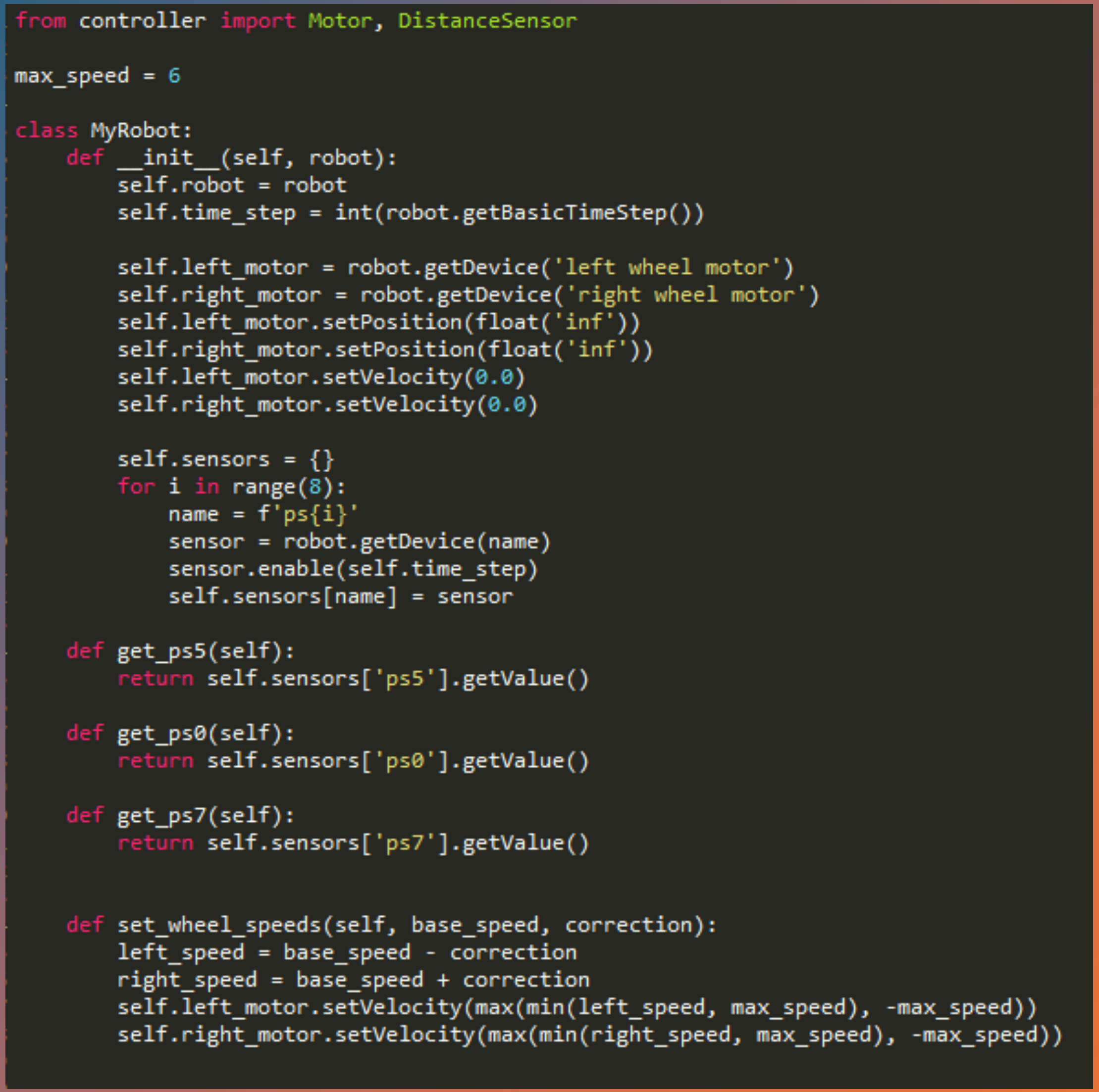
- Robot first searches for and aligns with a nearby wall
- Follows the wall while staying parallel to it
- Uses a PID controller to maintain correct distance
- Monitors sensors to detect:
- Right-hand turns (wall appears in front)
- Left-hand turns (wall on left disappears)

Implentation

In order to implement this, I use three python files

- My_robot.py
- Pid.py
- Wall_follower.py





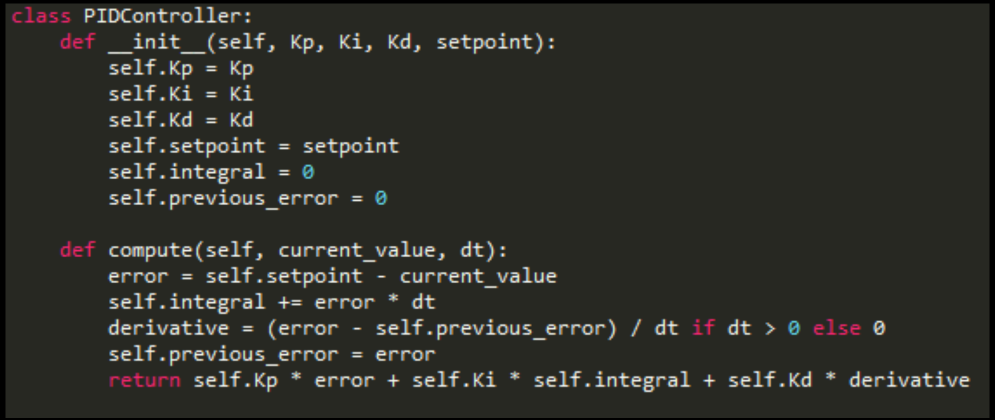
```python
from controller import Motor, DistanceSensor

max_speed = 6

class MyRobot:
    def __init__(self, robot):
        self.robot = robot
        self.time_step = int(robot.getBasicTimeStep())

        self.left_motor = robot.getDevice('left wheel motor')
        self.right_motor = robot.getDevice('right wheel motor')
        self.left_motor.setPosition(float('inf'))
        self.right_motor.setPosition(float('inf'))
        self.left_motor.setVelocity(0.0)
        self.right_motor.setVelocity(0.0)

        self.sensors = {}
        for i in range(8):
            name = f'ps{i}'
            sensor = robot.getDevice(name)
            sensor.enable(self.time_step)
            self.sensors[name] = sensor

    def get_ps5(self):
        return self.sensors['ps5'].getValue()

    def get_ps0(self):
        return self.sensors['ps0'].getValue()

    def get_ps7(self):
        return self.sensors['ps7'].getValue()


    def set_wheel_speeds(self, base_speed, correction):
        left_speed = base_speed - correction
        right_speed = base_speed + correction
        self.left_motor.setVelocity(max(min(left_speed, max_speed), -max_speed))
        self.right_motor.setVelocity(max(min(right_speed, max_speed), -max_speed))
```

My_robot.py

- Sets up the Webots robot and its proximity sensors
- Handles reading sensor values each timestep
- Provides access to wheel motors and sensor data for other modules

pid.py

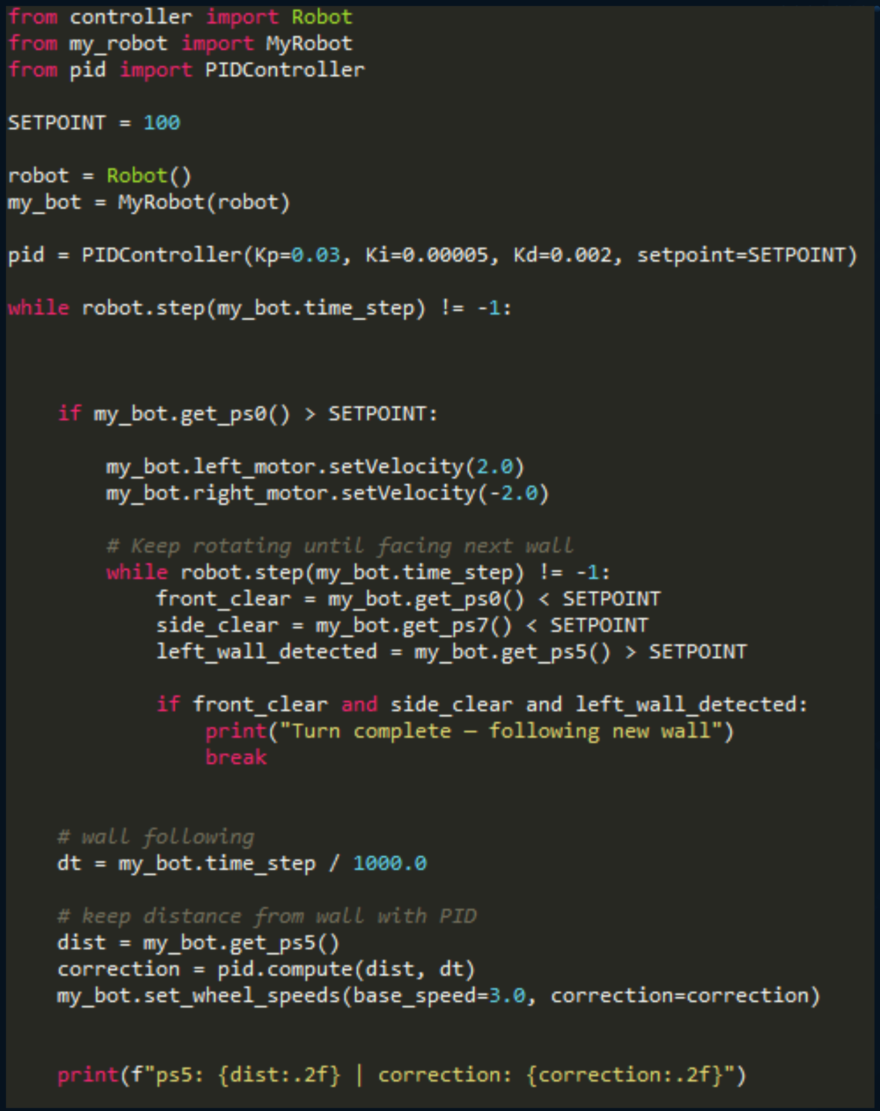
- Takes in the wall-following error and returns correction speed
- Used to keep the robot at the correct distance from the wall



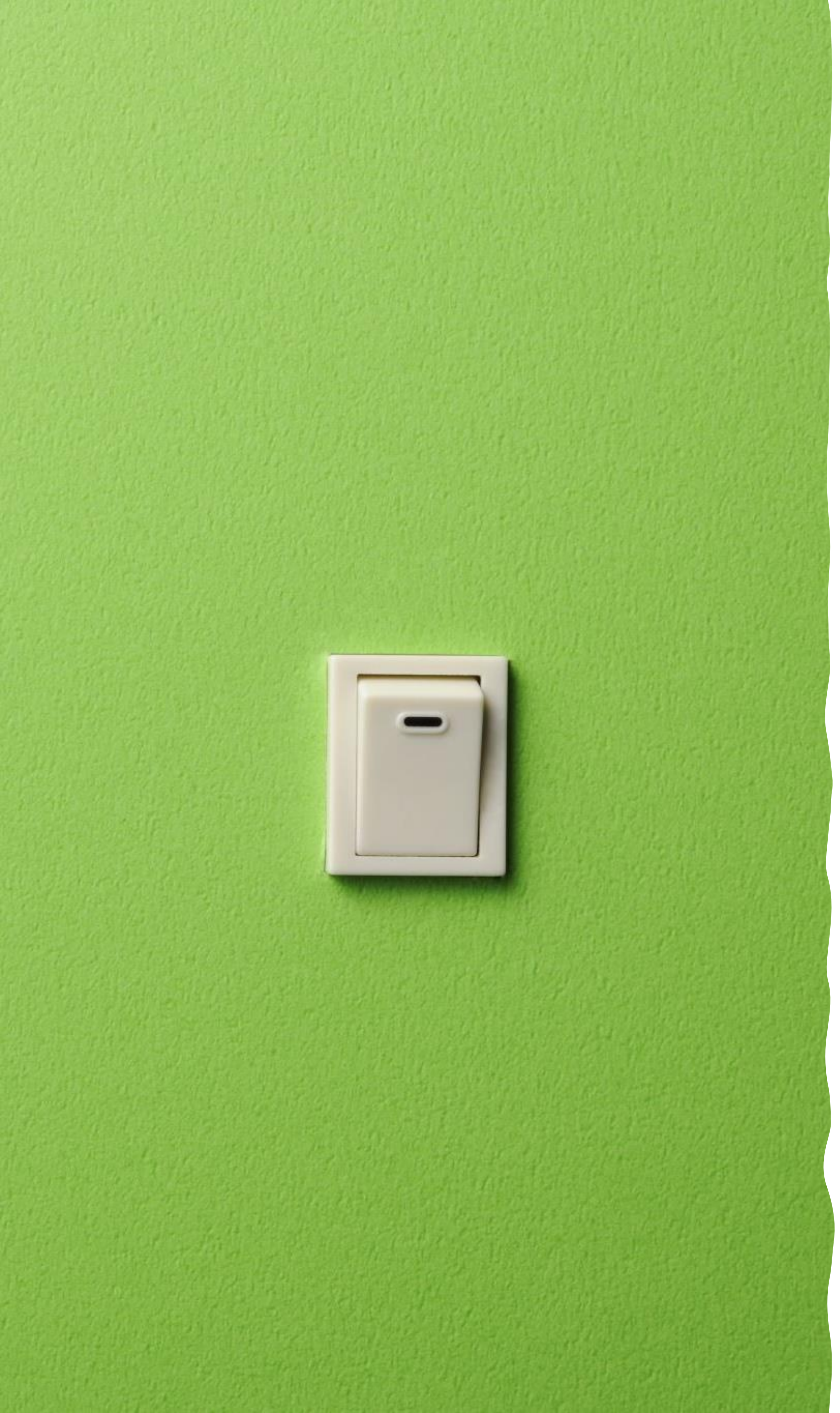
```python
class PIDController:
    def __init__(self, Kp, Ki, Kd, setpoint):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.setpoint = setpoint
        self.integral = 0
        self.previous_error = 0

    def compute(self, current_value, dt):
        error = self.setpoint - current_value
        self.integral += error * dt
        derivative = (error - self.previous_error) / dt if dt > 0 else 0
        self.previous_error = error
        return self.Kp * error + self.Ki * self.integral + self.Kd * derivative
```

Wall_follower.py

- Main control logic for following the wall
- Uses sensor data and PID output to adjust wheel speeds
- Detects corners and turns using left/right/front sensors



```python
from controller import Robot
from my_robot import MyRobot
from pid import PIDController

SETPOINT = 100

robot = Robot()
my_bot = MyRobot(robot)

pid = PIDController(Kp=0.03, Ki=0.00005, Kd=0.002, setpoint=SETPOINT)

while robot.step(my_bot.time_step) != -1:


    if my_bot.get_ps0() > SETPOINT:

        my_bot.left_motor.setVelocity(2.0)
        my_bot.right_motor.setVelocity(-2.0)

        # Keep rotating until facing next wall
        while robot.step(my_bot.time_step) != -1:
            front_clear = my_bot.get_ps0() < SETPOINT
            side_clear = my_bot.get_ps7() < SETPOINT
            left_wall_detected = my_bot.get_ps5() > SETPOINT

            if front_clear and side_clear and left_wall_detected:
                print("Turn complete – following new wall")
                break


    # wall following
    dt = my_bot.time_step / 1000.0

    # keep distance from wall with PID
    dist = my_bot.get_ps5()
    correction = pid.compute(dist, dt)
    my_bot.set_wheel_speeds(base_speed=3.0, correction=correction)


    print(f"ps5: {dist:.2f} | correction: {correction:.2f}")
```



Challenges/ Improvements

- Improve tuning PID controller (wobbly)
- Left hand turns
- Following wall on left or right