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

پلتفرم ساخت ربات آسان برای ROS2 و MicroBlock

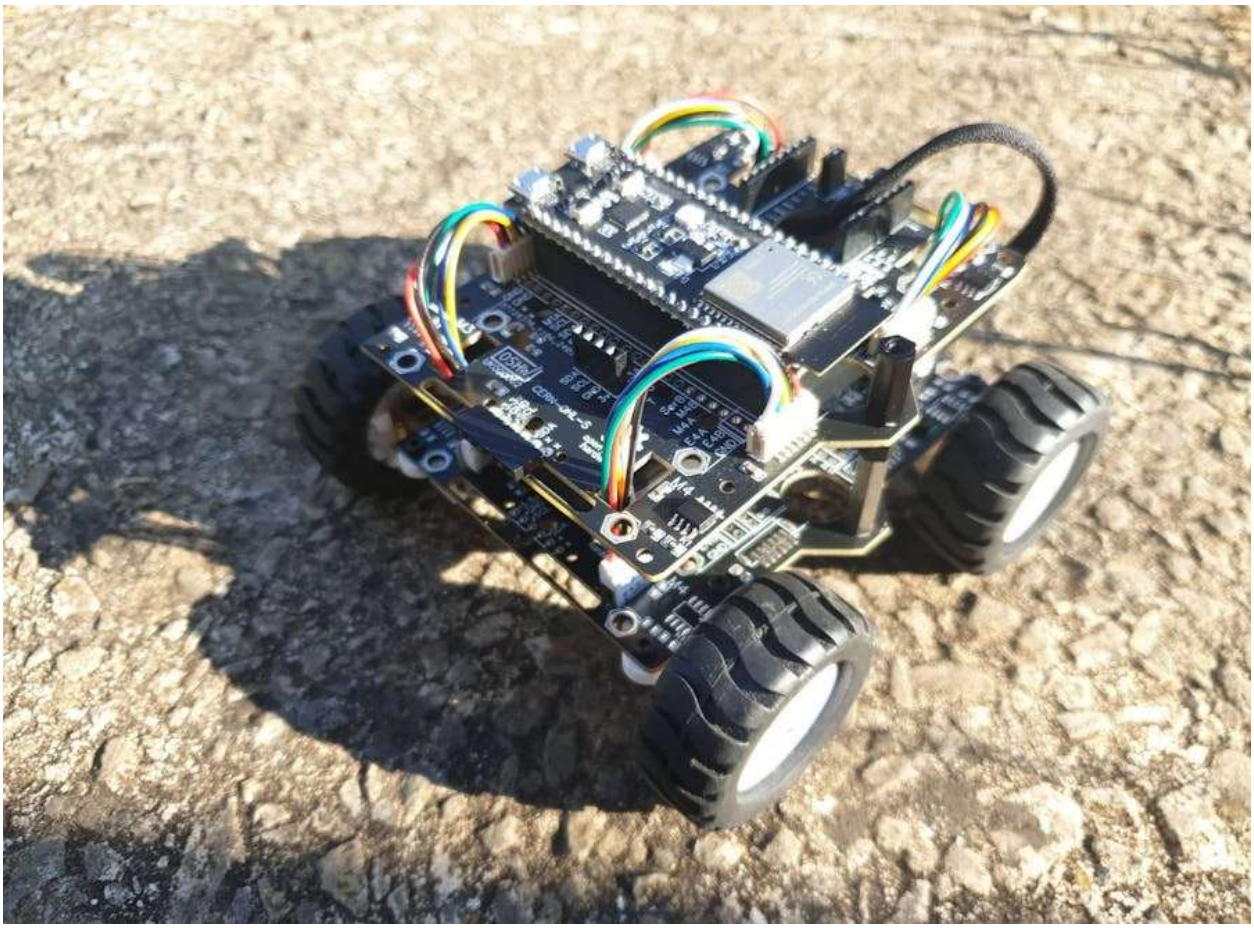
یک شاسی ربات چرخ دار با هزینه کم برای افرادی که می‌خواهند با سیستم عامل رباتیک ۲ (ROS 2) یا میکروبلاکس (MicroBlocks) آزمایش کنند.

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 سروموتورها 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

```
#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