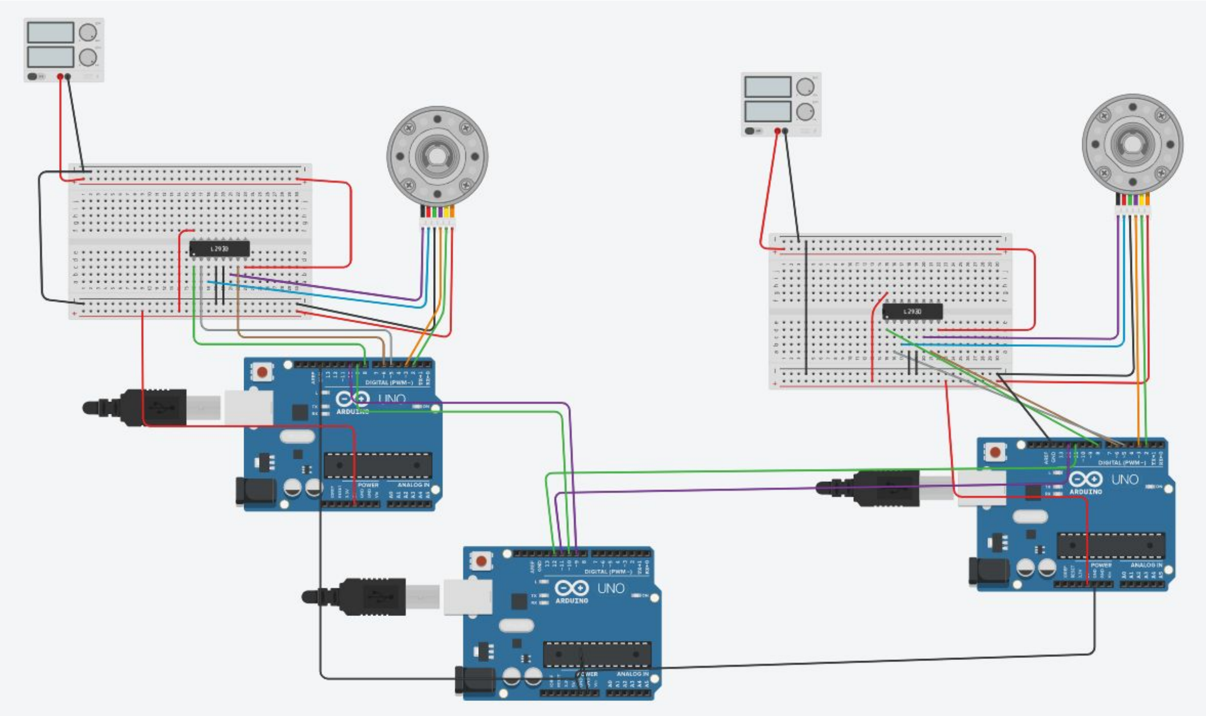
Nome: Vitor Luiz Lima Carazzi **N°USP:** 9834010
Nome: Alessandro Brugnera Silva **N°USP:** 10334040

PMR3402 - Controlador Robótico

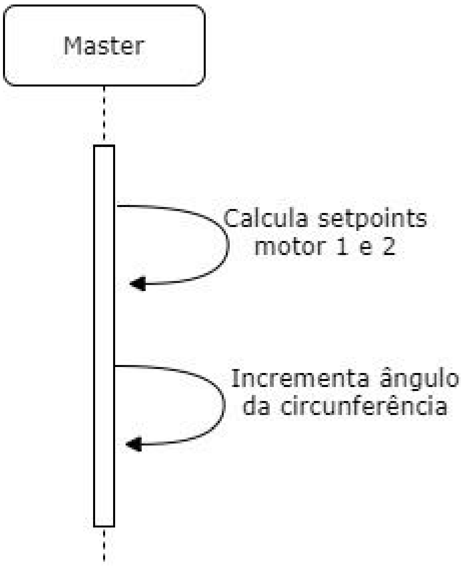
1. **Circuito:**



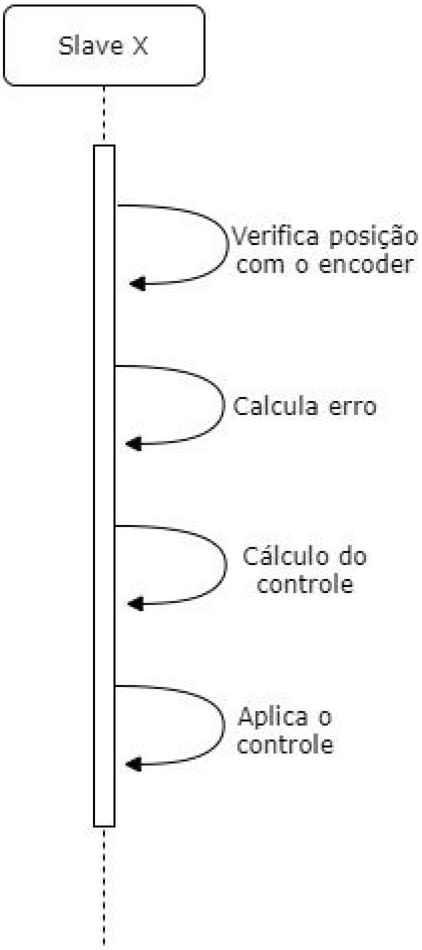
- O arduíno Master é o central;
- O arduíno Slave1 é o do lado esquerdo;
- O arduíno Slave 2 é o do lado direito;

2. Diagramas de sequência:

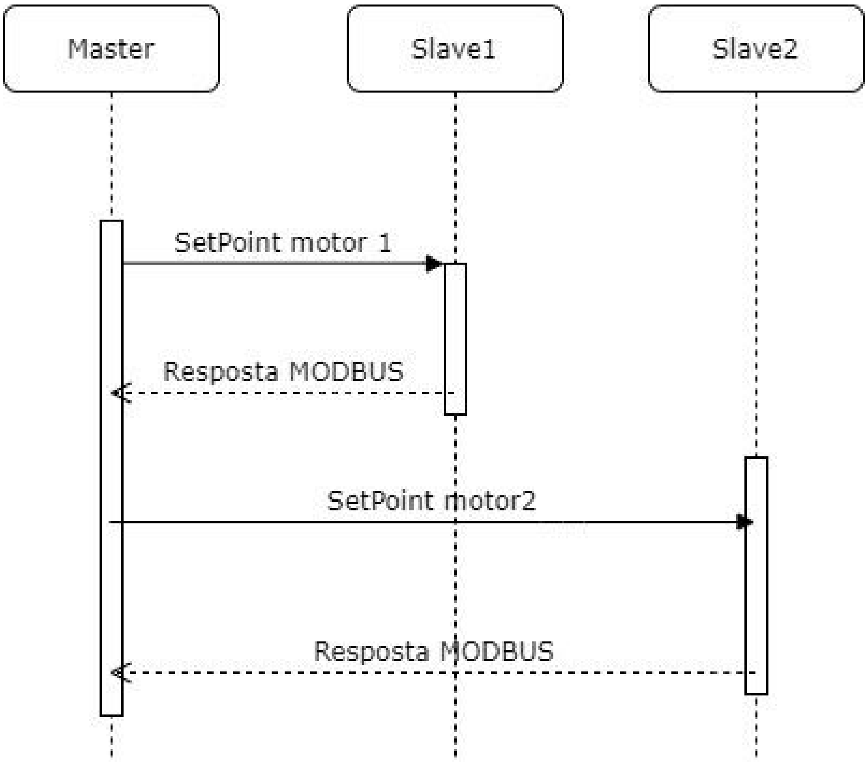
a. SetPoints:


Master
Calcula setpoints motor 1 e 2
Incrementa ângulo da circunferência

c. Controle PID:



b. Comunicação MODBUS:



3. **Códigos:**
    a. **Master:**

```
#include <SoftwareSerial.h>


# define rx1Pin 10
# define tx1Pin 9
# define rx2Pin 11
# define tx2Pin 12

// Comunicacao modbus
char c;
String recData;
String query = ":";
String sendData;
SoftwareSerial mySerial1 = SoftwareSerial(rx1Pin, tx1Pin);
SoftwareSerial mySerial2 = SoftwareSerial(rx2Pin, tx2Pin);
char dados[21];
int dec_lrc[4];;
int lrc;
int lrc_bin[16];

// calculo de trajetoria
int x = 1, y = 1, angle = 1;


// ============================== Fim das variaveis globais

//Funcoes utilizadas para o protocolo MODBUS

int BintoDec(int val1, int val2, int val3, int val4) {
    int result = 0;
    result += val4;
    result += val3 * 2;
    result += val2 * 4;
    result += val1 * 8;
    return result;
}

String formata_query(String query, int coord, int slave) {
    int s = 0;
    //1 e 2 - Slave Adress (0x)
    query += "0";
    query += slave;
    //3 e 4 - Function Code (06)
    query += "06";

    // 5 ,6 ,7 ,8 - Controlador (0001)
    query += " 0001 ";
    // 9 ,10 ,11 ,12 - coord setpoint ( ABCD )
    query += "00";
    query += coord;
    // 13 ,14 ,15 ,16 - LRC
    lrc = query[1] ^ query[2] ^ query[3] ^ query[4] ^ query[5] ^ query[6] ^ query[7] ^ query[8] ^ query[9] ^ query[10] ^
          query[11] ^ query[12];
    for (int k = 0; lrc > 0; k++) {
        lrc_bin[k] = lrc % 2;
        lrc = lrc / 2;
    }
    int dec_lrc[4];
    dec_lrc[3] = BintoDec(lrc_bin[3], lrc_bin[2], lrc_bin[1], lrc_bin[0]);// lsb
    dec_lrc[2] = BintoDec(lrc_bin[7], lrc_bin[6], lrc_bin[5], lrc_bin[4]);
    dec_lrc[1] = BintoDec(lrc_bin[11], lrc_bin[10], lrc_bin[9], lrc_bin[8]);
    dec_lrc[0] = BintoDec(lrc_bin[15], lrc_bin[14], lrc_bin[13], lrc_bin[12]);// msb
    int m = 0;
    for (int j = 0; j < 4; j++) {
        if (dec_lrc[m] > 10) {
            query += dec_lrc[m];
        } else {
            query += dec_lrc[m];
        }
        m++;
    }
    // 17 e 18 - CR ,LF
    query += '\r';
    query += '\n';
    return query;
}


// TASK1

# define INTERVALO1 1000

void task1() {
    x = round(10 * (1 + cos(angle * M_PI / 180)));
    y = round(10 * (1 + sin(angle * M_PI / 180)));
    angle++;

} // task1

// TASK2

# define INTERVALO2 200 // Tarefa2 a cada 0.005 s

void task2() {
    sendData = formata_query(query, x, 1);

    Serial.println(" Transmitting : " + sendData);
    mySerial1.print(sendData);
    sendData = "";
    query = ":";
    sendData = formata_query(query, y, 2);
    Serial.println(" Transmitting : " + sendData);
    mySerial2.print(sendData);
    sendData = "";
    query = ":";
    delay(50);
    if (mySerial1.available() > 0) {
        recData = "";
        while (mySerial1.available()) {
            c = mySerial1.read();
            recData += c;
        }
    }
    delay(50);
    if (mySerial2.available() > 0) {
        recData = "";
        while (mySerial2.available()) {
            c = mySerial2.read();
            recData += c;
        }
    }

} // task2


// TASK SWITCHER

typedef struct {
    void (*task )();

    long interval;
    long current_time;
    int status;
} TaskControl;

# define MAX_TASKS 2
# define READY 1
# define WAIT 0
TaskControl taskList[MAX_TASKS];

void createTask(int taskNum, void (*t )(), long interval) {
    // Cria uma task
    taskList[taskNum].task = t;
    taskList[taskNum].interval = interval;
    taskList[taskNum].current_time = 0;
    taskList[taskNum].status = WAIT;
} // createTask

void runCurrentTask() {
    // Executa a task atual
    int i;

    void (*task )();
    for (i = 0; i < MAX_TASKS; i++) {
        if (taskList[i].status == READY) {
            task = taskList[i].task;
            (*task)();
            noInterrupts();
            taskList[i].status = WAIT;
            taskList[i].current_time = 0;
            interrupts();
        } // if task is READY
    } // for each task
} // runCurrentTask

void updateTickCounter() {
    // Atualiza o contador para os intervalos
    // das tasks
    int i;
    for (i = 0; i < MAX_TASKS; i++) {
        if (taskList[i].status == WAIT) {
            taskList[i].current_time++;
            if (taskList[i].current_time >= taskList[i].interval) {
                taskList[i].status = READY;
            }
        } // if task is WAITing
    } // for each task
} // updateTickCounter







void setup() {

    pinMode(rx1Pin, INPUT);
    pinMode(tx1Pin, OUTPUT);
    pinMode(rx2Pin, INPUT);
    pinMode(tx2Pin, OUTPUT);

    // set the data rate for the SoftwareSerial port
    mySerial1.begin(2400);
    mySerial2.begin(2400);
    Serial.begin(2400);


    // Cria as tarefas
    createTask(0, &task1, INTERVALO1);
    createTask(1, &task2, INTERVALO2);
    // Inicia interrupcao do timer
    // para 1ms
    setTimerInterrupt(1000); // int @1ms (1000 us)
}


void loop() {
    runCurrentTask(); // executa tarefa atual
}


// Interrupcao do Timer
ISR ( TIMER1_COMPA_vect ) {
        updateTickCounter() ;
} // ISR


void setTimerInterrupt(long uSecs) {
    noInterrupts(); // Desabilita interrupcoes

    TCCR1A = 0;
    TCCR1B = 0;
    TCNT1 = 0;

    // compare match register 16 MHz /256 * t(s) - 1
    OCR1A = (16e6 / 256L * uSecs ) / 1e6 - 1;

    TCCR1B |= (1 << WGM12); // CTC mode
    TCCR1B |= (1 << CS12); // 256 prescaler
    TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt

    interrupts(); // enable all interrupts
}
```

b. **Slave1:**

```
#include <SoftwareSerial.h>




# define rxPin 10
# define txPin 9


// Pinos de controle do Motor
const int motorDirPin = 5; // Input 1
const int motorPWMPin = 6; // Input 2
const int EnablePin = 8; // Enable
const int LED = 13;

// Pino de encoder
const int encoderPinA = 2;

const int encoderPinB = 3;
int encoderPos = 0;

// encoder value change motor turn angles
const float ratio = 360. / 188.611 / 48.;
// 360. -> 1 turn
// 188.611 -> Gear Ratio
// 48. -> Encoder : Countable Events Per Revolution ( Motor Shaft )

//PID
float Kp = 20; // Proporcional
float Ki = 0.00001; // Integrativo
float Kd = 0.0000; // Derivativo
float target; //Posicao desejada
float last_target = 0;

float last_error = 0; // Armazenamento do erro anterior de PID
float Ierror; // Erro integral
float Derror; // Erro derivativo

// Tempo
unsigned long current_time; // Momento atual de calculo do PID
unsigned long last_time; // Ultimo momento calculado do PID

// Comunicacao modbus
char c;
String recData;
int i = 0;
String resposta = ":";
int setpoint_char[4];
int dec_lrc[4];
int lrc;
int lrc_bin[16];
String sendData = "";
SoftwareSerial mySerial = SoftwareSerial(rxPin, txPin);
int setPoint = 0;
int newSetPoint = 1;
int nova = 0;


//MODBUS
bool condicoes_ok(String recData) {
    if (!check_slave(recData)) {
        return false;
    }
    return true;
}

bool check_slave(String recData) {
    char c = recData[2];
    if (c == '1') {
        return true;
    }
    return false;
}


int BintoDec(int val1, int val2, int val3, int val4) {
    int result = 0;
    result += val4;
    result += val3 * 2;
    result += val2 * 4;
    result += val1 * 8;
    return result;
}

String formataResposta(String resposta, int setpoint) {
    int s = 0;
    //1 e 2 - Slave Adress (01)
    resposta += "0";
    resposta += "1";
    //3 e 4 - Function Code (06)
    resposta += "0";
    resposta += "6";
    // 5 ,6 ,7 ,8 - Controlador (0001)
    resposta += "0";
    resposta += "0";
    resposta += "0";
    resposta += "1";
    // 9 ,10 ,11 ,12 - Setpoint ( ABCD )
    resposta += "00";
    resposta += setpoint;
    // 13 ,14 ,15 ,16 - LRC
    lrc = resposta[1] ^ resposta[2] ^ resposta[3] ^ resposta[4] ^ resposta[5] ^ resposta[6] ^ resposta[7] ^
          resposta[8] ^ resposta[9] ^ resposta[10] ^ resposta[11] ^ resposta[12];
    for (int k = 0; lrc > 0; k++) {
        lrc_bin[k] = lrc % 2;
        lrc = lrc / 2;
    }
    int dec_lrc[4];
    dec_lrc[3] = BintoDec(lrc_bin[3], lrc_bin[2], lrc_bin[1], lrc_bin[0]);// lsb
    dec_lrc[2] = BintoDec(lrc_bin[7], lrc_bin[6], lrc_bin[5], lrc_bin[4]);
    dec_lrc[1] = BintoDec(lrc_bin[11], lrc_bin[10], lrc_bin[9], lrc_bin[8]);
    dec_lrc[0] = BintoDec(lrc_bin[12], lrc_bin[11], lrc_bin[10], lrc_bin[9]);// msb
    int m = 0;
    for (int j = 13; j <= 16; j++) {
        if (dec_lrc[m] > 10) {
            resposta += dec_lrc[m];
        } else {
            resposta += dec_lrc[m];
        }
        m++;
    }
    // 17 e 18 - CR ,LF
    resposta += '\r';
    resposta += '\n';
    return resposta;


}

// TASK1

# define INTERVALO1 50 // Tarefa1 a cada 0.005 s

void task1() {
    // Print na tela a posicao do motor

    Serial.println(target);
    Serial.println(float(encoderPos) * ratio);

} // task1


// TASK2

# define INTERVALO2 50 // Tarefa2 a cada 0.005 s

void task2() {
    // Verifica posicao atual e ajusta com PID

    current_time = millis(); // Atualiza instante atual
    float motorDeg = float(encoderPos) * ratio; //Posicao atual do motor
    int dt = current_time - last_time; // Diferenca de tempo entre os ajustes
    float error = target - motorDeg; // Erro de controle
    Ierror += error; // Incrementa erro integral
    Derror = error - last_error; // Atualiza erro derivativo
    float control = (Kp * error) + // Calcula output decontrole
                    (Ki * Ierror * dt) +
                    (Kp * Kd * Derror / dt);
    digitalWrite(EnablePin, 255); // Seta comando pro motor
    doMotor((control >= 0) ? HIGH : LOW, min(abs(control), 255));

    last_target = target;
    last_error = error; // Atualiza o erro de controle
    last_time = current_time; // Atualiza o instante de controle

} // task2

# define INTERVALO3 200

void task3() {
    if (mySerial.available() > 0) {
        recData = "";
        while (mySerial.available()) {
            c = mySerial.read();
            recData += c;
            nova = 1;
        }
        if (nova == 1 && condicoes_ok(recData)) {
            Serial.println("SS Data received = " + recData);

            setpoint_char[3] = recData[9] - 48;
            setpoint_char[2] = recData[10] - 48;
            setpoint_char[1] = recData[11] - 48;
            setpoint_char[0] = recData[12] - 48;
            for (int m = 3; m >= 0; m--) {
                setPoint = 10 * setPoint + setpoint_char[m];
            }
            sendData = formataResposta(resposta, setPoint);
            resposta = ":";
            newSetPoint = setPoint;
            Serial.println(" Transmiting : " + sendData);
            mySerial.print(sendData);
            sendData = "";
            setPoint = 0;
        }
        nova = 0;
    }

}
```

```c
// TASK SWITCHER

typedef struct {
    void (*task )();

    long interval;
    long current_time;
    int status;
} TaskControl;

# define MAX_TASKS 3
# define READY 1
# define WAIT 0
TaskControl taskList[MAX_TASKS];

void createTask(int taskNum, void (*t )(), long interval) {
    // Cria uma task
    taskList[taskNum].task = t;
    taskList[taskNum].interval = interval;
    taskList[taskNum].current_time = 0;
    taskList[taskNum].status = WAIT;
} // createTask

void runCurrentTask() {
    // Executa a task atual
    int i;
    void (*task )();
    for (i = 0; i < MAX_TASKS; i++) {
        if (taskList[i].status == READY) {
            task = taskList[i].task;
            (*task)();
            noInterrupts();
            taskList[i].status = WAIT;
            taskList[i].current_time = 0;
            interrupts();
        } // if task is READY
    } // for each task

} // runCurrentTask

void updateTickCounter() {
    // Atualiza o contador para os intervalos
    // das tasks
    int i;
    for (i = 0; i < MAX_TASKS; i++) {
        if (taskList[i].status == WAIT) {
            taskList[i].current_time++;
            if (taskList[i].current_time >= taskList[i].interval) {
                taskList[i].status = READY;
            }
        } // if task is WAITing
    } // for each task
} // updateTickCounter

// ===========================
// SETUP

void setup() {

    pinMode(rxPin, INPUT);
    pinMode(txPin, OUTPUT);

    // set the data rate for the SoftwareSerial port
    mySerial.begin(2400);

    Serial.begin(2400);

    // Configura os pinos
    pinMode(encoderPinA, INPUT_PULLUP);
    attachInterrupt(0, doEncoderA, CHANGE);

    pinMode(encoderPinB, INPUT_PULLUP);
    attachInterrupt(1, doEncoderB, CHANGE);

    pinMode(LED, OUTPUT);
    pinMode(motorDirPin, OUTPUT);
    pinMode(EnablePin, OUTPUT);

    // Cria as tarefas
    createTask(0, &task1, INTERVALO1);
    createTask(1, &task2, INTERVALO2);
    createTask(2, &task3, INTERVALO3);

    // Inicia interrupcao do timer
    // para 1ms
    setTimerInterrupt(1000); // int @1ms (1000 us)
}


void loop() {
    // target = 10; // Setando posicao desejada para o motor
    if (target != newSetPoint) {
        target = newSetPoint;

    }
    runCurrentTask(); // executa tarefa atual
}


// Interrupcao do Timer
ISR ( TIMER1_COMPA_vect ) {
        updateTickCounter();
```

```
} // ISR

// set up timer1 -
// compare interrupt @ uSecs microseconds
void setTimerInterrupt(long uSecs) {
    noInterrupts(); // Desabilita interrupcoes

    TCCR1A = 0;
    TCCR1B = 0;
    TCNT1 = 0;

    // compare match register 16 MHz /256 * t(s) - 1
    OCR1A = (16e6 / 256L * uSecs) / 1e6 - 1;

    TCCR1B |= (1 << WGM12); // CTC mode
    TCCR1B |= (1 << CS12); // 256 prescaler
    TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt

    interrupts(); // enable all interrupts
}


void doEncoderA() {
    encoderPos += (digitalRead(encoderPinA) == digitalRead(encoderPinB)) ? 1 : -1;
}

void doEncoderB() {
    encoderPos += (digitalRead(encoderPinA) == digitalRead(encoderPinB)) ? -1 : 1;
}

void doMotor(bool dir, int vel) {
    digitalWrite(motorDirPin, dir);
    digitalWrite(LED, dir);
    analogWrite(motorPWMPin, dir ? (255 - vel) : vel);
}
```

c. **Slave2:**

```
#include <SoftwareSerial.h>


# define rxPin 10
# define txPin 9


// Pinos de controle do Motor
const int motorDirPin = 5; // Input 1
const int motorPWMPin = 6; // Input 2
const int EnablePin = 8; // Enable
const int LED = 13;

// Pino de encoder
const int encoderPinA = 2;

const int encoderPinB = 3;
int encoderPos = 0;

// encoder value change motor turn angles
const float ratio = 360. / 188.611 / 48.;
// 360. -> 1 turn
// 188.611 -> Gear Ratio
// 48. -> Encoder : Countable Events Per Revolution ( Motor Shaft )

//PID
float Kp = 20; // Proporcional
float Ki = 0.00001; // Integrativo
float Kd = 0.0000; // Derivativo
float target; //Posicao desejada
float last_target = 0;

float last_error = 0; // Armazenamento do erro anterior de PID
float Ierror; // Erro integral
float Derror; // Erro derivativo

// Tempo
unsigned long current_time; // Momento atual de calculo do PID
unsigned long last_time; // Ultimo momento calculado do PID

// Comunicacao modbus
char c;
String recData;
int i = 0;
String resposta = ":";
int setpoint_char[4];
int dec_lrc[4];
int lrc;
int lrc_bin[16];
String sendData = "";
```

```
SoftwareSerial mySerial = SoftwareSerial(rxPin, txPin);
int setPoint = 0;
int newSetPoint = 1;
int nova = 0;


//MODBUS
bool condicoes_ok(String recData) {
    if (!check_slave(recData)) {
        return false;
    }
    return true;
}

bool check_slave(String recData) {
    char c = recData[2];
    if (c == '1') {
        return true;
    }
    return false;
}


int BintoDec(int val1, int val2, int val3, int val4) {
    int result = 0;
    result += val4;
    result += val3 * 2;
    result += val2 * 4;
    result += val1 * 8;
    return result;
}

String formataResposta(String resposta, int setpoint) {
    int s = 0;
    //1 e 2 - Slave Adress (01)
    resposta += "0";
    resposta += "1";
    //3 e 4 - Function Code (06)
    resposta += "0";
    resposta += "6";
    // 5 ,6 ,7 ,8 - Controlador (0001)
    resposta += "0";
    resposta += "0";
    resposta += "0";
    resposta += "1";
    // 9 ,10 ,11 ,12 - Setpoint ( ABCD )
    resposta += "00";
    resposta += setpoint;
    // 13 ,14 ,15 ,16 - LRC
    lrc = resposta[1] ^ resposta[2] ^ resposta[3] ^ resposta[4] ^ resposta[5] ^
resposta[6] ^ resposta[7] ^
```

```
          resposta[8] ^ resposta[9] ^ resposta[10] ^ resposta[11] ^ resposta[12];
    for (int k = 0; lrc > 0; k++) {
        lrc_bin[k] = lrc % 2;
        lrc = lrc / 2;
    }
    int dec_lrc[4];
    dec_lrc[3] = BintoDec(lrc_bin[3], lrc_bin[2], lrc_bin[1], lrc_bin[0]);// lsb
    dec_lrc[2] = BintoDec(lrc_bin[7], lrc_bin[6], lrc_bin[5], lrc_bin[4]);
    dec_lrc[1] = BintoDec(lrc_bin[11], lrc_bin[10], lrc_bin[9], lrc_bin[8]);
    dec_lrc[0] = BintoDec(lrc_bin[12], lrc_bin[11], lrc_bin[10], lrc_bin[9]);// msb
    int m = 0;
    for (int j = 13; j <= 16; j++) {
        if (dec_lrc[m] > 10) {
            resposta += dec_lrc[m];
        } else {
            resposta += dec_lrc[m];
        }
        m++;
    }
    // 17 e 18 - CR ,LF
    resposta += '\r';
    resposta += '\n';
    return resposta;


}

// TASK1

# define INTERVALO1 50 // Tarefa1 a cada 0.005 s

void task1() {
    // Print na tela a posi[U+FFFD][U+FFFD] o do motor

    Serial.println(target);
    Serial.println(float(encoderPos) * ratio);

} // task1


// TASK2

# define INTERVALO2 50 // Tarefa2 a cada 0.005 s

void task2() {
    // Verifica posicao atual e ajusta com PID

    current_time = millis(); // Atualiza instante atual
    float motorDeg = float(encoderPos) * ratio; //Posicao atual do motor
    int dt = current_time - last_time; // Diferenca de tempo entre os ajustes
    float error = target - motorDeg; // Erro de controle
    Ierror += error; // Incrementa erro integral
    Derror = error - last_error; // Atualiza erro derivativo
    float control = (Kp * error) + // Calcula output decontrole
                    (Ki * Ierror * dt) +
                    (Kp * Kd * Derror / dt);
    digitalWrite(EnablePin, 255); // Seta comando pro motor
    doMotor((control >= 0) ? HIGH : LOW, min(abs(control), 255));

    last_target = target;
    last_error = error; // Atualiza o erro de controle
    last_time = current_time; // Atualiza o instante de controle

} // task2

# define INTERVALO3 200

void task3() {
    if (mySerial.available() > 0) {
        recData = "";
        while (mySerial.available()) {
            c = mySerial.read();
            recData += c;
            nova = 1;
        }
        if (nova == 1 && condicoes_ok(recData)) {
            Serial.println("SS Data received = " + recData);

            setpoint_char[3] = recData[9] - 48;
            setpoint_char[2] = recData[10] - 48;
            setpoint_char[1] = recData[11] - 48;
            setpoint_char[0] = recData[12] - 48;
            for (int m = 3; m >= 0; m--) {
                setPoint = 10 * setPoint + setpoint_char[m];
            }
            sendData = formataResposta(resposta, setPoint);
            resposta = ":";
            newSetPoint = setPoint;
            Serial.println(" Transmiting : " + sendData);
            mySerial.print(sendData);
            sendData = "";
            setPoint = 0;
        }
        nova = 0;
    }

}



// TASK SWITCHER
```

```
typedef struct {
    void (*task )();

    long interval;
    long current_time;
    int status;
} TaskControl;

# define MAX_TASKS 3
# define READY 1
# define WAIT 0
TaskControl taskList[MAX_TASKS];

void createTask(int taskNum, void (*t )(), long interval) {
    // Cria uma task
    taskList[taskNum].task = t;
    taskList[taskNum].interval = interval;
    taskList[taskNum].current_time = 0;
    taskList[taskNum].status = WAIT;
} // createTask

void runCurrentTask() {
    // Executa a task atual
    int i;
    void (*task )();
    for (i = 0; i < MAX_TASKS; i++) {
        if (taskList[i].status == READY) {
            task = taskList[i].task;
            (*task)();
            noInterrupts();
            taskList[i].status = WAIT;
            taskList[i].current_time = 0;
            interrupts();
        } // if task is READY
    } // for each task

} // runCurrentTask

void updateTickCounter() {
    // Atualiza o contador para os intervalos
    // das tasks
    int i;
    for (i = 0; i < MAX_TASKS; i++) {
        if (taskList[i].status == WAIT) {
            taskList[i].current_time++;
            if (taskList[i].current_time >= taskList[i].interval) {
                taskList[i].status = READY;
            }
        } // if task is WAITing
    } // for each task
} // updateTickCounter
```

```
// ===========================
// SETUP

void setup() {

    pinMode(rxPin, INPUT);
    pinMode(txPin, OUTPUT);

    // set the data rate for the SoftwareSerial port
    mySerial.begin(2400);

    Serial.begin(2400);

    // Configura os pinos
    pinMode(encoderPinA, INPUT_PULLUP);
    attachInterrupt(0, doEncoderA, CHANGE);

    pinMode(encoderPinB, INPUT_PULLUP);
    attachInterrupt(1, doEncoderB, CHANGE);

    pinMode(LED, OUTPUT);
    pinMode(motorDirPin, OUTPUT);
    pinMode(EnablePin, OUTPUT);

    // Cria as tarefas
    createTask(0, &task1, INTERVALO1);
    createTask(1, &task2, INTERVALO2);
    createTask(2, &task3, INTERVALO3);

    // Inicia interrupcao do timer
    // para 1ms
    setTimerInterrupt(1000); // int @1ms (1000 us)
}


void loop() {
    // target = 10; // Setando posicao desejada para o motor
    if (target != newSetPoint) {
        target = newSetPoint;

    }
    runCurrentTask(); // executa tarefa atual
}


// Interrupcao do Timer
ISR ( TIMER1_COMPA_vect ) {
        updateTickCounter();
} // ISR
```

```
// set up timer1 -
// compare interrupt @ uSecs microseconds
void setTimerInterrupt(long uSecs) {
    noInterrupts(); // Desabilita interrupcoes

    TCCR1A = 0;
    TCCR1B = 0;
    TCNT1 = 0;

    // compare match register 16 MHz /256 * t(s) - 1
    OCR1A = (16e6 / 256L * uSecs) / 1e6 - 1;

    TCCR1B |= (1 << WGM12); // CTC mode
    TCCR1B |= (1 << CS12); // 256 prescaler
    TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt

    interrupts(); // enable all interrupts
}


void doEncoderA() {
    encoderPos += (digitalRead(encoderPinA) == digitalRead(encoderPinB)) ? 1 : -1;
}

void doEncoderB() {
    encoderPos += (digitalRead(encoderPinA) == digitalRead(encoderPinB)) ? -1 : 1;
}

void doMotor(bool dir, int vel) {
    digitalWrite(motorDirPin, dir);
    digitalWrite(LED, dir);
    analogWrite(motorPWMPin, dir ? (255 - vel) : vel);
}
```

4. **Questões:**
    a. **Qual o tempo de execução do PID?**
    O tempo médio de execução do PID é de 240µs.
    b. **Se você não tiver o Serial.Monitor (ou um display), como saber o tempo de execução do PID?**
    O tempo entre uma mudança e outra dos valores de Duty Cycle enviados ao PWM do motor é o tempo de execução do PID.
    c. **Tente estimar o jitter(variação) do tempo de amostragem usado no PID.**
    O jitter pode ser estimado calculando-se uma média de duas leituras consecutivas do encoder. Dessa forma a média é de cerca de 3,5 ms.
    d. **Como calcular o tempo máximo para as interrupções do encoder?**
    O tempo máximo é a soma do intervalo com a qual a task é chamada e do tempo de execução do PID.

5. **Link:**
https://www.tinkercad.com/things/kMIr53cgy50-magnificent-rottis/editel?sharecode=sADvlEsPD4vQhMLC3mInhSjBfO9EIMYFJRkXiFXTmhc