#include <stdint.h>

#include <stdbool.h>

// Constants for motor control

#define MAX\_PWM\_OUTPUT 255

#define MAX\_SPEED 100.0 // Maximum speed in units per second

#define MAX\_ACCELERATION 5.0 // Maximum acceleration in units per second^2

#define MAX\_DECELERATION 5.0 // Maximum deceleration in units per second^2

#define ENCODER\_TICKS\_PER\_REV 1000 // Example encoder ticks per revolution

#define TIMESTAMP\_INTERVAL\_MS 10 // Sample interval for speed calculation

// PID Control Structure

typedef struct {

float Kp;

float Ki;

float Kd;

float set\_point;

float last\_error;

float integral;

} PIDControl;

// Function Prototypes

void PID\_init(PIDControl \*pid, float Kp, float Ki, float Kd);

float PID\_calculate(PIDControl \*pid, float current);

void control\_motor\_x(float output);

void control\_motor\_y(float output);

void control\_motor\_z(float output);

float read\_encoder\_x();

float read\_encoder\_y();

float read\_encoder\_z();

void update\_speed();

void apply\_acceleration\_control(float\* current\_speed, float target\_speed);

// Speed control function prototypes

void set\_pwm\_x(int32\_t output);

void set\_pwm\_y(int32\_t output);

void set\_pwm\_z(int32\_t output);

// Motor and Encoder Variables

volatile int32\_t encoder\_x\_count = 0;

volatile int32\_t encoder\_y\_count = 0;

volatile int32\_t encoder\_z\_count = 0;

// PID Instances for Position and Speed

PIDControl pid\_position\_x, pid\_speed\_x;

PIDControl pid\_position\_y, pid\_speed\_y;

PIDControl pid\_position\_z, pid\_speed\_z;

// Variables for speed tracking

float speed\_x = 0.0;

float speed\_y = 0.0;

float speed\_z = 0.0;

float last\_position\_x = 0.0;

float last\_position\_y = 0.0;

float last\_position\_z = 0.0;

// Target speeds

float target\_speed\_x = 0.0;

float target\_speed\_y = 0.0;

float target\_speed\_z = 0.0;

void PID\_init(PIDControl \*pid, float Kp, float Ki, float Kd) {

pid->Kp = Kp;

pid->Ki = Ki;

pid->Kd = Kd;

pid->set\_point = 0.0;

pid->last\_error = 0.0;

pid->integral = 0.0;

}

float PID\_calculate(PIDControl \*pid, float current) {

float error = pid->set\_point - current;

pid->integral += error;

float derivative = error - pid->last\_error;

float output = (pid->Kp \* error) + (pid->Ki \* pid->integral) + (pid->Kd \* derivative);

pid->last\_error = error;

return output;

}

// This function calculates current speed based on encoder changes over time

void update\_speed() {

static uint32\_t last\_time = 0; // timestamp from the last speed update

uint32\_t current\_time = get\_time(); // function to get current timestamp in ms

if (current\_time - last\_time >= TIMESTAMP\_INTERVAL\_MS) {

float delta\_time = (current\_time - last\_time) / 1000.0; // Convert time to seconds

speed\_x = (encoder\_x\_count - last\_position\_x) / ENCODER\_TICKS\_PER\_REV / delta\_time;

speed\_y = (encoder\_y\_count - last\_position\_y) / ENCODER\_TICKS\_PER\_REV / delta\_time;

speed\_z = (encoder\_z\_count - last\_position\_z) / ENCODER\_TICKS\_PER\_REV / delta\_time;

last\_position\_x = encoder\_x\_count;

last\_position\_y = encoder\_y\_count;

last\_position\_z = encoder\_z\_count;

last\_time = current\_time;

}

}

// Function to apply acceleration control

void apply\_acceleration\_control(float\* current\_speed, float target\_speed) {

if (target\_speed > \*current\_speed) {

// Accelerating

float potential\_speed = \*current\_speed + MAX\_ACCELERATION \* (TIMESTAMP\_INTERVAL\_MS / 1000.0);

\*current\_speed = (potential\_speed < target\_speed) ? potential\_speed : target\_speed;

} else {

// Decelerating

float potential\_speed = \*current\_speed - MAX\_DECELERATION \* (TIMESTAMP\_INTERVAL\_MS / 1000.0);

\*current\_speed = (potential\_speed > target\_speed) ? potential\_speed : target\_speed;

}

}

void control\_motor\_x(float output) {

// Limit output to the range of PWM

if (output > MAX\_PWM\_OUTPUT) {

output = MAX\_PWM\_OUTPUT;

} else if (output < -MAX\_PWM\_OUTPUT) {

output = -MAX\_PWM\_OUTPUT;

}

set\_pwm\_x(output);

}

void control\_motor\_y(float output) {

// Limit output to the range of PWM

if (output > MAX\_PWM\_OUTPUT) {

output = MAX\_PWM\_OUTPUT;

} else if (output < -MAX\_PWM\_OUTPUT) {

output = -MAX\_PWM\_OUTPUT;

}

set\_pwm\_y(output);

}

void control\_motor\_z(float output) {

// Limit output to the range of PWM

if (output > MAX\_PWM\_OUTPUT) {

output = MAX\_PWM\_OUTPUT;

} else if (output < -MAX\_PWM\_OUTPUT) {

output = -MAX\_PWM\_OUTPUT;

}

set\_pwm\_z(output);

}

float read\_encoder\_x() {

return (float)encoder\_x\_count; // Assuming encoder counts directly to position

}

float read\_encoder\_y() {

return (float)encoder\_y\_count; // Assuming encoder counts directly to position

}

float read\_encoder\_z() {

return (float)encoder\_z\_count; // Assuming encoder counts directly to position

}

// Sample function to simulate setting PWM for motor drivers

void set\_pwm\_x(int32\_t output) {

// Implementation to set PWM for motor X

}

void set\_pwm\_y(int32\_t output) {

// Implementation to set PWM for motor Y

}

void set\_pwm\_z(int32\_t output) {

// Implementation to set PWM for motor Z

}

// Main function

int main(void) {

// Initialize PID controllers for Position and Speed for X, Y, Z

PID\_init(&pid\_position\_x, 1.0f, 0.01f, 0.1f);

PID\_init(&pid\_speed\_x, 0.5f, 0.01f, 0.1f);

PID\_init(&pid\_position\_y, 1.0f, 0.01f, 0.1f);

PID\_init(&pid\_speed\_y, 0.5f, 0.01f, 0.1f);

PID\_init(&pid\_position\_z, 1.0f, 0.01f, 0.1f);

PID\_init(&pid\_speed\_z, 0.5f, 0.01f, 0.1f);

while (1) {

// Update speed every few milliseconds

update\_speed();

float current\_position\_x = read\_encoder\_x();

float current\_position\_y = read\_encoder\_y();

float current\_position\_z = read\_encoder\_z();

// Fetch or set desired target positions & speeds here

pid\_position\_x.set\_point = /\* desired X position \*/;

target\_speed\_x = /\* desired X speed \*/; // Target speed for X

pid\_speed\_x.set\_point = target\_speed\_x; // Set target speed for PID control

pid\_position\_y.set\_point = /\* desired Y position \*/;

target\_speed\_y = /\* desired Y speed \*/; // Target speed for Y

pid\_speed\_y.set\_point = target\_speed\_y; // Set target speed for PID control

pid\_position\_z.set\_point = /\* desired Z position \*/;

target\_speed\_z = /\* desired Z speed \*/; // Target speed for Z

pid\_speed\_z.set\_point = target\_speed\_z; // Set target speed for PID control

// Calculate PID outputs for position and speed

float pid\_output\_x\_position = PID\_calculate(&pid\_position\_x, current\_position\_x);

float pid\_output\_y\_position = PID\_calculate(&pid\_position\_y, current\_position\_y);

float pid\_output\_z\_position = PID\_calculate(&pid\_position\_z, current\_position\_z);

// Apply acceleration control to current speeds

apply\_acceleration\_control(&speed\_x, target\_speed\_x);

apply\_acceleration\_control(&speed\_y, target\_speed\_y);

apply\_acceleration\_control(&speed\_z, target\_speed\_z);

// Control motors based on PID outputs and current speeds

control\_motor\_x(pid\_output\_x\_position + PID\_calculate(&pid\_speed\_x, speed\_x));

control\_motor\_y(pid\_output\_y\_position + PID\_calculate(&pid\_speed\_y, speed\_y));

control\_motor\_z(pid\_output\_z\_position + PID\_calculate(&pid\_speed\_z, speed\_z));

}

}