UNIVERSIDADE FEDERAL DE LAVRAS

Introdução Aos Sistemas Embarcados e Microcontroladores

TRABALHO PRÁTICO

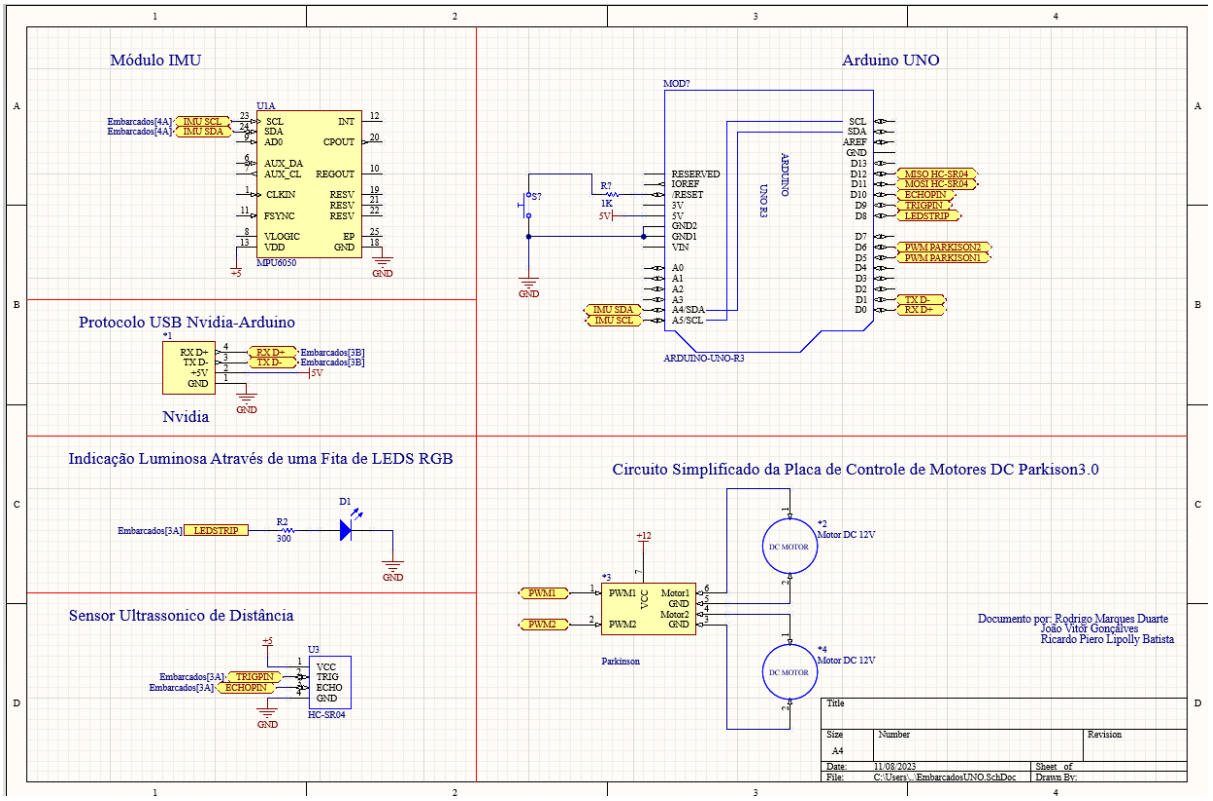
Rodrigo Marques Duarte

Ricardo Piero Lipolly Batista

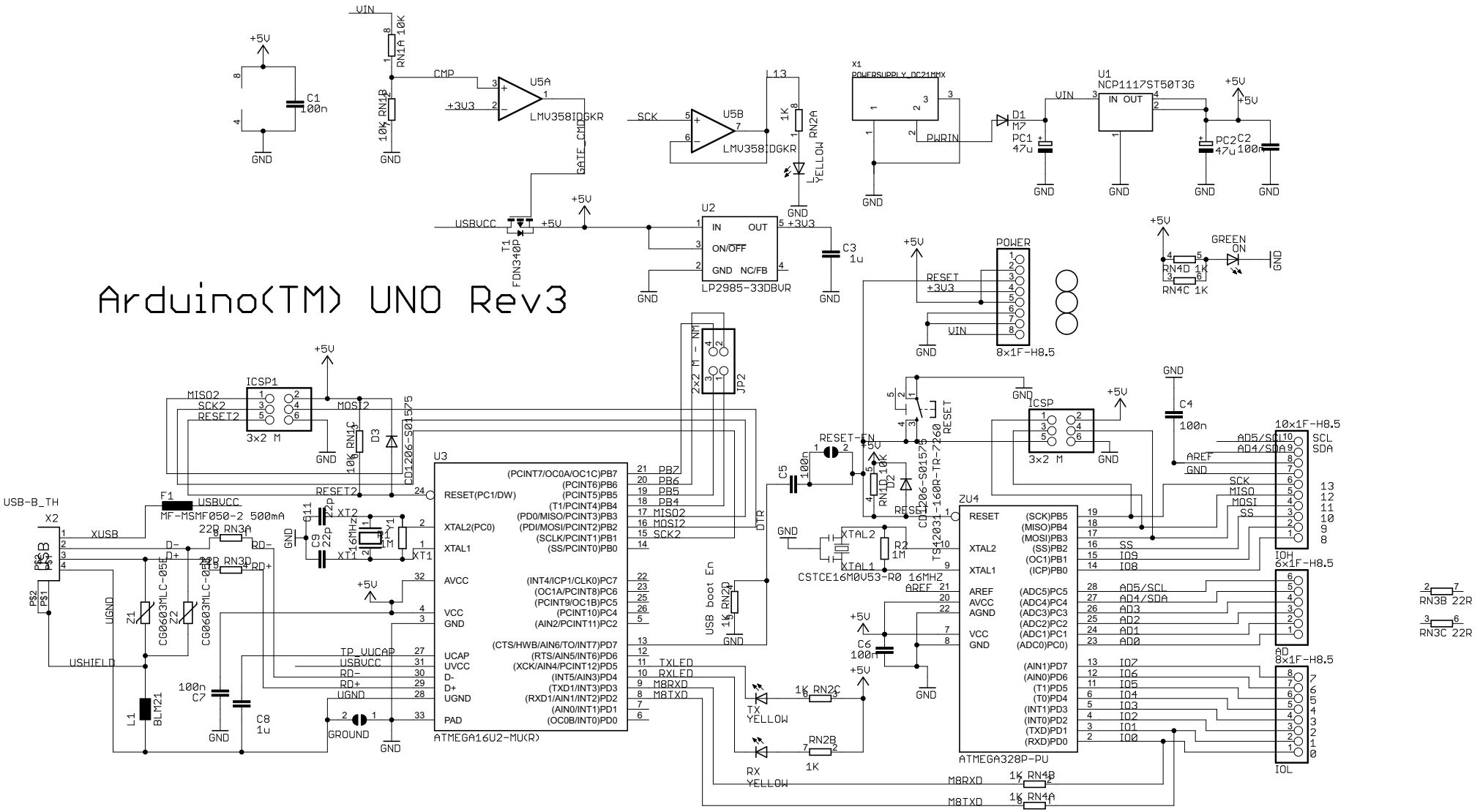
João Vitor Gonçalves

Turma: 22A

Hardware Projeto Trekking 2023/1 -Equipe TROIA de Robótica



O projeto esquemático foi feito com auxílio do software ALTIUM DESIGNER.
O Projeto dispõe de:
1 Sensor de Distância Ultrassônico
1 Sensor IMU para correção de trajetória
1 Conjunto de LEDS para indicação
1 placa NVIDIA JETSON NANO para processamento de imagem
1 placa Arduino UNO para o controle de sensores e atuadores.
1 placa de Potência para os motores DC's Parkison de autoria da Equipe TROIA (0-24VDC)
2 motores DC 12 V



Grupo: Ricardo Piero Lippoli Batista

Tema: Robô trekking

Texto descrevendo o robô e seu funcionamento em apenas um parágrafo:
O Robô Trekking utiliza duas placas principais para operar de maneira autônoma. A Nvidia Jetson Nano, alimentada por 5V, responsável pela visão computacional, identifica objetos pré-determinados e envia coordenadas para a segunda placa via comunicação serial. Esta segunda placa, um Arduino Uno, alimentado pela Nvidia Jetson Nano (5V), trata as coordenadas, gerencia os sensores ultrassônicos e giroscópio, implementa um controlador PID e conduz o robô na direção do objeto identificado. Equipado com a habilidade de desviar de obstáculos, o robô Trekking para próximo do objeto, emitindo um sinal visual através de um LED piscante para indicar o sucesso no trekking do objeto.

Alunos: Ricardo Piero, João Vitor e Rodrigo Marques

Projeto: Robô trekking

Arquivo main:

```
//Importar bibliotecas
#include <Arduino.h>
#include <Serial.h>
#include <Controle.h>
#include <Ultrassonico.h>

int leitura, eixoX, valEixoX;
void setup(){ //função de inicialização dos sensores
  pinMode(LED_BUILTIN, OUTPUT);
  initPKS();
  Serial.begin(9600);
  init_mpu();
  initUltrassonico();
}

void loop(){ //função principal
  eixoX = SerialConvertion();
  testeMotor(eixoX);
  distCone();
}
```

Arquivo motor:

```
//importando bibliotecas
#ifndef MOTOR_H
#define MOTOR_H
#include <defines.h>
```

```
#include <giroscopio.h>
#include <servo.h>
Servo FW_PKS, ANG_PKS;

void initPKS(){ //função de iniciar o motor através da placa parkinson
        FW_PKS.attach(fw_pks);
        ANG_PKS.attach(ang_pks);
}

void stopPKS(){ //função para parar o motor através da placa parkinson
    FW_PKS.writeMicroseconds(1500);
    ANG_PKS.writeMicroseconds(1500);
}

void motorsControl(){ //função de simulçao para controle do motor
    if (Serial.available()){
        if(Serial.read() == 'Direita'){
            FW_PKS.writeMicroseconds(2000);
            ANG_PKS.writeMicroseconds(1800);
        }
        if(Serial.read() == 'Esquerda'){
            FW_PKS.writeMicroseconds(2000);
            ANG_PKS.writeMicroseconds(1200);
        }
        else{
            FW_PKS.writeMicroseconds(1500);
            ANG_PKS.writeMicroseconds(1500);
        }
    }
}
```

```
#endif
```

Arquivo defines:

```
//Definindo pinos e variaveis
#ifndef DEFINES_H
#define DEFINES_H

#include <Arduino.h>

#define fw_pks 6
#define ang_pks 5

#define trigPin 9
#define echoPin 10
#define LEDCONE 8

#define DE1 A0
#define DE2 A1
#define DE3 A2
#define DE4 A3

#define SDA A4
#define SCl A5

#define PINO_LED 13

#define CENTRO_CONE 320
```

```
#define MPU_ADDR 0x68

String cmd = "";

#endif
```

Arquivo ultrassónico:

```
//Importa bibliotecas
#include <defines.h>
#include <Controle.h>

long duration;
int distance;
void initUltrassonico(){ //função para iniciar ultrassonico
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}
void distCone(){ //função para saber a distancia do robô ao cone
  // limpando trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // setando o trigPin como estado HIGH por 10 micro segundos
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // faz a leitura do echoPin, retorna o pulso do tempo
  duration = pulseIn(echoPin, HIGH);
  // calcula a distancia
  distance = duration * 0.034 / 2;
```

```
  if(distance <= 300){ //se a distancia for menor que 300 para o robô e pisca o
LED
    FW_PKS.writeMicroseconds(1500);
    ANG_PKS.writeMicroseconds(1500);
    pinMode(LEDCONE,LOW);
    delayMicroseconds(400);
    pinMode(LEDCONE,HIGH);
    delayMicroseconds(400);
  }
}
```

Arquivo serial:

```
//Importa bibliotecas
#include <Arduino.h>


int SerialConvertion(){ //função que faz a descriptografia das cordenadas
recebidas pelo serial
    String palavra = " ";
    char c;
    if (Serial.available() > 0){ //se não existir valor no Serial, retorna 0
        c = (Serial.read());
        if (c == (',')){
            while (c != '/'){
                int i = 0;
                if (Serial.available() > 0){
                    c = char(Serial.read());
                    i++;
                    if (i < 2){
                        palavra = palavra + c;
                    }
```

```
            }
        }
        int eixoX = palavra.toInt();
        return eixoX;
    }
  }
  else{
    return 0;
  }
}
```

Arquivo controle:

```
//Importando bibliotecas
#include <defines.h>
#include <motor.h>

int ang, frente;
void testeMotor(int eixoX){ // função de definir a direção e velocidade do motor
  if (eixoX != 0){ //se o eixo recebido for diferente de 0 ele altera o estado anterior
    if (eixoX > 0 && eixoX < 220){ // se o eixo for maior que 0 e menor que 220, vai para a esquerda
      frente = 1340;
      ang = 1156;
    }
    else if (eixoX > 420){ // se o eixo for maior que  420, vai para a direita
      frente = 1760;
      ang = 1894;
    }
    else if (eixoX >= 220 && eixoX <= 420){ // se o eixo for maior ou igual a 220 e menor ou igual a 420, segue em frente
```

```
            frente = 1876;
            ang = 1500;
        }
    }
    //funções de controle do motor
    FW_PKS.writeMicroseconds(frente);
    ANG_PKS.writeMicroseconds(ang);
    delay(400);
}

float PID(float target, float atual){// função de PID geral
    float kp = 60;
    float error = target - atual;
    float output = error * kp;
    return output;
}

float pidCamera(int target, int atual){// função de PID da camera
    int kp = 0.00035;
    int error = target - atual;
    int output = error * kp;
    return output;
}

void align(int eixoX){ // função para ajuste de posição através do giroscopio
    float gyro = readAngularSpeed();
}

void getpulse(int frente, int ang){ // função para simulação de controle do robô
    FW_PKS.writeMicroseconds(frente);
```

```
    ANG_PKS.writeMicroseconds(ang);
}
```

Arquivo giroscópio:

```
//Importando bibliotecas
#ifndef GIROSCOPIO_H
#define GIROSCOPIO_H
#include <defines.h>
#include <Wire.h>



void init_mpu(){ // iniciando giroscopio
    Wire.begin();
    Wire.beginTransmission(MPU_ADDR);
    Wire.write(0x6B);
    Wire.write(0);
    Wire.endTransmission(true);

    Wire.beginTransmission(MPU_ADDR);  // SET RANGE GYRO
    Wire.write(0x1B);                  // endereço do registrador
    Wire.write(0b00001000);            // valor para 500 g/s
    Wire.endTransmission(true);

    Wire.beginTransmission(MPU_ADDR);  // SET BAND WIDTH
    Wire.write(0x1A);                  // endereço do registrador
    Wire.write(0b00000100);            // valor para 21Hz
    Wire.endTransmission(true);
}
```

```
float readAngularSpeed(){ // função que le o valor da velocidade angular
    int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
    Wire.beginTransmission(MPU_ADDR);
    Wire.write(0x3B);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU_ADDR, 14, true);
    AcX = Wire.read() << 8 |
        Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
    AcY = Wire.read() << 8 |
        Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
    AcZ = Wire.read() << 8 |
        Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
    Tmp = Wire.read() << 8 |
        Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
    GyX = Wire.read() << 8 |
        Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
    GyY = Wire.read() << 8 |
        Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
    GyZ = Wire.read() << 8 |
        Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)

    float gyro_scale = 1;
    if (false) gyro_scale = 131;
    if (true) gyro_scale = 65.5;
    if (false) gyro_scale = 32.8;
    if (false) gyro_scale = 16.4;

    float gyroX = ((float)GyX) / gyro_scale;
    float gyroY = ((float)GyY) / gyro_scale;
    float gyroZ = ((float)GyZ) / gyro_scale;
```

```
    gyroX *= (0.017453293F);
    gyroY *= (0.017453293F);
    gyroZ *= (0.017453293F);
    return gyroY;
}

#endif
```