#include <iostream>

#include <algorithm> // for std::min, std::max>

#include <chrono>

#include <thread>

class Quadcopter {

private:

int motor1, motor2, motor3, motor4;

// Clamp value between 0 and 255

int clamp(int value) {

return std::min(std::max(value, 0), 255);

}

// PID controller variables for pitch, roll, and yaw (optional)

float pitchKp = 0.6, pitchKi = 0.2, pitchKd = 0.05; // PID constants for pitch

float rollKp = 0.6, rollKi = 0.2, rollKd = 0.05; // PID constants for roll

float yawKp = 0.6, yawKi = 0.2, yawKd = 0.05; // PID constants for yaw

float prevPitchError = 0, integralPitch = 0;

float prevRollError = 0, integralRoll = 0;

float prevYawError = 0, integralYaw = 0;

public:

Quadcopter() : motor1(0), motor2(0), motor3(0), motor4(0) {}

void setup() {

// Simulate setting up the motor pins as outputs

std::cout << "Setting up motor pins as outputs..." << std::endl;

}

void loop() {

// Simulate sensor readings (throttle, roll, pitch, yaw)

int throttle = 500; // Fixed throttle value for simplicity

int roll = 10; // Sensor input for roll

int pitch = -5; // Sensor input for pitch

int yaw = 0; // Sensor input for yaw (no yaw correction here)

// Use PID controllers to stabilize the quadcopter based on errors

int motor1Speed = calculateMotorSpeed(throttle, roll, pitch, yaw, 1);

int motor2Speed = calculateMotorSpeed(throttle, -roll, pitch, yaw, 2);

int motor3Speed = calculateMotorSpeed(throttle, roll, -pitch, yaw, 3);

int motor4Speed = calculateMotorSpeed(throttle, -roll, -pitch, yaw, 4);

// Output the motor speeds

std::cout << "Motor 1 Speed: " << motor1Speed << std::endl;

std::cout << "Motor 2 Speed: " << motor2Speed << std::endl;

std::cout << "Motor 3 Speed: " << motor3Speed << std::endl;

std::cout << "Motor 4 Speed: " << motor4Speed << std::endl;

// Simulate time delay between loop iterations

std::this\_thread::sleep\_for(std::chrono::milliseconds(50)); // 50 ms delay for simulation

}

// Calculate the motor speed with PID adjustments

int calculateMotorSpeed(int throttle, int roll, int pitch, int yaw, int motorID) {

int target = throttle; // Base target speed from throttle

// Calculate PID correction for each axis (roll, pitch, yaw)

if (motorID == 1 || motorID == 3) {

// Front-right or rear-right motor

target += applyPIDCorrection(pitch, roll, yaw, motorID);

}

else if (motorID == 2 || motorID == 4) {

// Front-left or rear-left motor

target += applyPIDCorrection(pitch, roll, yaw, motorID);

}

return clamp(target); // Ensure motor speed stays between 0 and 255

}

// Apply PID corrections for motor speed adjustment

int applyPIDCorrection(int pitch, int roll, int yaw, int motorID) {

float error = 0;

if (motorID == 1 || motorID == 2) {

// Motor 1 and 2 - pitch and roll correction

error = pitch + roll;

} else if (motorID == 3 || motorID == 4) {

// Motor 3 and 4 - pitch and roll correction

error = -pitch - roll;

}

float &prevError = (motorID == 1 || motorID == 2) ? prevPitchError : prevRollError;

float &integral = (motorID == 1 || motorID == 2) ? integralPitch : integralRoll;

float pidOutput = calculatePID(error, prevError, integral, motorID);

// Update previous error and integral for next iteration

prevError = error;

integral += error;

return pidOutput;

}

// Calculate PID correction output

float calculatePID(float error, float prevError, float &integral, int motorID) {

float kp = 0, ki = 0, kd = 0;

// Select appropriate PID constants based on motor ID

if (motorID == 1 || motorID == 2) {

// Pitch or Roll correction

kp = rollKp; ki = rollKi; kd = rollKd;

} else if (motorID == 3 || motorID == 4) {

// Pitch or Roll correction

kp = pitchKp; ki = pitchKi; kd = pitchKd;

}

float derivative = error - prevError;

return kp \* error + ki \* integral + kd \* derivative;

}

};

int main() {

// Initialize quadcopter object

Quadcopter quad;

// Setup the quadcopter motors

quad.setup();

// Run the loop simulating the quadcopter's flight control

while (true) {

quad.loop();

}

return 0;

}