# BattleBot

## Introducere

Proiectul este reprezentat de un robot controlat wireless cu o radiocomanda FlySky FS-I6. Funcționalitățile robotului sunt: deplasarea plană(directie de tip tanc), aruncătorul de flăcări(acționat de un switch de pe telecomandă) și controlul vitezei spinnerului cilindric.

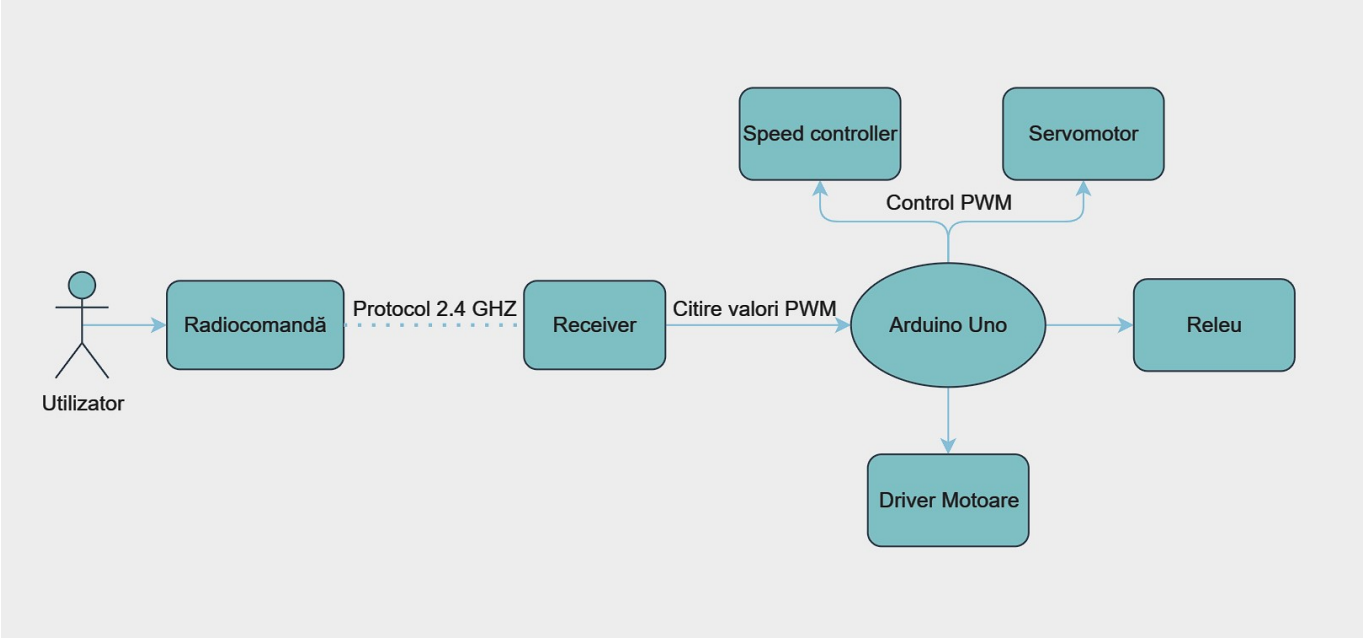
Inspirat de seria BattleBots, am dorit să proiectez un battlebot de la 0. Astfel, am ajuns la design-ul curent: Un șasiu simetric(pentru a funcționa și dacă este răsturnat), cu două roți motorizate pentru deplasare și un cilindru asimetric ca armă principală.

Robotul a fost printat 3D folosind o imprimantă Creality CR-10S Pro V2, iar design-ul a fost creat în Autodesk Fusion 360.

# Descriere generală

Schema bloc include șapte componente principale: radiocomanda, receiverul, microcontrollerul Arduino Uno, driverul de motoare pentru direcție, modulul cu releu care pornește transformatorul step-up, servomotorul și controllerul pentru motorul brushless.

Schema de mai jos descrie modul prin care componentele interacționează între ele.



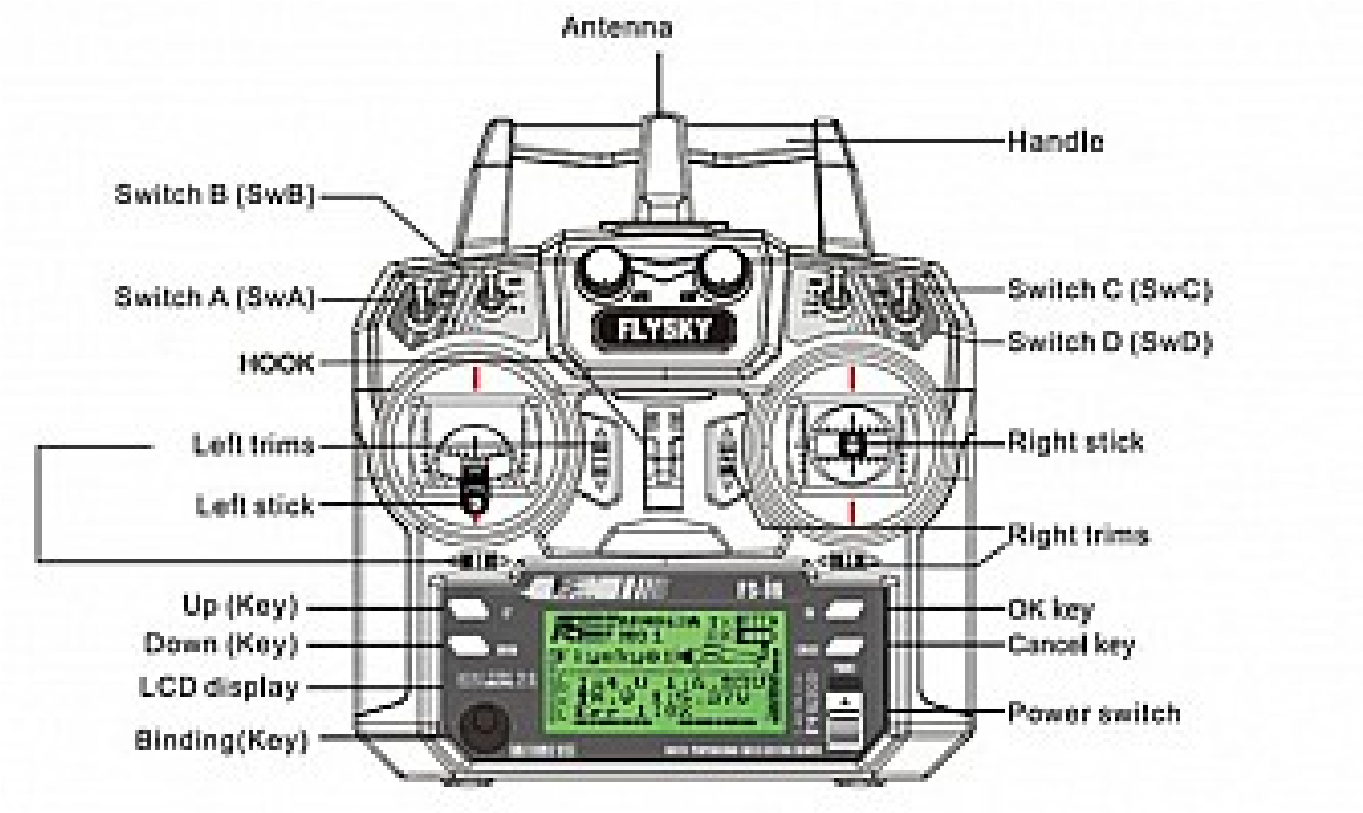

**Descriere componente:**

- **Radiocomanda:** Un sistem radio RC folosit pentru a controla diverse electronice. Transmite inputul la un receiver.
- **Receiverul:** Primește datele de la radiocomandă și le codifică într-ul semnal PWM, care poate fi citit de microcontroller.
- **Microcontrollerul:** Citește semnalul PWM din receiver, și îl folosește pentru a controla restul componentelor electronice.
- **Driverul de motoare:** Primește semnale de la microcontroller și controlează motoarele robotului pentru a aigura deplasarea.
- **Releul:** Primește un semnal de la microcontroller pentru a conecta la o sursă de curent un transformator stepUp.
- **Servomotorul:** Un motor cu rotație fixă, unghiul de rotație al acestuia este controlat de microcontroller folosind un semnal PWM.
- **Controllerul brushless:** Acesta controlează viteza de rotație a motorului brushless, primește valoarea PWM de la microcontroller.

Radiocomanda transmite receiverului 6 canale, dintre care noi folosim 4:

- Canalul 1: Joystick-ul drept, axa Ox - folosit pentru a controla virajul robotului.

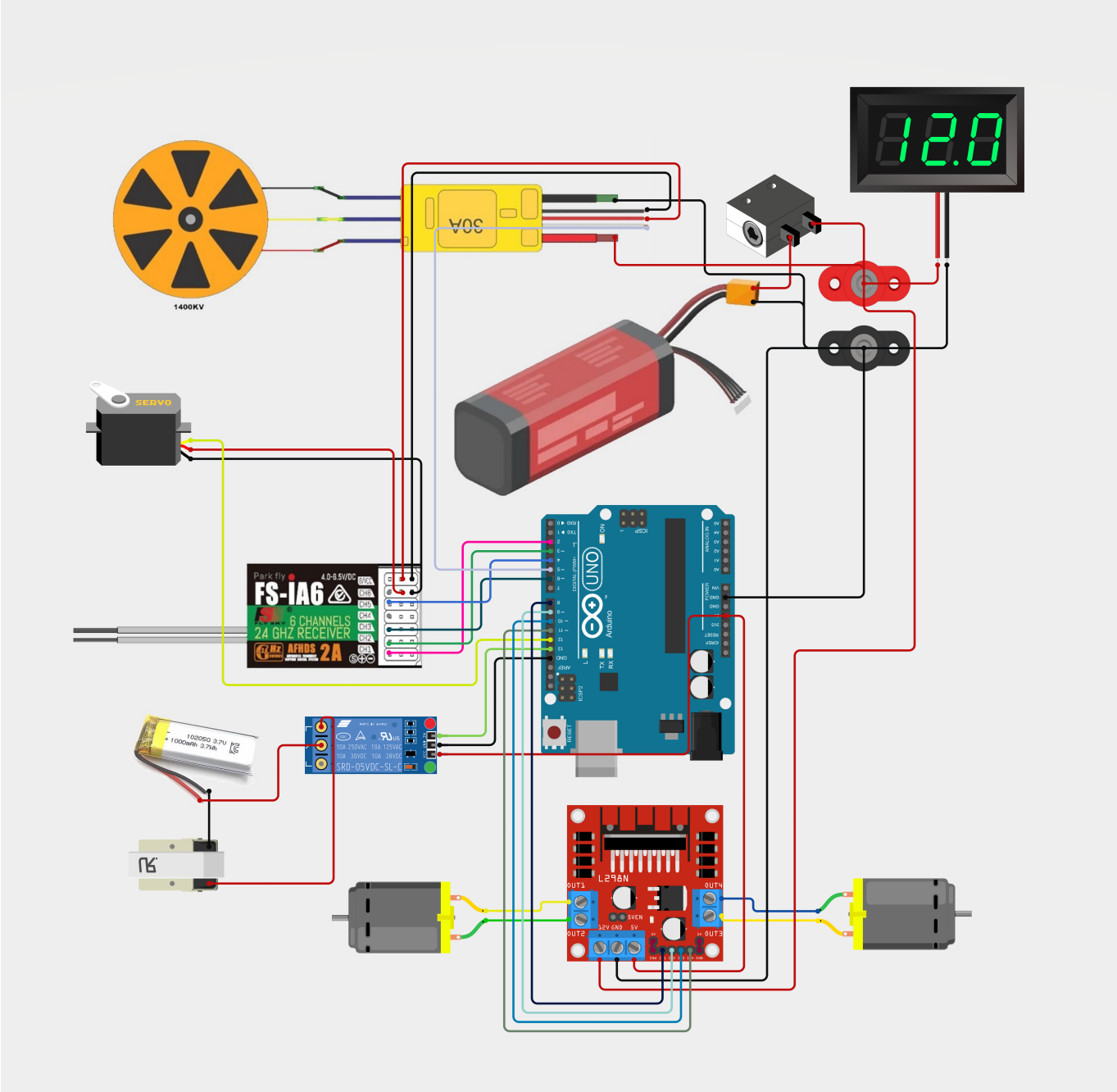
- Canalul 2: Joystick-ul drept, axa Oy - folosit pentru a controla direcția de deplasare a robotului.
- Canalul 3: Joystick-ul stâng, axa Oy - folosit pentru a controla viteza motorului brushless.
- Canalul 5: Switch-ul A - folosit pentru a activa flamethrowerul.




Astfel, acționând canalul 5 al teleccomenzii, robotul se va comporta în următorul fel:

# Hardware Design

În imaginea de mai jos se poate observa schema electrică a proiectului.

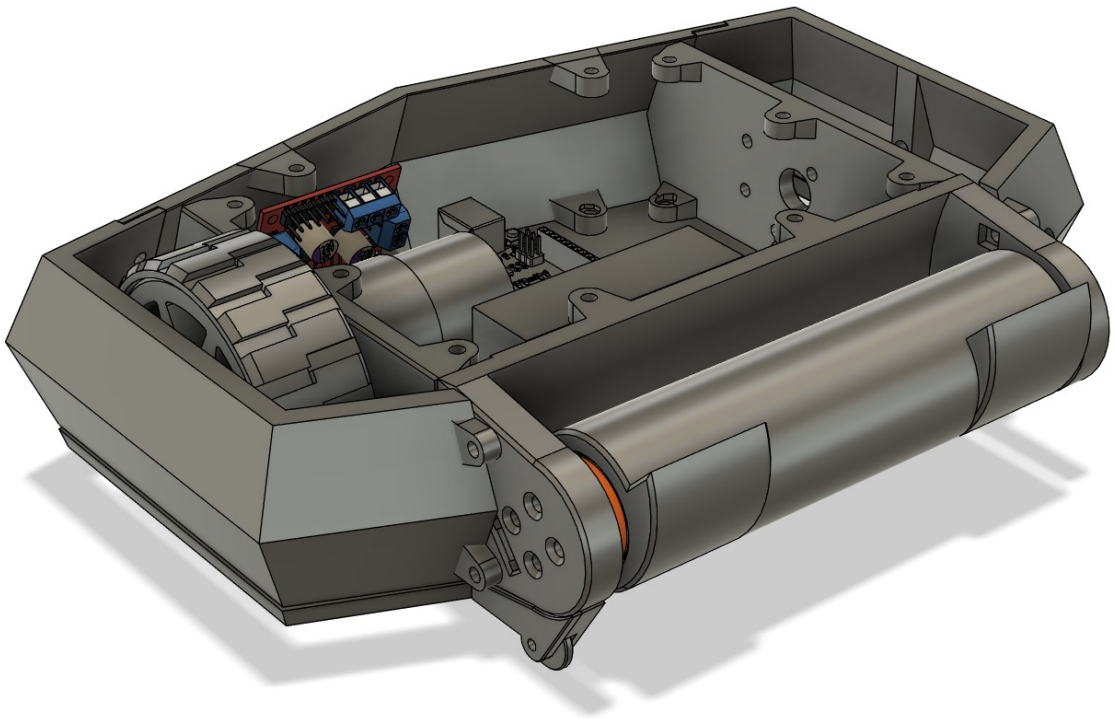



**Listă componente:**

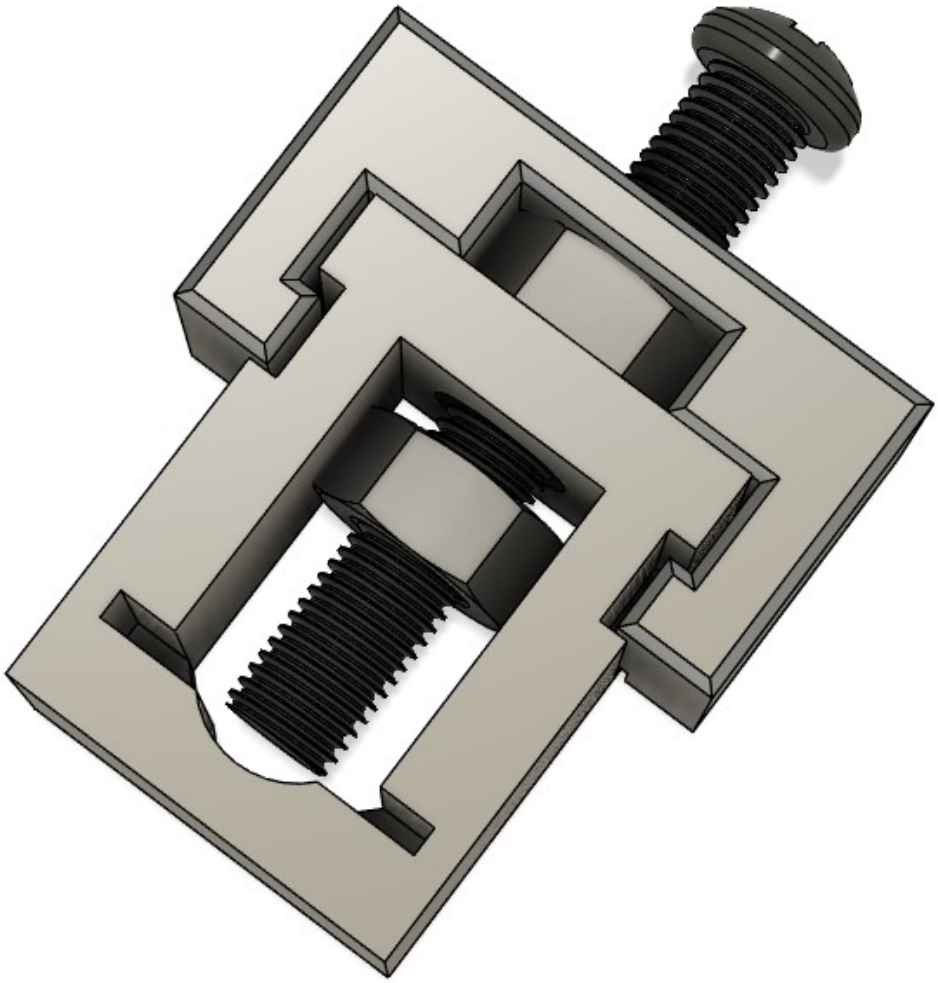
- Radiocomanda FlySky Fs-I6 2.4GHz
- Receiver radio FlySky Fs-IA6 2.4GHz
- Placă de dezvoltare Arduino Uno R3 ATmega328P
- Driver motoare DC L298N
- Motoare DC cu reductor JGB37

- Baterie LiPo Gens Ace Soaring 1300mAh 3S
- Speed Controller 30A BLDC
- Motor brushless A2212/6T 2200KV
- Servomotor SG90 9G
- Modul releu comandat 5V
- Modul transformator step-up 20kV
- Voltmetru OKY4092

Modelul 3D al robotului a fost conceput în jurul componentelor hardware. Am început cu o platformă cu găuri de montare pentru un Arduino Uno, pe care am extins-o pentru a adăuga bateria și motoarele pentru deplasarea robotului și pentru armă.

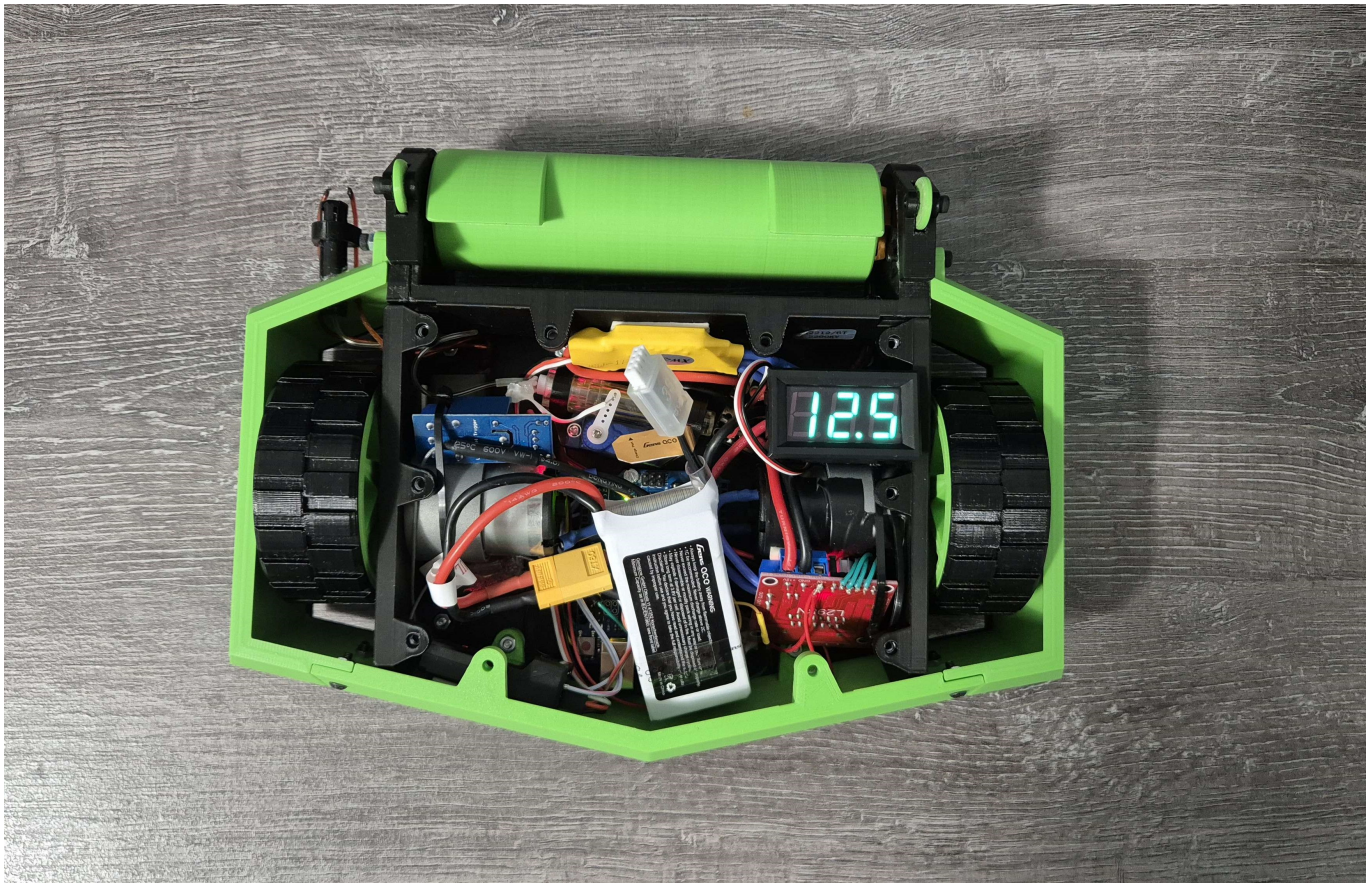


Pentru a porni robotul din exterior fără a-l dezasambla pentru a accesa bateria, am proiectat un întrerupător cu șurub: acesta asigură conexiunea circuitului de alimentare al robotului cu borna pozitivă a bateriei atunci când se înfiletează șurubul.



Întrerupătorul este accesibil din exterior și este plasat pe panoul central inferior al robotului. Săgeata de pe panoul din imaginea de mai jos semnifică direcția de rotire(spre dreapta) a șurubului pentru a porni robotul. Pentru a opri robotul, întrerupătorul va fi rotit spre stânga.

După asamblarea șasiului printat 3D și montarea componentelor electrice în acesta, robotul va arăta astfel:




## Software Design

După finalizarea etapei de proiectare de hardware, am programat inițial microcontrollerul folosind environmentul Arduino IDE pentru a testa funcționalitățile acestuia și a mă asigura că acesta funcționează corect. Codul inițial în Arduino pentru robot a fost:

```
#include <Servo.h>

#define RCPinFWD 2
#define RCPinSide 3
#define RCPinSwitch 4

#define Motor1Pin1 9
#define Motor1Pin2 8
#define Motor2Pin1 11
#define Motor2Pin2 10
#define RelayPin 13
#define ServoPin 12

volatile long StartTimeFWD = 0;
```

```
volatile long CurrentTimeFWD = 0;
volatile long PulsesFWD = 0;
int PulseWidthFWD = 0;

volatile long StartTimeSide = 0;
volatile long CurrentTimeSide = 0;
volatile long PulsesSide = 0;
int PulseWidthSide = 0;

int SwitchValue;

Servo servo;

void setup() {
  Serial.begin(9600);
  pinMode(RCPinFWD, INPUT_PULLUP);
  pinMode(RCPinSide, INPUT_PULLUP);
  pinMode(RCPinSwitch, INPUT);

  attachInterrupt(digitalPinToInterrupt(RCPinFWD),PulseTimerFWD,CHANGE);
  attachInterrupt(digitalPinToInterrupt(RCPinSide),PulseTimerSide,CHANGE);

  pinMode(Motor1Pin1, OUTPUT);
  pinMode(Motor1Pin2, OUTPUT);
  pinMode(Motor2Pin1, OUTPUT);
  pinMode(Motor2Pin2, OUTPUT);

  servo.attach(ServoPin);
  pinMode(RelayPin, OUTPUT);
}

void loop() {
  if (PulsesFWD < 2000){
    PulseWidthFWD = PulsesFWD;
  }
  if (PulsesSide < 2000){
    PulseWidthSide = PulsesSide;
  }
  Serial.print(PulseWidthFWD);
  Serial.print("    ");
  Serial.println(PulseWidthSide);

  if (PulseWidthFWD > 1000 && PulseWidthSide > 1000) {
    if (PulseWidthFWD > 1550) {
      digitalWrite(Motor1Pin1, HIGH);
      digitalWrite(Motor1Pin2, LOW);
    }
    else if (PulseWidthFWD < 1450) {
      digitalWrite(Motor1Pin1, LOW);
      digitalWrite(Motor1Pin2, HIGH);
    }
```

```
    else {
      digitalWrite(Motor1Pin1, LOW);
      digitalWrite(Motor1Pin2, LOW);
    }

    if (PulseWidthSide > 1550) {
      digitalWrite(Motor2Pin1, HIGH);
      digitalWrite(Motor2Pin2, LOW);
    }
    else if (PulseWidthSide < 1450) {
      digitalWrite(Motor2Pin1, LOW);
      digitalWrite(Motor2Pin2, HIGH);
    }
    else {
      digitalWrite(Motor2Pin1, LOW);
      digitalWrite(Motor2Pin2, LOW);
    }
  }
  else {
    digitalWrite(Motor2Pin1, LOW);
    digitalWrite(Motor2Pin1, LOW);
    digitalWrite(Motor1Pin1, LOW);
    digitalWrite(Motor1Pin1, LOW);
  }

  SwitchValue = pulseIn(RCPinSwitch, HIGH);
  Serial.print("___");
  Serial.print(SwitchValue);
  Serial.print("___");

  servo.write(SwitchValue);

  if (SwitchValue > 1000)
    digitalWrite(RelayPin, HIGH);
  else
    digitalWrite(RelayPin, LOW);
}

void PulseTimerFWD(){
  CurrentTimeFWD = micros();
  if (CurrentTimeFWD > StartTimeFWD){
    PulsesFWD = CurrentTimeFWD - StartTimeFWD;
    StartTimeFWD = CurrentTimeFWD;
  }
}
void PulseTimerSide(){
  CurrentTimeSide = micros();
  if (CurrentTimeSide > StartTimeSide){
    PulsesSide = CurrentTimeSide - StartTimeSide;
    StartTimeSide = CurrentTimeSide;
  }
```

```
}
```

Codul de Arduino a rămas referința de bază pentru design-ul codului in AVR C pentru microcontrollerul AtMega328P. Acesta este codul pentru microcontroller scris în AVR C:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define CLOCK_SPEED 16000000UL
#define BAUD 9600
#define MYUBRR CLOCK_SPEED/16/BAUD-1

#define RCPinFWD PD2
#define RCPinSide PD3
#define RCPinSwitch PD4

#define Motor1Pin1 PB1
#define Motor1Pin2 PB0
#define Motor2Pin1 PB3
#define Motor2Pin2 PB2
#define RelayPin PB5
#define ServoPin PB4

volatile long StartTimeFWD = 0;
volatile long CurrentTimeFWD = 0;
volatile long PulsesFWD = 0;
int PulseWidthFWD = 0;

volatile long StartTimeSide = 0;
volatile long CurrentTimeSide = 0;
volatile long PulsesSide = 0;
int PulseWidthSide = 0;

int SwitchValue;

void setup();
void loop();
void PulseTimerFWD();
void PulseTimerSide();
void init_timer1();
void init_timer0();
long pulseIn(uint8_t pin, uint8_t state);
long micros();

volatile unsigned long timer0_overflow_count = 0;

int main(void) {
    setup();
```

```
    while (1) {
        loop();
    }
}

void setup() {
    // Set input and output pins
    DDRD &= ~((1 << RCPinFWD) | (1 << RCPinSide) | (1 << RCPinSwitch)); //
Set as input
    PORTD |= (1 << RCPinFWD) | (1 << RCPinSide); // Enable pull-up resistors

    DDRB |= (1 << Motor1Pin1) | (1 << Motor1Pin2) | (1 << Motor2Pin1) | (1
<< Motor2Pin2) | (1 << RelayPin) | (1 << ServoPin);

    // Set up external interrupts for FWD and Side
    EICRA |= (1 << ISC00) | (1 << ISC10); // Any logical change on INT0 and
INT1 generates an interrupt
    EIMSK |= (1 << INT0) | (1 << INT1); // Enable INT0 and INT1

    init_timer1();
    init_timer0();

    sei(); // Enable global interrupts
}

void loop() {
    if (PulsesFWD < 2000) {
        PulseWidthFWD = PulsesFWD;
    }
    if (PulsesSide < 2000) {
        PulseWidthSide = PulsesSide;
    }

    if (PulseWidthFWD > 1000 && PulseWidthSide > 1000) {
        if (PulseWidthFWD > 1550) {
            PORTB |= (1 << Motor1Pin1);
            PORTB &= ~(1 << Motor1Pin2);
        } else if (PulseWidthFWD < 1450) {
            PORTB &= ~(1 << Motor1Pin1);
            PORTB |= (1 << Motor1Pin2);
        } else {
            PORTB &= ~((1 << Motor1Pin1) | (1 << Motor1Pin2));
        }

        if (PulseWidthSide > 1550) {
            PORTB |= (1 << Motor2Pin1);
            PORTB &= ~(1 << Motor2Pin2);
        } else if (PulseWidthSide < 1450) {
            PORTB &= ~(1 << Motor2Pin1);
            PORTB |= (1 << Motor2Pin2);
        } else {
```

```
            PORTB &= ~((1 << Motor2Pin1) | (1 << Motor2Pin2));
        }
    } else {
        PORTB &= ~((1 << Motor1Pin1) | (1 << Motor1Pin2) | (1 << Motor2Pin1)
| (1 << Motor2Pin2));
    }

    SwitchValue = pulseIn(RCPinSwitch, HIGH);

    OCR1A = SwitchValue; // Set PWM for servo

    if (SwitchValue > 1000) {
        PORTB |= (1 << RelayPin);
    } else {
        PORTB &= ~(1 << RelayPin);
    }
}

ISR(INT0_vect) {
    PulseTimerFWD();
}

ISR(INT1_vect) {
    PulseTimerSide();
}

void PulseTimerFWD() {
    CurrentTimeFWD = micros();
    if (CurrentTimeFWD > StartTimeFWD) {
        PulsesFWD = CurrentTimeFWD - StartTimeFWD;
        StartTimeFWD = CurrentTimeFWD;
    }
}

void PulseTimerSide() {
    CurrentTimeSide = micros();
    if (CurrentTimeSide > StartTimeSide) {
        PulsesSide = CurrentTimeSide - StartTimeSide;
        StartTimeSide = CurrentTimeSide;
    }
}

void init_timer1() {
    // Timer1 setup for servo control
    TCCR1A = (1 << WGM11) | (1 << COM1A1);
    TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS11);
    ICR1 = 39999; // 50Hz frequency
}

void init_timer0() {
    // Timer0 setup for micros
```

```
    TCCR0A = 0;
    TCCR0B = (1 << CS01) | (1 << CS00); // Prescaler 64
    TIMSK0 = (1 << TOIE0); // Enable overflow interrupt
}

ISR(TIMER0_OVF_vect) {
    timer0_overflow_count++;
}

long micros() {
    unsigned long m;
    uint8_t oldSREG = SREG;

    cli();
    m = timer0_overflow_count;
    uint8_t t = TCNT0;

    if ((TIFR0 & (1 << TOV0)) && (t < 255)) {
        m++;
    }

    SREG = oldSREG;
    return ((m << 8) + t) * (64 / (F_CPU / 1000000L));
}

long pulseIn(uint8_t pin, uint8_t state) {
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);
    uint8_t stateMask = (state ? bit : 0);
    unsigned long width = 0;

    // Wait for any previous pulse to end
    while ((*portInputRegister(port) & bit) == stateMask);

    // Wait for the pulse to start
    while ((*portInputRegister(port) & bit) != stateMask);

    // Wait for the pulse to stop
    while ((*portInputRegister(port) & bit) == stateMask) {
        width++;
        if (width >= 1000000) {
            return 0;
        }
    }

    return width;
}
```

În codul de AVR, folosim hardware interrupts pe pinii de input pentru a identifica semnalele PWM de la receiverul radio, iar cu timerul 0 le măsurăm valoarea. Vom folosi apoi acele valori pentru a activa driverul de motoare DC pentru direcție, împreună cu servomotorul si controllerul de viteză pentru

motorul brushless. Pentru a trimite un semnal PWM corespunzător valorii primite de la receiver la speed controller și servomotor, vom folosi timerul 1.

# Concluzii

Acest proiect m-a ajutat să învăț despre programarea microprocesoarelor Atmel in AVR C, fără a folosi abstraction layer-ul oferit de Arduino IDE. De asemenea mi-a consolidat aptitudinile de CAD și de design de circuite.

Abilitatea de a citi semnalele dintr-un receiver hobby-grade cu ajutorul unui microprocesor deschide o plajă largă de potențiale proiecte viitoare.

# Download

Link Repository Git: BattleBot

Arhiva care conține următoarele fisiere:

- 3D - un fisier cu toate piesele 3D folosite în proiectarea robotului.
- Battlebot - implementarea proiectului în Arduino
- media - conținut media cu robotul
- Schematic photos - figurile schematicelor robotului
- src - implementarea proiectului în AVR C

arhiva

# Bibliografie/Resurse

### Resurse Software

- Laboratorul 0: GPIO
- Laboratorul 2: Întreruperi, Timere
- Laboratorul 3: Timere, PWM

### Resurse Hardware

- HowToMechatronics
- ElectroNoobs

Export to PDF