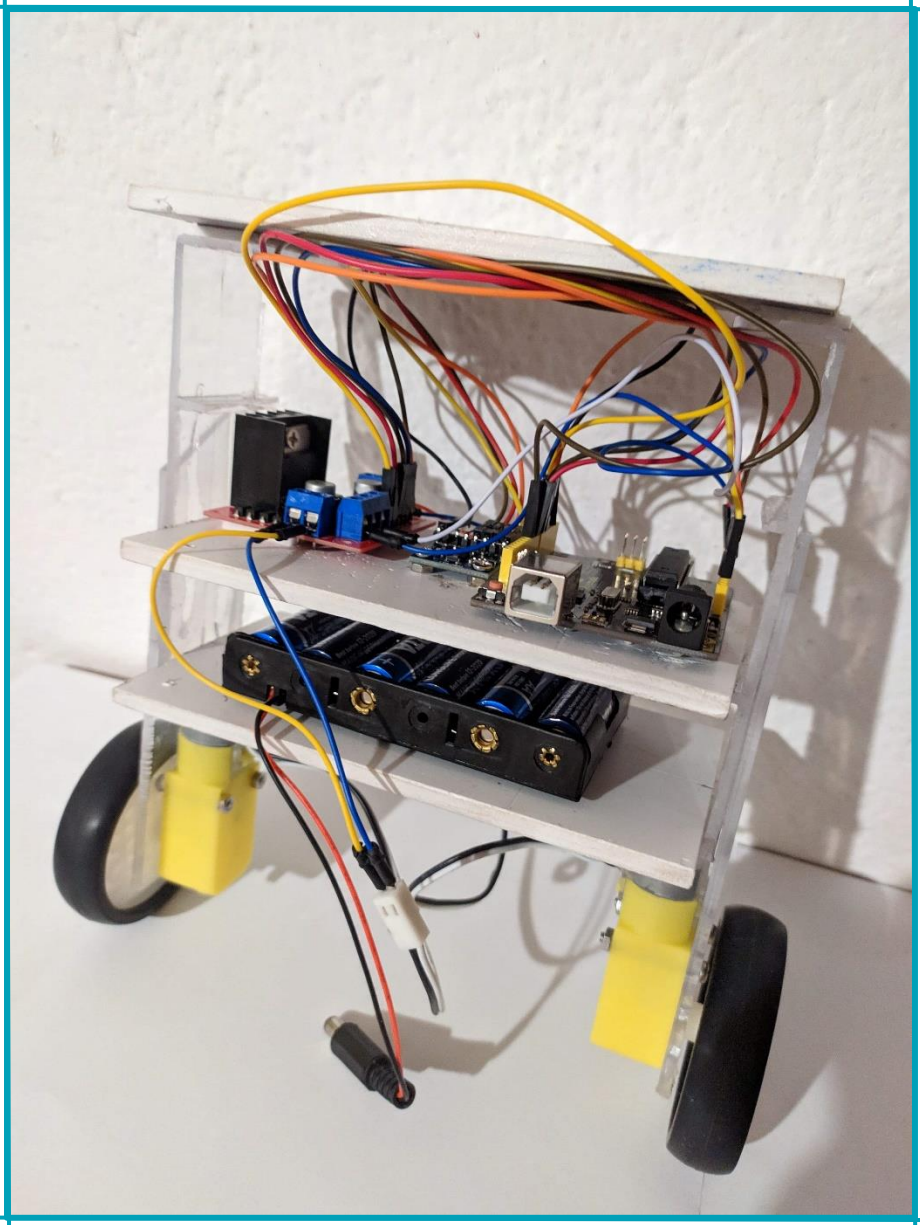


Self-balancing robot

САМОБАЛАНСИРАЩ СЕ РОБОТ С АРДУИНО

Изготвили:
Ади Хаджиев | Амира Емин | Амина Бисерова
Селви Хутев | Дениза Дурева

Съдържание

ОПИСАНИЕ НА ПРОЕКТА/ SELF-BALANCING ROBOT

Двуколесен или самобалансиращ се робот е нестабилна динамична система за разлика от другите четириколесни роботи, които са винаги в равновесно състояние. Под нестабилна тук имаме предвид, че роботът е свободен да пада напред или назад, без да му бъде приложена сила. А самобалансиращ се, означава, че роботът сам балансира в равновесно състояние, тоест изправено на 90 градуса. Този проект работи върху концепцията за обърнато махало.

За да се поддържа балансиран роботът, двигателите трябва да противодействат на падащия робот. Това действие изисква обратна връзка и коригиращи елементи. Елементът за обратна връзка е жироскопът MPU6050 + акселерометър, който дава както ускорение, така и въртене и в трите оси. Arduino използва това, за да знае текущата ориентация на робота. Коригиращият елемент е комбинацията от двигател/мотор и колело.

СЪСТАВНИ ЧАСТИ

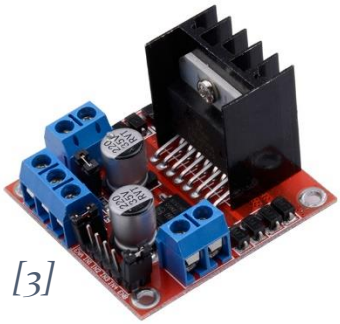
В изработката на проекта са използвани следните части:

- Arduino UNO (Ардуино УНО платка) [1]
- Geared DC motors (мотори с постоянен ел. ток) – 2 броя [2]
- L298N Motor Driver Module (моторен драйвер L298N) [3]
- MPU6050 [4]
- Чифт гуми [5]
- 7.4V Li-ion батерии – 6 броя [6]
- Свързващи кабели [7]
- Тяло за робота, изградено от пластмасови пластини [8]

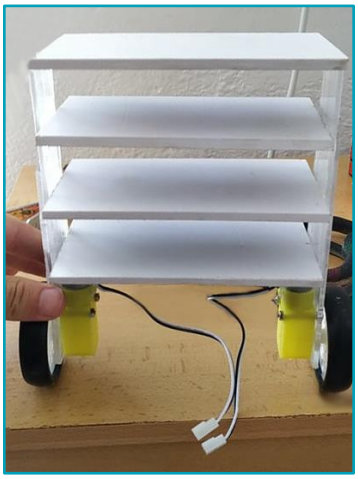
[1]

[2]

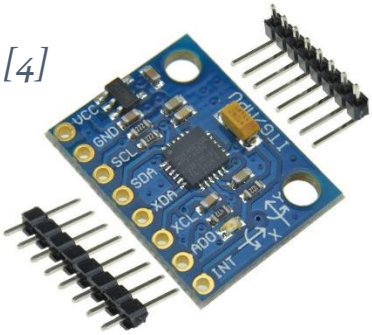
[3]



[8]



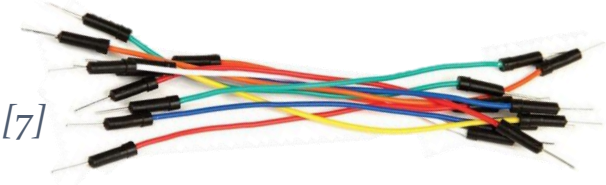
[4]



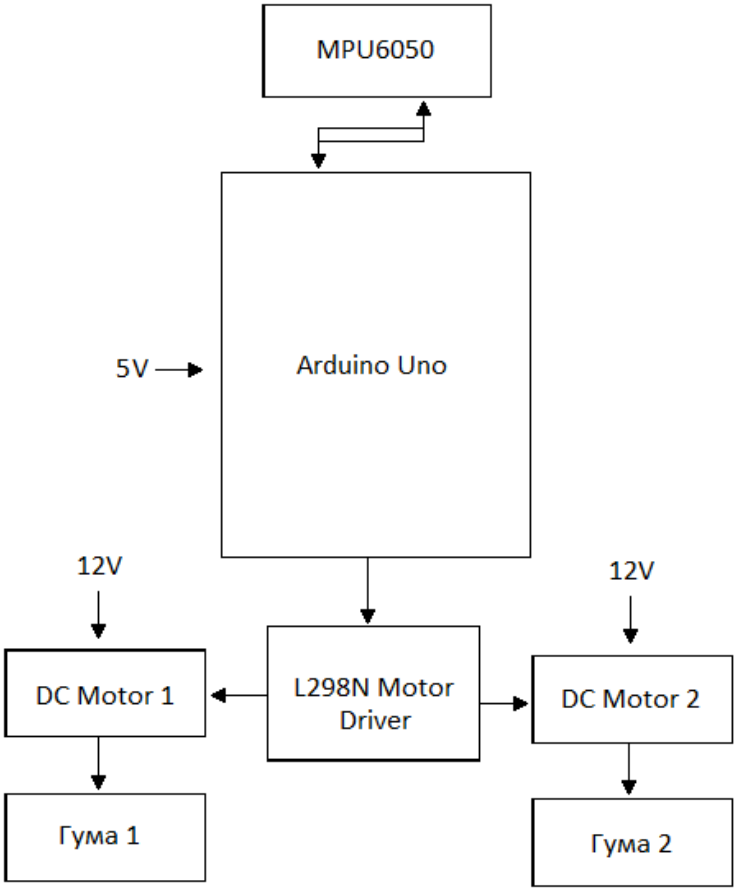
[5]

[6]

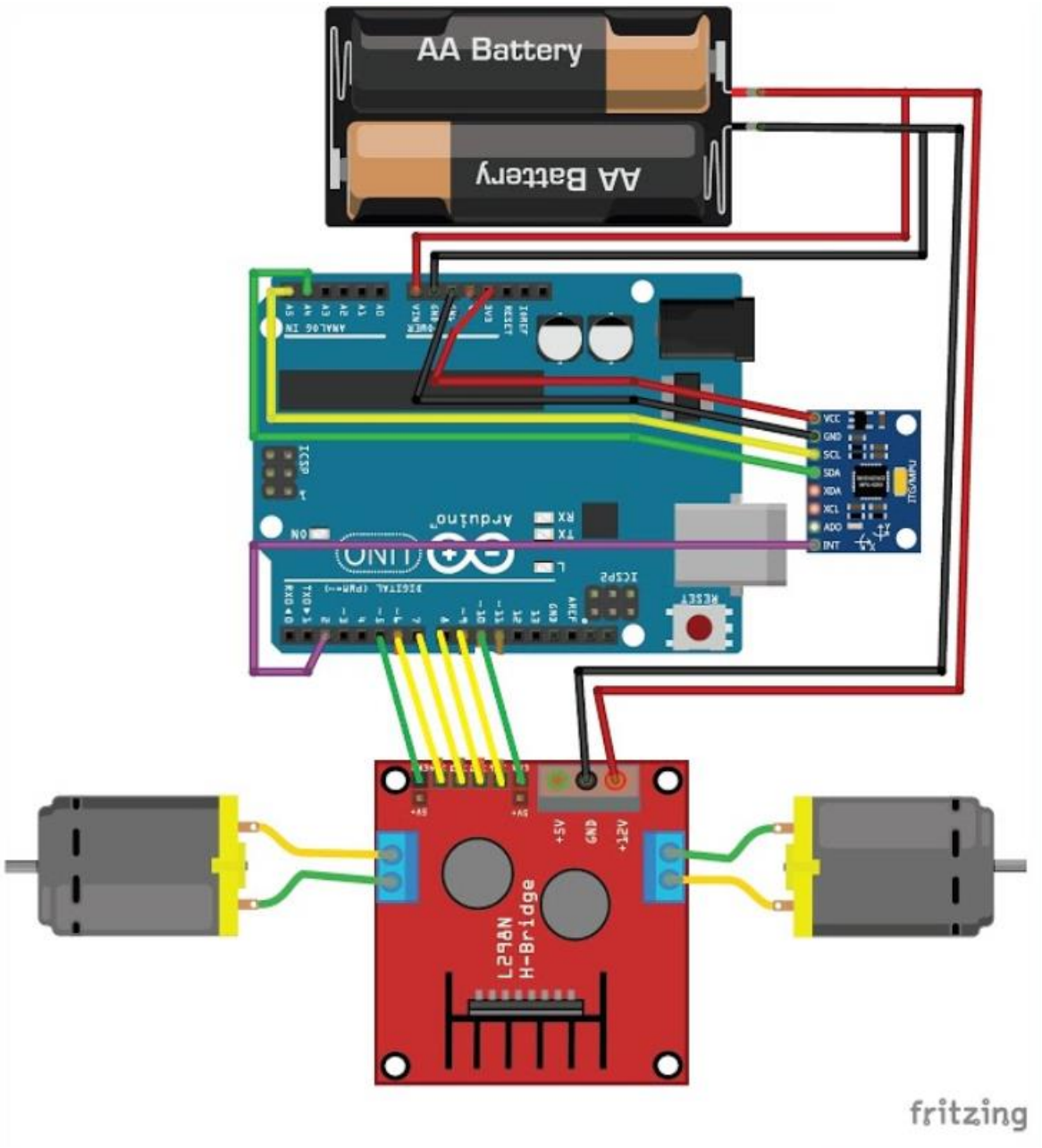
[7]



БЛОКОВА СХЕМА



- ЕЛЕКТРИЧЕСКА СХЕМА



СГЛОБЯВАНЕ НА РОБОТА

След направата на шасито (тялото на робота) от пластмасови пластини, при това взето на предвид, че то трябва да бъде здраво и да не се клати при балансирането на робота, следва свързването на MPU6050 с Ардуино платката и свързването на двигателите чрез модулният моторен драйвер, цялата конструкция захранена от батериите.

Arduino и моторният драйвер L298N се захранват директно през Vin пина и 12-волтовия терминал. Вграденият регулатор на Arduino платката преобразува входа 7.4V в 3V и MPU6050 се захранва от него. Свързваме 7.4-волтовия положителен проводник/жица от батерията към 12-волтовия входен терминал на моторния драйвер. Това карайки моторите да работят с 7.4V.

Таблиците по-долу показват как са свързани MPU6050 и L298N към Ардуино платката.

ПИН	АРДУИНО	3V	GND	A5	A4	D2
	MPU6050	VCC	GND	SCL	SDA	INT

ПИН	АРДУИНО	D6	D7	D8	D9	D5	D10
	L298N	IN1	IN2	IN3	IN4	ENA	ENB

MPU6050 комуникира с Arduino чрез I2C интерфейс, затова ние използваме SPI пиновете A4 и A5 на Arduino. DC двигателите са свързани към PWM пиновете D6, D7 D8 и D9. Те са свързани към PWM пинове, защото чрез тях, ние ще контролираме скоростта на постояннотоковия двигател, като променяме работния цикъл на PWM сигналите.

- ## ПРОГРАМИРАНЕ НА РОБОТА (СОРС КОД)

Концепцията зад работата на робота е проста – трябва да проверим дали е наклонен напред или назад с помощта на MPU6050, и в зависимост от това, да завъртим колелата в съответната посока, така, че роботът да възвръща равновесното си състояние. В същото време трябва да контролираме скоростта, с която се въртят колелата, така ако ботът е леко дезориентиран от централното положение, колелата се въртят бавно и скоростта се увеличава, при отдалечаване от централната позиция. За да постигнем това, използваме PID (proportional integral derivative) алгоритъм, който има централната позиция като зададена стойност и нивото на дезориентация като изход.

Използвали сме готови библиотеки, които да изпълнят PID калкулацията и да ни върнат стойността на отклонението от MPU6050. Библиотеката LMotorController се използва за задвижване на двата мотора с L298N модула. Библиотеките I2Cdev и MPU6050_6_Axis_MotionApps20 са за четене на данни от MPU6050.

Библиотеките са взети от следните GitHub клонове, след което са добавени към нашата Arduino lib директория.

https://github.com/kurimawxxoo/arduino-self-balancing-robot

Започваме програмата с добавянето на същите тези библиотеки, и на вградената I2C такава.

```
#include <PID_v1.h>
#include <LMotorController.h>
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

#define MIN_ABS_SPEED 20
```

След това декларираме променливите, които са необходими за получаване на данните от сензора MPU6050. Ние четем както гравитационния вектор, така и кватернионните стойности и след това изчисляваме височината на наклона и стойността на въртене на робота. Плаващият масив ypr[3] ще съдържа крайния резултат.

```
MPU6050 mpu;

// MPU control/status vars
bool dmpReady = false; // set true if DMP init was successful
uint8_t mpuIntStatus; // holds actual interrupt status byte from MPU
uint8_t devStatus; // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize; // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount; // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q; // [w, x, y, z] quaternion container
VectorFloat gravity; // [x, y, z] gravity vector
float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector
```

Въвеждаме и настройваме правилните стойности нужни за PID алгоритъма, в нашия случай са тези:

```
//PID
double originalSetpoint = 175.5;
double setpoint = originalSetpoint;
double movingAngleOffset = 0.1;
double input, output;

//adjust these values to fit your own design
double Kp = 85;
double Kd = 2;
double Ki = 650;
```

На следващия ред инициализираме PID алгоритъма, като предаваме входните променливи input, output, setpoint, Kp, Ki и Kd. От тях, setpoint, Kp, Ki и Kd вече са зададени, стойността на input ще бъде текущата стойност на отклонението, прочетена от MPU6050 сензора, а стойността на output ще бъде тази, която се изчислява от PID алгоритъма, което означава, че този алгоритъм ще ни даде изходна стойност, която ще се използва за коригиране на входната такава, така че тя да е близка до зададената точка (setpoint).

```
PID pid(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);
```

След което задаваме стойностите за моторния контролер.

```
double motorSpeedFactorLeft = 0.8;
double motorSpeedFactorRight = 0.8;
//MOTOR CONTROLLER
int ENA = 5;
int IN1 = 6;
int IN2 = 7;
int IN3 = 8;
int IN4 = 9;
int ENB = 10;
LMotorController motorController(ENA, IN1, IN2, ENB, IN3, IN4, motorSpeedFactorLeft, motorSpeedFactorRight);

volatile bool mpuInterrupt = false; // indicates whether MPU interrupt pin has gone high
void dmpDataReady()
{
mpuInterrupt = true;
}
```

Следва void Setup() функцията.

```
void setup()
{
// join I2C bus (I2Cdev library doesn't do this automatically)
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
Wire.begin();
TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
Fastwire::setup(400, true);
#endif
```

В нея инициализираме MPU6050 чрез конфигурирането на DMP (Digital Motion Processor), което ни помага да комбинираме данните от акселерометъра с тези на жироскопа и да получим надеждни стойности за Yaw, Pitch и Roll.

```
mpu.initialize();

devStatus = mpu.dmpInitialize();

// supply your own gyro offsets here, scaled for min sensitivity
mpu.setXGyroOffset(220);
mpu.setYGyroOffset(76);
mpu.setZGyroOffset(-85);
mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

// make sure it worked (returns 0 if so)
if (devStatus == 0)
{
// turn on the DMP, now that it's ready
mpu.setDMPEnabled(true);

// enable Arduino interrupt detection
attachInterrupt(0, dmpDataReady, RISING);
mpuIntStatus = mpu.getIntStatus();

// set our DMP Ready flag so the main loop() function knows it's okay to use it
dmpReady = true;

// get expected DMP packet size for later comparison
packetSize = mpu.dmpGetFIFOPacketSize();
```

Завършваме void Setup() метода със задаването на стойностите на PID.

```
//setup PID
pid.SetMode(AUTOMATIC);
pid.SetSampleTime(10);
pid.SetOutputLimits(-255, 255);
}
else
{
// ERROR!
// 1 = initial memory load failed
// 2 = DMP configuration updates failed
// (if it's going to break, usually the code will be 1)
Serial.print(F("DMP Initialization failed (code "));
Serial.print(devStatus);
Serial.println(F(")"));
}
}
```

В Loop() метода, проверяваме дали данните от MPU6050 са готови да бъда прочетени, и ако да, ние ги използваме, за да пресметнем стойността на PID, базирано на която след това, моторният контролер започва да се движи.

```
void loop()
{
// if programming failed, don't try to do anything
if (!dmpReady) return;

// wait for MPU interrupt or extra packet(s) available
while (!mpuInterrupt && fifoCount < packetSize)
{
//no mpu data - performing PID calculations and output to motors
pid.Compute();
motorController.move(output, MIN_ABS_SPEED);

}
```

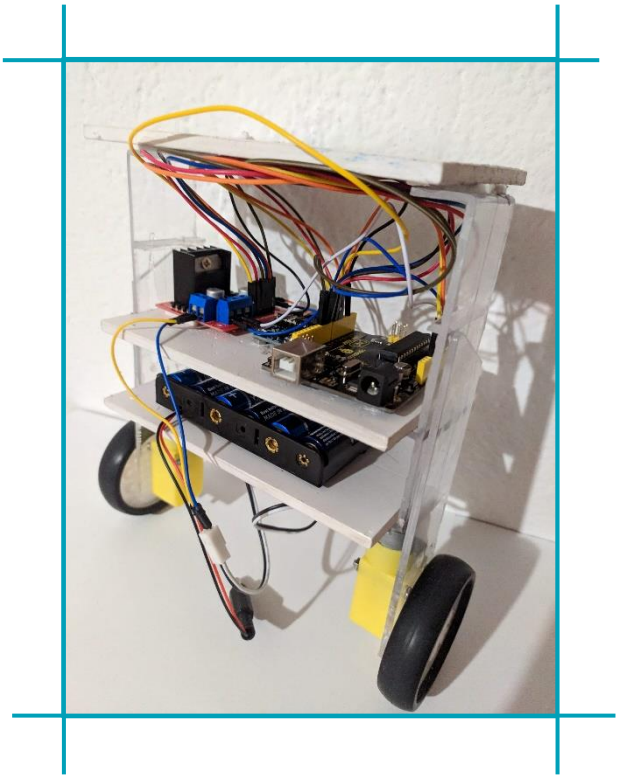
```
// reset interrupt flag and get INT_STATUS byte
mpuInterrupt = false;
mpuIntStatus = mpu.getIntStatus();

// get current FIFO count
fifoCount = mpu.getFIFOCount();

// check for overflow (this should never happen unless our code is too inefficient)
if ((mpuIntStatus & 0x10) || fifoCount == 1024)
{
// reset so we can continue cleanly
mpu.resetFIFO();
Serial.println(F("FIFO overflow!"));

// otherwise, check for DMP data ready interrupt (this should happen frequently)
}
else if (mpuIntStatus & 0x02)
{
// wait for correct available data length, should be a VERY short wait
while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

// read a packet from FIFO
mpu.getFIFOBytes(fifoBuffer, packetSize);

// track FIFO count here in case there is > 1 packet available
// (this lets us immediately read more without waiting for an interrupt)
fifoCount -= packetSize;

mpu.dmpGetQuaternion(&q, fifoBuffer);
mpu.dmpGetGravity(&gravity, &q);
mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
input = ypr[1] * 180/M_PI + 180;
}
}
```

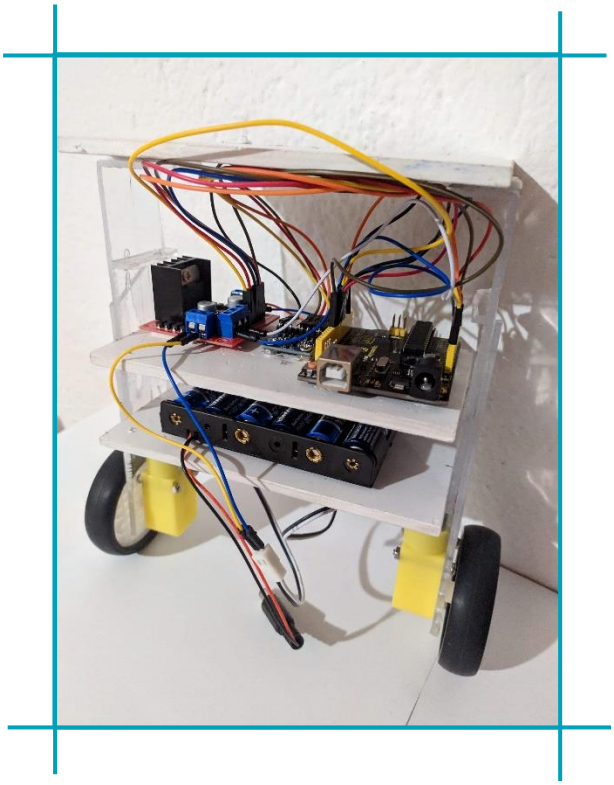
ЗАКЛЮЧЕНИЕ

Роботът се балансира на две колела, без да пада и с леко трептене.

Стойностите на PID контролера са изпробвани и тествани, както следва:

- Пропорционално/ Proportional (Kp) = 85
- Производно/ Derivative (Kd) = 2
- Интегрално (Цялостно)/ Integral (Ki) = 650.





Всички материали по проекта могат да бъдат разгледани тук:
https://github.com/ami566/Self-Balancing-Robot_Arduino-Project