

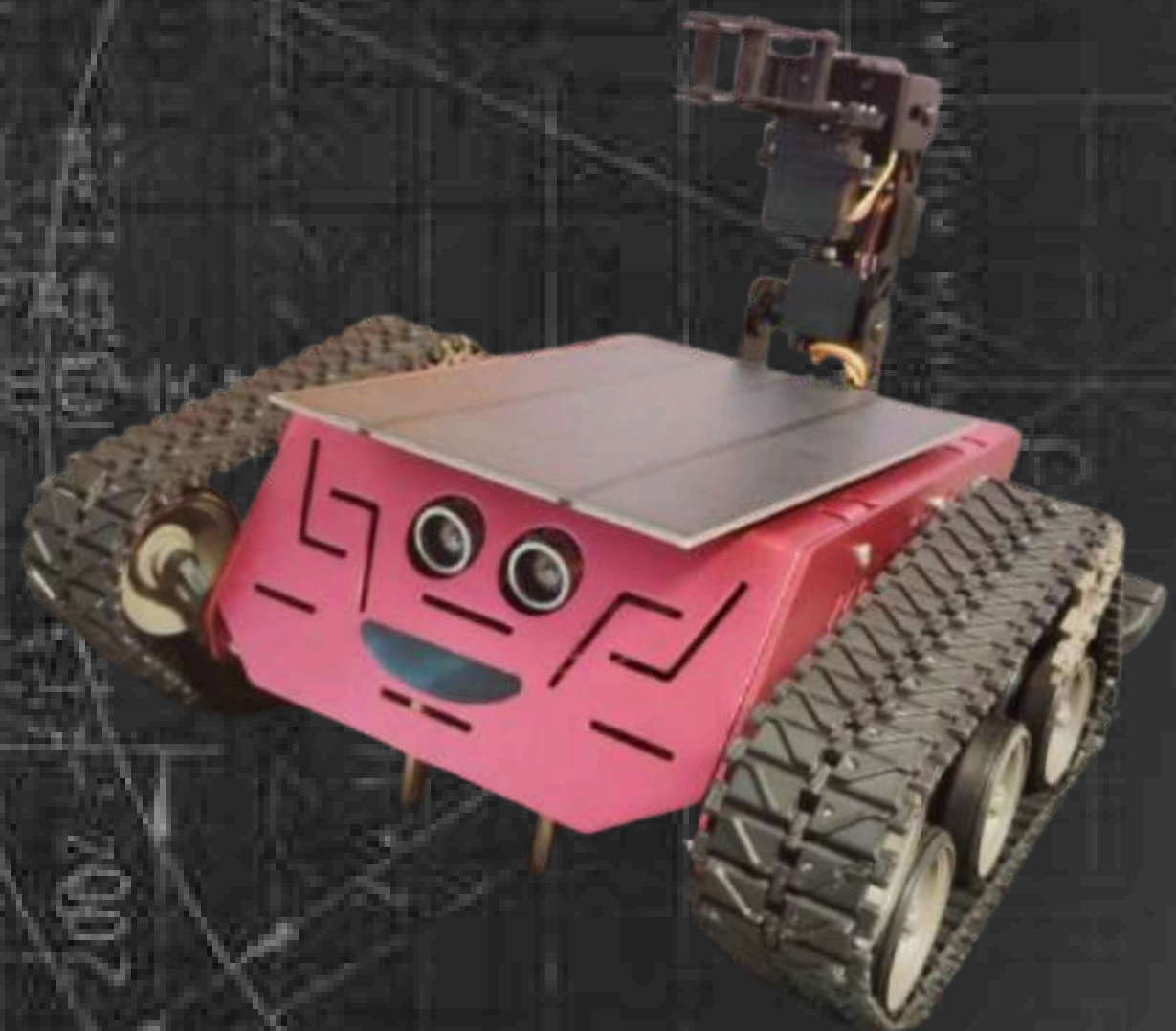
TRACKY

ROBOT CURIER ȘI EXPLORATOR

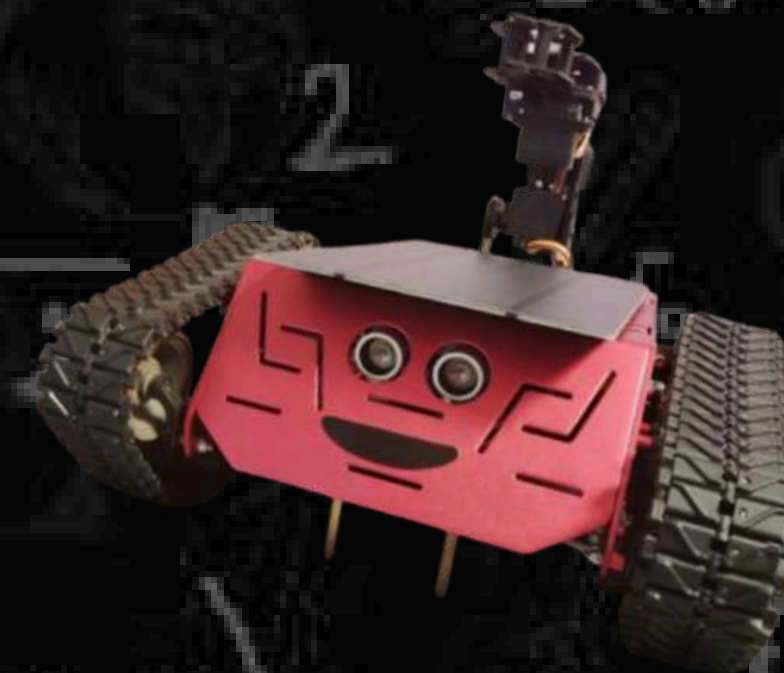
NUME : POPESCU ANDREI GEORGE CONSTANTIN

CLASA : X-A

LOCALITATE : PETROȘANI , HUNEDOARA



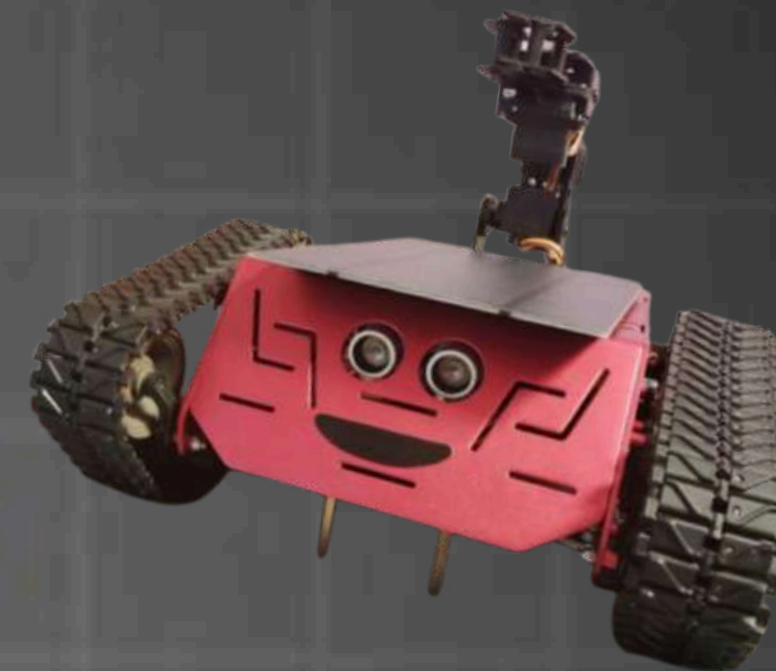
UTILITATE PRACTICĂ



Definirea problemei

În multe situații, transportul de obiecte mici și explorarea în medii variate necesită soluții autonome, eficiente energetic și adaptabile. Tracky este un robot autonom creat pentru a rezolva problema transportului obiectelor mici și a explorării în spații interioare și exterioare, într-un mod eficient, sigur și sustenabil. În plus, el adaugă o componentă inovatoare de divertisment și educație, transformând munca robotică într-o experiență interactivă.

SOLUȚIA PROPUȘĂ



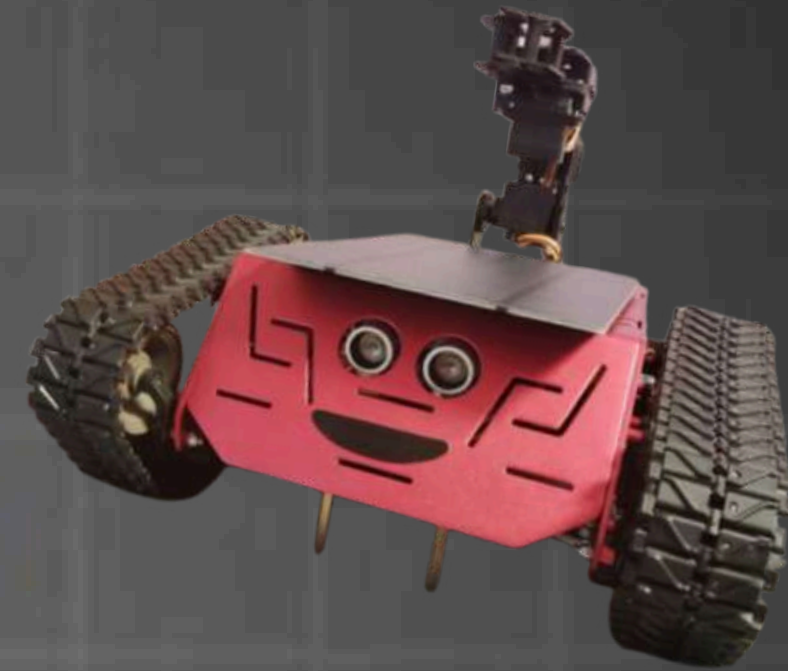
PROTOTIPUL EXPERIMENTAL TRACKY

TRACKY ESTE UN ROBOT AUTONOM CARE COMBINĂ MAI MULTE METODE DE CONTROL ȘI OPTIMIZARE ENERGETICĂ:

- MIȘCARE AUTONOMĂ ÎN FUNCȚIE DE SURSA DE LUMINĂ, UTILIZÂND UN PANOU SOLAR PENTRU A-ȘI EXTINDE AUTONOMIA.
- EVITAREA OBSTACOLELOR PRINTR-UN SENZOR ULTRASONIC, ASIGURÂND DEPLASAREA FĂRĂ COLIZIUNI.
- CONTROL MANUAL PRIN TELECOMANDĂ IR SAU PRIN INTERMEDIUL UNUI CONTROLLER DE TIP PLAYSTATION2, OFERIND UTILIZATORULUI POSIBILITATEA DE A DIREȚIONA ROBOTUL ÎN FUNCȚIE DE NEVOI SPECIFICE.
- INCLUDE UN BRAȚ ROBOTIC – PENTRU MANIPULAREA OBIECTELOR MICI.
- OBȚINEREA DE IMAGINI ÎN TIMP REAL , CÂT SI SALVATE PE UN CARD SD.
- POATE FI UTILIZAT ȘI ÎN ACTIVITĂȚI RECREATIVE ȘI EDUCATIVE – PENTRU ELEVI, STUDENȚI SAU COPII PASIONAȚI DE TEHNOLOGIE.



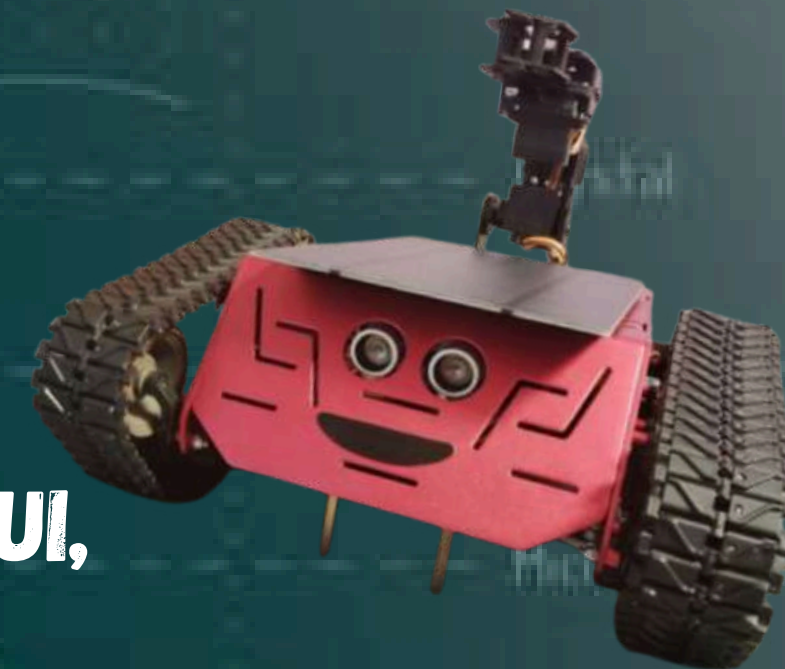
ÎN CE MĂSURĂ PROBLEMA ESTE REZOLVATĂ MAI EFICIENT UTILIZÂND ROBOTUL DECÂT PRIN ALTE METODE ?



TRACKY OFERĂ AVANTAJE CLARE FAȚĂ DE METODELE CLASICE:

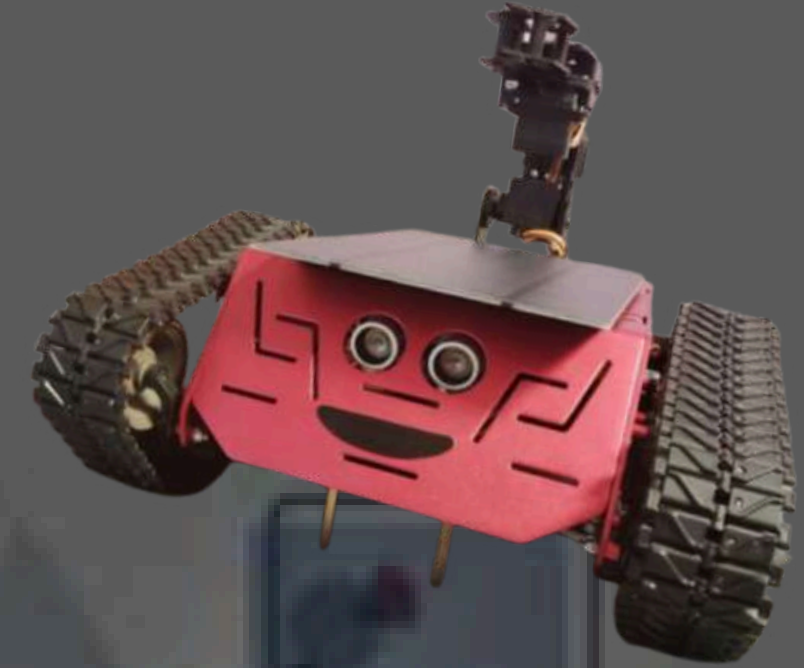
<u>Aspect</u>	<u>Metodă clasică</u>	<u>Tracky</u>
Efort fizic	Necesită muncă umană	Complet automatizat
Navigare în spații mici	Limitată	Ușor de adaptat și manevrat
Autonomie energetică	Nu există	Panou solar integrat
Colectare de date	Nu este posibilă	Salvare de imagini
Utilizare educativă	Nu poate fi folosită în scopuri didactice	Ideal pentru ateliere STEM, școli
Divertisment	Inexistent	Poate fi programat pentru jocuri de labirint, urmărire lumini, competiții între

ELEMENTE TEHNICE INCLUSE:



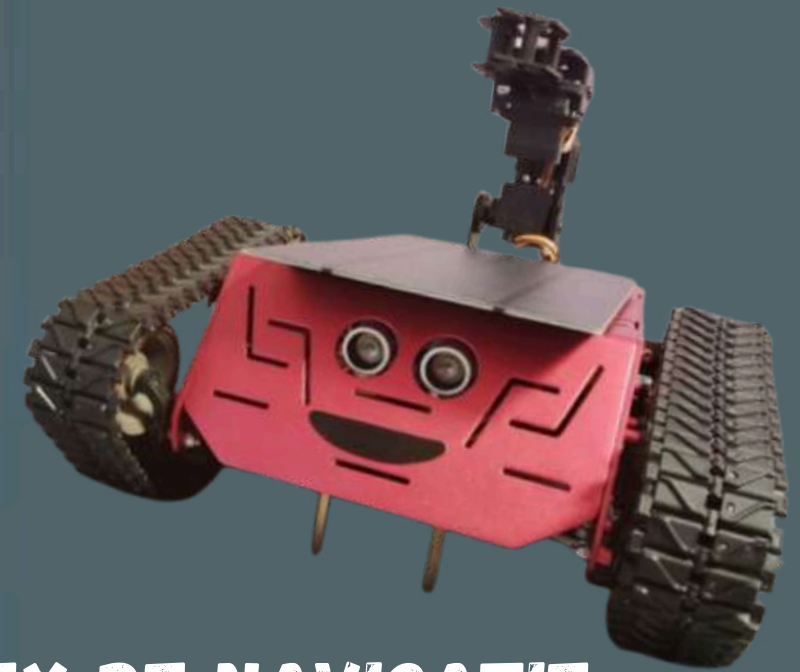
- SCHEME ELECTRICE ȘI ELECTRONICE PENTRU GESTIONAREA MOTORULUI, SENZORILOR
- ALGORITMI CARE CONTROLEAZĂ DIRECȚIA DE DEPLASARE PE BAZA INTENSITĂȚII LUMINII.
- SISTEM DE EVITARE A OBSTACOLELOR FOLOSIND DATELE SENZORULUI ULTRASONIC.
- SISTEME DE OBTINERE A IMAGINI PE UN CARD SD.
- PROTOCOL DE COMUNICARE PRIN INFRAROȘU PENTRU CONTROL MANUAL CU TELECOMANDA IR & PS2 (SONY PS2 WIRELESS CONTROLLER)
- DESENE DE EXECUȚIE PENTRU STRUCTURA ELECTRICĂ ȘI INTEGRAREA COMPONENTELOR ELECTRONICE.

REZULTATE OBȚINUTE



- **TESTELE REALIZATE CU PROTOTIPUL EXPERIMENTAL AU DEMONSTRAT CĂ TRACKY POATE:**
- **DETECTA ȘI URMĂRI SURSELE DE LUMINĂ PENTRU A OPTIMIZA COLECTAREA ENERGIEI SOLARE.**
- **EVITA OBSTACOLELE UTILIZÂND SENZORUL ULTRASONIC PENTRU O NAVIGAȚIE SIGURĂ.**
- **FUNCȚIONA AUTONOM SAU MANUAL, OFERIND FLEXIBILITATE UTILIZATORULUI.**
- **TRANSPORTA OBIECTE MICI ÎNTR-UN MOD EFICIENT.**
- **GENERAREA DE IMAGINI ÎN DIFERITE MEDII CU AJUTORUL CAMERELOR.**

REZULTATE OBȚINUTE



CONCLUZII ȘI DIRECȚII DE DEZVOLTARE

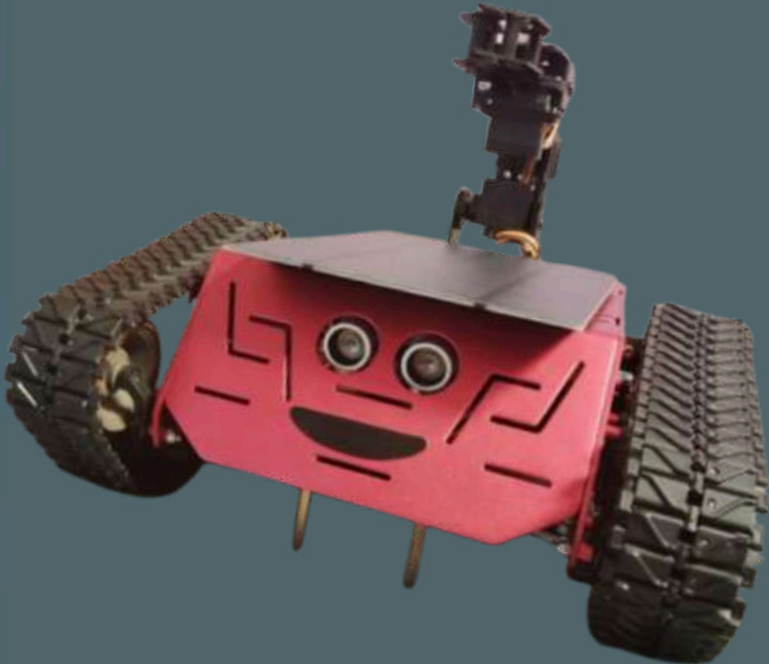
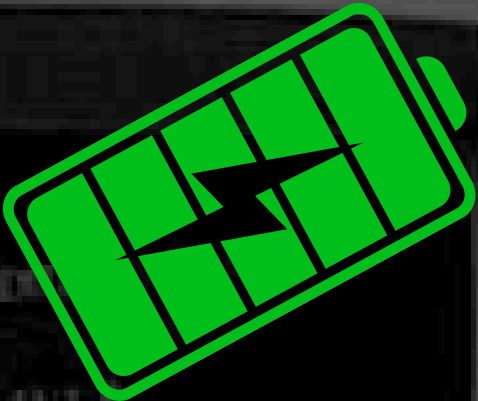
TRACKY ADUCE O CONTRIBUȚIE INOVATIVĂ PRIN INTEGRAREA UNUI SISTEM COMPLEX DE NAVIGAȚIE AUTONOMĂ BAZAT PE LUMINA AMBIENTALĂ ȘI EVITAREA OBSTACOLELOR, COMPLETAT DE UN MOD DE CONTROL MANUAL PRIN TELECOMANDA IR SI PS2.

PENTRU DEZVOLTAREA VIITOARE, SE POT EXPLORA:

- **ÎMBUNĂTĂȚIREA ALGORITMILOR AI PENTRU O NAVIGAȚIE MAI EFICIENTĂ ȘI ADAPTIVĂ.**
- **EXTINDEREA CAPACITĂȚII PANOULUI SOLAR PENTRU AUTONOMIE MAI MARE.**
- **ADĂUGAREA UNUI BRAȚ MAI PUTERNIC PENTRU MANIPULAREA OBIECTELOR TRANSPORTATE.**
- **OPTIMIZAREA INTERFEȚEI DE CONTROL PENTRU O EXPERIENȚĂ MAI INTUITIVĂ CU TELECOMANDA IR SAU ALTE MODALITĂȚII DE CONTROL .**

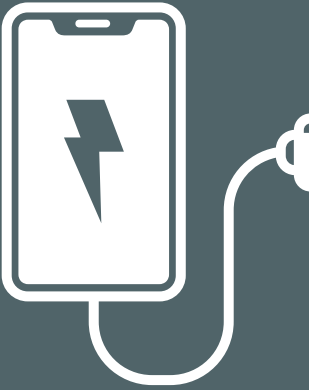
TRACKY DEMONSTREAZĂ CUM TEHNOLOGIA POATE COMBINA EXPLORAREA AUTONOMĂ, EFICIENȚA ENERGETICĂ ȘI CONTROLUL FLEXIBIL, DESCHIZÂND NOI POSIBILITĂȚI ÎN DOMENIUL ROBOTICII MOBILE.

REZULTATE OBȚINUTE



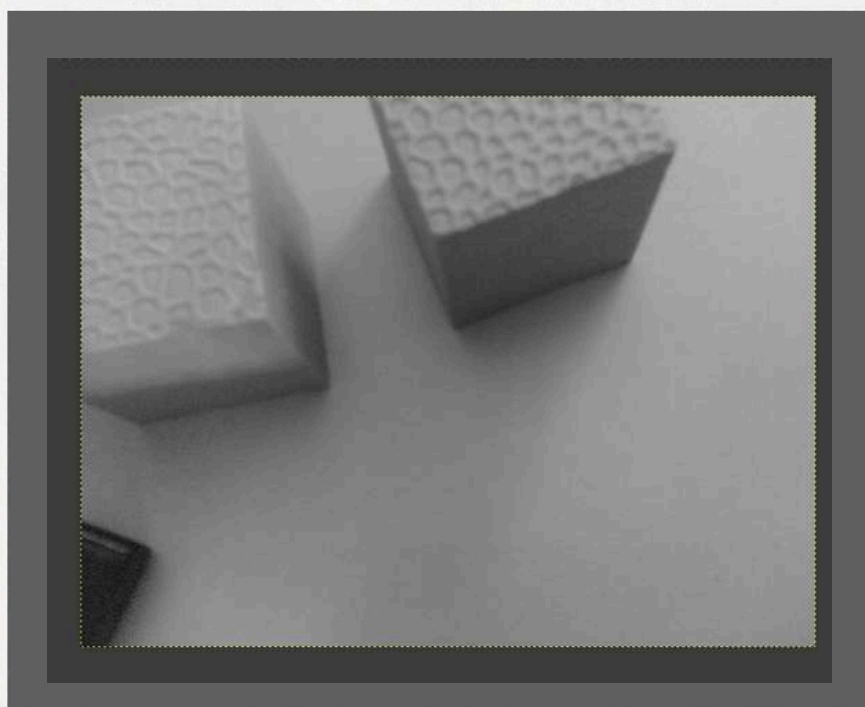
<u>Componente</u>	<u>Activitate</u>	<u>Consum estimat</u>	<u>Durată estimată</u>
2x ESP32-CAM (2x 18650)	Poză la 3 secunde → pe SD	~1.5W total	~12.5 ore
ROBOTUL (2x 18650)	Standby (Arduino + senzori)	~0.4W	~47 ore
ROBOTUL (2x 18650)	Activ continuu (motoare, senzori,receiver,servo-uri)	~5W	3.5-4 ore











- Timp estimat încărcare pentru 2 acumulatori 18650 (3000 mAh fiecare):
- PORT DE ÎNCĂRCARE: MICRO USB (5V / 2A)
 - TIMP ESTIMAT: 3 – 4 ORE
 - PORT DE ÎNCĂRCARE: USB-C CU POWER DELIVERY (9V / 2A)
 - TIMP ESTIMAT: 2 – 3 ORE
 - PANOUL SOLAR 2.6W-3.5W
 - TIMP ESTIMAT: 2-6 ORE

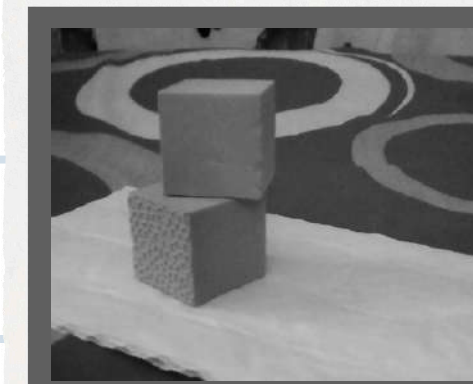
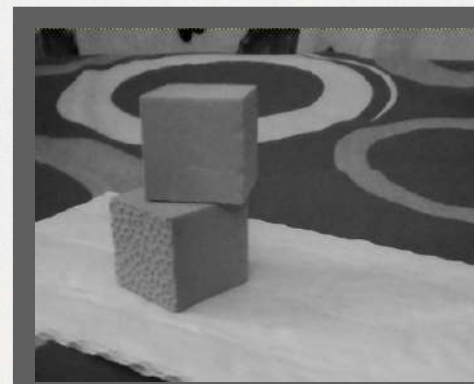


Notă: Valorile de autonomie sunt estimative și se bazează pe acumulatori 18650 complet încărcăți (3000 mAh fiecare). Durata reală poate varia în funcție de nivelul de încărcare, uzura bateriilor și condițiile de utilizare.

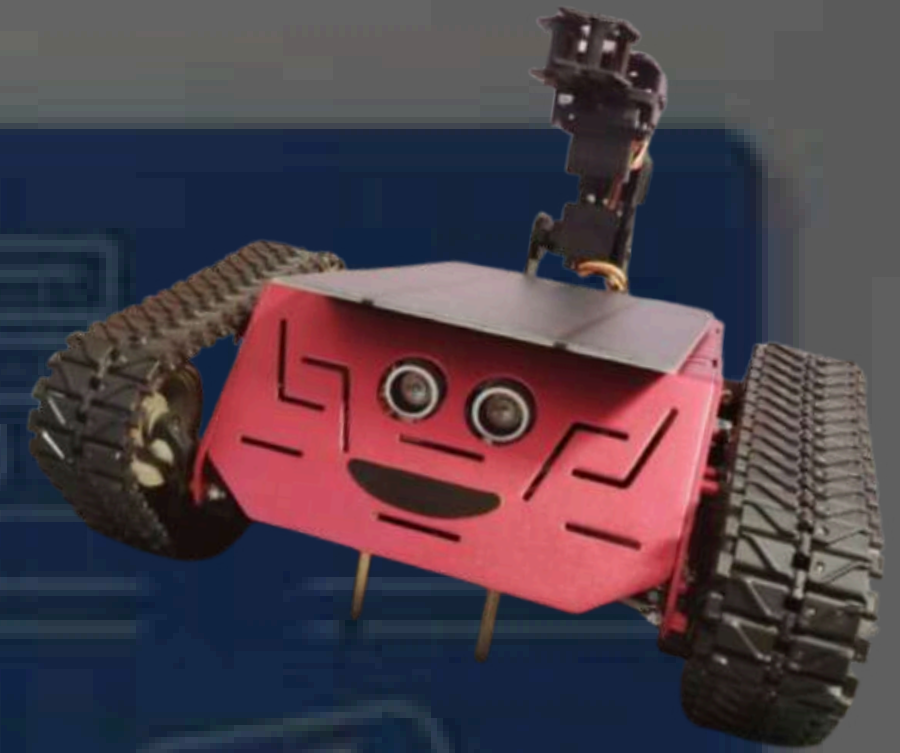
Rezultate ESP32-CAM-uri



Name	Date modified	Type	Size
 foto_0.pgm		GIMP 3.0.2-1 PGM	1,876 KB
 foto_1.pgm		GIMP 3.0.2-1 PGM	1,876 KB
 foto_2.pgm		GIMP 3.0.2-1 PGM	1,876 KB
 foto_3.pgm		GIMP 3.0.2-1 PGM	1,876 KB
 foto_4.pgm		GIMP 3.0.2-1 PGM	1,876 KB
 foto_5.pgm		GIMP 3.0.2-1 PGM	1,876 KB
 foto_6.pgm		GIMP 3.0.2-1 PGM	1,876 KB
 foto_7.pgm		GIMP 3.0.2-1 PGM	1,876 KB
 foto_8.pgm		GIMP 3.0.2-1 PGM	1,876 KB
 foto_9.pgm		GIMP 3.0.2-1 PGM	1,876 KB



DETALII PRIVIND PROCESUL DE DEZVOLTARE



A) CONCEPEREA IDEII:

IDENTIFICAREA NEVOII PENTRU UN ROBOT AUTONOM CAPABIL SĂ TRANSPORTE OBIECTE MICI ȘI SĂ EXPLOREZE AUTONOM.

STABILIREA CERINTELOR: AUTONOMIE ENERGETICĂ EXTINSĂ, EVITAREA OBSTACOLELOR, MOD E URMARIRE SOLARĂ, CONTROL DUAL (MANUAL ȘI AUTOMAT).

B) ALEGEREA COMPONENTELOR:

AM ANALIZAT PERFORMANȚA COMPONENTELOR DISPONIBILE PE PIAȚĂ ȘI AM SELECTAT SOLUȚIILE OPTIME LUÂND ÎN CONSIDERARE:

FIABILITATE (EX: ARDUINO UNO R3 PENTRU STABILITATEA PLATFORMEI).

EFICIENȚĂ ENERGETICĂ (EX: UTILIZAREA UNUI PANOU SOLAR PENTRU PRELUNGIREA AUTONOMIEI).

COSTURI ȘI DISPONIBILITATE (COMPONENTE ACCESIBILE, CU LIVRARE RAPIDĂ).

C) PROTOTIPAREA:

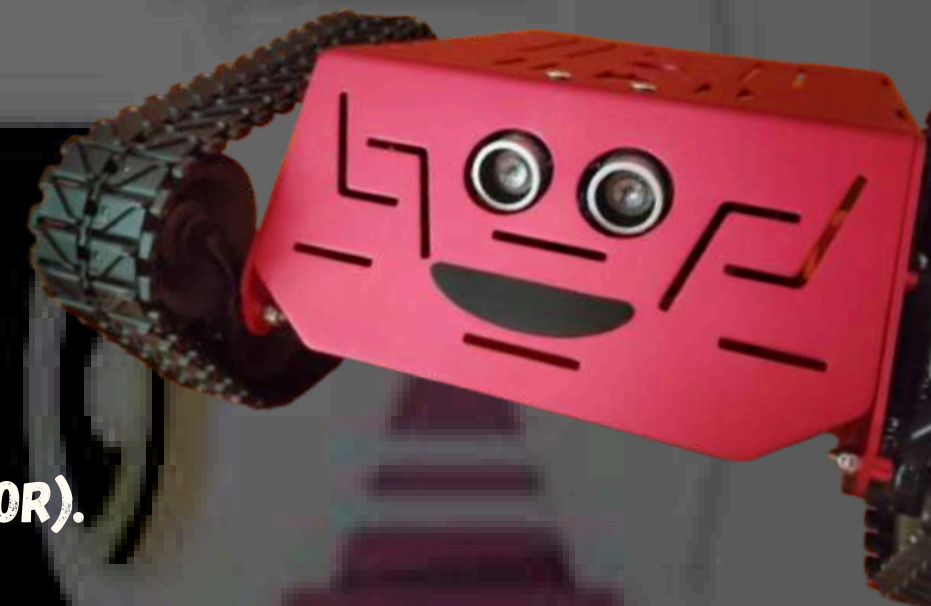
PRIMA VERSIUNE A FOST BAZATĂ PE UN SISTEM SIMPLU DE EVITARE A OBSTACOLELOR.

AM INTEGRAT TREPTAT MODULELE SUPLIMENTARE: RECEPTOR IR, MODULELE ESP32-CAM, CONTROLLER PS2, PANOU SOLAR, BRAT ROBOTIC.

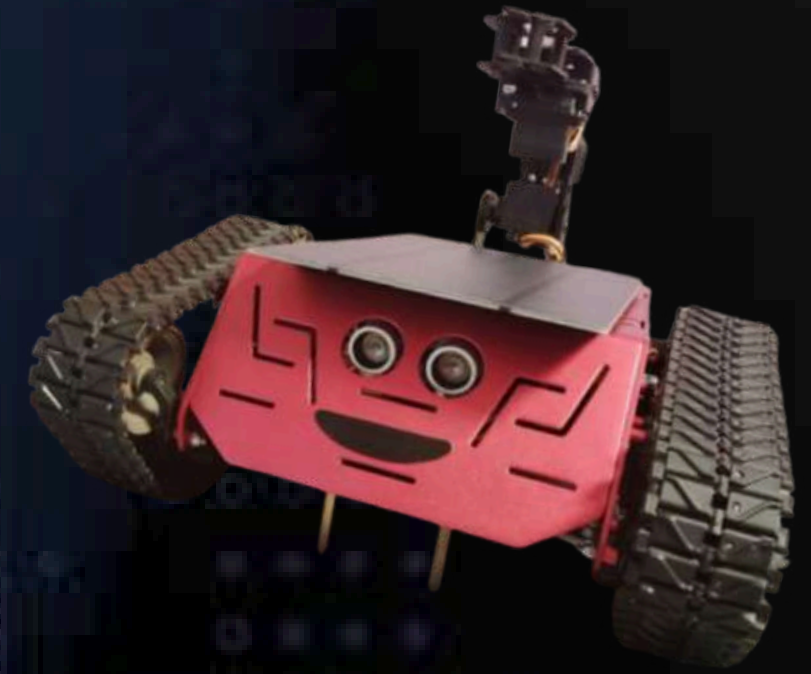
D) TESTARE ȘI OPTIMIZARE:

TESTE DE TEREN PENTRU VERIFICAREA NAVIGĂRII AUTONOME ȘI FIABILITĂȚII ÎN DIVERSE CONDIȚII (INTERIOR/EXTERIOR).

OPTIMIZAREA ALGORITMULUI DE EVITARE A OBSTACOLELOR ȘI A CONTROLULUI MANUAL.



IMPLICAREA PERSONALĂ



- PROIECTUL TRACKY ESTE REZULTATUL UNEI PASIUNI PERSONALE PENTRU ROBOTICĂ ȘI INOVAȚIE ÎN DOMENIUL SISTEMELOR AUTONOME.
- DE LA IDEE PÂNĂ LA PROTOTIPUL FUNCȚIONAL, AM CONCEPUT FIECARE ETAPĂ A PROIECTULUI, FOLOSIND UN BUGET PERSONAL.
- ÎN ALEGEREA COMPONENTELOR, AM ȚINUT CONT DE RAPORTUL PERFORMANȚĂ-PREȚ ȘI DE COSTURILE DE TRANSPORT PENTRU A OPTIMIZA RESURSELE DISPONIBILE.
- ACEASTĂ ABORDARE PRAGMATICĂ MI-A PERMIS SĂ REALIZEZ UN ROBOT EFICIENT ENERGETIC, CU FUNCȚII AVANSATE DE NAVIGARE ȘI CONTROL MANUAL.

MECANICA

COMPLEXITATE

MOTOARE: 2 MOTOARE DC (1 AX FORMA L) PENTRU DEPLASARE PE ȘENILE,

GRADE DE LIBERTATE: SE POATE DEPLASA ÎNAINTE, ÎNAPOI, STÂNGA, DREAPTA.

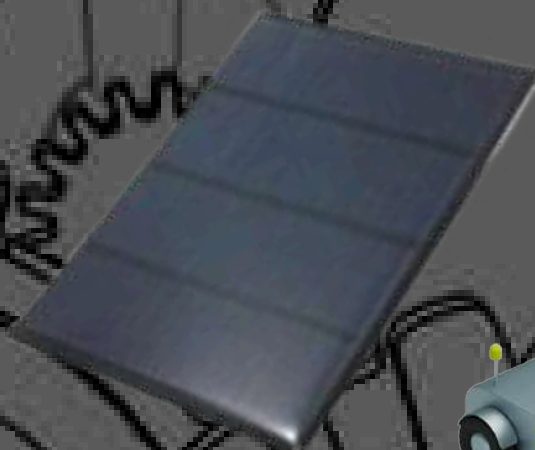
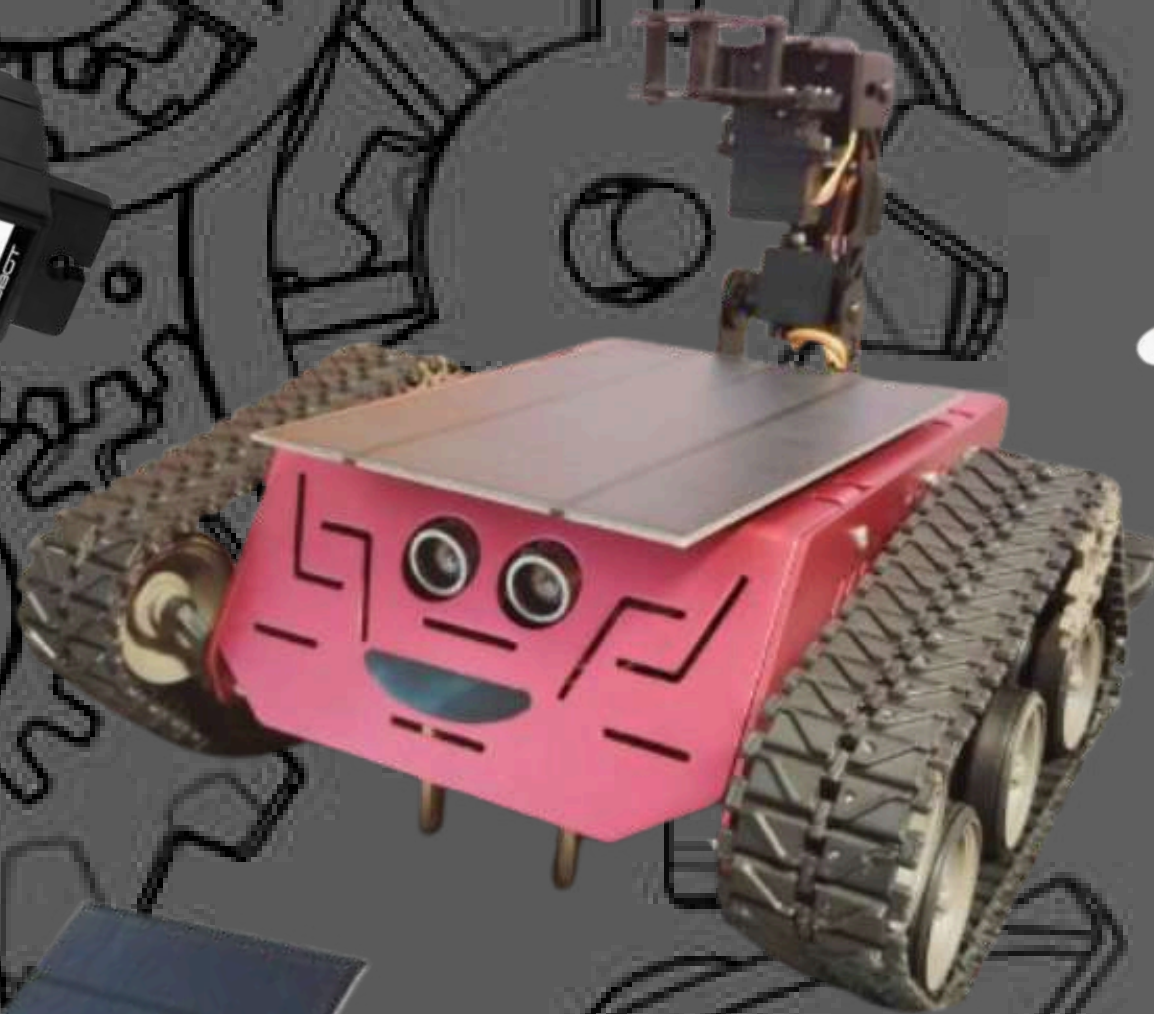
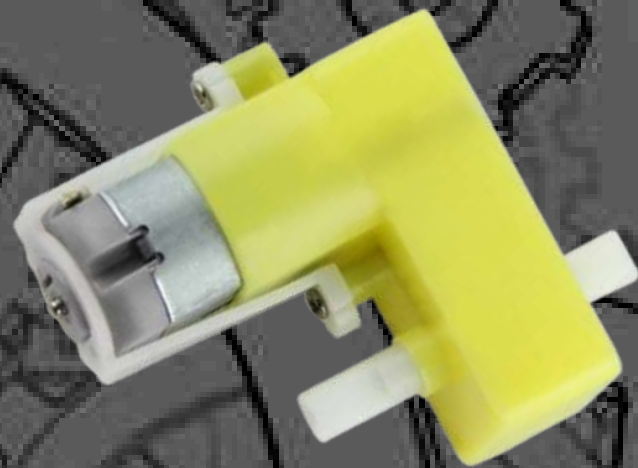
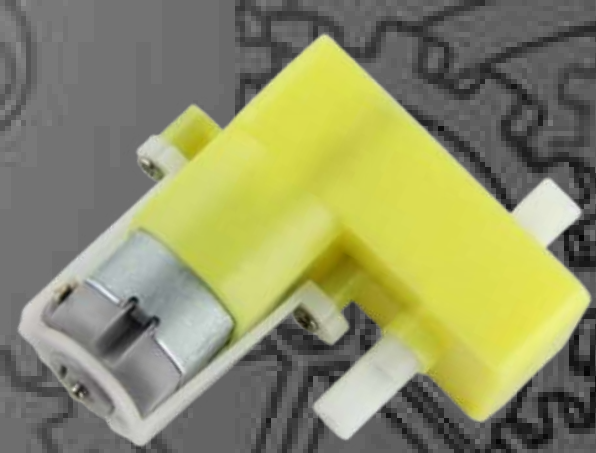
3 SERVO (LFD-01M) PENTRU BRAȚUL ROBOTIC.


BRAȚUL ROBOTIC POATE RIDICA/ÎMPINGE OBIECTE. (0-180°)

EFICIENȚĂ ÎN CONSTRUCȚIE

CONSUM MINIM DE ENERGIE: MOTOARELE SUNT GESTIONATE EFICIENT, IAR ALGORITMI DE NAVIGAȚIE EVITĂ MIȘCĂRILE INUTILE.

SURSĂ REGENERABILĂ: ARE UN PANOU SOLAR MIC PENTRU EXTINDEREA AUTONOMIEI.





ELECTRONICA

SENZORI FOLOSITI:

Ultrasonic (HC-SR04): Detectează obstacolele, folosit pentru navigație autonomă.

2X Module senzor cu fotorezistor LDR, pentru detectarea sursei de lumină

Receptor IR: Primește comenzi de la telecomandă IR pentru control manual.

ARHITECTURĂ ELECTRONICĂ:

Microcontroller: Arduino UNO R3 (ATmega328P) – gestionează motoarele și senzorii.

Doua plăci de dezvoltare ESP32-CAM, Wifi, Bluetooth, cu camere de tip RHYX-M21-45 2M & OV2640 2M -pentru generarea de fotografii

Driver motoare: Motor Driver Shield .

AMS1117 5V->3.3V : PENTRU A FOLOSI RECEIVER-UL de PS2

Alimentare: 2x 18650 Li-Ion.

Modul Buzzer Pasiv - folosit pentru a emite sunete.

PS2 Wireless Controller Receiver 3.3VDC

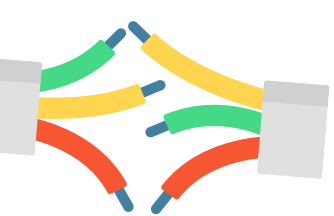
Modul powerbank 18650(2P, 5V/3VDC, Micro USB si Tip C

Modul masurare curent/tensiune pentru dispozitive alimentate prin USB

Complexitate: Tracky poate fi autonom PRIN INTERMEDIUL SENZORULUI ULTRASONIC si LDR, dar și telecomandat, permițând schimbarea modurilor.

Tabel : Intervalele senzorilor și reacțiile robotului

Senzor	Tip / Funcție	Interval de măsurare	Condiții logice în cod	Comportamentul robotului
<div>  Ultrasonic </div>	<div> Distanță față de obstacol  </div>	<div> Aproximativ 2 – 400 cm </div>	Front_Distance <= 10	Se deplasează înapoi , apoi se rotește aleator (stânga/dreapta), buzzer 1000 Hz
			10 < Front_Distance <= 20	Se oprește , apoi se rotește aleator , buzzer 800 Hz
			20 < Front_Distance <= 30	Se deplasează înainte (viteză 150), buzzer 600 Hz
			Front_Distance > 30	Se oprește
<div>  LDR (fotorezistor) </div>	<div> Detectare diferență de lumină   </div>	<div> 0 – 1023 (ADC 10-bit) </div>	ldrLeft > 800	Prag minim de lumină
			abs(ldrLeft - ldrRight) < 50	Diferență mică de lumină → STOP
			ldrLeft - ldrRight > 50	Lumină mai puternică în stânga → rotește stânga
			ldrRight - ldrLeft > 50 (sau < -50)	Lumină mai puternică în dreapta → rotește dreapta
			ldrLeft ≤ 800 && ldrRight ≤ 800	Întuneric → STOP



CIRCUIT

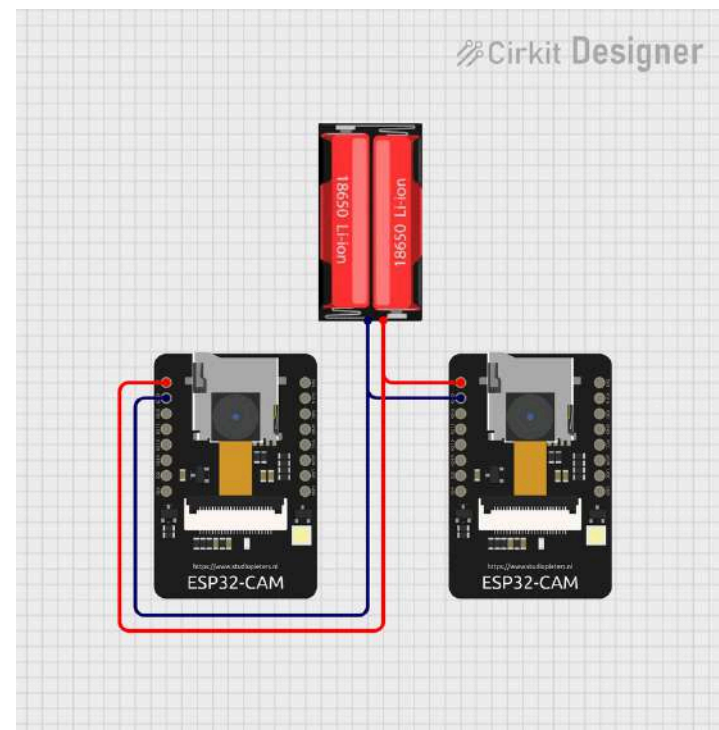
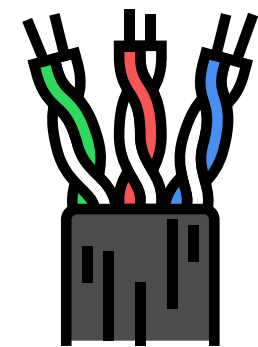
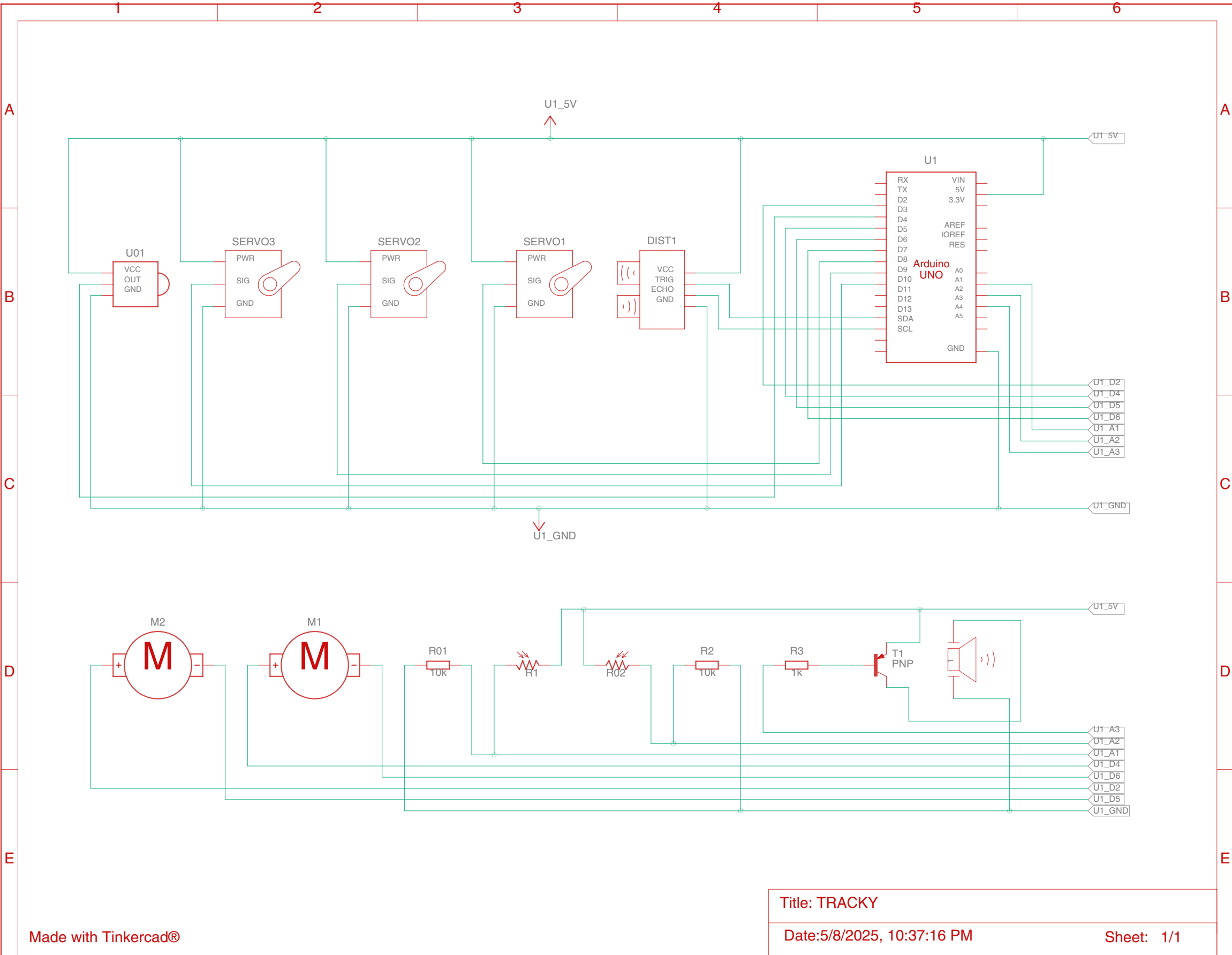
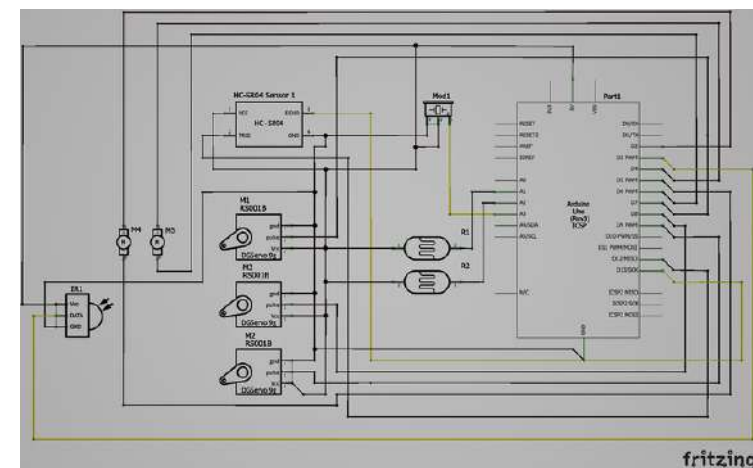
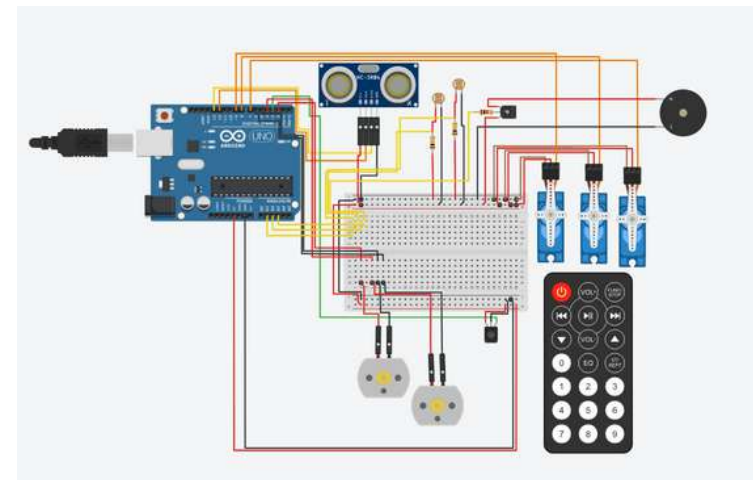


RECEIVER-UL PS2 ESTE CONECTAT ASTFEL:

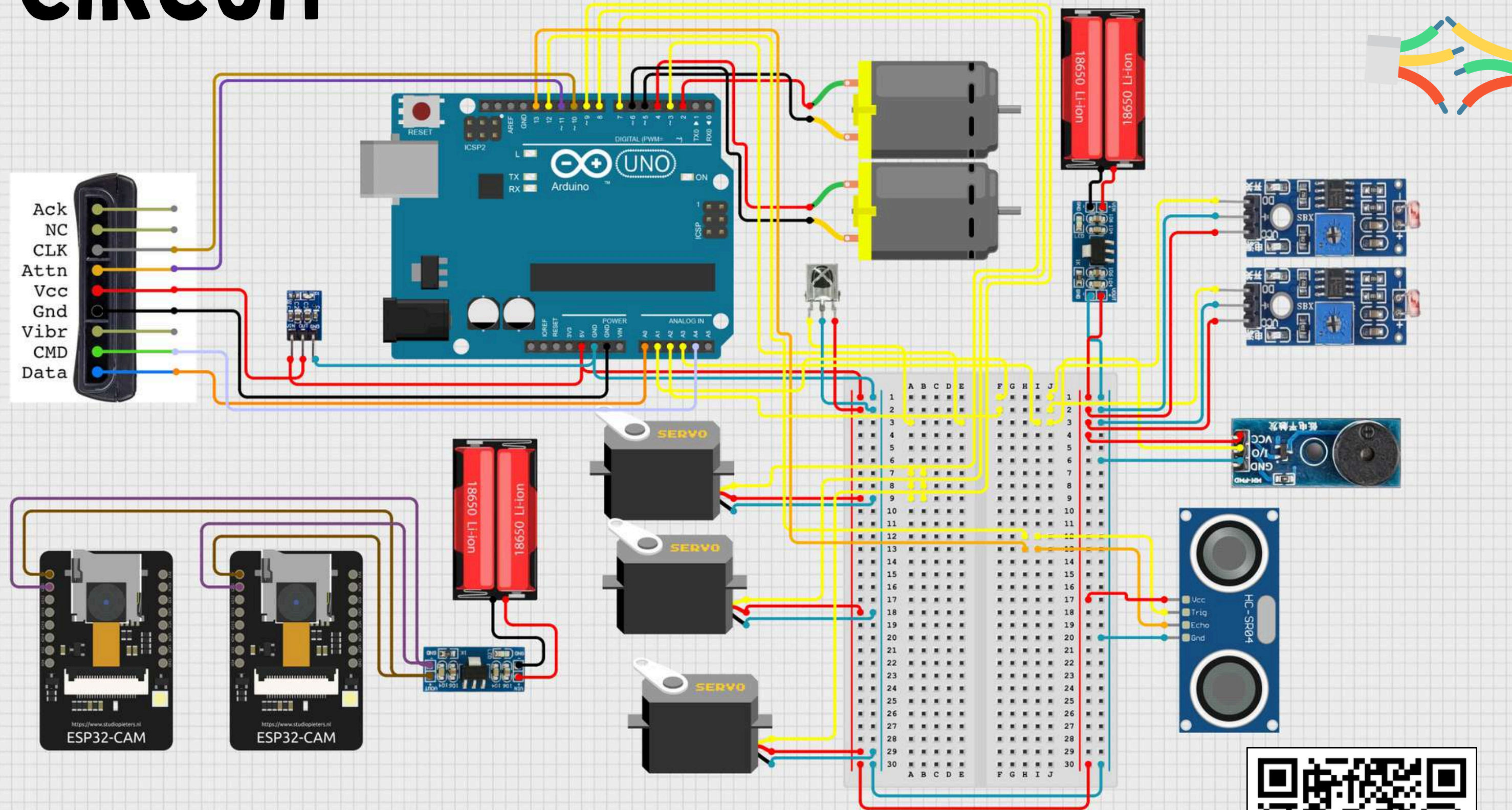
- Pinii A0 și A4 sunt utilizați ca pini digitali .
- GND comun cu Arduino
- Alimentarea este asigurată de Vout-ul AMS1117, care furnizează 3.3V DC.

REGULATORUL AMS1117 ESTE CONECTAT ASTFEL:

- Intrare (VIN): 5V de la Arduino
- Masa (GND): GND de la Arduino
- Iesire (VOUT): 3.3V DC

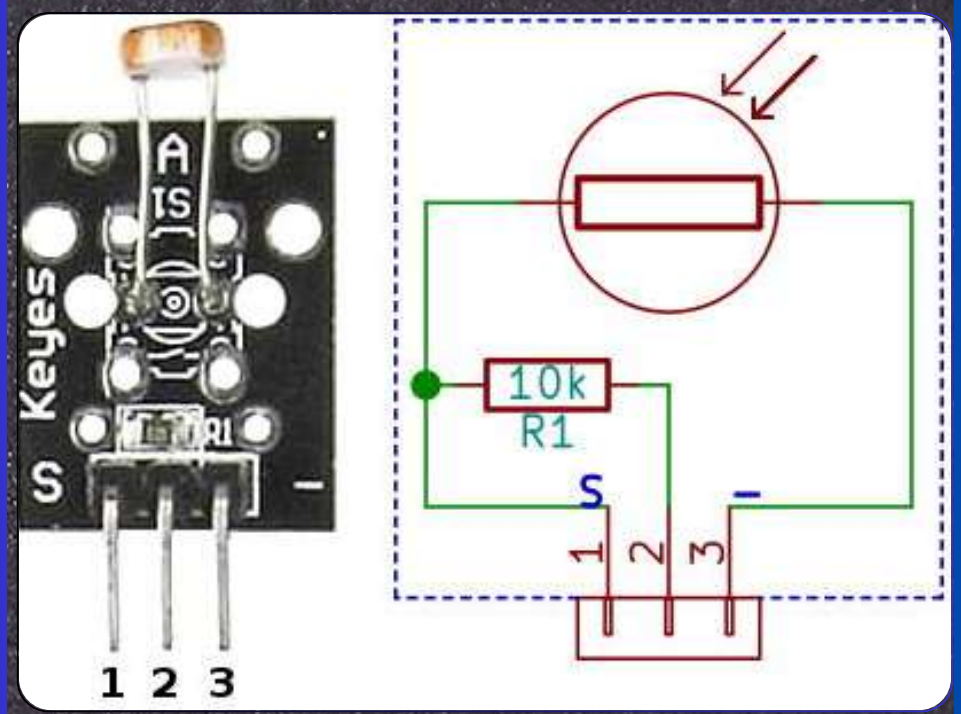
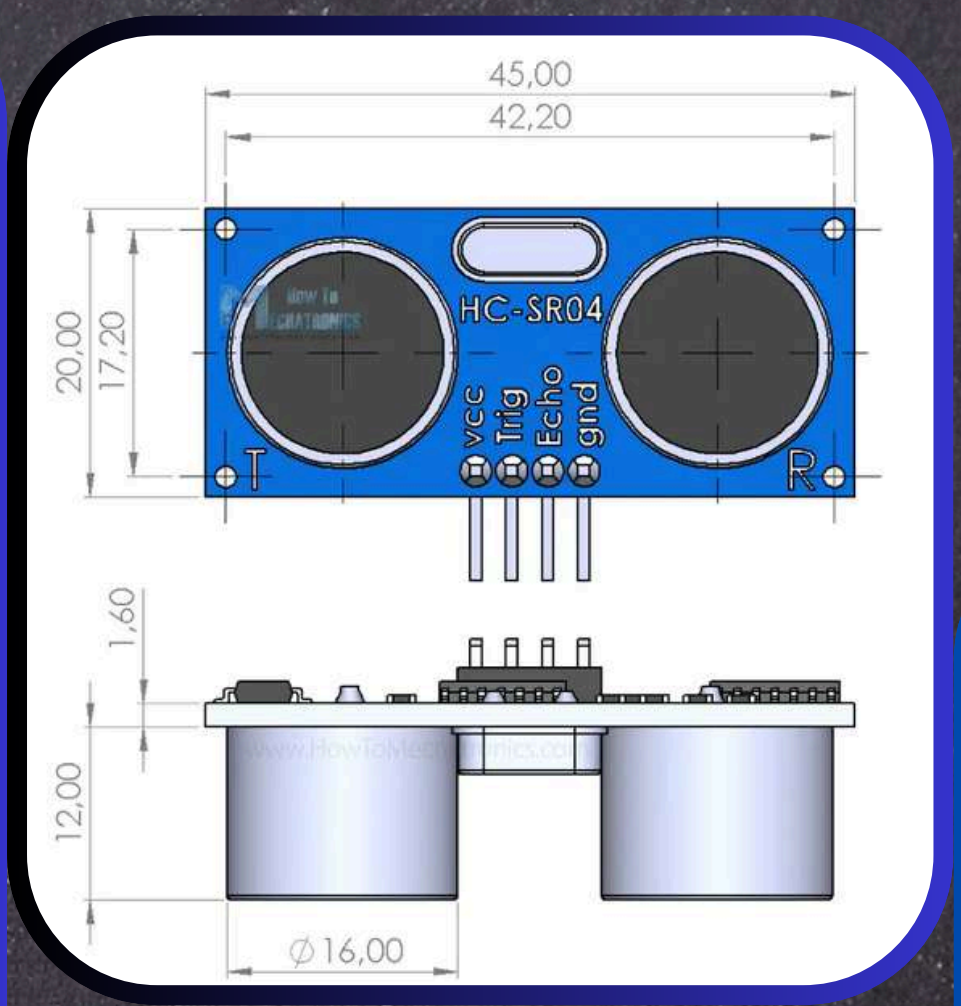
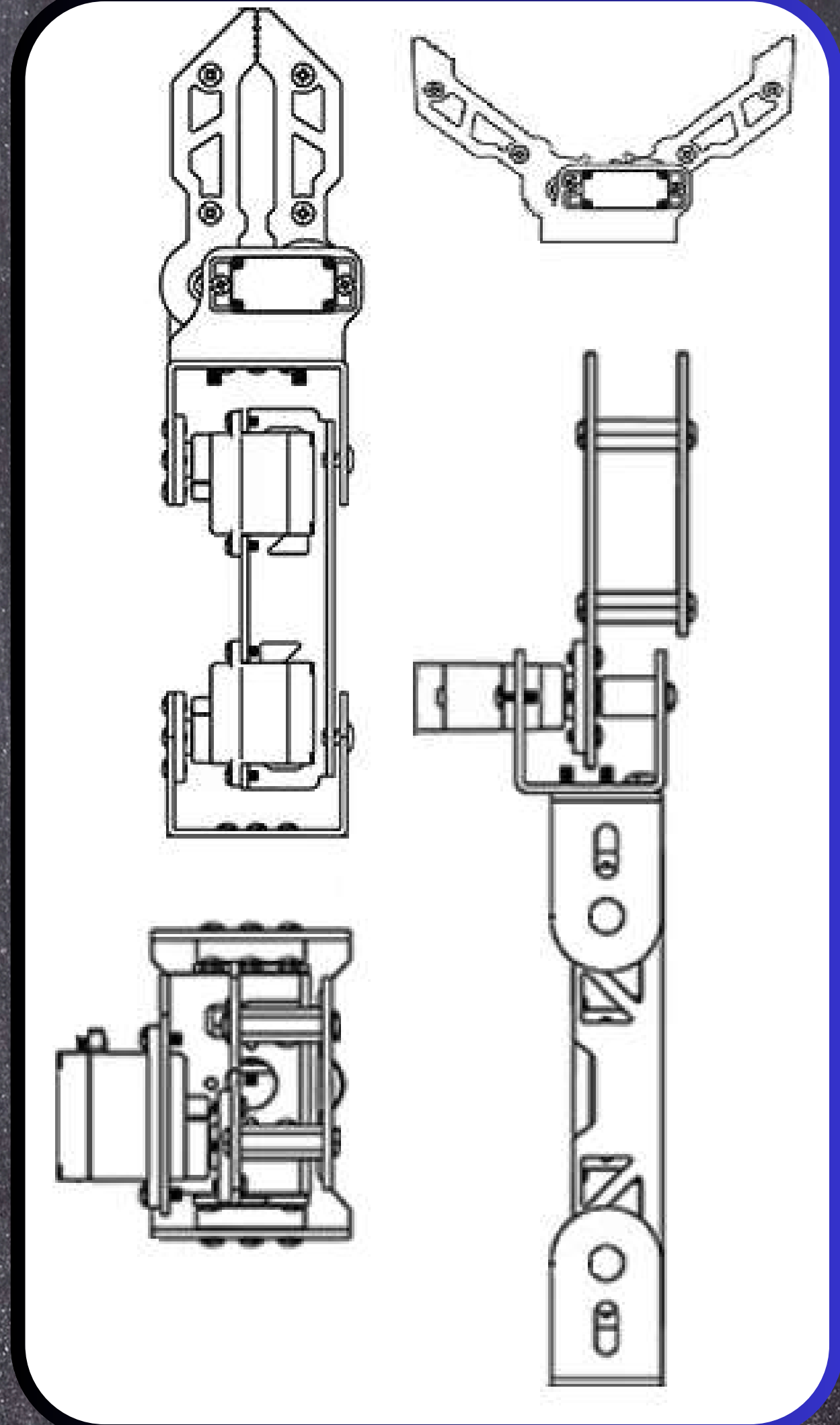


CIRCUIT

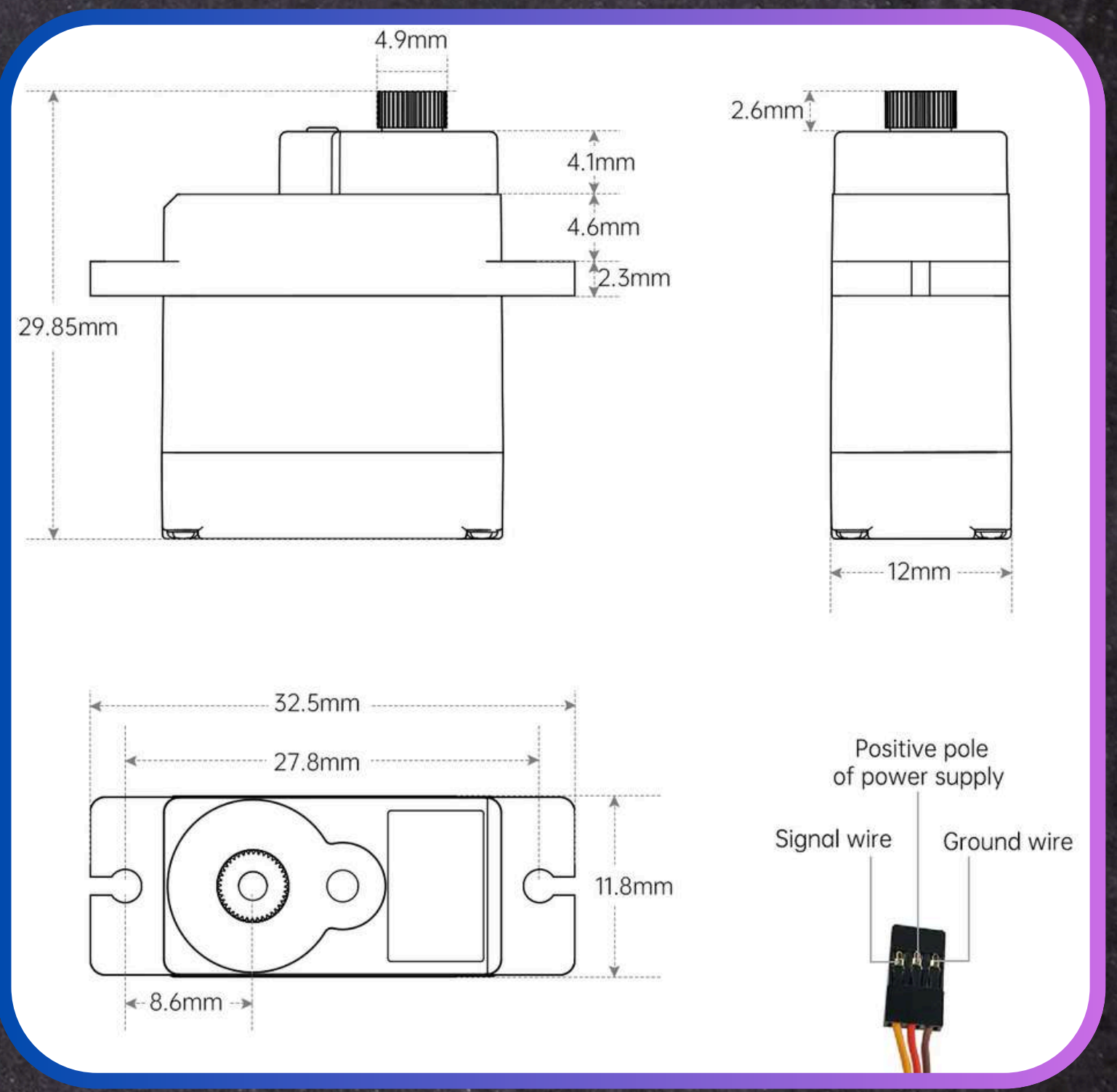


Acest circuit reprezintă o variantă similară, realizată cu componente compatibile, pentru sistemul Tracky realizat pe platforma Cirkit Designer.

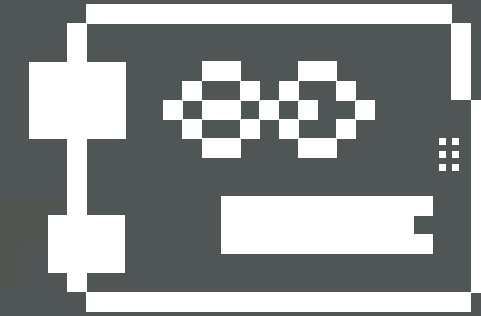




DIAGRAMME



SOFTWARE



MOD AUTONOM:

ALGORITM DE EVITARE A OBSTACOLELOR

(EX.DEPLASARE ÎNAINTE → DETECTARE OBSTACOL → SCHIMBARE DIRECȚIE).

ALGORITM DE MASURAREA INTENSITATII LUMINOASE SI DETERMINAREA PREZENTEI SAU ABSENTEI ACESTEIA, DEPLASÂNDU-SE ÎN CEA MAI PUTERNICĂ ZONĂ.

MOD MANUAL: CONTROL PRIN TELECOMANDĂ IR SAU PS2.

MOD MIXT:

POSIBILITATEA DE A COMUTA ÎNTRE AUTOMAT/MANUAL PRINTR-UN BUTON.

EFICIENȚĂ:

COD OPTIMIZAT FĂRĂ DELAY INUTIL, FOLOSIND ÎNTRERUPERI (INTERRUPTS) PENTRU SENZORI ȘI CONTROL MOTOARE.





DESIGN INDUSTRIAL

DESIGN MODULAR:

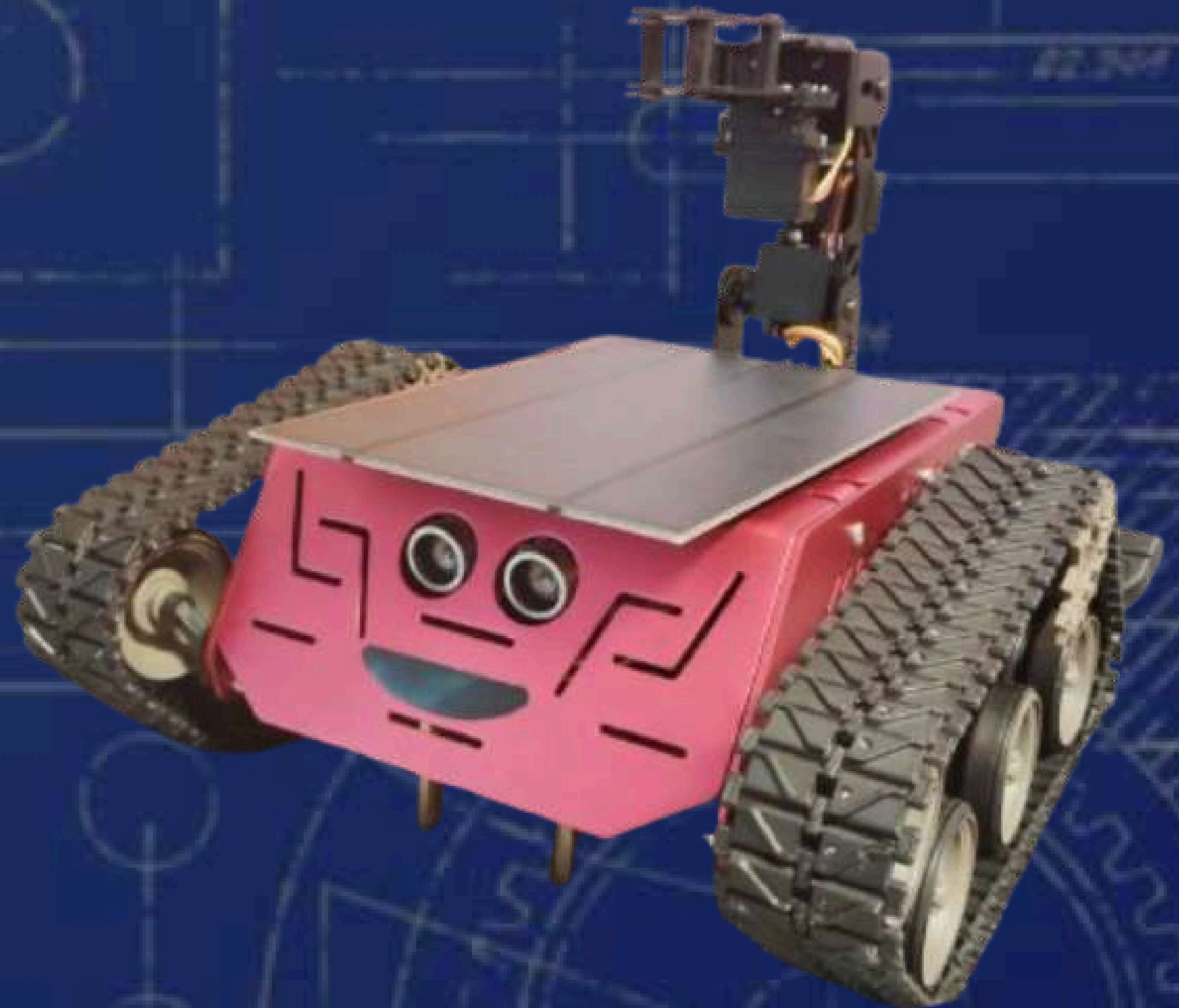
COMPONENTELE SUNT UȘOR DE
ASAMBLAT ȘI ÎNLOCUIT.

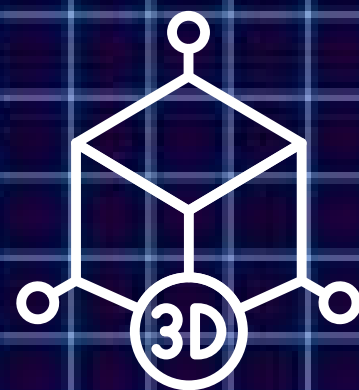
MATERIALE:

CARCASĂ UȘOARĂ, DIN ALUMINIU
REZISTENT, ȘENILE 3D (SLA).

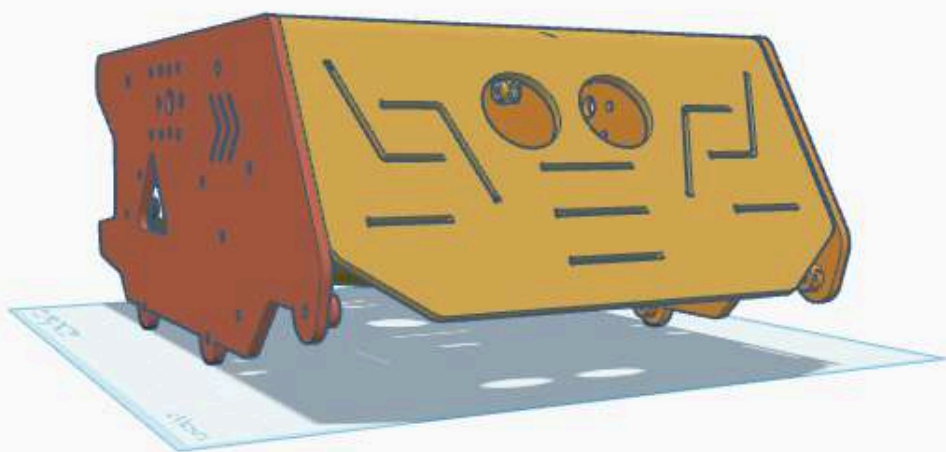
AUTOMATIZARE:

POATE FI PRODUS ÎN SERIE
DATORITĂ DESIGNULUI SIMPLU.

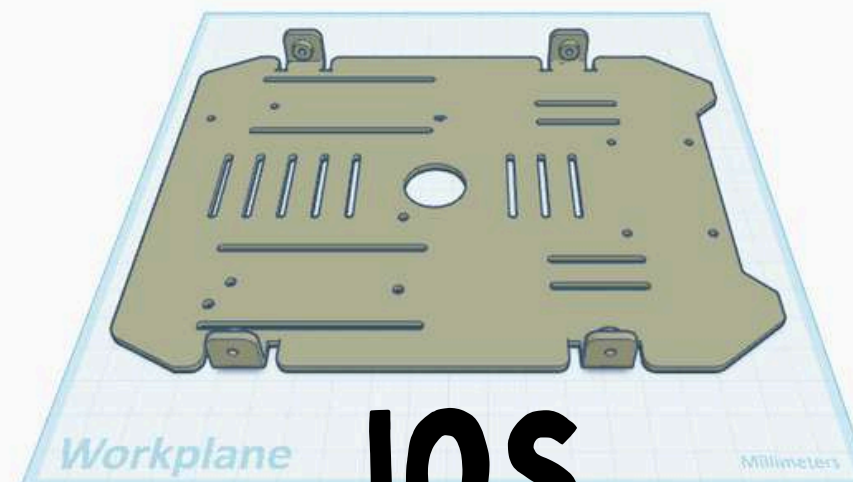




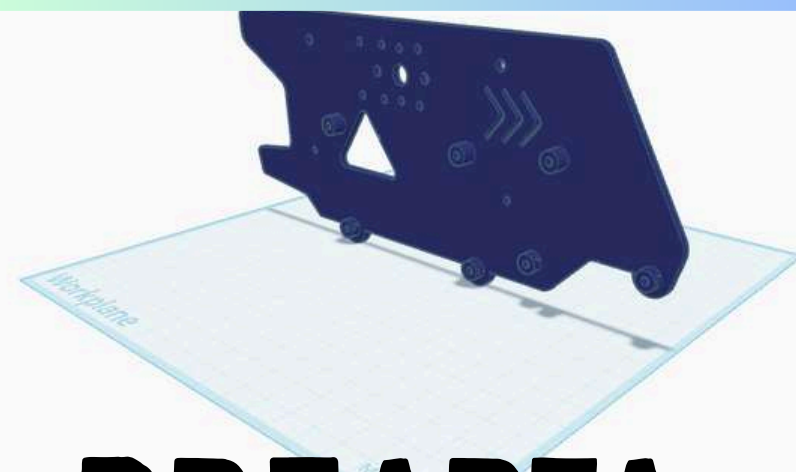
MODELE 3D



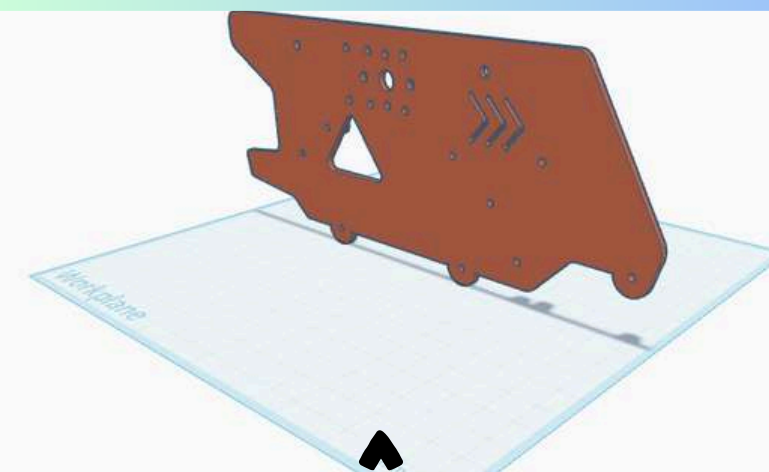
SUS



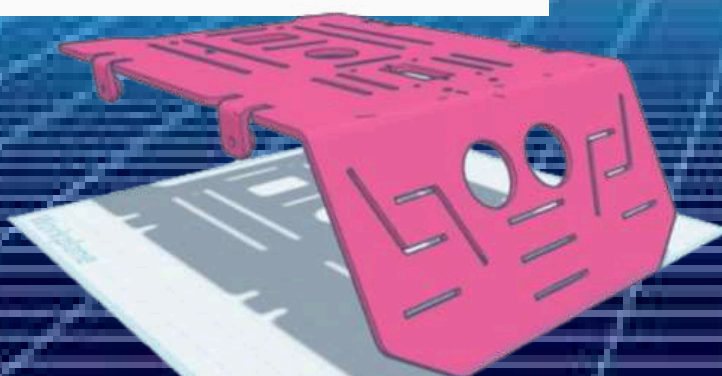
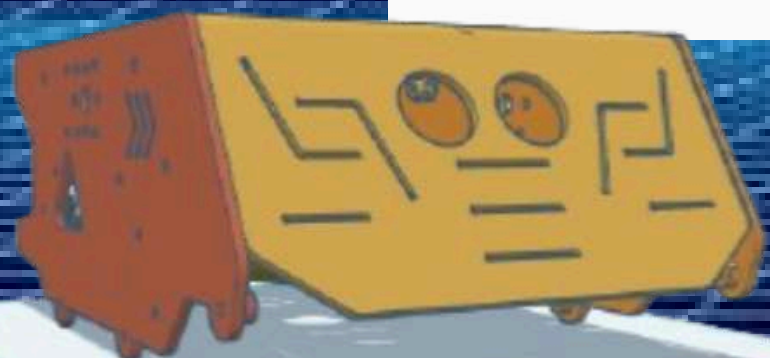
JOS



DREAPTA



STÂNGA



<Code>

<ESP32CAM>

```
#include "esp_camera.h"
#include "FS.h"
#include "SD_MMC.h"
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "driver/rtc_io.h"

// Pinout pentru RHYX-M21-45 &
OV2640
#define PWDN_GPIO_NUM    32
#define RESET_GPIO_NUM  -1
#define XCLK_GPIO_NUM    0
#define SIOD_GPIO_NUM    26
#define SIOC_GPIO_NUM    27
#define Y9_GPIO_NUM      35
#define Y8_GPIO_NUM      34
#define Y7_GPIO_NUM      39
#define Y6_GPIO_NUM      36
#define Y5_GPIO_NUM      21
#define Y4_GPIO_NUM      19
#define Y3_GPIO_NUM      18
#define Y2_GPIO_NUM       5
#define VSYNC_GPIO_NUM   25
#define HREF_GPIO_NUM    23
#define PCLK_GPIO_NUM    22
```

```
unsigned long photoCounter = 0;

void initMicroSDCard(){
    Serial.println("Inițializare SD Card...");
    if(!SD_MMC.begin()){
        Serial.println("Eroare la montarea SD Cardului");
        return;
    }
    uint8_t cardType = SD_MMC.cardType();
    if(cardType == CARD_NONE){
        Serial.println("Nu există un card SD conectat");
        return;
    }
    Serial.println("SD Card montat cu succes!");
}

String getPictureFilename(){
    String filename;
    fs::FS &fs = SD_MMC;
    File file;
    do {

        filename = "/foto_" + String(photoCounter++) + ".pgm";
        file = fs.open(filename, FILE_READ);
        file.close();
    } while (file);

    return filename;
}
```

```
void takeSavePhoto(){
    camera_fb_t * fb = esp_camera_fb_get();

    if(!fb) {
        Serial.println("Captura camerei a eșuat");
        delay(1000);
        ESP.restart();
    }
    String path = getPictureFilename();
    Serial.printf("Fișier PGM: %s\n", path.c_str());

    fs::FS &fs = SD_MMC;
    File file = fs.open(path.c_str(), FILE_WRITE);
    if(!file){
        Serial.println("Eroare la deschiderea fișierului
        pentru scriere");
    } else {

        file.printf("P5\n%d %d\n255\n", fb->width, fb-
        >height);

        file.write(fb->buf, fb->len);
        Serial.printf("Imagine salvată: %s\n",
        path.c_str());
        file.close();
    }

    esp_camera_fb_return(fb);
}
```

<Code>

<ESP32CAM>

```
void configCamera(){
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_GRAYSCALE;
  // Format de culoare grayscale (alb-negru)
```

```
if(psramFound()){
  config.frame_size = FRAMESIZE_UXGA;
  config.jpeg_quality = 10;
  config.fb_count = 1;
} else {
  config.frame_size = FRAMESIZE_UXGA;
  config.jpeg_quality = 12;
  config.fb_count = 1;
}

esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
  Serial.printf("Eroare la inițializarea camerei: 0x%x", err);
  return;
}

void setup() {
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
  Serial.begin(115200);
  delay(1000);

  Serial.print("Inițializare cameră...");
  configCamera();
  Serial.println("Camera inițializată cu succes!");

  initMicroSDCard();
}
```

```
void loop() {
  takeSavePhoto();
  delay(3000);
}
```


<Codul> <Robotului>

```
#include "IR_remote.h" // Include biblioteca pentru controlul telecomenzii IR
#include "keymap.h"     // Include biblioteca pentru maparea tastelor telecomenzii IR
#include <Servo.h>      // Include biblioteca pentru controlul servomotorului
#include "PS2X_lib.h"   //for v1.6

Servo servoD7, servoD9, servoD8;
PS2X ps2x;             // create PS2 Controller Class
IRremote ir(3);        // controlul telecomenzii IR, inițializat pe pinul 3
volatile char IR_Car_Mode = ' '; // Mod de control al robotului (primit prin IR)
volatile boolean IR_Mode_Flag = false; // Flag pentru indicarea modului de control IR
volatile int Front_Distance; // Variabilă pentru distanța frontală (senzor ultrasonic)
bool ultrasonicAvoidanceMode = false; // Mod de evitare a obstacolelor cu senzor ultrasonic
bool ultrasonicFollowMode = false; // Mod de urmărire a persoanelor cu senzor ultrasonic
volatile boolean continuous_mode = false; // Flag pentru indicarea modului continuu
volatile boolean ldrMode = false; // Flag pentru modul LDR

const int ldrPin2 = A1; // Pinul pentru al doilea senzor LDR
const int ldrPin3 = A2; // Pinul pentru al treilea senzor LDR

const int LDR_MIN = 800; // Prag minim de lumină
const int LDR_MAX = 1023; // Prag maxim de lumină

// Variabile pentru stocarea unghiurilor servomotoarelor
int angleD8 = 90;
int angleD7 = 90;
int angleD9 = 90;

// Variabile pentru detectarea tranziției de la „neapăsător” la „apăsător”
bool lastButtonState_RED = false;
bool lastButtonState_PINK = false;
bool lastButtonState_GREEN = false;
bool lastButtonState_BLUE = false;
```

```
int error = 0;
byte type = 0;
byte vibrate = 0;
// Declarații funcții
void STOP();
void Move_Forward(int car_speed);
void Move_Backward(int car_speed);
void Rotate_Left(int car_speed);
void Rotate_Right(int car_speed);
void IR_remote_control();
float checkdistance();
void Ultrasonic_Avoidance();
void Ultrasonic_Follow();

#define BUZZER_PIN A3

void playTone(int frequency, int duration) {
    long period = 1000000L / frequency;
    long cycles = (long)duration * 1000L / period;

    for (long i = 0; i < cycles; i++) {
        digitalWrite(BUZZER_PIN, HIGH);
        delayMicroseconds(period / 2);
        digitalWrite(BUZZER_PIN, LOW);
        delayMicroseconds(period / 2);
    }
}

void startupSound() {
    pinMode(BUZZER_PIN, OUTPUT);
```


<Codul>

<Robotului>

```
playTone(440, 150); // A4 - Ton scurt de start  
delay(100);
```

```
playTone(523, 150); // C5 - Ton scurt pentru confirmare  
delay(100);
```

```
playTone(440, 150); // A4 - Ton scurt pentru finalizare  
delay(100);  
}
```

```
void setup() {
```

```
  startupSound();
```

```
  servoD7.attach(7);  
  servoD9.attach(9);  
  servoD8.attach(8);
```

```
  servoD7.write(90);  
  servoD9.write(90);  
  servoD8.write(0);
```

```
  pinMode(2, OUTPUT);  
  pinMode(5, OUTPUT);
```

```
  pinMode(4, OUTPUT);  
  pinMode(6, OUTPUT);
```

```
  pinMode(12, OUTPUT);  
  pinMode(13, INPUT);
```

```
  pinMode(LDRPIN2, INPUT);  
  pinMode(LDRPIN3, INPUT);
```

```
  DELAY(0);  
  STOP();  
  SERIAL.BEGIN(115200);
```

```
  DELAY(500);
```

```
  ERROR = PS2X.CONFIG_GAMEPAD(11,18,14,10, TRUE, TRUE);
```

```
  IF (ERROR == 0) {  
    SERIAL.PRINTLN("FOUND CONTROLLER, CONFIGURED SUCCESSFUL");  
    SERIAL.PRINTLN("TRY OUT ALL THE BUTTONS, X WILL VIBRATE THE CONTROLLER,  
    FASTER AS YOU PRESS HARDER;");  
    SERIAL.PRINTLN("HOLDING L1 OR R1 WILL PRINT OUT THE ANALOG STICK VALUES.");  
  }
```

```
  else if (error == 1)  
    Serial.println("No controller found, check wiring, see readme.txt  
    to enable debug. visit www.billporter.info for troubleshooting tips");
```

```
  else if (error == 2)  
    Serial.println("Controller found but not accepting commands. see readme.txt to  
    enable debug.  
    Visit www.billporter.info for troubleshooting tips");
```

```
  else if (error == 3)  
    Serial.println("Controller refusing to enter Pressures mode, may not support it.  
    ");
```

```
  type = ps2x.readType();
```

```
  switch (type) {  
    case 0:  
      Serial.println("Unknown Controller type");  
      break;  
  }  
}
```


<Code> <RobotUI>

```
else if (error == 1)
  Serial.println("No controller found, check wiring, see readme.txt to enable debug.
visit www.billporter.info for troubleshooting tips");

else if (error == 2)
  Serial.println("Controller found but not accepting commands. see readme.txt to
enable debug. Visit www.billporter.info for troubleshooting tips");

else if (error == 3)
  Serial.println("Controller refusing to enter Pressures mode, may not support it. ");

type = ps2x.readType();

switch (type) {
  case 0:
    Serial.println("Unknown Controller type");
    break;
}

if (ldrMode) {
  int ldrLeft = analogRead(ldrPin2);
  int ldrRight = analogRead(ldrPin3);

  const int sensitivity = 50;

  if (ldrLeft > LDR_MIN || ldrRight > LDR_MIN) {
    int diff = ldrLeft - ldrRight;
```

```
if (abs(diff) < sensitivity) {
  STOP();
} else if (diff > 0) {
  Rotate_Left(150);
} else {
  Rotate_Right(150);
}
} else {
  STOP();
}

delay(100);
}

ps2x.read_gamepad(false, vibrate);

bool R1Pressed = ps2x.Button(PSB_R1);

auto updateServo = [&R1Pressed](bool
currentButton, bool &lastButtonState, int &angle,
Servo &servo) {
  if (currentButton && !lastButtonState) {
    if (R1Pressed) {
      angle = constrain(angle - 15, 0, 180);
    } else {
      angle = constrain(angle + 15, 0, 180);
    }
    servo.write(angle);
  }
  lastButtonState = currentButton;
};
```


<Codul> <Robotului>

```
updateServo(ps2x.Button(PSB_RED), lastButtonState_RED, angleD8, servoD8);  
updateServo(ps2x.Button(PSB_GREEN), lastButtonState_GREEN, angleD7, servoD7);  
updateServo(ps2x.Button(PSB_BLUE), lastButtonState_BLUE, angleD9, servoD9);
```

```
ps2x.read_gamepad(false, 0); // Citim datele de la controller
```

```
if (ps2x.Button(PSB_PAD_UP)) {  
    delay(50); // Întârziere pentru debounce  
    if (ps2x.Button(PSB_PAD_UP)) { Serial.println("UP is being held"); Move_Forward(255); delay(100); STOP(); }  
}
```

```
if (ps2x.Button(PSB_PAD_RIGHT)) {  
    delay(50);  
    if (ps2x.Button(PSB_PAD_RIGHT)) { Serial.println("Right is being held"); Rotate_Right(255); delay(100); STOP(); }  
}
```

```
if (ps2x.Button(PSB_PAD_LEFT)) {  
    delay(50);  
    if (ps2x.Button(PSB_PAD_LEFT)) { Serial.println("LEFT is being held"); Rotate_Left(255); delay(100); STOP(); }  
}
```

```
if (ps2x.Button(PSB_PAD_DOWN)) {  
    delay(50);  
    if (ps2x.Button(PSB_PAD_DOWN)) { Serial.println("DOWN is being held"); Move_Backward(255); delay(100); STOP(); }  
}
```

```
ps2x.read_gamepad(false, false); // Citim datele de la controller
```

```
if (ps2x.NewButtonState())  
{  
    if (ps2x.Button(PSB_L3))  
        Serial.println("L3 pressed");  
    if (ps2x.Button(PSB_R3))  
        Serial.println("R3 pressed");  
    if (ps2x.Button(PSB_L2))  
        Serial.println("L2 pressed");  
    if (ps2x.Button(PSB_R2))  
        Serial.println("R2 pressed");  
}  
if (ps2x.Button(PSB_START)) {  
}  
delay(50); // Mică întârziere pentru stabilitate  
}
```

```
void IR_remote_control() {  
    int keyCode = ir.getIrKey(ir.getCode(), 1);
```

```
    if (keyCode == IR_KEYCODE_UP) IR_Car_Mode = 'f';  
    if (keyCode == IR_KEYCODE_LEFT) IR_Car_Mode = 'l';  
    if (keyCode == IR_KEYCODE_DOWN) IR_Car_Mode = 'b';  
    if (keyCode == IR_KEYCODE_RIGHT) IR_Car_Mode = 'r';  
    if (keyCode == IR_KEYCODE_OK) {  
        continuous_mode = false;  
        ldrMode = false;  
        STOP();  
    }  
}
```

```
if (keyCode == IR_KEYCODE_2) continuous_mode = true;  
if (keyCode == IR_KEYCODE_STAR) ldrMode = true;
```

```
if (keyCode == IR_KEYCODE_2) {  
    continuous_mode = true;  
}
```


<Codul> <Robotului>

```
if (keyCode == IR_KEYCODE_5) { angleD9 += 15;
  if (angleD9 > 180) angleD9 = 180; servoD9.write(angleD9); delay(200);
}
if (keyCode == IR_KEYCODE_6) { angleD9 -= 15;
  if (angleD9 < 0) angleD9 = 0; servoD9.write(angleD9); delay(200);
}
if (keyCode == IR_KEYCODE_8) { angleD8 -= 15;
  if (angleD8 < 0) angleD8 = 0; servoD8.write(angleD8); delay(200);
}
if (keyCode == IR_KEYCODE_9) { angleD8 += 15;
  if (angleD8 > 180) angleD8 = 180; servoD8.write(angleD8); delay(200);
}
if (keyCode == IR_KEYCODE_0) { angleD7 += 15;
  if (angleD7 > 180) angleD7 = 180; servoD7.write(angleD7); delay(200);
}
if (keyCode == IR_KEYCODE_POUND) { angleD7 -= 15;
  if (angleD7 < 0) angleD7 = 0; servoD7.write(angleD7); delay(200);
}

switch (IR_Car_Mode) {
  case 'b': Move_Backward(255); delay(100); STOP(); break;
  case 'f': Move_Forward(255); delay(100); STOP(); break;
  case 'l': Rotate_Left(255); delay(100); STOP(); break;
  case 'r': Rotate_Right(255); delay(100); STOP(); break;
  case 's': STOP(); break;
}
IR_Car_Mode = ' ';
}
```

```
void Move_Forward(int car_speed = 255) {
  // Motor A2 (stânga) - înainte
  digitalWrite(2, HIGH); // Direcție înainte
  analogWrite(5, car_speed); // Viteză motor A2
  // Motor B1 (dreapta) - înainte
  digitalWrite(4, LOW); // Direcție înainte (inversat)
  analogWrite(6, car_speed); // Viteză motor B1
}
```

```
void Move_Backward(int car_speed = 255) {
  // Motor A2 (stânga) - înapoi
  digitalWrite(2, LOW); // Direcție înapoi
  analogWrite(5, car_speed); // Viteză motor A2
  // Motor B1 (dreapta) - înapoi
  digitalWrite(4, HIGH); // Direcție înapoi (inversat)
  analogWrite(6, car_speed); // Viteză motor B1
}
```

```
void Rotate_Left(int car_speed = 255) {
  // Motor A2 (stânga) - înainte
  digitalWrite(2, HIGH); // Direcție înainte
  analogWrite(5, car_speed); // Viteză motor A2
  // Motor B1 (dreapta) - înapoi
  digitalWrite(4, HIGH); // Direcție înapoi (inversat)
  analogWrite(6, car_speed); // Viteză motor B1
}
```

```
void Rotate_Right(int car_speed = 255) {
  // Motor A2 (stânga) - înapoi
  digitalWrite(2, LOW); // Direcție înapoi
  analogWrite(5, car_speed); // Viteză motor A2
  // Motor B1 (dreapta) - înainte
  digitalWrite(4, LOW); // Direcție înainte (inversat)
  analogWrite(6, car_speed); // Viteză motor B1
}
```


<Codul> <Robotului>

```
void STOP() {  
  // Oprește motor A2 (stânga)  
  analogWrite(5, 0); // Viteză 0  
  digitalWrite(2, LOW); // Direcție oprită  
  // Oprește motor B1 (dreapta)  
  analogWrite(6, 0); // Viteză 0  
  digitalWrite(4, LOW); // Direcție oprită  
}  
float checkdistance() {  
  digitalWrite(12, LOW); delayMicroseconds(2); digitalWrite(12, HIGH); delayMicroseconds(10); digitalWrite(12, LOW);  
  unsigned long duration = pulseIn(13, HIGH);  
  float distance = (duration * 0.0343) / 2;  
  return distance;  
}
```

⚠ **Codul nu va funcționa fără instalarea bibliotecilor: IR_remote.h, keymap.h, Servo.h și PS2X_lib.h.**

⚠ **Pentru ca acest cod să funcționeze, bibliotecile IR_remote.h, keymap.h, Servo.h și PS2X_lib.h trebuie să fie instalate și plasate în același folder cu fișierul sursă.**