به نام خدا   
پروژه درس اصول رباتیکز  
استاد: دکتر محمد زارع  
دانشجو: سنا حسینی

# پلتفرم ساخت ربات آسان برای ROS2 و MicroBlock

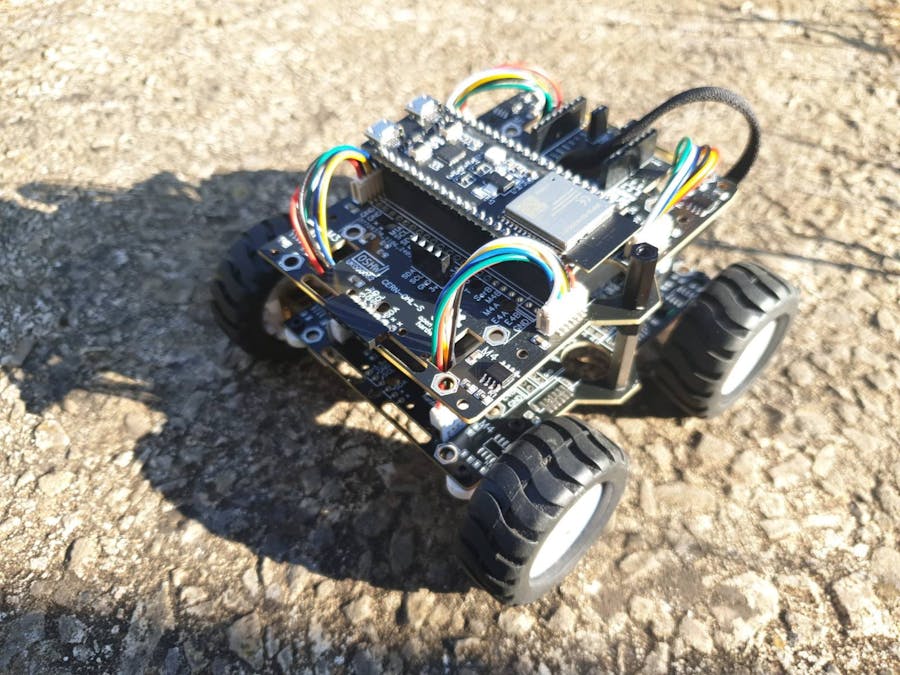
یک شاسی ربات چرخ دار با هزینه کم برای افرادی که می‌خواهند با سیستم عامل رباتیک ۲ (ROS 2) یا میکروبلاکس (MicroBlocks) آزمایش کنند.

Rosmo، یک پلتفرم منبع باز طراحی شده تا بدون نیاز به جوشکاری یا دسترسی به چاپگر سه‌بعدی، ساخته شود. Rosmoیک شاسی ربات دو یا چهار چرخ است که توسط یک ماژول Espressif ESP32-S3 کنترل می‌شود و به دو یا چهار موتور با انکودرها متصل شده و توسط یک بانک انرژی USB با هزینه کم تغذیه می‌شود. پیکربندی اصلی آن برای سادگی و قابلیت تهیه به صرفه طراحی شده است، اما به راحتی قابل توسعه است. پشتیبانی از چرخ‌های مکانوم، "چشمان" OLED و حسگرهای از جمله واحدهای اندازه‌گیری لرزش (IMUs)، حسگرهای فاصله زمان پرواز (ToF) و ماژول‌های LIDAR، همراه با یک سوکت MikroBUS و یک سوکت Qwiic برای افزودن بیشتر. پشتیبانی از دختربردهای اضافی سفارشی، با گسترش‌های پیشنهاد شده شامل یک ماژول دوربین ESP32-S3

سخت‌افزار طراحی شده تا به دو روش قابل برنامه‌ریزی باشد، با امکانات بیشتر در آینده. برای مبتدیان، پشتیبانی از محیط کدگذاری بصری مبتنی بر بلوک‌ها MicroBlocks وجود دارد که در حال حاضر کار می‌کند اما منتظر پیکربندی انکودر موتور است.

پلتفرم همچنین از یک نسخه از نرم‌افزار Linorobot2 پشتیبانی می‌کند، که سازگاری با سیستم عامل رباتیک ۲ (ROS 2) را فراهم می‌کند.

**Rosmo robot**



# قطعات سخت افزاری و نرم افزار به کار رفته برای این پروژه:

در پروژه ساخت ربات با استفاده از شاسی Rosmo، معمولاً از قطعات سخت‌افزاری و نرم‌افزاری مختلف استفاده می‌شود. در زیر لیستی از قطعات استفاده شده در این پروژه آورده شده است:

قطعات سخت‌افزاری:

1. شاسی Rosmo: این شاسی برای ساخت ربات‌های خودکار استفاده می‌شود.

2. موتورها: برای حرکت ربات نیاز به موتورهای DC یا سرووموتورها دارید.

3. چرخ‌ها: برای انتقال حرکت از موتور به زمین.

4. ماژول‌های الکترونیکی: مانند ماژول‌های سنسور، ماژول‌های کنترلر و ماژول‌های بلوتوث.

5. باتری: برای تامین انرژی به ربات.

6. سنسورها: مانند سنسورهای فاصله، سنسورهای خط و سنسورهای رنگ.

قطعات نرم‌افزاری:

1. کد برنامه‌نویسی: برنامه‌نویسی میکروکنترلر یا برد کامپیوتر برای کنترل حرکت‌ها و عملکرد ربات.

2. الگوریتم‌های هوش مصنوعی: برای اجرای وظایف هوشمندانه مانند پیگیری خط، جستجوی شئ و جلوگیری از برخورد.

3. نرم‌افزار کنترل: برای ارتباط با ربات از طریق بلوتوث یا Wi-Fi و ارسال دستورات به ربات.

در کل، این پروژه نیازمند ترکیب هوش مصنوعی، الکترونیک و برنامه‌نویسی است تا یک ربات خودکار قابل کنترل و پروگرام‌پذیر را ایجاد کنید.

# روش کار ربات ساخته شده با Rosmo :

روش کار ربات ساخته شده با شاسی Rosmo به صورت زیر است:

1. جمع‌آوری اطلاعات: ربات با استفاده از سنسورهای موجود بر روی آن، اطلاعات محیط را جمع‌آوری می‌کند. این اطلاعات شامل فاصله تا موانع، شدت نور، دما و ... است.

2. پردازش اطلاعات: پس از جمع‌آوری اطلاعات، ربات با استفاده از الگوریتم‌های هوش مصنوعی و برنامه‌نویسی میکروکنترلر، اطلاعات را پردازش می‌کند. در این مرحله، ربات تصمیم می‌گیرد که چه کاری را باید انجام دهد. به عنوان مثال، در صورتی که فاصله تا مانع کمتر از یک حداقل تعیین شده باشد، ربات تصمیم می‌گیرد که باید توقف کند و یا به سمت چپ یا راست حرکت کند.

3. کنترل حرکت: پس از پردازش اطلاعات، ربات با استفاده از موتورهای خود، حرکت خود را کنترل می‌کند. برای مثال، اگر ربات تصمیم گرفت که باید به سمت چپ حرکت کند، موتورهای سمت چپ را فعال کرده و موتورهای سمت راست را غیرفعال می‌کند.

4. تشخیص و جلوگیری از برخورد: در صورتی که ربات به مانع برخورد کند، با استفاده از سنسورهای خود، به سرعت تشخیص می‌دهد و تلاش می‌کند تا از برخورد جلوگیری کند. برای مثال، در صورتی که فاصله تا مانع کمتر از یک حداقل تعیین شده باشد، ربات تصمیم می‌گیرد که باید توقف کند و یا به سمت چپ یا راست حرکت کند.

5. ارسال داده‌ها: در صورت نیاز، ربات می‌تواند داده‌های جمع‌آوری شده را به دستگاه دیگری، مانند کامپیوتر یا تلفن همراه، ارسال کند.

6. پایان عملیات: در صورت پایان عملیات، ربات به حالت استاندارد خود بازمی‌گردد و منتظر دستورات جدید می‌شود.

به طور خلاصه، ربات با استفاده از سنسورهای خود، اطلاعات محیط را جمع‌آوری کرده و با استفاده از الگوریتم‌های هوش مصنوعی و برنامه‌نویسی میکروکنترلر، تصمیمات لازم را برای کنترل حرکت خود می‌گیرد.

# مراحل اجرای پروژه:

مراحل اجرای پروژه ساخت ربات با استفاده از شاسی Rosmo عموماً به صورت زیر است:

## 1. طراحی و ساخت سخت‌افزار:

- ابتدا باید شاسی Rosmo را آماده کنید و قطعات سخت‌افزاری مانند موتورها، چرخ‌ها، باتری، ماژول‌های الکترونیکی و سنسورها را به آن متصل کنید.

- بهتر است در این مرحله از نقشه‌ها و دستورالعمل‌های تولید کننده استفاده کنید.

## 2. برنامه‌نویسی نرم‌افزار:

- برای کنترل حرکت‌ها و عملکردهای ربات نیاز به برنامه‌نویسی میکروکنترلر یا برد کامپیوتر دارید.

- باید کدهای برنامه‌نویسی را بر اساس وظایف مورد انتظار ربات تهیه کنید. این شامل الگوریتم‌های هوش مصنوعی، کنترل حرکت، و ارتباط با ماژول‌های الکترونیکی است.

## 3. تست و عیب‌یابی:

- پس از آماده‌سازی سخت‌افزار و نرم‌افزار، باید ربات را تست کرده و عملکرد آن را بررسی کنید.

- در صورت وجود مشکلات، باید عیب‌یابی کنید و تغییرات لازم را اعمال کنید.

## 4. بهینه‌سازی و توسعه:

- پس از تست و عیب‌یابی، ممکن است نیاز به بهینه‌سازی و توسعه داشته باشید تا عملکرد و قابلیت‌های ربات را بهبود بخشید.

## 5. استفاده و ادامه توسعه:

- پس از اطمینان از عملکرد صحیح ربات، می‌توانید آن را به کار بگیرید و در صورت نیاز، آن را برای وظایف جدید گسترش دهید.

با توجه به پیچیدگی پروژه، همکاری با تیم‌های الکترونیک، برنامه‌نویسی و هوش مصنوعی می‌تواند به اجرای موفق‌تر پروژه کمک کند.

# **Code:**

**Adc\_calibrate**

#include <Arduino.h>

#include <driver/dac.h>

// Based on original work from Helmut Weber (https://github.com/MacLeod-D/ESP32-ADC)

// that he described at https://esp32.com/viewtopic.php?f=19&t=2881&start=30#p47663

// Modified with bug-fixed by Henry Cheung

//

// Build a ESP32 ADC Lookup table to correct ESP32 ADC linearity issue

// Run this sketch to build your own LUT for each of your ESP32, copy and paste the

// generated LUT to your sketch for using it, see example sketch on how to use it

//

// Version 2.0 - switch to use analogRead() instead of esp-idf function adcStart()

// Version 1.0 - original adoptation and bug fix based on Helmut Weber code

// #define GRAPH // uncomment this for print on Serial Plotter

// #define FLOAT\_LUT // uncomment this if you need float LUT

#include <micro\_ros\_platformio.h>

#include <stdio.h>

#include <rcl/rcl.h>

#include <rcl/error\_handling.h>

#include <rclc/rclc.h>

#include <rclc/executor.h>

#include <nav\_msgs/msg/odometry.h>

#include <sensor\_msgs/msg/imu.h>

#include <sensor\_msgs/msg/magnetic\_field.h>

#include <sensor\_msgs/msg/battery\_state.h>

#include <sensor\_msgs/msg/range.h>

#include <geometry\_msgs/msg/twist.h>

#include <geometry\_msgs/msg/vector3.h>

#include "config.h"

#include "syslog.h"

#include "motor.h"

#include "kinematics.h"

#include "pid.h"

#include "odometry.h"

#include "imu.h"

#include "mag.h"

#define ENCODER\_USE\_INTERRUPTS

#define ENCODER\_OPTIMIZE\_INTERRUPTS

#include "encoder.h"

#include "battery.h"

#include "range.h"

#include "lidar.h"

#include "wifis.h"

#include "ota.h"

#include "pwm.h"

#ifdef WDT\_TIMEOUT

#include <esp\_task\_wdt.h>

#endif

#ifndef BAUDRATE

#define BAUDRATE 115200

#endif

#define ADC\_PIN BATTERY\_PIN // GPIO 35 = A7, uses any valid Ax pin as you wish

float Results[4097];

float Res2[4096\*5];

void dumpResults() {

for (int i=0; i<4096; i++) {

if (i % 16 == 0) {

Serial.println();

Serial.print(i); Serial.print(" - ");

}

Serial.print(Results[i], 2); Serial.print(", ");

}

Serial.println();

}

void dumpRes2() {

Serial.println(F("Dump Res2 data..."));

for (int i=0; i<(5\*4096); i++) {

if (i % 16 == 0) {

Serial.println(); Serial.print(i); Serial.print(" - ");

}

Serial.print(Res2[i],3); Serial.print(", ");

}

Serial.println();

}

void setup() {

#ifdef BOARD\_INIT // board specific setup

BOARD\_INIT;

#endif

Serial.begin(BAUDRATE);

pinMode(LED\_PIN, OUTPUT);

dac\_output\_enable(DAC\_CHANNEL\_1); // pin 25

dac\_output\_voltage(DAC\_CHANNEL\_1, 0);

analogReadResolution(12);

delay(1000);

}

void loop() {

Serial.print(F("Test Linearity "));

for (int j=0; j<500; j++) {

if (j % 100 == 0) Serial.print(".");

for (int i=0;i<256;i++) {

dac\_output\_voltage(DAC\_CHANNEL\_1, (i & 0xff));

delayMicroseconds(100);

Results[i\*16]=0.9\*Results[i\*16] + 0.1\*analogRead(ADC\_PIN);

}

}

Serial.println();

// dumpResults();

Serial.println(F("Calculate interpolated values .."));

Results[4096] = 4095.0;

for (int i=0; i<256; i++) {

for (int j=1; j<16; j++) {

Results[i\*16+j] = Results[i\*16] + (Results[(i+1)\*16] - Results[(i)\*16])\*(float)j / (float)16.0;

}

}

// dumpResults();

Serial.println(F("Generating LUT .."));

for (int i=0; i<4096; i++) {

Results[i]=0.5 + Results[i];

}

// dumpResults();

Results[4096]=4095.5000;

for (int i=0; i<4096; i++) {

for (int j=0; j<5; j++) {

Res2[i\*5+j] = Results[i] + (Results[(i+1)] - Results[i]) \* (float)j / (float)10.0;

}

}

// dumpRes2();

for (int i=1; i<4096; i++) {

int index;

float minDiff=99999.0;

for (int j=0; j<(5\*4096); j++) {

float diff=fabs((float)(i) - Res2[j]);

if(diff<minDiff) {

minDiff=diff;

index=j;

}

}

Results[i]=(float)index;

}

// dumpResults();

for (int i=0; i<(4096); i++) {

Results[i]/=5;

}

#ifdef GRAPH

while(1) {

for (int i=2; i<256; i++) {

dac\_output\_voltage(DAC\_CHANNEL\_1, (i & 0xff));

delayMicroseconds(100);

float r = Results[analogRead(ADC\_PIN)];

Serial.print(i\*16); Serial.print(" "); Serial.println(r);

}

}

#else

Serial.println();

#ifdef FLOAT\_LUT

Serial.println("const float ADC\_LUT[4096] = { 0,");

for (int i=1; i<4095; i++) {

Serial.print(Results[i],4); Serial.print(",");

if ((i%15)==0) Serial.println();

}

Serial.println(Results[4095]);

Serial.println("};");

#else

Serial.println("const int16\_t ADC\_LUT[4096] = { 0,");

for (int i=1; i<4095; i++) {

Serial.print((int)Results[i]); Serial.print(",");

if ((i%15)==0) Serial.println();

}

Serial.println((int)Results[4095]);

Serial.println("};");

#endif

while(1);

#endif

}

**Calibration code**

// Copyright (c) 2021 Juan Miguel Jimeno

//

// Licensed under the Apache License, Version 2.0 (the "License");

// you may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

#include <Arduino.h>

#include "config.h"

#include "motor.h"

#define ENCODER\_USE\_INTERRUPTS

#define ENCODER\_OPTIMIZE\_INTERRUPTS

#include "encoder.h"

#include "kinematics.h"

#ifndef BAUDRATE

#define BAUDRATE 9600

#endif

#define SAMPLE\_TIME 10 //s

Encoder motor1\_encoder(MOTOR1\_ENCODER\_A, MOTOR1\_ENCODER\_B, COUNTS\_PER\_REV1, MOTOR1\_ENCODER\_INV);

Encoder motor2\_encoder(MOTOR2\_ENCODER\_A, MOTOR2\_ENCODER\_B, COUNTS\_PER\_REV2, MOTOR2\_ENCODER\_INV);

Encoder motor3\_encoder(MOTOR3\_ENCODER\_A, MOTOR3\_ENCODER\_B, COUNTS\_PER\_REV3, MOTOR3\_ENCODER\_INV);

Encoder motor4\_encoder(MOTOR4\_ENCODER\_A, MOTOR4\_ENCODER\_B, COUNTS\_PER\_REV4, MOTOR4\_ENCODER\_INV);

Motor motor1\_controller(PWM\_FREQUENCY, PWM\_BITS, MOTOR1\_INV, MOTOR1\_PWM, MOTOR1\_IN\_A, MOTOR1\_IN\_B);

Motor motor2\_controller(PWM\_FREQUENCY, PWM\_BITS, MOTOR2\_INV, MOTOR2\_PWM, MOTOR2\_IN\_A, MOTOR2\_IN\_B);

Motor motor3\_controller(PWM\_FREQUENCY, PWM\_BITS, MOTOR3\_INV, MOTOR3\_PWM, MOTOR3\_IN\_A, MOTOR3\_IN\_B);

Motor motor4\_controller(PWM\_FREQUENCY, PWM\_BITS, MOTOR4\_INV, MOTOR4\_PWM, MOTOR4\_IN\_A, MOTOR4\_IN\_B);

Kinematics kinematics(

Kinematics::LINO\_BASE,

MOTOR\_MAX\_RPM,

MAX\_RPM\_RATIO,

MOTOR\_OPERATING\_VOLTAGE,

MOTOR\_POWER\_MAX\_VOLTAGE,

WHEEL\_DIAMETER,

LR\_WHEELS\_DISTANCE

);

long long int counts\_per\_rev[4];

int total\_motors = 4;

Motor \*motors[4] = {&motor1\_controller, &motor2\_controller, &motor3\_controller, &motor4\_controller};

Encoder \*encoders[4] = {&motor1\_encoder, &motor2\_encoder, &motor3\_encoder, &motor4\_encoder};

String labels[4] = {"FRONT LEFT - M1: ", "FRONT RIGHT - M2: ", "REAR LEFT - M3: ", "REAR RIGHT - M4: "};

void printSummary()

{

Serial.println("\r\n================MOTOR ENCODER READINGS================");

Serial.print(labels[0]);

Serial.print(encoders[0]->read());

Serial.print(" ");

Serial.print(labels[1]);

Serial.println(encoders[1]->read());

Serial.print(labels[2]);

Serial.print(encoders[2]->read());

Serial.print(" ");

Serial.print(labels[3]);

Serial.println(encoders[3]->read());

Serial.println("");

Serial.println("================COUNTS PER REVOLUTION=================");

Serial.print(labels[0]);

Serial.print(counts\_per\_rev[0]);

Serial.print(" ");

Serial.print(labels[1]);

Serial.println(counts\_per\_rev[1]);

Serial.print(labels[2]);

Serial.print(counts\_per\_rev[2]);

Serial.print(" ");

Serial.print(labels[3]);

Serial.println(counts\_per\_rev[3]);

Serial.println("");

Serial.println("====================MAX VELOCITIES====================");

float max\_rpm = kinematics.getMaxRPM();

Kinematics::velocities max\_linear = kinematics.getVelocities(max\_rpm, max\_rpm, max\_rpm, max\_rpm);

Kinematics::velocities max\_angular = kinematics.getVelocities(-max\_rpm, max\_rpm,-max\_rpm, max\_rpm);

Serial.print("Linear Velocity: +- ");

Serial.print(max\_linear.linear\_x);

Serial.println(" m/s");

Serial.print("Angular Velocity: +- ");

Serial.print(max\_angular.angular\_z);

Serial.println(" rad/s");

}

void sampleMotors(bool show\_summary)

{

if(Kinematics::LINO\_BASE == Kinematics::DIFFERENTIAL\_DRIVE)

{

total\_motors = 2;

}

float measured\_voltage = constrain(MOTOR\_POWER\_MEASURED\_VOLTAGE, 0, MOTOR\_OPERATING\_VOLTAGE);

float scaled\_max\_rpm = ((measured\_voltage / MOTOR\_OPERATING\_VOLTAGE) \* MOTOR\_MAX\_RPM);

float total\_rev = scaled\_max\_rpm \* (SAMPLE\_TIME / 60.0);

for(int i=0; i<total\_motors; i++)

{

Serial.print("SPINNING ");

Serial.print(labels[i]);

unsigned long start\_time = micros();

unsigned long last\_status = micros();

encoders[i]->write(0);

while(true)

{

if(micros() - start\_time >= SAMPLE\_TIME \* 1000000)

{

motors[i]->spin(0);

Serial.println("");

break;

}

if(micros() - last\_status >= 1000000)

{

last\_status = micros();

Serial.print(".");

}

motors[i]->spin(PWM\_MAX);

}

counts\_per\_rev[i] = encoders[i]->read() / total\_rev;

}

if(show\_summary)

printSummary();

}

void setup()

{

#ifdef BOARD\_INIT

BOARD\_INIT;

#endif

Serial.begin(BAUDRATE);

while (!Serial) {

}

Serial.println("Sampling process will spin the motors at its maximum RPM.");

Serial.println("Please ensure that the robot is ELEVATED and there are NO OBSTRUCTIONS to the wheels.");

Serial.println("");

Serial.println("Type 'spin' and press enter to spin the motors.");

Serial.println("Type 'sample' and press enter to spin the motors with motor summary.");

Serial.println("Press enter to clear command.");

Serial.println("");

}

void loop()

{

static String cmd = "";

while (Serial.available())

{

char character = Serial.read();

cmd.concat(character);

Serial.print(character);

delay(1);

if(character == '\r' and cmd.equals("spin\r"))

{

cmd = "";

Serial.println("\r\n");

sampleMotors(0);

}

else if(character == '\r' and cmd.equals("sample\r"))

{

cmd = "";

Serial.println("\r\n");

sampleMotors(1);

}

else if(character == '\r')

{

Serial.println("");

cmd = "";

}

}

}

**Test\_acc code**

// Copyright (c) 2021 Juan Miguel Jimeno

// Copyright (c) 2023 Thomas Chou

//

// Licensed under the Apache License, Version 2.0 (the "License");

// you may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

#include <Arduino.h>

#include <micro\_ros\_platformio.h>

#include <stdio.h>

#include <nav\_msgs/msg/odometry.h>

#include <sensor\_msgs/msg/imu.h>

#include <sensor\_msgs/msg/magnetic\_field.h>

#include <sensor\_msgs/msg/battery\_state.h>

#include <sensor\_msgs/msg/range.h>

#include <geometry\_msgs/msg/twist.h>

#include <geometry\_msgs/msg/vector3.h>

#include "config.h"

#include "syslog.h"

#include "motor.h"

#include "kinematics.h"

#include "pid.h"

#include "odometry.h"

#include "imu.h"

#include "mag.h"

#define ENCODER\_USE\_INTERRUPTS

#define ENCODER\_OPTIMIZE\_INTERRUPTS

#include "encoder.h"

#include "battery.h"

#include "range.h"

#include "lidar.h"

#include "wifis.h"

#include "ota.h"

#include "pwm.h"

#ifndef BAUDRATE

#define BAUDRATE 115200

#endif

// #define DEBUG

nav\_msgs\_\_msg\_\_Odometry odom\_msg;

sensor\_msgs\_\_msg\_\_Imu imu\_msg;

sensor\_msgs\_\_msg\_\_MagneticField mag\_msg;

geometry\_msgs\_\_msg\_\_Twist twist\_msg;

sensor\_msgs\_\_msg\_\_BatteryState battery\_msg;

sensor\_msgs\_\_msg\_\_Range range\_msg;

void setLed(int value)

{

#ifdef LED\_PIN

digitalWrite(LED\_PIN, value);

#endif

}

int getLed(void)

{

#ifdef LED\_PIN

return digitalRead(LED\_PIN);

#else

return 0;

#endif

}

void initLed(void)

{

#ifdef LED\_PIN

pinMode(LED\_PIN, OUTPUT);

#endif

}

Encoder motor1\_encoder(MOTOR1\_ENCODER\_A, MOTOR1\_ENCODER\_B, COUNTS\_PER\_REV1, MOTOR1\_ENCODER\_INV);

Encoder motor2\_encoder(MOTOR2\_ENCODER\_A, MOTOR2\_ENCODER\_B, COUNTS\_PER\_REV2, MOTOR2\_ENCODER\_INV);

Encoder motor3\_encoder(MOTOR3\_ENCODER\_A, MOTOR3\_ENCODER\_B, COUNTS\_PER\_REV3, MOTOR3\_ENCODER\_INV);

Encoder motor4\_encoder(MOTOR4\_ENCODER\_A, MOTOR4\_ENCODER\_B, COUNTS\_PER\_REV4, MOTOR4\_ENCODER\_INV);

Motor motor1\_controller(PWM\_FREQUENCY, PWM\_BITS, MOTOR1\_INV, MOTOR1\_PWM, MOTOR1\_IN\_A, MOTOR1\_IN\_B);

Motor motor2\_controller(PWM\_FREQUENCY, PWM\_BITS, MOTOR2\_INV, MOTOR2\_PWM, MOTOR2\_IN\_A, MOTOR2\_IN\_B);

Motor motor3\_controller(PWM\_FREQUENCY, PWM\_BITS, MOTOR3\_INV, MOTOR3\_PWM, MOTOR3\_IN\_A, MOTOR3\_IN\_B);

Motor motor4\_controller(PWM\_FREQUENCY, PWM\_BITS, MOTOR4\_INV, MOTOR4\_PWM, MOTOR4\_IN\_A, MOTOR4\_IN\_B);

PID motor1\_pid(PWM\_MIN, PWM\_MAX, K\_P, K\_I, K\_D);

PID motor2\_pid(PWM\_MIN, PWM\_MAX, K\_P, K\_I, K\_D);

PID motor3\_pid(PWM\_MIN, PWM\_MAX, K\_P, K\_I, K\_D);

PID motor4\_pid(PWM\_MIN, PWM\_MAX, K\_P, K\_I, K\_D);

Kinematics kinematics(

Kinematics::LINO\_BASE,

MOTOR\_MAX\_RPM,

MAX\_RPM\_RATIO,

MOTOR\_OPERATING\_VOLTAGE,

MOTOR\_POWER\_MAX\_VOLTAGE,

WHEEL\_DIAMETER,

LR\_WHEELS\_DISTANCE

);

Odometry odometry;

IMU imu;

MAG mag;

unsigned total\_motors = 4;

void setup()

{

Serial.begin(BAUDRATE);

initLed();

#ifdef BOARD\_INIT // board specific setup

BOARD\_INIT;

#endif

initPwm();

motor1\_controller.begin();

motor2\_controller.begin();

motor3\_controller.begin();

motor4\_controller.begin();

initWifis();

initOta();

imu.init();

mag.init();

initBattery();

initRange();

if(Kinematics::LINO\_BASE == Kinematics::DIFFERENTIAL\_DRIVE)

{

total\_motors = 2;

}

motor1\_encoder.getRPM();

motor2\_encoder.getRPM();

motor3\_encoder.getRPM();

motor4\_encoder.getRPM();

#ifdef BOARD\_INIT\_LATE // board specific setup

BOARD\_INIT\_LATE;

#endif

syslog(LOG\_INFO, "%s Ready %lu", \_\_FUNCTION\_\_, millis());

delay(2000);

}

unsigned runs = 12;

const unsigned ticks = 20;

const float dt = ticks \* 0.001;

const unsigned run\_time = 1000; // 1s

const unsigned buf\_size = run\_time / ticks \* 4;

Kinematics::velocities buf[buf\_size];

float batt[buf\_size];

float imu\_max\_acc\_x, imu\_min\_acc\_x;

unsigned idx = 0;

void record(unsigned n) {

unsigned i;

for (i = 0; i < n; i++, idx++) {

float rpm1 = motor1\_encoder.getRPM();

float rpm2 = motor2\_encoder.getRPM();

float rpm3 = motor3\_encoder.getRPM();

float rpm4 = motor4\_encoder.getRPM();

imu\_msg = imu.getData();

battery\_msg = getBattery();

float imu\_acc\_x = imu\_msg.linear\_acceleration.x;

if (imu\_acc\_x > imu\_max\_acc\_x) imu\_max\_acc\_x = imu\_acc\_x;

if (imu\_acc\_x < imu\_min\_acc\_x) imu\_min\_acc\_x = imu\_acc\_x;

if (idx < buf\_size) {

buf[idx] = kinematics.getVelocities(rpm1, rpm2, rpm3, rpm4);

batt[idx] = battery\_msg.voltage;

}

delay(ticks);

runWifis();

runOta();

}

}

void dump\_record(void) {

float max\_vel\_x = 0, min\_vel\_x = 0, max\_acc\_x = 0, min\_acc\_x = 0;

float max\_vel\_y = 0, min\_vel\_y = 0, max\_acc\_y = 0, min\_acc\_y = 0;

float max\_vel\_z = 0, min\_vel\_z = 0, max\_acc\_z = 0, min\_acc\_z = 0;

float dist = 0;

float avg\_batt = 0, min\_batt = batt[0];

for (idx = 0; idx < buf\_size; idx++) {

float vel\_x = buf[idx].linear\_x;

float vel\_y = buf[idx].linear\_y;

float vel\_z = buf[idx].angular\_z;

if (vel\_x > max\_vel\_x) max\_vel\_x = vel\_x;

if (vel\_x < min\_vel\_x) min\_vel\_x = vel\_x;

if (vel\_y > max\_vel\_y) max\_vel\_y = vel\_y;

if (vel\_y < min\_vel\_y) min\_vel\_y = vel\_y;

if (vel\_z > max\_vel\_z) max\_vel\_z = vel\_z;

if (vel\_z < min\_vel\_z) min\_vel\_z = vel\_z;

if (min\_batt > batt[idx]) min\_batt = batt[idx];

avg\_batt += batt[idx];

#ifdef DEBUG

Serial.printf("%04d VEL %6.2f %6.2f m/s %6.2f rad/s BAT %5.2fV\n",

idx, vel\_x, vel\_y, vel\_z, batt[idx]);

syslog(LOG\_INFO, "%04d VEL %6.2f %6.2f m/s %6.2f rad BAT %5.2fV/s",

idx, vel\_x, vel\_y, vel\_z, batt[idx]);

#endif

}

for (idx = 0; idx < buf\_size; idx++) {

unsigned prev = idx ? (idx -1) : 0;

float acc\_x = (buf[idx].linear\_x - buf[prev].linear\_x) / dt;

float acc\_y = (buf[idx].linear\_y - buf[prev].linear\_y) / dt;

float acc\_z = (buf[idx].angular\_z - buf[prev].angular\_z) / dt;

if (acc\_x > max\_acc\_x) max\_acc\_x = acc\_x;

if (acc\_x < min\_acc\_x) min\_acc\_x = acc\_x;

if (acc\_y > max\_acc\_y) max\_acc\_y = acc\_y;

if (acc\_y < min\_acc\_y) min\_acc\_y = acc\_y;

if (acc\_z > max\_acc\_z) max\_acc\_z = acc\_z;

if (acc\_z < min\_acc\_z) min\_acc\_z = acc\_z;

#ifdef DEBUG

Serial.printf("%04d ACC %6.2f %6.2f m/s2 %6.2f rad/s2 BAT %5.2fV\n",

idx, acc\_x, acc\_y, acc\_z, batt[idx]);

syslog(LOG\_INFO, "%04d ACC %6.2f %6.2f m/s2 %6.2f rad/s2 BAT %5.2fV",

idx, acc\_x, acc\_y, acc\_z, batt[idx]);

#endif

}

if (runs & 1) {

for (idx = buf\_size / 4; idx < buf\_size / 2; idx++)

dist += buf[idx].linear\_x \* dt;

for (idx = 0; idx < buf\_size / 4; idx++)

if (buf[idx].linear\_x > max\_vel\_x \* 0.9) break;

} else {

for (idx = buf\_size / 4; idx < buf\_size / 2; idx++)

dist += buf[idx].angular\_z \* dt;

for (idx = 0; idx < buf\_size / 4; idx++)

if (buf[idx].angular\_z > max\_vel\_z \* 0.9) break;

}

avg\_batt /= buf\_size;

Serial.printf("MAX VEL %6.2f %6.2f m/s %6.2f rad/s\n",

max\_vel\_x, max\_vel\_y, max\_vel\_z);

Serial.printf("MIN VEL %6.2f %6.2f m/s %6.2f rad/s\n",

min\_vel\_x, min\_vel\_y, min\_vel\_z);

Serial.printf("MAX ACC %6.2f %6.2f m/s2 %6.2f rad/s2\n",

max\_acc\_x, max\_acc\_y, max\_acc\_z);

Serial.printf("MIN ACC %6.2f %6.2f m/s2 %6.2f rad/s2\n",

min\_acc\_x, min\_acc\_y, min\_acc\_z);

Serial.printf("IMU ACC %6.2f %6.2f m/s2\n", imu\_max\_acc\_x, imu\_min\_acc\_x);

Serial.printf("time to 0.9x max vel %6.2f sec\n", idx \* dt);

Serial.printf("distance to stop %6.2f %s\n", dist, (runs & 1) ? "m" : "rad");

Serial.printf("BAT %5.2fV MIN %5.2fV\n", avg\_batt, min\_batt);

syslog(LOG\_INFO, "MAX VEL %6.2f %6.2f m/s %6.2f rad/s",

max\_vel\_x, max\_vel\_y, max\_vel\_z);

syslog(LOG\_INFO, "MIN VEL %6.2f %6.2f m/s %6.2f rad/s",

min\_vel\_x, min\_vel\_y, min\_vel\_z);

syslog(LOG\_INFO, "MAX ACC %6.2f %6.2f m/s2 %6.2f rad/s2",

max\_acc\_x, max\_acc\_y, max\_acc\_z);

syslog(LOG\_INFO, "MIN ACC %6.2f %6.2f m/s2 %6.2f rad/s2",

min\_acc\_x, min\_acc\_y, min\_acc\_z);

syslog(LOG\_INFO, "IMU ACC %6.2f %6.2f m/s2", imu\_max\_acc\_x, imu\_min\_acc\_x);

syslog(LOG\_INFO, "time to 0.9x max vel %6.2f sec", idx \* dt);

syslog(LOG\_INFO, "distance to stop %6.2f %s\n", dist, (runs & 1) ? "m" : "rad");

syslog(LOG\_INFO, "BAT %5.2fV MIN %5.2fV\n", avg\_batt, min\_batt);

}

void loop() {

float pwm\_max = PWM\_MAX;

float pwm\_min = -pwm\_max;

while (runs > 0) {

runs--;

idx = 0;

imu\_max\_acc\_x = 0;

imu\_min\_acc\_x = 0;

// full speed forward / spin counterclockwise

setLed(HIGH);

motor1\_controller.spin((runs & 1) ? pwm\_max : pwm\_min);

motor2\_controller.spin(pwm\_max);

motor3\_controller.spin((runs & 1) ? pwm\_max : pwm\_min);

motor4\_controller.spin(pwm\_max);

record(run\_time / ticks);

// stop

setLed(LOW);

motor1\_controller.spin(0);

motor2\_controller.spin(0);

motor3\_controller.spin(0);

motor4\_controller.spin(0);

record(run\_time / ticks);

// full speed backward / spin clockwise

setLed(HIGH);

motor1\_controller.spin((runs & 1) ? pwm\_min : pwm\_max);

motor2\_controller.spin(pwm\_min);

motor3\_controller.spin((runs & 1) ? pwm\_min : pwm\_max);

motor4\_controller.spin(pwm\_min);

record(run\_time / ticks);

// stop

setLed(LOW);

motor1\_controller.spin(0);

motor2\_controller.spin(0);

motor3\_controller.spin(0);

motor4\_controller.spin(0);

record(run\_time / ticks);

// print result

Serial.printf("MAX PWM %6.1f %6.1f\n", pwm\_max, pwm\_min);

syslog(LOG\_INFO, "MAX PWM %6.1f %6.1f", pwm\_max, pwm\_min);

dump\_record();

if ((runs & 3) == 0) {

pwm\_max /= 2;

pwm\_min /= 2;

}

}

// idle

delay(100);

runWifis();

runOta();

}

**Test\_motor code**

// Copyright (c) 2021 Juan Miguel Jimeno

// Copyright (c) 2023 Thomas Chou

//

// Licensed under the Apache License, Version 2.0 (the "License");

// you may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

#include <Arduino.h>

#include <micro\_ros\_platformio.h>

#include <stdio.h>

#include <i2cdetect.h>

#include <nav\_msgs/msg/odometry.h>

#include <sensor\_msgs/msg/imu.h>

#include <sensor\_msgs/msg/magnetic\_field.h>

#include <sensor\_msgs/msg/battery\_state.h>

#include <sensor\_msgs/msg/range.h>

#include <geometry\_msgs/msg/twist.h>

#include <geometry\_msgs/msg/vector3.h>

#include "config.h"

#include "syslog.h"

#include "motor.h"

#include "kinematics.h"

#include "pid.h"

#include "odometry.h"

#include "imu.h"

#include "mag.h"

#define ENCODER\_USE\_INTERRUPTS

#define ENCODER\_OPTIMIZE\_INTERRUPTS

#include "encoder.h"

#include "battery.h"

#include "range.h"

#include "lidar.h"

#include "wifis.h"

#include "ota.h"

#include "pwm.h"

#ifndef BAUDRATE

#define BAUDRATE 115200

#endif

nav\_msgs\_\_msg\_\_Odometry odom\_msg;

sensor\_msgs\_\_msg\_\_Imu imu\_msg;

sensor\_msgs\_\_msg\_\_MagneticField mag\_msg;

geometry\_msgs\_\_msg\_\_Twist twist\_msg;

sensor\_msgs\_\_msg\_\_BatteryState battery\_msg;

sensor\_msgs\_\_msg\_\_Range range\_msg;

void setLed(int value)

{

#ifdef LED\_PIN

digitalWrite(LED\_PIN, value);

#endif

}

int getLed(void)

{

#ifdef LED\_PIN

return digitalRead(LED\_PIN);

#else

return 0;

#endif

}

void initLed(void)

{

#ifdef LED\_PIN

pinMode(LED\_PIN, OUTPUT);

#endif

}

Encoder motor1\_encoder(MOTOR1\_ENCODER\_A, MOTOR1\_ENCODER\_B, COUNTS\_PER\_REV1, MOTOR1\_ENCODER\_INV);

Encoder motor2\_encoder(MOTOR2\_ENCODER\_A, MOTOR2\_ENCODER\_B, COUNTS\_PER\_REV2, MOTOR2\_ENCODER\_INV);

Encoder motor3\_encoder(MOTOR3\_ENCODER\_A, MOTOR3\_ENCODER\_B, COUNTS\_PER\_REV3, MOTOR3\_ENCODER\_INV);

Encoder motor4\_encoder(MOTOR4\_ENCODER\_A, MOTOR4\_ENCODER\_B, COUNTS\_PER\_REV4, MOTOR4\_ENCODER\_INV);

Motor motor1\_controller(PWM\_FREQUENCY, PWM\_BITS, MOTOR1\_INV, MOTOR1\_PWM, MOTOR1\_IN\_A, MOTOR1\_IN\_B);

Motor motor2\_controller(PWM\_FREQUENCY, PWM\_BITS, MOTOR2\_INV, MOTOR2\_PWM, MOTOR2\_IN\_A, MOTOR2\_IN\_B);

Motor motor3\_controller(PWM\_FREQUENCY, PWM\_BITS, MOTOR3\_INV, MOTOR3\_PWM, MOTOR3\_IN\_A, MOTOR3\_IN\_B);

Motor motor4\_controller(PWM\_FREQUENCY, PWM\_BITS, MOTOR4\_INV, MOTOR4\_PWM, MOTOR4\_IN\_A, MOTOR4\_IN\_B);

PID motor1\_pid(PWM\_MIN, PWM\_MAX, K\_P, K\_I, K\_D);

PID motor2\_pid(PWM\_MIN, PWM\_MAX, K\_P, K\_I, K\_D);

PID motor3\_pid(PWM\_MIN, PWM\_MAX, K\_P, K\_I, K\_D);

PID motor4\_pid(PWM\_MIN, PWM\_MAX, K\_P, K\_I, K\_D);

Kinematics kinematics(

Kinematics::LINO\_BASE,

MOTOR\_MAX\_RPM,

MAX\_RPM\_RATIO,

MOTOR\_OPERATING\_VOLTAGE,

MOTOR\_POWER\_MAX\_VOLTAGE,

WHEEL\_DIAMETER,

LR\_WHEELS\_DISTANCE

);

Odometry odometry;

IMU imu;

MAG mag;

unsigned total\_motors = 4;

void setup()

{

Serial.begin(BAUDRATE);

initLed();

#ifdef BOARD\_INIT // board specific setup

BOARD\_INIT;

#endif

initWifis();

initOta();

i2cdetect(); // default range from 0x03 to 0x77

initPwm();

motor1\_controller.begin();

motor2\_controller.begin();

motor3\_controller.begin();

motor4\_controller.begin();

imu.init();

mag.init();

initBattery();

initRange();

if(Kinematics::LINO\_BASE == Kinematics::DIFFERENTIAL\_DRIVE)

{

total\_motors = 2;

}

motor1\_encoder.getRPM();

motor2\_encoder.getRPM();

motor3\_encoder.getRPM();

motor4\_encoder.getRPM();

#ifdef BOARD\_INIT\_LATE // board specific setup

BOARD\_INIT\_LATE;

#endif

syslog(LOG\_INFO, "%s Ready %lu", \_\_FUNCTION\_\_, millis());

}

void loop() {

static unsigned tk = 0; // tick

const unsigned run\_time = 8; // run time of each motor

const unsigned cycle = run\_time \* total\_motors;

unsigned current\_motor = tk / run\_time % total\_motors;

unsigned direction = tk / cycle % 2; // 0 forward, 1 reverse

const int pwm\_max = (1 << PWM\_BITS) - 1;

static float max\_rpm, stopping;

setLed(direction ? LOW : HIGH);

motor1\_controller.spin((current\_motor == 0) ? (direction ? -pwm\_max : pwm\_max) : 0);

motor2\_controller.spin((current\_motor == 1) ? (direction ? -pwm\_max : pwm\_max) : 0);

motor3\_controller.spin((current\_motor == 2) ? (direction ? -pwm\_max : pwm\_max) : 0);

motor4\_controller.spin((current\_motor == 3) ? (direction ? -pwm\_max : pwm\_max) : 0);

delay(1000);

float current\_rpm1 = motor1\_encoder.getRPM();

float current\_rpm2 = motor2\_encoder.getRPM();

float current\_rpm3 = motor3\_encoder.getRPM();

float current\_rpm4 = motor4\_encoder.getRPM();

if (current\_motor == 0 && tk % run\_time == run\_time - 1) max\_rpm = current\_rpm1;

if (current\_motor == 1 && tk % run\_time == 0) stopping = current\_rpm1;

if (current\_motor == 1 && tk % run\_time == run\_time - 1) max\_rpm = current\_rpm2;

if (total\_motors == 2 && current\_motor == 0 && tk % run\_time == 0) stopping = current\_rpm2;

if (current\_motor == 2 && tk % run\_time == 0) stopping = current\_rpm2;

if (current\_motor == 2 && tk % run\_time == run\_time - 1) max\_rpm = current\_rpm3;

if (current\_motor == 3 && tk % run\_time == 0) stopping = current\_rpm3;

if (current\_motor == 3 && tk % run\_time == run\_time - 1) max\_rpm = current\_rpm4;

if (total\_motors == 4 && current\_motor == 0 && tk % run\_time == 0) stopping = current\_rpm4;

if (tk && tk % run\_time == 0) {

Kinematics::velocities max\_linear = kinematics.getVelocities(max\_rpm, max\_rpm, max\_rpm, max\_rpm);

Kinematics::velocities max\_angular = kinematics.getVelocities(-max\_rpm, max\_rpm,-max\_rpm, max\_rpm);

Serial.printf("MOTOR%d SPEED %6.2f m/s %6.2f rad/s STOP %6.3f m\n", current\_motor ? current\_motor : total\_motors,

max\_linear.linear\_x, max\_angular.angular\_z, max\_linear.linear\_x \* stopping / max\_rpm);

syslog(LOG\_INFO, "MOTOR%d SPEED %6.2f m/s %6.2f rad/s STOP %6.3f m\n", current\_motor ? current\_motor : total\_motors,

max\_linear.linear\_x, max\_angular.angular\_z, max\_linear.linear\_x \* stopping / max\_rpm);

}

Serial.printf("MOTOR%d %s RPM %8.1f %8.1f %8.1f %8.1f\n",

current\_motor + 1, direction ? "REV" : "FWD",

current\_rpm1, current\_rpm2, current\_rpm3, current\_rpm4);

syslog(LOG\_INFO, "MOTOR%d %s RPM %8.1f %8.1f %8.1f %8.1f\n",

current\_motor + 1, direction ? "REV" : "FWD",

current\_rpm1, current\_rpm2, current\_rpm3, current\_rpm4);

tk++;

runWifis();

runOta();

}

**Test\_sensors code**

// Copyright (c) 2021 Juan Miguel Jimeno

//

// Licensed under the Apache License, Version 2.0 (the "License");

// you may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

#include <Arduino.h>

#include <micro\_ros\_platformio.h>

#include <stdio.h>

#include <i2cdetect.h>

#include <nav\_msgs/msg/odometry.h>

#include <sensor\_msgs/msg/imu.h>

#include <sensor\_msgs/msg/magnetic\_field.h>

#include <sensor\_msgs/msg/battery\_state.h>

#include <sensor\_msgs/msg/range.h>

#include <geometry\_msgs/msg/twist.h>

#include <geometry\_msgs/msg/vector3.h>

#include "config.h"

#include "syslog.h"

#include "motor.h"

#include "kinematics.h"

#include "pid.h"

#include "odometry.h"

#include "imu.h"

#include "mag.h"

#define ENCODER\_USE\_INTERRUPTS

#define ENCODER\_OPTIMIZE\_INTERRUPTS

#include "encoder.h"

#include "battery.h"

#include "range.h"

#include "lidar.h"

#include "wifis.h"

#include "ota.h"

#include "pwm.h"

#ifndef BAUDRATE

#define BAUDRATE 115200

#endif

nav\_msgs\_\_msg\_\_Odometry odom\_msg;

sensor\_msgs\_\_msg\_\_Imu imu\_msg;

sensor\_msgs\_\_msg\_\_MagneticField mag\_msg;

geometry\_msgs\_\_msg\_\_Twist twist\_msg;

sensor\_msgs\_\_msg\_\_BatteryState battery\_msg;

sensor\_msgs\_\_msg\_\_Range range\_msg;

Odometry odometry;

IMU imu;

MAG mag;

void setup()

{

Serial.begin(BAUDRATE);

#ifdef BOARD\_INIT // board specific setup

BOARD\_INIT;

#endif

initWifis();

initOta();

i2cdetect(); // default range from 0x03 to 0x77

initPwm();

imu.init();

mag.init();

initBattery();

initRange();

#ifdef BOARD\_INIT\_LATE // board specific setup

BOARD\_INIT\_LATE

#endif

syslog(LOG\_INFO, "%s Ready %lu", \_\_FUNCTION\_\_, millis());

}

void loop() {

delay(1000);

imu\_msg = imu.getData();

mag\_msg = mag.getData();

#ifdef MAG\_BIAS

const float mag\_bias[3] = MAG\_BIAS;

mag\_msg.magnetic\_field.x -= mag\_bias[0];

mag\_msg.magnetic\_field.y -= mag\_bias[1];

mag\_msg.magnetic\_field.z -= mag\_bias[2];

#endif

battery\_msg = getBattery();

range\_msg = getRange();

Serial.printf("ACC %5.2f %5.2f %5.2f GYR %5.2f %5.2f %5.2f MAG %5.2f %5.2f %5.2f\n"

" BAT %5.2fV RANGE %5.2fm\n",

imu\_msg.linear\_acceleration.x, imu\_msg.linear\_acceleration.y, imu\_msg.linear\_acceleration.z,

imu\_msg.angular\_velocity.x, imu\_msg.angular\_velocity.y, imu\_msg.angular\_velocity.x,

mag\_msg.magnetic\_field.x \* 1000000, mag\_msg.magnetic\_field.y \* 1000000,

mag\_msg.magnetic\_field.z \* 1000000, battery\_msg.voltage, range\_msg.range);

syslog(LOG\_INFO, "ACC %5.2f %5.2f %5.2f GYR %5.2f %5.2f %5.2f MAG %5.2f %5.2f %5.2f"

" BAT %5.2fV RANGE %5.2fm",

imu\_msg.linear\_acceleration.x, imu\_msg.linear\_acceleration.y, imu\_msg.linear\_acceleration.z,

imu\_msg.angular\_velocity.x, imu\_msg.angular\_velocity.y, imu\_msg.angular\_velocity.x,

mag\_msg.magnetic\_field.x \* 1000000, mag\_msg.magnetic\_field.y \* 1000000,

mag\_msg.magnetic\_field.z \* 1000000, battery\_msg.voltage, range\_msg.range);

runWifis();

runOta();

}

# برخی توابع استفاده شده:

در این کد، توابع زیر استفاده شده‌اند:

1. adc1\_config\_width():

- این تابع برای تنظیم دقت خواندن ADC به 12 بیت استفاده می‌شود.

2. adc1\_config\_channel\_atten():

- این تابع برای تنظیم ویژگی‌های ورودی ADC مانند ولتاژ مرجع (attenuation) استفاده می‌شود.

3. adc1\_get\_raw():

- این تابع برای خواندن مقدار ADC خام (raw) استفاده می‌شود.

4. dac\_output\_enable():

- این تابع برای فعال کردن ورودی DAC استفاده می‌شود.

5. dac\_output\_voltage():

- این تابع برای تنظیم ولتاژ خروجی DAC استفاده می‌شود.

6. vTaskDelay():

- این تابع برای تاخیر دادن اجرای برنامه به مدت زمان مشخص استفاده می‌شود.

7. esp\_adc\_cal\_characterize():

- این تابع برای کالیبراسیون ADC با استفاده از Lookup table استفاده می‌شود.

8. esp\_adc\_cal\_get\_characteristics():

- این تابع برای دریافت ویژگی‌های کالیبراسیون ADC استفاده می‌شود.

این توابع به شما کمک می‌کنند تا ADC و DAC ESP32 را به درستی پیکربندی کرده و کالیبره کنید.

# نتیجه گیری:

در این پروژه، ما ابتدا یک ESP32 را برنامه‌ریزی کردیم تا ولتاژ ورودی از یک سنسور را با استفاده از ADC خوانده و سپس ولتاژ خروجی را با استفاده از DAC تنظیم کردیم.

برای خواندن ولتاژ ورودی، ما از تابع ADC.read() برای خواندن مقدار آنالوگ استفاده کردیم. سپس مقدار خوانده شده را به ولتاژ معادل تبدیل کردیم.

برای تنظیم ولتاژ خروجی، ما از تابع DAC.write() برای تنظیم مقدار ولتاژ خروجی استفاده کردیم.

این پروژه نشان می‌دهد که ESP32 به عنوان یک میکروکنترلر قدرتمند با قابلیت‌های ADC و DAC، قابلیت‌های کنترل و نمونه‌برداری با دقت بالا را داراست. این قابلیت‌ها مناسب برای کاربردهای مختلفی از جمله سنسورهای آنالوگ و کنترل خروجی‌های آنالوگ می‌باشد.

# **منابع**

<https://www.hackster.io/Rosmo/rosmo-easy-to-build-robot-platform-for-ros-2-and-microblock-9da8eb>

https://t.me/GPT4Telegrambot