بسمه تعالی

پروژه درس سیستم های دیجیتال 2

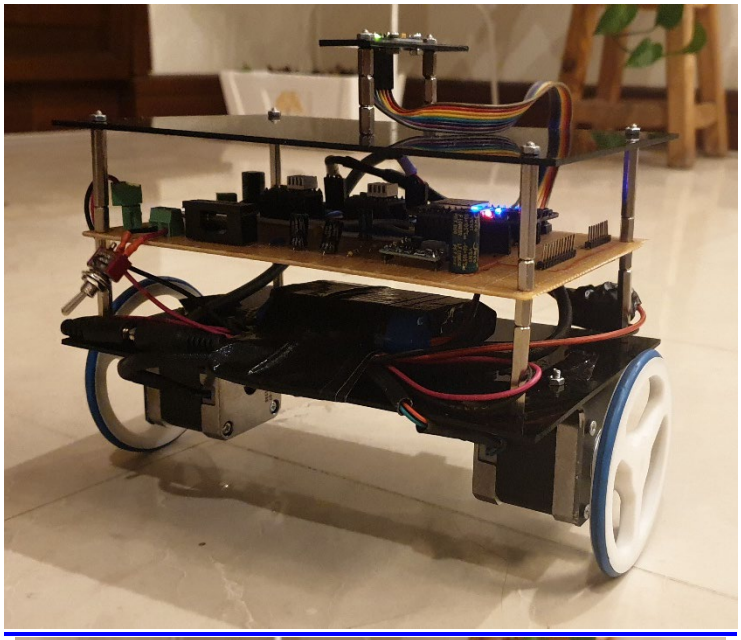
ربات تعادلی
(Self Balancing Robot)

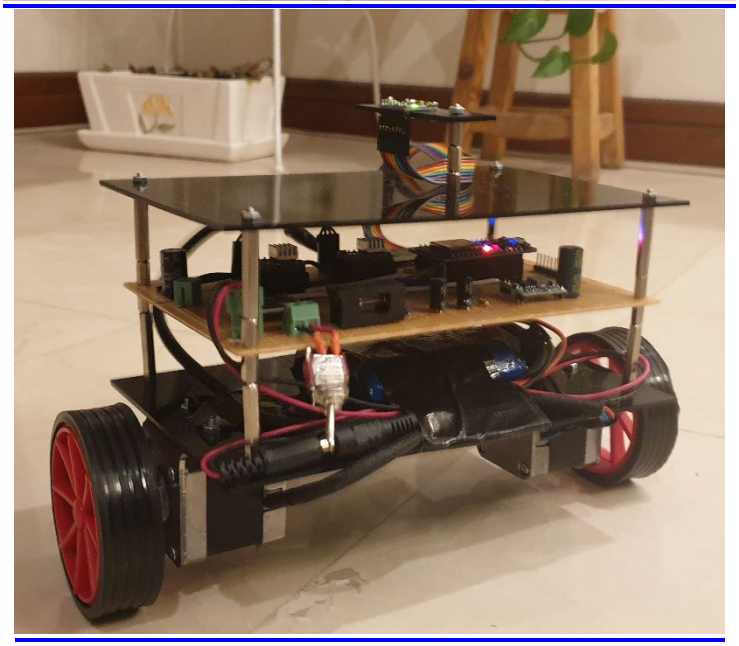
اعضای گروه:

امیرحسین باباپور، علیرضا صادقی، مهرزاد گلابی، محمد هراتی زاده

تیر 1403

برای مشاهده کد و فایل های ربات به گیتهاب پروژه مراجعه کنید:
https://github.com/MehrzadGolabi/selfbalancingrobot

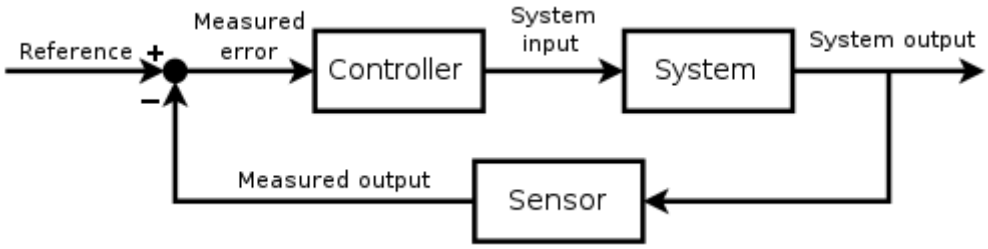




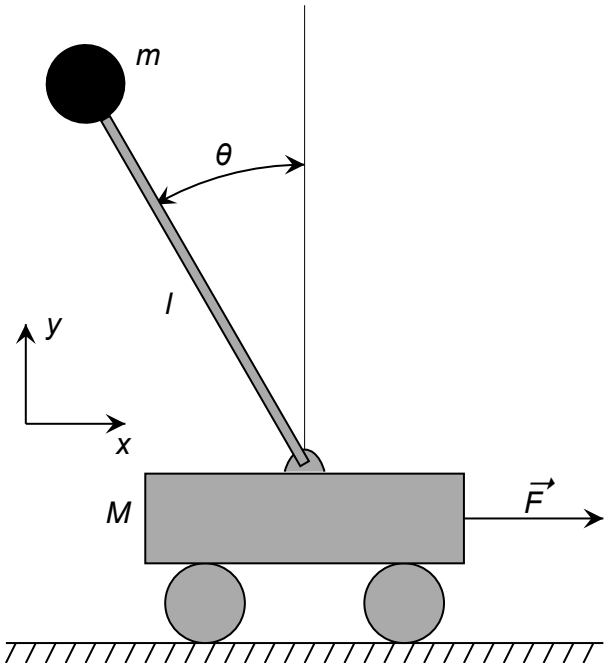
فصل اول: شرح و تعریف پروژه

ربات تعادلی، یک سیستم است که به صورت سیستم پاندول معکوس میتوان آن را شبیه سازی کرد. در آونگ معکوس مرکز جرم (نقطه ثقل) آن بالای نقطه چرخش آن قرار دارد. آونگ معکوس وضعیتی ناپایدار دارد و بدون کمک جانبی، سقوط می‌کند. می توان آن را در این موقعیت معکوس با استفاده از یک سیستم کنترل برای نظارت بر زاویه قطب و حرکت دادن نقطه محوری به صورت افقی در زیر مرکز جرم هنگامی که شروع به سقوط کرد، به حالت تعلیق در آورد و تعادل آن را حفظ کرد.

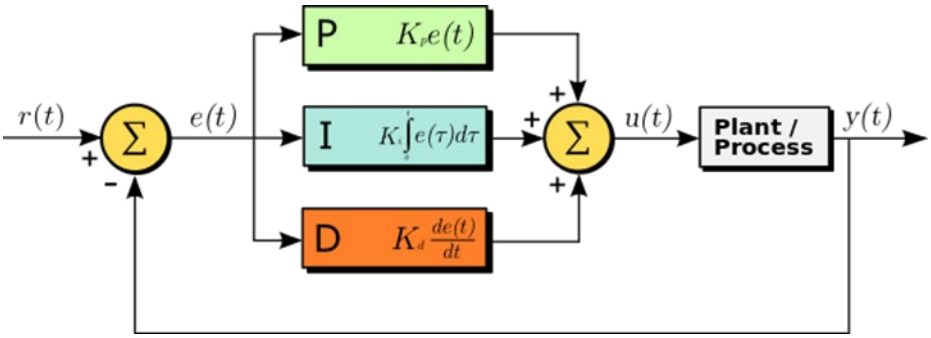
برای متعادل کردن یک سیستم ناپایدار، فیدبک منفی مورد نیاز است. نمودار زیر این رویکرد را نشان می دهد. علاوه بر یک حلقه فیدبک، یک حسگر برای اندازه‌گیری خطا (یا بهتر است بگوییم پاسخ سیستم)، و یک محرک برای به حداقل رساندن خطا مورد نیاز است.



آونگ معکوس:



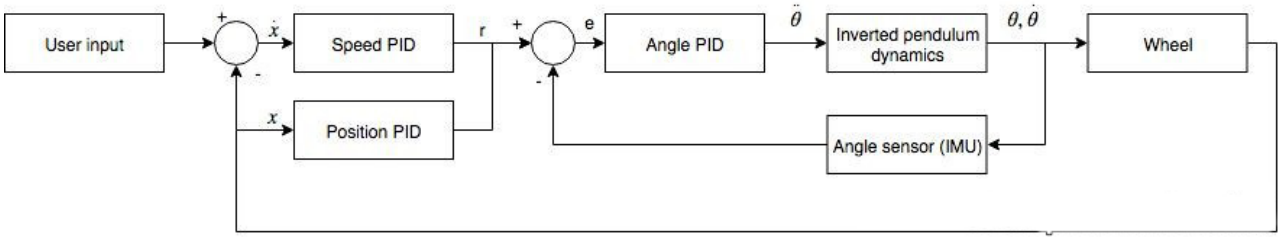
ربات تعادلی به عنوان یک پروژه متداول در برای بررسی کنترل کننده ها مورد استفاده قرار می گیرد. قابلیت اجرای روش های مختلف کنترلی از مزیت های این سیستم است. برای این نمونه از سیستم کنترلی PID استفاده شده است. کنترل‌کننده PID مقدار «خطا» بین خروجی فرایند و مقدار ورودی مطلوب (setpoint) محاسبه می‌کند. هدف کنترل‌کننده، به حداقل رساندن خطا با تنظیم ورودی‌های کنترل فرایند است. PID از سه قسمت مجزا به نام‌های Proportional (تناسبی)،Integral (انتگرال‌گیر) و Derivative (مشتق‌گیر) تشکیل شده که هر کدام از آن‌ها سیگنال خطا را به عنوان ورودی گرفته و عملیاتی را روی ان انجام می‌دهند و در نهایت خروجی شان با هم جمع می‌شود. خروجی این مجموعه که همان خروجی کنترل‌کننده PID است برای اصلاح خطا (error) به سیستم فرستاده می‌شود.



$$\text{Output(t)} = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau)\, d\tau + T_d \frac{de(t)}{dt} \right)$$

در بسیاری از کنترل‌کننده‌ها به علت حساسیت عبارتِ مشتق نسبت به نویز و دشواری اجرا، از آن صرف نظر و کنترل را به صورت PI پیاده‌سازی می‌کنند. سیگنال خروجی PID بر اساس نسبتی از خطای کنونی سیستم (عملکرد حاضر)، به اضافه مجموع خطاهای سیستم (رفتار گذشته)، به اضافه مشتق خطای کنونی (تخمین خطی رفتار آینده) محاسبه می‌شود و برای اصلاح خطا به سیستم اعمال می‌گردد.

یکی از روشهای بسیار متداول در محاسبه ضرایب کنترل کننده های PID روش زیگلر نیکولز است، که لازمه آن ارزیابی سیستم در حالت نوسانی است. اگرچه این روش از جمله روش های متداول است اما کماکان در کاربردهای عملی با آزمون و خطا و مشاهده رفتار سیستم تا حد بسیار قابل قبولی می توان به سیستمی پایار دست یافت.



روند کلی سیستم کنترلی به صورت بالا است. نرم افزار 2 حلقه کنترلی در فرکانس200Hz دارد. حلقه اولی سعی دارد تا خطای زاویه را کاهش دهد، یعنی سعی در تعادل و عمود نگه داشتن ربات دارد. حلقه دوم با توجه به ورودی که از کاربر (با استفاده از صفحه کنترل حرکت) میگیرد مکان و سرعت را تنظیم میکند. اگر ورودی ای از طرف کاربر موجود نباشد، این حلقه فعال نخواهد بود و فقط با توجه به حلقه اولی ربات سعی در تعادل خواهد داشت. ولی اگر ورودی ای گرفته شود، به طور مثال مقدار سرعت مثبت داده شود، ربات به توجه به سرعت مورد نیاز، به جلو خم شده تا به تندی مورد نظر برسد و حرکت صورت بگیرد.

قابلیت ها:

- حفظ تعادل و مقاومت در برابر اغتشاشات کوچک
- با هر بار شارژ ربات توانایی حفظ تعادل تا حداکثر 1 ساعت و 30 دقیقه را دارد.
- ارتباط از طریق WEBSOCKET به صورت REAL-TIME میان ربات و مرورگر کامپیوتر که امکانات زیر را مهیا میکند:
  - تنظیم پارامتر های PID از طریق WIFI
  - کنترل حرکت از طریق WIFI (حدود 20 متر)
  - تنظیم سرعت موتور ها از طریق WIFI (از مقادیر0 تا 1500)
  - نمایش ولتاژ باتری برای تخمین مقدار شارژ از طریق WIFI
  - نمایش وضعیت کنونی PID بر روی نمودار

محدودیت ها

سیستم های کنترلی نا پایدار مانند آونگ معکوس به عوامل متعددی برای حفظ پایداری خود نیاز دارند که اخلال در این موارد منجر به از دست رفتن پایداری می شود.
از جمله مواردی که می تواند اخلال در روند پایداری داشته باشد :

- **وجود و دریافت نویز توسط حسگر MPU6050 :** سنسور وظیفه دریافت و ضعیت کنونی و انتقال آن به پروسسور را دارد که با توجه به داده های ورودی محاسبات ریاضی روی آن انجام می دهد و با توجه به آن موتور ها را به حرکت در می آورد.
- **اندازه چرخ:** از عوامل موثر بر پایداری شعاع و ضخامت چرخ مورد استفاده است. شعاع بیشتر موجب افزایش دقت در تنظیم تعادل ربات می شود و ضخامت نیز پایداری کلی ربات را افزایش می دهد. اندازه چرخ با توجه به اندازه ربات تعیین می شود که خود تابعی از اندازه مدار است.
- **وزن ربات:** در صورتی که بالای ربات سنگین تر باشد، تنظیم وضعیت زاویه آن راحت تر است.

فصل دوم: معرفی قطعات و شرح آنها

لیست قطعات استفاده شده:

1. استپر موتور NEMA17 سایز 42 میلیمتری با گشتاور 4کیلوگرم و دقت 1.8درجه – 2عدد
2. درایور استپر موتور DRV8825 – 2عدد
3. سنسور MPU6050
4. ESP32 DEVKIT V1
5. مبدل ولتاژ DC به DC مدل MP1584
6. باتری 12 ولت 3سل
7. قطعات دیگر شامل:
    a. خازن ها:
        i. 1میکرو فاراد
        ii. 100 میکرو فاراد
        iii. 100 نانو فاراد
        iv. 470 میکرو فاراد
    b. مقاومت ها:
        i. 3.3 کیلو اهم
        ii. 100 کیلو اهم
    c. فیوز 2 آمپر
    d. کانکتور ها برای باتری و شارژ
    e. پین هدر ها برای اتصال به بورد

استپر موتور NEMA17

از استپر موتور به دلیل دقت بالای آن نسبت به موتور های DC استفاده شده است. موتور های DC برای داشتن گشتاور مورد نیاز برای پایداری ربات، به جعبه دنده نیاز دارند، ولی وجود جعبه دنده واکنش متقابل ایجاد کرده و اصطکاک آن، متعادل کردن ربات را بسیار سخت میکند.

Specification

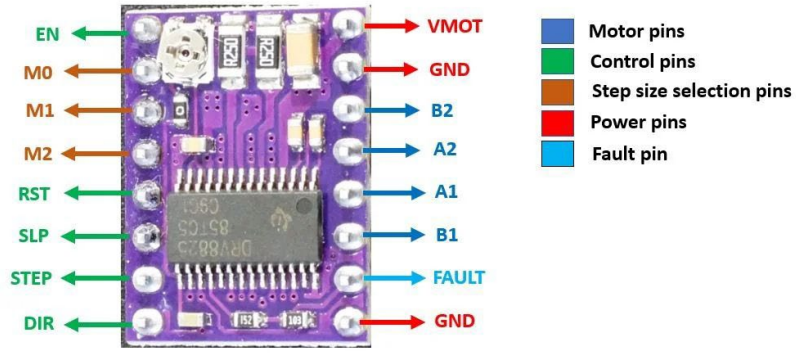
Number of Phase or Coil: 2
Model: CD-CB-PP-EE
Weight: 294g
Dimension: 42x42x39mm
Angle (Degree) Per Step: 1.8°
Torque: 4Kg.cm
Current Per Phase: 1.7A (BiPloar drive)
Voltage: 12-24V
Shaft Length: 10mm
Shaft Diameter: 5 mm
Number of Cables: 4
Phase resistance is 1.5 ohms

درایور استپر موتور DRV8825

برای استفاده از استپر موتور نیاز به درایور آن داریم، ماژول DRV8825 یک درایور استپر برای راه اندازی موتورهای استپر دوقطبی بصورت میکرواستپ است. از ویژگی های این ماژول می توان به تنظیم محدوده جریان ،داشتن محافظ در برابر مصرف جریان اضافی و افزایش دما بیش از حد و همچنین دارای 6 میکرواستپ با تفکیک پذیری تا 32/1 استپ نام برد. این ماژول با ولتاژ 8.2 تا 45 ولت کار می کند و می تواند در هر فاز بدون هرگونه هیت سینک و جریان هوای فن حداکثر تا A1.5 جریان دهی کند.

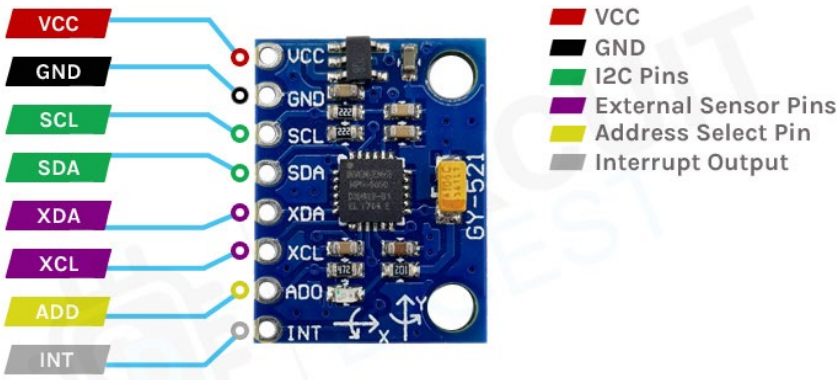
1 Features

- PWM Microstepping Stepper Motor Driver
  - Built-In Microstepping Indexer
  - Up to 1/32 Microstepping
- Multiple Decay Modes
  - Mixed Decay
  - Slow Decay
  - Fast Decay
- 8.2-V to 45-V Operating Supply Voltage Range
- 2.5-A Maximum Drive Current at 24 V and T_A = 25°C
- Simple STEP/DIR Interface
- Low Current Sleep Mode
- Built-In 3.3-V Reference Output
- Small Package and Footprint
- Protection Features
  - Overcurrent Protection (OCP)
  - Thermal Shutdown (TSD)
  - VM Undervoltage Lockout (UVLO)
  - Fault Condition Indication Pin (nFAULT)



سنسور MPU6050

MPU6050 سنسور ژیروسکوپ و زاویه سنج با دقت بالا است و برای اندازه گیری زاویه فعلی و شتاب ربات مناسب است.

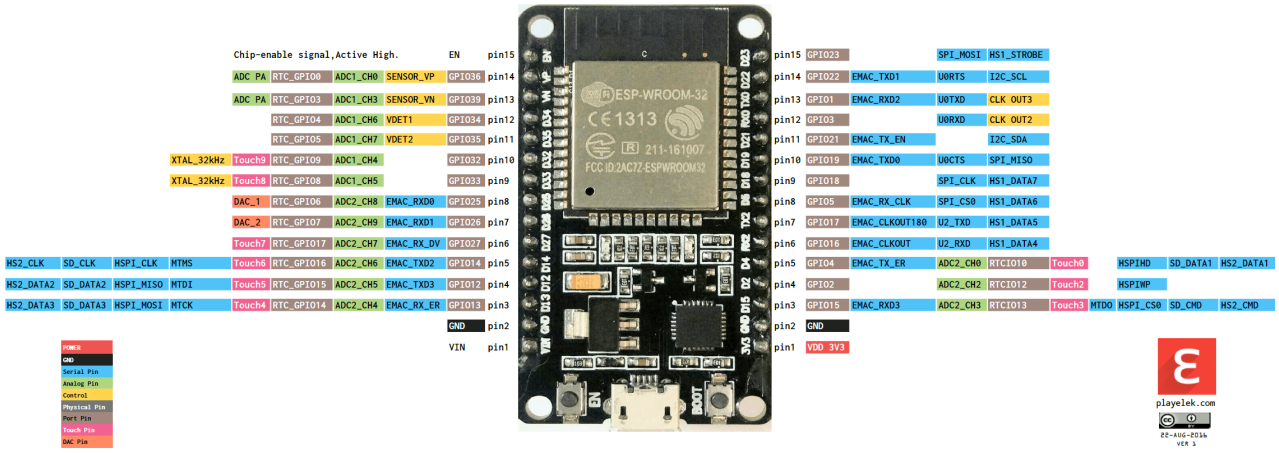


ESP32 DEVKIT V1

برای میکروکنترلر در این پروژه از ESP32 استفاده شده است، قابلیت های

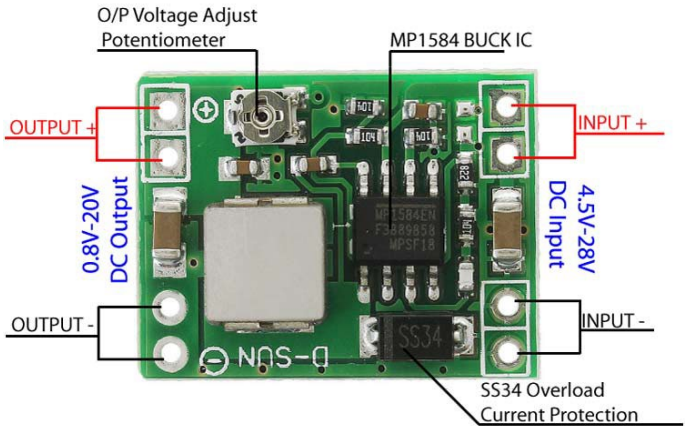
- دقت قابل قبول در ADC
- هسته وافای ۲,۴ گیگا هرتز و بلوتوث داخلی
- سرعت پردازش بالا
- بهترین نسبت پرفورمنس به مقدار مصرف انرژی که در یک وسیله شارژی حیاتی است

انتخاب آن را نسبت به گزینه های دیگر آسان کرده است.



مبدل ولتاژ DC به DC مدل MP1584

ماژول کاهنده DC به DC با ولتاژ ورودی 4.5 تا 28 ولت - جریان 3 آمپر برای تامین VCC قطعات



باتری 12 ولت 3سل

برای تامین موتور ها به 12 ولت نیاز است و بقیه قطعات از طریق مبدل کاهنده ولتاژ، 5 ولت دریافت میکنند. این باتری با توجه به محدود کردن جریان موتور ها، قابلیت تامین انرژی ربات را تا حدود 40 دقیقه تا یک ساعت دارد.

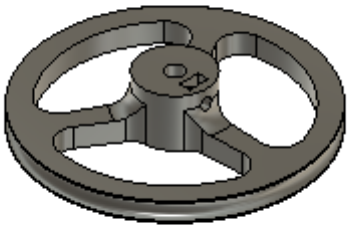
فصل سوم: طراحی سخت افزار

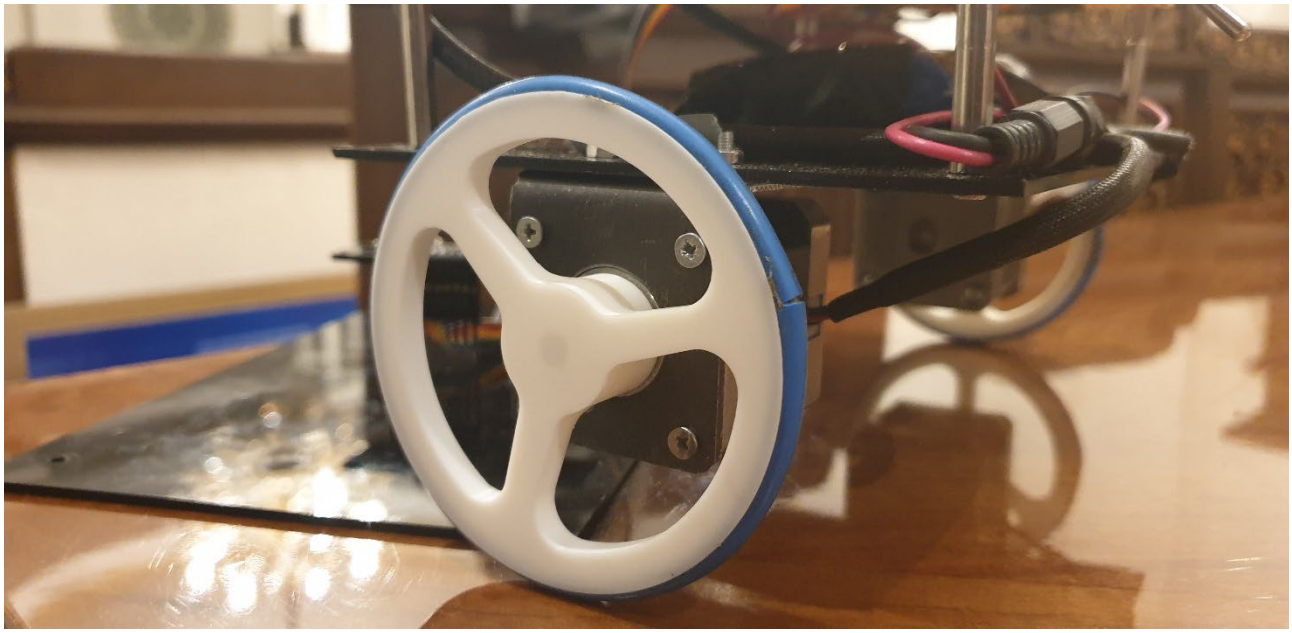
شاسی ربات

برای ساخت شاسی ربات از ورقه های پلکسی گلاس 2 میلی متری استفاده شده است که به سایز مدار بریده شده، و با استفاده از اسپیسر های فلزی به یک دیگر متصل شده اند. برای اتصال موتور ها از براکت های 90 درجه مخصوص موتور سایز NEMA17 استفاده شده است که با استفاده از پیچ به پایینترین طبقه متصل شده اند.
سنسور MPU6050 به نویز بسیار حساس است، علاوه بر آن دقیق ترین اندازه گیری را در بالاترین نقطه انجام میدهد به همین سبب با استفاده از پایه های فلزی در بالاترین نقطه قرار گرفته است.
برای اتصال باتری از جک DC استفاده شده است که قابلیت اتصال و قطع کردن ساده باتری و همچنین شارژ کردن آسان آن را میدهد. علاوه بر آن امکان اتصال منبع تغذیه 12 ولت به ربات برای تست و بررسی را مهیا میکند.

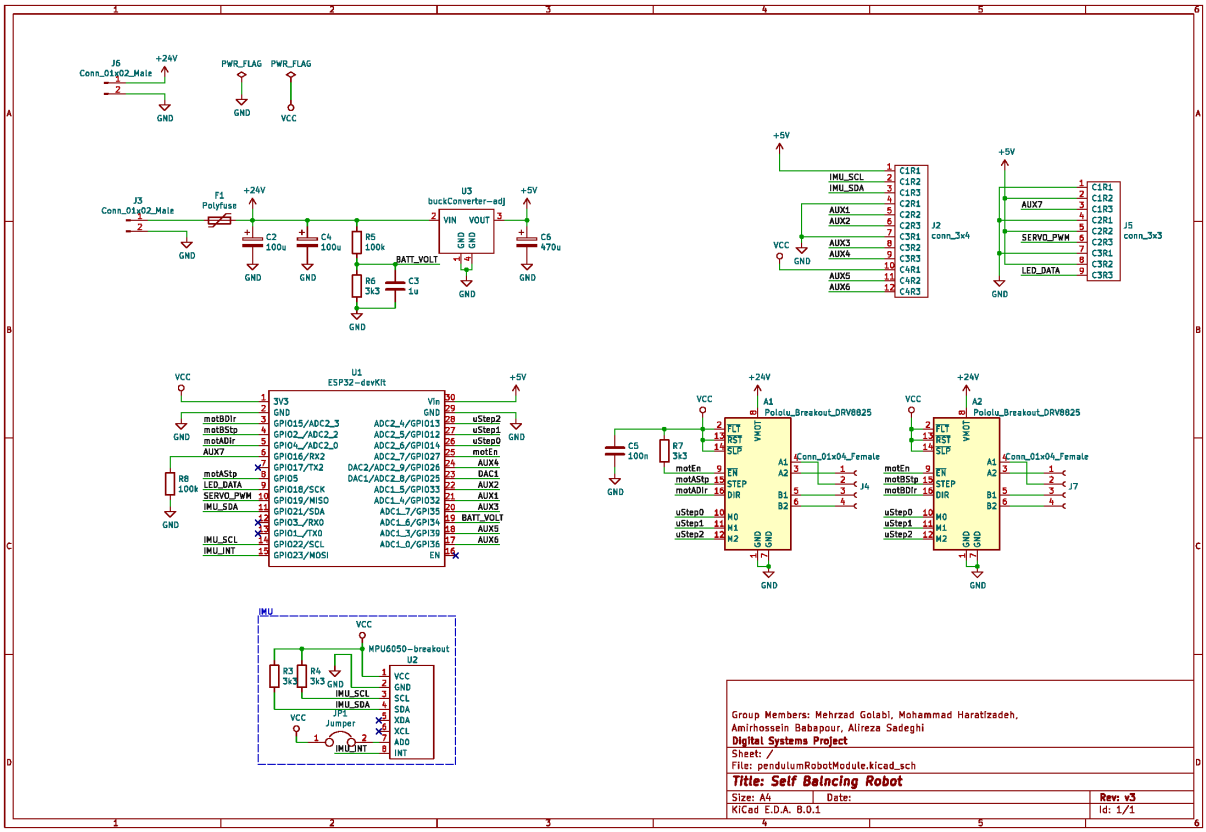
چرخ ها

یکی از چالش های مواجه شده، اندازه شفت موتور ها است که 5 میلی متر است، این موضوع باعث شد تا اکثر چرخ های آماده برای ربات ها که در بازار موجود است، به طور قابل قبولی، قابل استفاده برای ربات نباشد. چرخ هایی با اندازه شفت 5 میلی متری به صورت 3 بعدی طراحی و سپس چاپ شده اند. برای لاستیک های چرخ ها از یک سیم USB که قطر مورد نظر را داشت استفاده شده است.



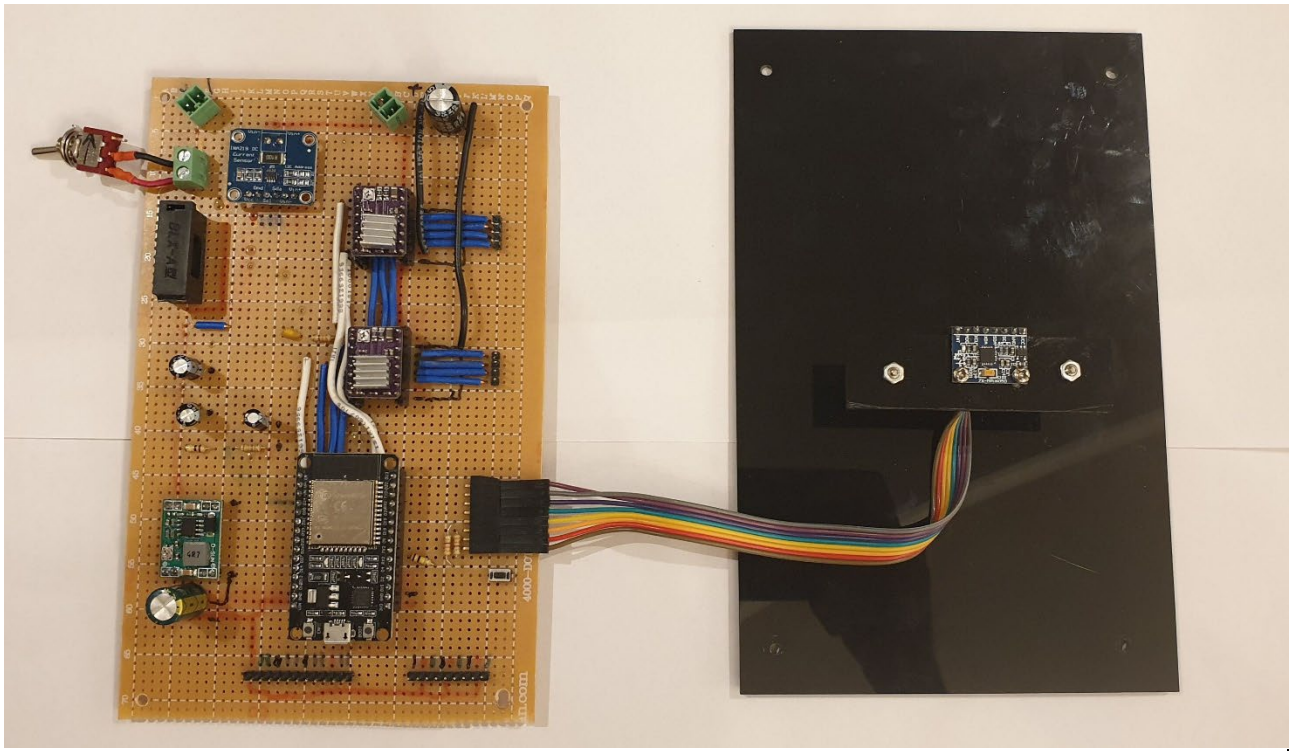


طراحی مدار

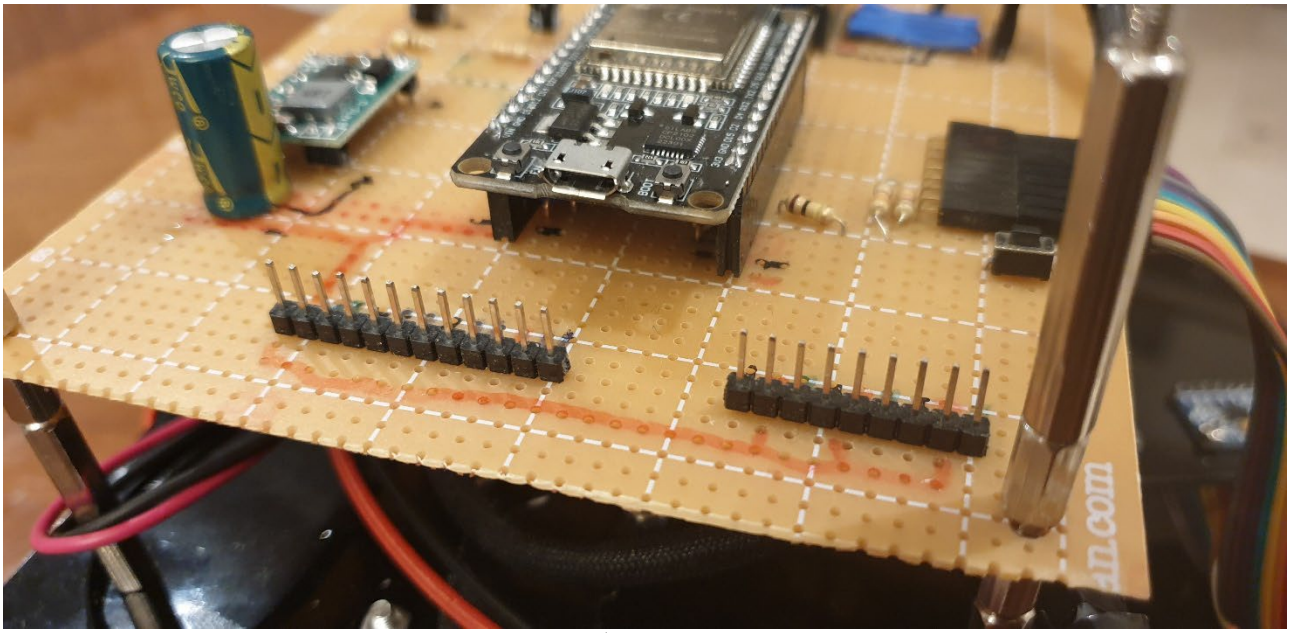


شماتیک مدار طراحی شده به صورت بالا است.

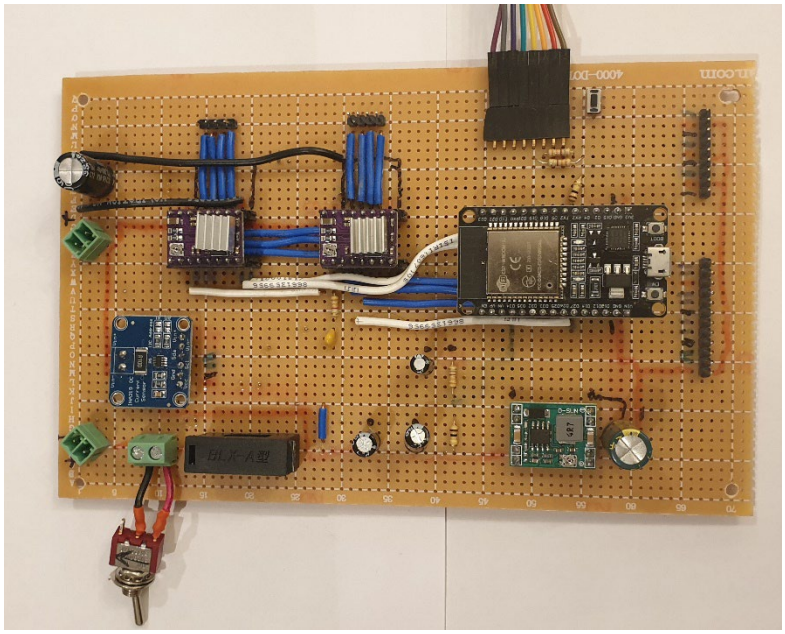
ورودی منبع مدار باتری 12 تا 24 ولت است که از فیوز تعبیه شده عبور کرده و با استفاده از مقاومت های مقسم ولتاژ به ESP32 برای اندازه گیری مقدار ولتاژ فعلی متصل شده است. سپس به مبدل کاهنده ولتاژ که قبل از اتصال به مدار بر روی 5 ولت تنظیم شده است، متصل میگردد. این مبدل وظیفه تامین Vcc قطعات را دارد. در ورودی و خروجی این مبدل از خازن های کوپلاژ برای کاهش نویز استفاده شده اند. در درایور های DRV8825، مطابق دیتا شیت متصل شده اند. Vmot در این درایور ها به ورودی باتری متصل است. IMU نیز مطابق دیتا شیت متصل شده است، برای پایه AD0 که برای آدرس دهی است، در مدار طراحی از یک جامپر و در مدار فیزیکی از یک Push Button کوچک استفاده شده است. استفاده از این دکمه برای آدرس دهی I2C تنها زمانی لازم خواهد بود که دیگر قطعات I2C نیز به مدار اضافه شود. تمام قطعات به ESP32 متصل گردیده اند، در مدار برای استفاده و بهره برداری بهینه در آینده، تمامی پایه های استفاده نشده ESP با استفاده از پین هدر های متعدد قابل دسترسی است که اضافه کردن قطعات و قابلیت های جدید را بدون تغییر کلی در مدار، تسهیل می بخشد.

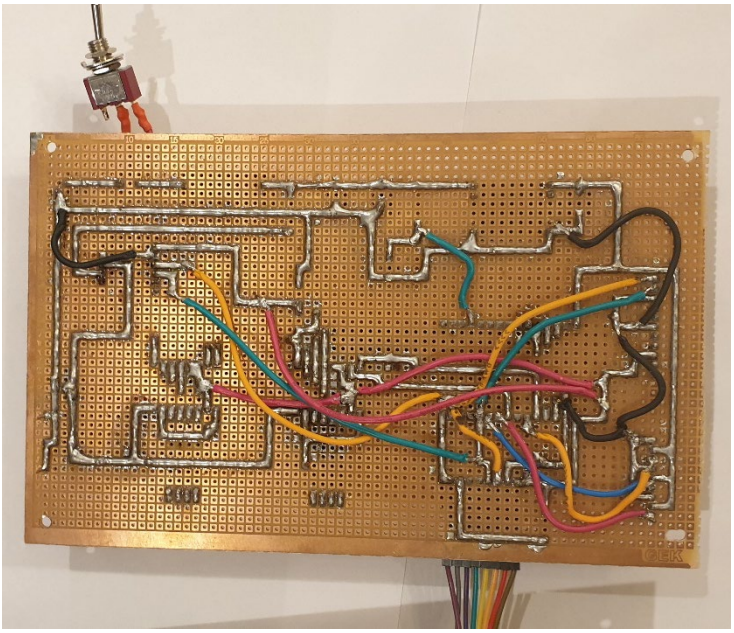


مدار ربات



پین هدر های قابل استفاده





فصل چهارم: کد

کل کد در این گزارش گذاشته شده است، ولی برای مشاهده با کیفیت تر حتما به گیتهاب پروژه مراجعه کنید.
کد در بستر VScode و با استفاده از ابزار PlatformIO نوشته شده است. برای استفاده نیاز است تا این پیشنیاز ها نصب شوند و بستر استفاده با انتخاب برد درست (ESP32) مهیا شود. بعد از نصب، کد را از گیتهاب دانلود کرده و در PlatformIO به عنوان existing project باز میکنیم. با این کار تمامی library های پیشنیاز به صورت خودکار نصب خواهند شد. اکنون میتوان با اتصال ESP32 به کامپیوتر و نوشتن COMPORT درست در فایل platformio.ini کد را کامپایل و آپلود کرد. برای استفاده از صفحات وب نیاز به آپلود Filesystem Image نیز میباشد که در منو PlatformIO قابل مشاهده است. بعد از این مراحل میتوان با باز کردن Serial Monitor وضعیت ربات را مشاهده و دستورات لازم را داد. توضیحات بیشتر در بخش راه اندازی خواهد بود.

```
#include <Arduino.h>
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#include <Streaming.h>
#include <MPU6050.h>
#include <PID.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <WebSocketsServer.h>
#include <FS.h>
#include <SPIFFS.h>
#include <SPIFFSEditor.h>
#include <fastStepper.h>
// #include <par.h>
#include <Preferences.h>  // for storing settings
#include "driver/adc.h"
#include "esp_adc_cal.h"
#include <Update.h>

// ----- Input method

// Driving behaviour
float speedFactor = 0.7;  // how strong it reacts to inputs, lower = softer (limits max speed) (between 0 and 1)
float steerFactor = 1.0;  // how strong it reacts to inputs, lower = softer (limits max speed) (between 0 and 1)
float speedFilterConstant = 0.9;  // how fast it reacts to inputs, higher = softer (between 0 and 1, but not 0 or 1)
float steerFilterConstant = 0.9;  // how fast it reacts to inputs, higher = softer (between 0 and 1, but not 0 or 1)


// ----- Type definitions
typedef union {
  struct {
    float val; // Float (4 bytes) comes first, as otherwise padding will be applied
    uint8_t cmd;
    uint8_t checksum;
  };
  uint8_t array[6];
} command;

typedef union {
  uint8_t arr[6];
  struct {
    uint8_t grp;
    uint8_t cmd;
    union {
      float val;
      uint8_t valU8[4];
    };
  } __attribute__((packed));
} cmd;

// Plot settings
struct {
  boolean enable = 0; // Enable sending data
  uint8_t prescaler = 4;
} plot;

/* Remote control structure
To adjust "driving experience" a gain can be adjusted for the speed and steer inputs.
```

```
Additionaly, a selfRight input can be used. When setting this bit to 1,
the robot will enable control in an attempt to self right.
The override input can be used to control the robot when it is lying flat.
The robot will switch automatically from override to balancing mode, if it happens to right itself.
The disable control input can be used to
1) disable the balancing mode
2) disable the self-right attempt
3) disable the override mode
Depending on which state the robot is in.
*/
struct {
  float speed = 0;
  float steer = 0;
  float speedGain = 0.25;
  float steerGain = 0.25;
  float speedOffset = 0.0;
  bool selfRight = 0;
  bool disableControl = 0;
  bool override = 0;
} remoteControl;

#define FORMAT_SPIFFS_IF_FAILED true

// ----- Function prototypes
void sendWifiList(void);
void parseSerial();
void parseCommand(char* data, uint8_t length);
void calculateGyroOffset(uint8_t nSample);
void readSensor();
void initSensor(uint8_t n);
void setMicroStep(uint8_t uStep);
void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length);
void sendConfigurationData(uint8_t num);

void IRAM_ATTR motLeftTimerFunction();
void IRAM_ATTR motRightTimerFunction();

// ----- Definitions and variables
// -- Web server
const char* http_username = "admin";
const char* http_password = "admin";
AsyncWebServer httpServer(80);
size_t content_len;
WebSocketsServer wsServer = WebSocketsServer(81);

// -- EEPROM
Preferences preferences;

// -- Stepper motors
#define motEnablePin 27
#define motUStepPin1 14
#define motUStepPin2 12
#define motUStepPin3 13

fastStepper motLeft(5, 4, 0, motLeftTimerFunction, true);
fastStepper motRight(2, 15, 1, motRightTimerFunction);

uint8_t microStep = 16;
uint8_t motorCurrent = 150;
float maxStepSpeed = 1500;

// -- PID control
#define dT_MICROSECONDS 5000
#define dT dT_MICROSECONDS/1000000.0

#define PID_ANGLE 0
#define PID_POS 1
#define PID_SPEED 2

#define PID_ANGLE_MAX 12
PID pidAngle(cPID, dT, PID_ANGLE_MAX, -PID_ANGLE_MAX);
#define PID_POS_MAX 35
PID pidPos(cPD, dT, PID_POS_MAX, -PID_POS_MAX);
PID pidSpeed(cP, dT, PID_POS_MAX, -PID_POS_MAX);

uint8_t controlMode = 1; // 0 = only angle, 1 = angle+position, 2 = angle+speed
```

```
// Threshold for fall detection. If integral of error of angle controller is larger than this value, controller is disabled
#define angleErrorIntegralThreshold 30.0
#define angleErrorIntegralThresholdDuringSelfright angleErrorIntegralThreshold*3
#define angleEnableThreshold 5.0 // If (absolute) robot angle is below this threshold, enable control
#define angleDisableThreshold 70.0 // If (absolute) robot angle is above this threshold, disable control (robot has fallen down)

// -- IMU
MPU6050 imu;

#define GYRO_SENSITIVITY 65.5

int16_t gyroOffset[3];
float accAngle = 0;
float filterAngle = 0;
float angleOffset = 2.0;
float gyroFilterConstant = 0.996;
float gyroGain = 1.0;

// Temporary values for debugging sensor algorithm
float rxg, ayg, azg;

// -- Others
#define PIN_LED 32
#define PIN_MOTOR_CURRENT 25
#define PIN_LED_LEFT 33
#define PIN_LED_RIGHT 26

// ADC definitions (for reading battery voltage)
#define ADC_CHANNEL_BATTERY_VOLTAGE ADC1_CHANNEL_6 // GPIO number 34
// Battery voltage is measured via a 100 and 3.3 kOhm resistor divider. Reference voltage is 1.1 V (if attenuation is set to 0dB)
#define BATTERY_VOLTAGE_SCALING_FACTOR (100+3.3)/3.3
#define BATTERY_VOLTAGE_FILTER_COEFFICIENT 0.99
esp_adc_cal_characteristics_t adc_chars;

// -- WiFi
#define ROBOT_NAME_DEFAULT "balancingrobot"
char robotName[63] = ROBOT_NAME_DEFAULT;

// BT MAC
char BTaddress[20] = "00:00:00:00:00:00";

// Noise source (for system identification)
boolean noiseSourceEnable = 0;
float noiseSourceAmplitude = 1;

// ----- Interrupt functions -----
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

void IRAM_ATTR motLeftTimerFunction() {
  portENTER_CRITICAL_ISR(&timerMux);
  motLeft.timerFunction();
  portEXIT_CRITICAL_ISR(&timerMux);
}
void IRAM_ATTR motRightTimerFunction() {
  portENTER_CRITICAL_ISR(&timerMux);
  motRight.timerFunction();
  portEXIT_CRITICAL_ISR(&timerMux);
}


void setMotorCurrent() {
  dacWrite(PIN_MOTOR_CURRENT, motorCurrent);
}

void sendData(uint8_t *b, uint8_t l) {
  wsServer.sendBIN(0,b,l);
}

void wirelessTask(void * parameters) {
  while (1) {
    #ifdef INPUT_IBUS
    IBus.loop();
    #endif
    wsServer.loop();
    delay(2);
```

```
  }
}

void handleUpdate(AsyncWebServerRequest *request) {
  char* html = "<form method='POST' action='/doUpdate' enctype='multipart/form-data'><input type='file' name='update'><input type='submit'
value='Update'></form>";
  request->send(200, "text/html", html);
}

void handleDoUpdate(AsyncWebServerRequest *request, const String& filename, size_t index, uint8_t *data, size_t len, bool final) {
  if (!index){
    Serial.println("Update");
    content_len = request->contentLength();
    // if filename includes spiffs, update the spiffs partition
    int cmd = (filename.indexOf("spiffs") > -1) ? U_SPIFFS : U_FLASH;

    if (!Update.begin(UPDATE_SIZE_UNKNOWN, cmd)) {
      Update.printError(Serial);
    }
  }

  if (Update.write(data, len) != len) {
    Update.printError(Serial);
  }

  if (final) {
    AsyncWebServerResponse *response = request->beginResponse(302, "text/plain", "Please wait while the device reboots");
    response->addHeader("Refresh", "20");
    response->addHeader("Location", "/");
    request->send(response);
    if (!Update.end(true)){
      Update.printError(Serial);
    } else {
      Serial.println("Update complete");
      Serial.flush();
      ESP.restart();
    }
  }
}

void printProgress(size_t prg, size_t sz) {
  Serial.printf("Progress: %d%%\n", (prg*100)/content_len);
}


// ----- Main code
void setup() {

  Serial.begin(115200);

  preferences.begin("settings", false);  // false = RW-mode
  // preferences.clear();  // Remove all preferences under the opened namespace

  pinMode(motEnablePin, OUTPUT);
  pinMode(motUStepPin1, OUTPUT);
  pinMode(motUStepPin2, OUTPUT);
  pinMode(motUStepPin3, OUTPUT);
  digitalWrite(motEnablePin, 1); // Disable steppers during startup
  setMicroStep(microStep);

  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_LED_LEFT, OUTPUT);
  pinMode(PIN_LED_RIGHT, OUTPUT);
  digitalWrite(PIN_LED, 0);
  digitalWrite(PIN_LED_LEFT, 1); // Turn on one LED to indicate we are live
  digitalWrite(PIN_LED_RIGHT, 0);

  motLeft.init();
  motRight.init();
  motLeft.microStep = microStep;
  motRight.microStep = microStep;

  // SPIFFS setup
  if(!SPIFFS.begin(FORMAT_SPIFFS_IF_FAILED)){
    Serial.println("SPIFFS mount failed");
    return;
```

```
} else {
  Serial.println("SPIFFS mount success");
}

// Gyro setup
delay(200);
Wire.begin(21, 22, 400000UL);
delay(100);
Serial.println(imu.testConnection());
imu.initialize();
imu.setFullScaleGyroRange(MPU6050_GYRO_FS_500);
// Calculate and store gyro offsets
delay(50);

// Init EEPROM, if not done before
#define PREF_VERSION 1  // if setting structure has been changed, count this number up to delete all settings
if (preferences.getUInt("pref_version", 0) != PREF_VERSION) {
  preferences.clear();  // Remove all preferences under the opened namespace
  preferences.putUInt("pref_version", PREF_VERSION);
  Serial << "EEPROM init complete, all preferences deleted, new pref_version: " << PREF_VERSION << "\n";
}

// Read gyro offsets
Serial << "Gyro calibration values: ";
for (uint8_t i=0; i<3; i++) {
  char buf[16];
  sprintf(buf, "gyro_offset_%u", i);
  gyroOffset[i] = preferences.getShort(buf, 0);
  Serial << gyroOffset[i] << "\t";
}
Serial << endl;

// Read angle offset
angleOffset = preferences.getFloat("angle_offset", 0.0);

// Perform initial gyro measurements
initSensor(50);

// Read robot name
uint32_t len = preferences.getBytes("robot_name", robotName, 63);

Serial.println(robotName);

// Connect to Wifi and setup OTA if known Wifi network cannot be found
boolean wifiConnected = 0;
if (preferences.getUInt("wifi_mode", 0)==1) {
  char ssid[63];
  char key[63];
  preferences.getBytes("wifi_ssid", ssid, 63);
  preferences.getBytes("wifi_key", key, 63);
  Serial << "Connecting to '" << ssid << "'" << endl;
  // Serial << "Connecting to '" << ssid << "', '" << key << "'" << endl;
  WiFi.mode(WIFI_STA);
  WiFi.setHostname(robotName);
  WiFi.begin(ssid, key);
  if (!(WiFi.waitForConnectResult() != WL_CONNECTED)) {
    Serial.print("Connected to WiFi with IP address: ");
    Serial.println(WiFi.localIP());
    wifiConnected = 1;

    digitalWrite(PIN_LED_LEFT, 0);
  } else {
    Serial.println("Could not connect to known WiFi network");
  }
}
if (!wifiConnected) {
  Serial.println("Starting AP...");
  WiFi.mode(WIFI_AP_STA);
  // WiFi.softAPConfig(apIP, apIP, IPAddress(192,168,178,24));
  WiFi.softAP(robotName, "turboturbo");
  Serial << "AP named '" << WiFi.softAPSSID() << "' started, IP address: " << WiFi.softAPIP() << endl;

  for (uint8_t k=0; k<3; k++) {
    digitalWrite(PIN_LED_LEFT, 1);
    delay(100);
    digitalWrite(PIN_LED_LEFT, 0);
```

```
    delay(100);
  }
}

ArduinoOTA.setHostname(robotName);
ArduinoOTA
  .onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH) {
      type = "sketch";
    } else {  // U_SPIFFS
      type = "filesystem";
    }

    Serial.println("Start updating " + type);
  })
  .onEnd([]() {
    Serial.println("\nEnd");
  })
  .onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  })
  .onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) {
      Serial.println("Auth Failed");
    } else if (error == OTA_BEGIN_ERROR) {
      Serial.println("Begin Failed");
    } else if (error == OTA_CONNECT_ERROR) {
      Serial.println("Connect Failed");
    } else if (error == OTA_RECEIVE_ERROR) {
      Serial.println("Receive Failed");
    } else if (error == OTA_END_ERROR) {
      Serial.println("End Failed");
    }
  });

ArduinoOTA.begin();

// Start DNS server
if (MDNS.begin(robotName)) {
  Serial.print("MDNS responder started, name: ");
  Serial.println(robotName);
} else {
  Serial.println("Could not start MDNS responder");
}

httpServer.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  Serial.println("Loading index.htm");
  request->send(SPIFFS, "/control2.htm");
});


httpServer.on("/update", HTTP_GET, [](AsyncWebServerRequest *request){handleUpdate(request);});
httpServer.on("/doUpdate", HTTP_POST,
  [](AsyncWebServerRequest *request) {},
  [](AsyncWebServerRequest *request, const String& filename, size_t index, uint8_t *data,
             size_t len, bool final) {handleDoUpdate(request, filename, index, data, len, final);}
);

httpServer.serveStatic("/", SPIFFS, "/");
httpServer.onNotFound([](AsyncWebServerRequest *request){
    request->send(404, "text/plain", "FileNotFound");
});

httpServer.addHandler(new SPIFFSEditor(SPIFFS,http_username,http_password));
httpServer.begin();

Update.onProgress(printProgress);

wsServer.begin();
wsServer.onEvent(webSocketEvent);

MDNS.addService("http", "tcp", 80);
MDNS.addService("ws", "tcp", 81);
```

```
  // Make some funny sounds
  // for (uint8_t i=0; i<150; i++) {
  //   motRight.speed = 500 + i*10;
  //   updateStepper(&motRight);
  //   delay(5);
  // }

  dacWrite(PIN_MOTOR_CURRENT, motorCurrent);

  pidAngle.setParameters(0.65,1.0,0.075,15);
  pidPos.setParameters(1,0,1.2,50);
  pidSpeed.setParameters(6,5,0,20);

  Serial.println("Ready");


  // Characterize ADC at particular atten
  esp_adc_cal_value_t val_type = esp_adc_cal_characterize(ADC_UNIT_1, ADC_ATTEN_0db, ADC_WIDTH_BIT_12, 1100, &adc_chars);
  if (val_type == ESP_ADC_CAL_VAL_EFUSE_VREF) {
     Serial.println("eFuse Vref");
  } else if (val_type == ESP_ADC_CAL_VAL_EFUSE_TP) {
     Serial.println("Two Point");
  } else {
     Serial.println("Default");
  }
  Serial << "ADC calibration values (attenuation, vref, coeff a, coeff b):" << adc_chars.atten << "\t"<< adc_chars.vref << "\t"<< adc_chars.coeff_a <<
"\t"<< adc_chars.coeff_b << endl;

  // Configure ADC
  adc1_config_channel_atten(ADC_CHANNEL_BATTERY_VOLTAGE, ADC_ATTEN_0db);
  adc_set_data_inv(ADC_UNIT_1, true); // For some reason, data is inverted...

  Serial.println("Booted, ready for driving!");

}


float mapfloat(float x, float in_min, float in_max, float out_min, float out_max)
{
 return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

void loop() {
  static unsigned long tLast = 0;
  float pidAngleOutput = 0;
  float avgMotSpeed;
  float steer = 0;
  static float avgSteer;
  static float avgSpeed;
  static boolean enableControl = 0;
  static float avgMotSpeedSum = 0;
  int32_t avgMotStep;
  float pidPosOutput = 0, pidSpeedOutput = 0;
  static uint8_t k = 0;
  static float avgBatteryVoltage = 0;
  static uint32_t lastInputTime = 0;
  uint32_t tNowMs;
  float absSpeed = 0;
  float noiseValue = 0;
  static boolean overrideMode = 0, lastOverrideMode = 0;
  static boolean selfRight = 0;
  static boolean disableControl = 0;
  static float angleErrorIntegral = 0;

  unsigned long tNow = micros();
  tNowMs = millis();

  if (tNow-tLast > dT_MICROSECONDS) {
    readSensor();
    // Read receiver inputs


    if (remoteControl.selfRight && !enableControl) { // Start self-right action (stops when robot is upright)
      selfRight = 1;
      disableControl = 0;
      remoteControl.selfRight = 0; // Reset single action bool
```

```
} else if (remoteControl.disableControl && enableControl ) { // Sort of kill-switch
 disableControl = 1;
 selfRight = 0;
 remoteControl.disableControl = 0;
}

// Filter speed and steer input
avgSpeed = speedFilterConstant*avgSpeed + (1-speedFilterConstant)*remoteControl.speed/5.0;
avgSteer = steerFilterConstant*avgSteer + (1-steerFilterConstant)*remoteControl.steer;

if (enableControl) {
 // Read receiver inputs

 if (abs(avgSpeed)<0.2) {
  // remoteControl.speed = 0;
 } else {
  lastInputTime = tNowMs;
  if (controlMode==1) {
   controlMode = 2;
   motLeft.setStep(0);
   motRight.setStep(0);
   pidSpeed.reset();
  }
 }

 steer = avgSteer;

 // Switch to position control if no input is received for a certain amount of time
 if (tNowMs-lastInputTime>2000 && controlMode == 2) {
  controlMode = 1;
  motLeft.setStep(0);
  motRight.setStep(0);
  pidPos.reset();
 }

 // Actual controller computations
 if (controlMode == 0) {
  pidAngle.setpoint = avgSpeed*2;
 } else if (controlMode == 1) {
  avgMotStep = (motLeft.getStep() + motRight.getStep())/2;
  pidPos.setpoint = avgSpeed;
  pidPos.input = -((float) avgMotStep) / 1000.0;
  pidPosOutput = pidPos.calculate();
  pidAngle.setpoint = pidPosOutput;
 } else if (controlMode == 2) {
  pidSpeed.setpoint = avgSpeed;
  pidSpeed.input = -avgMotSpeedSum/100.0;
  pidSpeedOutput = pidSpeed.calculate();
  pidAngle.setpoint = pidSpeedOutput;
 }


 pidAngle.input = filterAngle;

 pidAngleOutput = pidAngle.calculate();

 // Optionally, add some noise to angle for system identification purposes
 if (noiseSourceEnable) {
  noiseValue = noiseSourceAmplitude*((random(1000)/1000.0)-0.5);
  pidAngleOutput += noiseValue;
 }

 avgMotSpeedSum += pidAngleOutput/2;
 if (avgMotSpeedSum>maxStepSpeed) {
  avgMotSpeedSum  = maxStepSpeed;
 } else if (avgMotSpeedSum<-maxStepSpeed) {
  avgMotSpeedSum  = -maxStepSpeed;
 }
 avgMotSpeed = avgMotSpeedSum;
 motLeft.speed = avgMotSpeed + steer;
 motRight.speed = avgMotSpeed - steer;

 // Detect if robot has fallen. Concept: integrate angle controller error over time.
 // If absolute integrated error surpasses threshold, disable controller
 angleErrorIntegral += (pidAngle.setpoint - pidAngle.input) * dT;
 if (selfRight) {
```

```
      if (abs(angleErrorIntegral) > angleErrorIntegralThresholdDuringSelfright) {
        selfRight = 0;
        disableControl = 1;
      }
    } else {
      if (abs(angleErrorIntegral) > angleErrorIntegralThreshold) {
        disableControl = 1;
      }
    }


    // Switch microstepping
    absSpeed = abs(avgMotSpeed);
    uint8_t lastMicroStep = microStep;

    if (absSpeed > (150 * 32 / microStep) && microStep > 1) microStep /= 2;
    if (absSpeed < (130 * 32 / microStep) && microStep < 32) microStep *= 2;


    // Disable control if robot is almost horizontal. Re-enable if upright.
    if ((abs(filterAngle)>angleDisableThreshold && !selfRight) || disableControl) {
      enableControl = 0;
      // disableControl = 0; // Reset disableControl flag
      motLeft.speed = 0;
      motRight.speed = 0;
      digitalWrite(motEnablePin, 1); // Inverted action on enable pin
      digitalWrite(PIN_LED_RIGHT, 0);
    }
    if (abs(filterAngle)<angleEnableThreshold && selfRight) {
      selfRight = 0;
      angleErrorIntegral = 0; // Reset, otherwise the fall detection will be triggered immediately
    }
  } else { // Control not active

    // Override control
    if (overrideMode && !lastOverrideMode) { // Transition from disable to enable
      // Enable override mode
      motLeft.speed = 0;
      motRight.speed = 0;
      digitalWrite(motEnablePin, 0); // Enable motors
      overrideMode = 1;
    } else if (!overrideMode && lastOverrideMode) {
      digitalWrite(motEnablePin, 1); // Inverted action on enable pin
      overrideMode = 0;
    }
    lastOverrideMode = overrideMode;

    if (abs(filterAngle)>angleEnableThreshold+5) { // Only reset disableControl flag if angle is out of "enable" zone, otherwise robot will keep
cycling between enable and disable states
      disableControl = 0;
    }

    if ((abs(filterAngle)<angleEnableThreshold || selfRight) && !disableControl) { // (re-)enable and reset stuff
      enableControl = 1;
      digitalWrite(PIN_LED_RIGHT, 1);

      controlMode = 1;
      // avgMotSpeedSum = 0;

      if (!overrideMode) {
        avgMotSpeedSum = 0;
        digitalWrite(motEnablePin, 0); // Inverted action on enable pin
        pidAngle.reset();
      } else {
        avgMotSpeedSum = (motLeft.speed + motRight.speed) / 2;
        overrideMode = 0;
      }

      motLeft.setStep(0);
      motRight.setStep(0);
      pidPos.reset();
      pidSpeed.reset();

      angleErrorIntegral = 0;
      // delay(1);
    }
```

```
      if (overrideMode) {
        float spd = avgSpeed;
        float str = avgSteer;
        // if (spd<3) spd = 0;
        // if (str<3) str = 0;
        motLeft.speed = -30*spd + 2*str;
        motRight.speed = -30*spd - 2*str;

        // Run angle PID controller in background, such that it matches when controller takes over, if needed
        pidAngle.input = filterAngle;
        pidAngleOutput = pidAngle.calculate();
        // pidSpeed.setpoint = avgSpeed;
        // pidSpeed.input = -(motLeft.speed+motRight.speed)/2/100.0;
        // pidSpeedOutput = pidSpeed.calculate();
      }
      // Serial << motLeft.speed << "\t" << motRight.speed << "\t" << overrideMode << endl;
    }

    motLeft.update();
    motRight.update();
    // updateStepper(&motLeft);
    // updateStepper(&motRight);

    // Measure battery voltage, and send to connected client(s), if any
    float newBatteryVoltage = 0; //analogRead(PIN_BATTERY_VOLTAGE);
    uint32_t reading =  adc1_get_raw(ADC_CHANNEL_BATTERY_VOLTAGE);
    uint32_t voltage = esp_adc_cal_raw_to_voltage(reading, &adc_chars);
    avgBatteryVoltage = avgBatteryVoltage*BATTERY_VOLTAGE_FILTER_COEFFICIENT +
(voltage/1000.0)*BATTERY_VOLTAGE_SCALING_FACTOR*(1-BATTERY_VOLTAGE_FILTER_COEFFICIENT);

    // Send battery voltage readout periodically to web page, if any clients are connected
    static unsigned long tLastBattery;
    if (tNowMs - tLastBattery > 5000) {
      if (wsServer.connectedClients(0)>0) {
        char wBuf[10];

        sprintf(wBuf, "b%.1f", avgBatteryVoltage);
        wsServer.broadcastTXT(wBuf);
      }
      tLastBattery = tNowMs;
    }

    if (k==plot.prescaler) {
      k = 0;

      if (wsServer.connectedClients(0)>0 && plot.enable) {
        union {
          struct {
            uint8_t cmd = 255;
            uint8_t fill1;
            uint8_t fill2;
            uint8_t fill3;
            float f[13];
          };
          uint8_t b[56];
        } plotData;

        plotData.f[0] = micros()/1000000.0;
        plotData.f[1] = accAngle;
        plotData.f[2] = filterAngle;
        plotData.f[3] = pidAngle.setpoint;
        plotData.f[4] = pidAngle.input;
        plotData.f[5] = pidAngleOutput;
        plotData.f[6] = pidPos.setpoint;
        plotData.f[7] = pidPos.input;
        plotData.f[8] = pidPosOutput;
        plotData.f[9] = pidSpeed.setpoint;
        plotData.f[10] = pidSpeed.input;
        plotData.f[11] = pidSpeedOutput;
        // plotData.f[12] = noiseValue;?
        // plotData.f[9] = ayg;
        // plotData.f[10] = azg;
        // plotData.f[11] = rxg;
        // plotData.f[11] = microStep;
        wsServer.sendBIN(0, plotData.b, sizeof(plotData.b));
```

```
      }
    }
    k++;


    // Serial << filterAngle << "\t" << angleErrorIntegral << "\t" << enableControl << "\t" << disableControl << "\t" << selfRight << endl;

    // Serial << selfRight;
    // Serial << remoteControl.speed << "\t" << remoteControl.steer << endl;

    // Serial << microStep << "\t" << absSpeed << "\t" << endl;

    // Serial << endl;

    parseSerial();

    // Serial << micros()-tNow << "\t";

    tLast = tNow;

     // Run other tasks
    ArduinoOTA.handle();

    wsServer.loop();


    // Serial << micros()-tNow << endl;
  }

  // delay(1);
}

void parseSerial() {
  static char serialBuf[63];
  static uint8_t pos = 0;
  char currentChar;

  while (Serial.available()) {
    currentChar = Serial.read();
    serialBuf[pos++] = currentChar;
    if (currentChar == 'x') {
      parseCommand(serialBuf, pos);
      pos = 0;
      while (Serial.available()) Serial.read();
      memset(serialBuf, 0, sizeof(serialBuf));
    }
  }

}

void parseCommand(char* data, uint8_t length) {
  float val2;
  if ((data[length-1]=='x') && length>=3) {
    switch (data[0]) {
      case 'c': { // Change controller parameter
        uint8_t controllerNumber = data[1] - '0';
        char cmd2 = data[2];
        float val = atof(data+3);

        // Make pointer to PID controller
        PID* pidTemp;
        switch (controllerNumber) {
          case 1: pidTemp = &pidAngle; break;
          case 2: pidTemp = &pidPos;   break;
          case 3: pidTemp = &pidSpeed; break;
        }

        switch (cmd2) {
          case 'p': pidTemp->K = val;  break;
          case 'i': pidTemp->Ti = val; break;
          case 'd': pidTemp->Td = val; break;
          case 'n': pidTemp->N = val; break;
          case 't': pidTemp->controllerType = (uint8_t) val; break;
          case 'm': pidTemp->maxOutput = val; break;
          case 'o': pidTemp->minOutput = -val; break;
        }
```

```
      pidTemp->updateParameters();

      Serial << controllerNumber << "\t" << pidTemp->K << "\t" << pidTemp->Ti << "\t" << pidTemp->Td << "\t" << pidTemp->N << "\t" <<
pidTemp->controllerType << endl;
      break;
    }
    case 'a': // Change angle offset
      angleOffset = atof(data+1);
      Serial << angleOffset << endl;
      break;
    case 'f':
      gyroFilterConstant = atof(data+1);
      Serial << gyroFilterConstant << endl;
      break;
    case 'v':
      motorCurrent = atof(data+1);
      Serial << motorCurrent << endl;
      dacWrite(PIN_MOTOR_CURRENT, motorCurrent);
      break;
    case 'm':
      val2 = atof(data+1);
      Serial << val2 << endl;
      controlMode = val2;
      break;
    case 'u':
      microStep = atoi(data+1);
      setMicroStep(microStep);
      break;
    case 'g':
      gyroGain = atof(data+1);
      break;
    case 'p': {
      switch (data[1]) {
        case 'e':
          plot.enable = atoi(data+2);
          break;
        case 'p':
          plot.prescaler = atoi(data+2);
          break;
        case 'n': // Noise source enable
          noiseSourceEnable = atoi(data+2);
          break;
        case 'a': // Noise source amplitude
          noiseSourceAmplitude = atof(data+2);
          break;
      }
      break;
    }
    case 'j':
      gyroGain = atof(data+1);
      break;
    case 'k': {
      uint8_t cmd2 = atoi(data+1);
      if (cmd2==1) {  // calibrate gyro
        calculateGyroOffset(100);
      } else if (cmd2==2) {  // calibrate acc
        Serial << "Updating angle offset from " << angleOffset;
        angleOffset = filterAngle;
        Serial << " to " << angleOffset << endl;
        preferences.putFloat("angle_offset", angleOffset);
      }
      break;}
    case 'l':
      maxStepSpeed = atof(&data[1]);
      break;
    case 'n':
      gyroFilterConstant = atof(&data[1]);
      break;
    case 'r': { // r for remote control
      char cmd2 = data[1];
      switch (cmd2) {
        case 's':
          remoteControl.selfRight = 1;
          break;
        case 'd':
          remoteControl.disableControl = 1;
```

```
        break;
       case 'o':
        remoteControl.override = 1;
        break;
       case 'c': {
        char *p = data+1;
        int16_t param1 = 0, param2 = 0;
        param1 = atoi(++p);
        p = strchr(p, ',');
        if (p)
        {
         param2 = atoi(++p);
        }
        remoteControl.speed = param2 * remoteControl.speedGain;
        remoteControl.steer = param1 * remoteControl.steerGain;
        Serial << "Speed, steer: " << param1 << ", " << param2 << endl;
        break;
       }
       case 'g':
        remoteControl.speedGain = atof(data+2);
        break;
       case 'h':
        remoteControl.steerGain = atof(data+2);
        break;
      }
      break;
     }
     case 'w': {
      char cmd2 = data[1];
      char buf[63];
      uint8_t len;

      switch (cmd2) {
       case 'r':
        Serial.println("Rebooting...");
        ESP.restart();
        // pidParList.sendList(&wsServer);
        break;
       case 'l': // Send wifi networks to WS client
        sendWifiList();
        break;
       case 's': // Update WiFi SSID
        len = length-3;
        memcpy(buf, &data[2], len);
        buf[len] = 0;
        preferences.putBytes("wifi_ssid", buf, 63);
        Serial << "Updated WiFi SSID to: " << buf << endl;
        break;
       case 'k': // Update WiFi key
        len = length-3;
        memcpy(buf, &data[2], len);
        buf[len] = 0;
        preferences.putBytes("wifi_key", buf, 63);
        Serial << "Updated WiFi key to: " << buf << endl;
        break;
       case 'm': // WiFi mode (0=AP, 1=use SSID)
        preferences.putUInt("wifi_mode", atoi(&data[2]));
        Serial << "Updated WiFi mode to (0=access point, 1=connect to SSID): " << atoi(&data[2]) << endl;
        break;
       case 'n': // Robot name
        len = length-3;
        memcpy(buf, &data[2], len);
        buf[len] = 0;
        if (len>=4) {
         preferences.putBytes("robot_name", buf, 63);
        }
        Serial << "Updated robot name to: " << buf << endl;
        break;
      }
      break;}
     case 'z':
      sendConfigurationData(0);
      break;
    }
   }
  }
```

```
void sendWifiList(void) {
  char wBuf[200];
  uint8_t n;
  uint16_t pos = 2;

  wBuf[0] = 'w';
  wBuf[1] = 'l';

  Serial.println("Scan started");
  n = WiFi.scanNetworks();

  if (n>5) n = 5; // Limit to first 5 SSIDs

  // Make concatenated list, separated with commas
  for (uint8_t i=0; i<n; i++) {
    pos += sprintf(wBuf + pos, "%s,", WiFi.SSID(i).c_str());
  }
  wBuf[pos-1] = 0;

  Serial.println(wBuf);
  wsServer.sendTXT(0, wBuf);
}

void calculateGyroOffset(uint8_t nSample) {
  int32_t sumX = 0, sumY = 0, sumZ = 0;
  int16_t x, y, z;

  for (uint8_t i=0; i<nSample; i++) {
    imu.getRotation(&x, &y, &z);
    sumX += x;
    sumY += y;
    sumZ += z;
    delay(5);
  }

  gyroOffset[0] = sumX/nSample;
  gyroOffset[1] = sumY/nSample;
  gyroOffset[2] = sumZ/nSample;

  for (uint8_t i=0; i<3; i++) {
    char buf[16];
    sprintf(buf, "gyro_offset_%u", i);
    preferences.putShort(buf, gyroOffset[i]);
  }

  Serial << "New gyro calibration values: " << gyroOffset[0] << "\t" << gyroOffset[1] << "\t" << gyroOffset[2] << endl;
}

void readSensor() {
  int16_t ax, ay, az, gx, gy, gz;
  float deltaGyroAngle;

  imu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

  // accAngle = atan2f((float) ax, (float) az) * 180.0/M_PI;
  // deltaGyroAngle = -((float)((gy - gyroOffset[1])) / GYRO_SENSITIVITY) * dT * gyroGain;
    accAngle = atan2f((float) ay, (float) az) * 180.0/M_PI - angleOffset;
    deltaGyroAngle = ((float)((gx - gyroOffset[0])) / GYRO_SENSITIVITY) * dT * gyroGain;

  filterAngle = gyroFilterConstant * (filterAngle + deltaGyroAngle) + (1 - gyroFilterConstant) * (accAngle);

  // Serial << ay/1000.0 << "\t" << az/1000.0 << "\t" << accAngle << "\t" << filterAngle << endl;
  ayg = (ay*9.81)/16384.0;
  azg = (az*9.81)/16384.0;
  rxg = ((float)((gx - gyroOffset[0])) / GYRO_SENSITIVITY);
  // Serial << ayf << "\t"<< azf << "\t" << accAngle << endl;
}

void initSensor(uint8_t n) {
  float gyroFilterConstantBackup = gyroFilterConstant;
  gyroFilterConstant = 0.8;
  for (uint8_t i=0; i<n; i++) {
    readSensor();
  }
  gyroFilterConstant = gyroFilterConstantBackup;
```

```
}

void setMicroStep(uint8_t uStep) {
 // input:                  1 2 4 8 16 32
 // uStep table corresponds to 0 1 2 3 4  5  in binary on uStep pins
 // So, we need to take the log2 of input
 uint8_t uStepPow = 0;
 uint8_t uStepCopy = uStep;
 while (uStepCopy >>= 1) uStepPow++;

 digitalWrite(motUStepPin1, uStepPow&0x01);
 digitalWrite(motUStepPin2, uStepPow&0x02);
 digitalWrite(motUStepPin3, uStepPow&0x04);


}

void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length) {

   switch(type) {
      case WStype_DISCONNECTED:
         Serial.printf("[%u] Disconnected!\n", num);
         digitalWrite(PIN_LED_LEFT, 0);
         break;
      case WStype_CONNECTED: {
            IPAddress ip = wsServer.remoteIP(num);
            Serial.printf("[%u] Connected from %d.%d.%d.%d url: %s\n", num, ip[0], ip[1], ip[2], ip[3], payload);
            wsServer.sendTXT(num, "Connected");
            digitalWrite(PIN_LED_LEFT, 1);
         }
         break;
      case WStype_TEXT:
         Serial.printf("[%u] get Text: %s\n", num, payload);
         parseCommand((char*) payload, length);
         break;
      case WStype_BIN: {
         // Serial.printf("[%u] get binary length: %u\n", num, length);

         if (length==6) {
           cmd c;
           memcpy(c.arr, payload, 6);
           Serial << "Binary: " << c.grp << "\t" << c.cmd << "\t" << c.val << "\t" << sizeof(cmd) << endl;
         }

         break;
       }
                        case WStype_ERROR:
                        case WStype_FRAGMENT_TEXT_START:
                        case WStype_FRAGMENT_BIN_START:
                        case WStype_FRAGMENT:
                        case WStype_FRAGMENT_FIN:
                                break;
   }

}

void sendConfigurationData(uint8_t num) {
 // send message to client
 char wBuf[63];
 char buf[63];
 sprintf(wBuf, "c%dp%.4f", 1, pidAngle.K);
 wsServer.sendTXT(num, wBuf);
 sprintf(wBuf, "c%di%.4f", 1, pidAngle.Ti);
 wsServer.sendTXT(num, wBuf);
 sprintf(wBuf, "c%dd%.4f", 1, pidAngle.Td);
 wsServer.sendTXT(num, wBuf);
 sprintf(wBuf, "c%dn%.4f", 1, pidAngle.N);
 wsServer.sendTXT(num, wBuf);
 sprintf(wBuf, "c%dr%.4f", 1, pidAngle.R);
 wsServer.sendTXT(num, wBuf);
 sprintf(wBuf, "c%dm%.4f", 1, pidAngle.maxOutput);
 wsServer.sendTXT(num, wBuf);
 sprintf(wBuf, "c%do%.4f", 1, -pidAngle.minOutput);
 wsServer.sendTXT(num, wBuf);
 sprintf(wBuf, "c%dp%.4f", 2, pidPos.K);
```

```
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "c%di%.4f", 2, pidPos.Ti);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "c%dd%.4f", 2, pidPos.Td);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "c%dn%.4f", 2, pidPos.N);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "c%dr%.4f", 2, pidPos.R);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "c%dm%.4f", 2, pidPos.maxOutput);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "c%do%.4f", 2, -pidPos.minOutput);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "c%dp%.4f", 3, pidSpeed.K);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "c%di%.4f", 3, pidSpeed.Ti);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "c%dd%.4f", 3, pidSpeed.Td);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "c%dn%.4f", 3, pidSpeed.N);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "c%dr%.4f", 3, pidSpeed.R);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "c%dm%.4f", 3, pidSpeed.maxOutput);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "c%do%.4f", 3, -pidSpeed.minOutput);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "h%.4f", speedFilterConstant);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "i%.4f", steerFilterConstant);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "v%d", motorCurrent);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "j%.4f", gyroGain);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "n%.4f", gyroFilterConstant);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "l%.4f", maxStepSpeed);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "wm%d", preferences.getUInt("wifi_mode", 0));  // 0=AP, 1=Client
  wsServer.sendTXT(num, wBuf);
  preferences.getBytes("wifi_ssid", buf, 63);
  sprintf(wBuf, "ws%s", buf);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "wn%s", robotName);
  wsServer.sendTXT(num, wBuf);
  sprintf(wBuf, "wb%s", BTaddress);
  wsServer.sendTXT(num, wBuf);
}
```

فصل پنجم: راه اندازی و استفاده از دستگاه

برای استفاده و راه اندازی از ربات، ابتدا باتری شارژ کرده و با استفاده از جک DC تعبیه شده، به مدار متصل میکنیم. با روشن کردن ربات، wifi داخلی آن با نام SelfBalancing قابل مشاهده خواهد بود که رمز آن turboturbo است. امکان تعویض wifi داخلی به wifi دیگر، از 2 طریق دستورات سریال و یا صفحه تنظیمات امکان پذیر میباشد.
برای آغاز تعادل ربات باید در حالت افقی روشن شده و سپس با دست به حالت عمودی و تعادل نگه داشته شود، در این حالت، ربات شروع به کار خواهد کرد و دیگر نیازی به اعمال نیروی خارجی برای حفظ تعادل نخواهد بود. به دلیل تعریف زاویه threshold نیاز است تا ربات در حالت اولیه افقی روشن و سپس به صورت دستی برای آغاز به کار به حالت عمودی در بیاید.
برای کنترل کردن حرکت ربات، باید به صفحه های وب آن مراجعه کرد. در صورتی که از wifi داخلی استفاده شود، ایپی آن 192.168.4.1 خواهد بود. لیست صفحات به قرار زیر است:

- 192.168.4.1/control.htm
- 192.168.4.1/control2.htm
- 192.168.4.1/index.htm
- 192.168.4.1/index2.htm
- 192.168.4.1/index3.htm
- 192.168.4.1/plotTest.htm
- 192.168.4.1/simpletuning.htm

در صورتی که wifi خارجی استفاده شود، نیاز است که آدرس ایپی داده شده به ربات را با استفاده از serial monitor مشاهده کنید و با آدرس پیشفرض تغییر بدهید. یا میتوانید از آدرس balancingrobot.local/ADDRESS.htm استفاده کنید و ADDRESS را با صفحه مورد نظر تغییر بدهید.

صفحه index دسترسی به صفحات دیگر شامل تنظیم ساده PID، تنظیم دقیق تر PID و صفحه کنترل حرکت را میدهد. علاوه بر آن، مقدار ولتاژ باتری را نشان میدهد.

با استفاده از صفحه control میتوان حرکت ربات را کنترل کرد و به آن جهت داد. قابلیت تنظیم مقدار حساسیت کنترل را نیز وجود دارد که برای حرکت های دقیق مناسب تر است.

صفحه PlotTest، از مقادیر مشخص شده بر حسب زمان نمودار میکشد که با استفاده از آن میتوان تنظیم PID بسیار دقیقی به دست آورد.

دستورات serial monitor

با استفاده از یک برنامه serial monitor مورد نظر (در این پروژه از قابلیت monitor در PlatformIO در VScode استفاده شده است) با baudrate 115200 میتوان به ربات دستور داد.

دستور تغییر wifi

وایفای های اطراف

wlx

ساختار دستور به این صورت است:

wsYOURSSIDx
wkYOURKEYx
wm1x

برای تنظیمات مربوط به wireless از یک w، s برای SSID، k برای key، m برای mode وایفای (0=access point, 1=connect to SSID) و در نهایت x برای پایان.

دستور Restart کردن

wrx

دستور کالیبره کردن IMU:

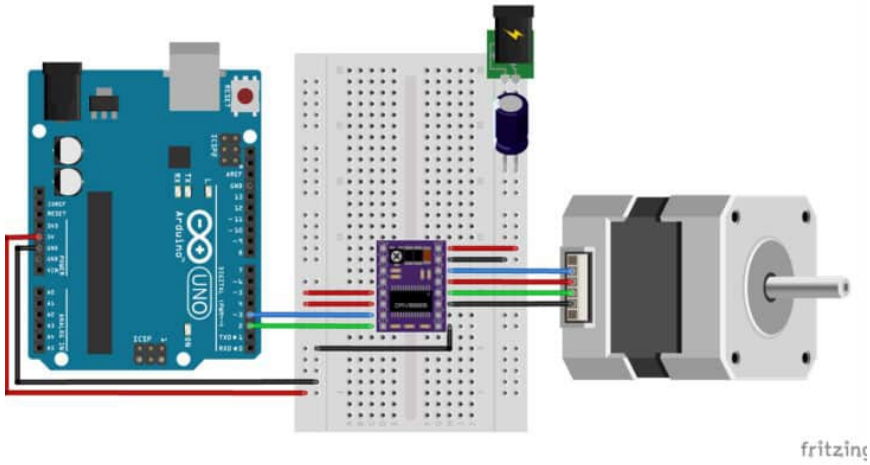
- Gyroscope باید ربات به صورت افقی بدون حرکت روی زمین قرار گیرد
k1x
- Accelerometer باید ربات به صورت عمودی و بدون حرکت قرار گیرد
k2x

مشاهده gain های کنترل

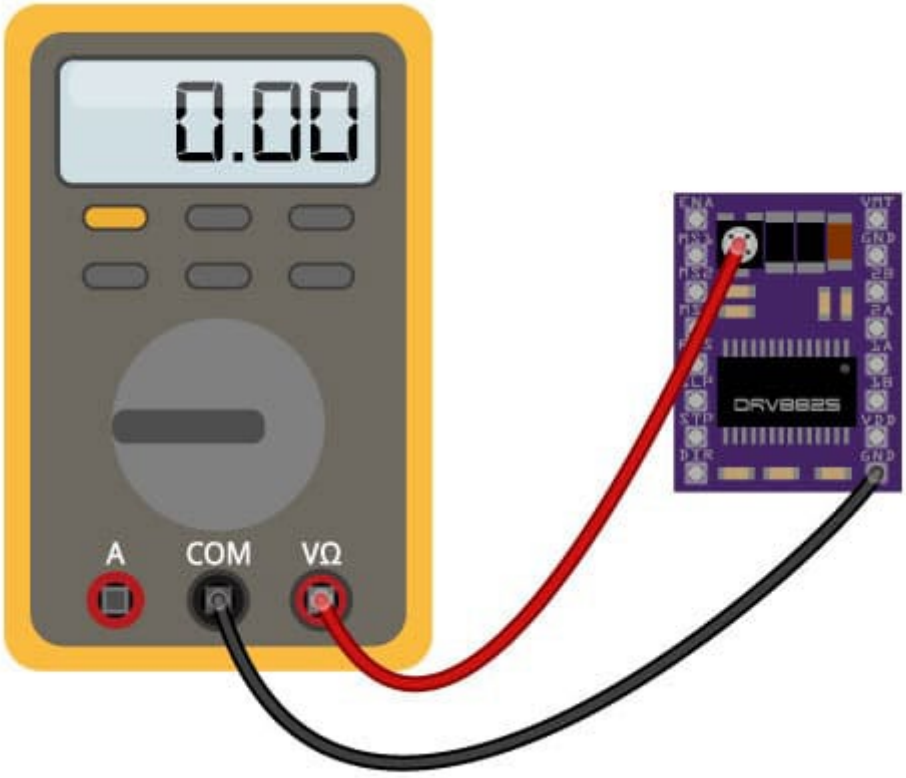
rcx

تنظیم جریان موتور ها

درایور drv8825 قابلیت تنظیم حداکثر جریان انتقالی را دارد، برای استفاده بهینه از ربات پیشنهاد میشود که جریان هر موتور بر روی حداکثر 400 میلی آمپر تنظیم شود، برای این کار نیاز به یک مدار دیگر به صورت زیر است:



با استفاده از مولتی متر، VREF در درایور را اندازه گرفته، مقدار جریان $CurrentLimit = v_{ref} \times 2$ خواهد بود. برای اطلاعات بیشتر به این سایت مراجعه شود.



Vref