Smart Vacuum Cleaner

Introducere

Prezentarea pe scurt a proiectului:

- Scopul final al acestui device este de a fi un aspirator smart care aspira singur evitand obstacolele din jur.
- In momentul actual nu contine pompa de aspirat, ci doar 'carcasa' si sistemul de miscare/evitare a obstacolelor.
- Ideea de la care am pornit acest proiect a fost de a crea propriul aspirator smart si de-al utiliza in casa.

Descriere generală

Cadrul circular al aspiratorului este facut dintr-o bucata de polistiren extrudat de 3cm grosime. Pe acest cadru vin prinse in holsuruburi componentele.

Pentru a porni aspiratorul, exista doua surse de alimentare: 9V respectiv 6V. Aceste surse pot fi pornite/oprite prin doua switch-uri. Odata pornita alimentarea, aspiratorul sta 5 secunde in idle asteptand ulterior un 'impuls' intr-unul din senzori pentru a porni rutina de functionare. Acesta va merge non-stop evitand obstacolele din jur.

Din pacate aveam nevoie de 4 senzori in loc de 3, deoarece exista un unghi mort in care aspiratorul se blocheaza.

Schema bloc:

Hardware Design

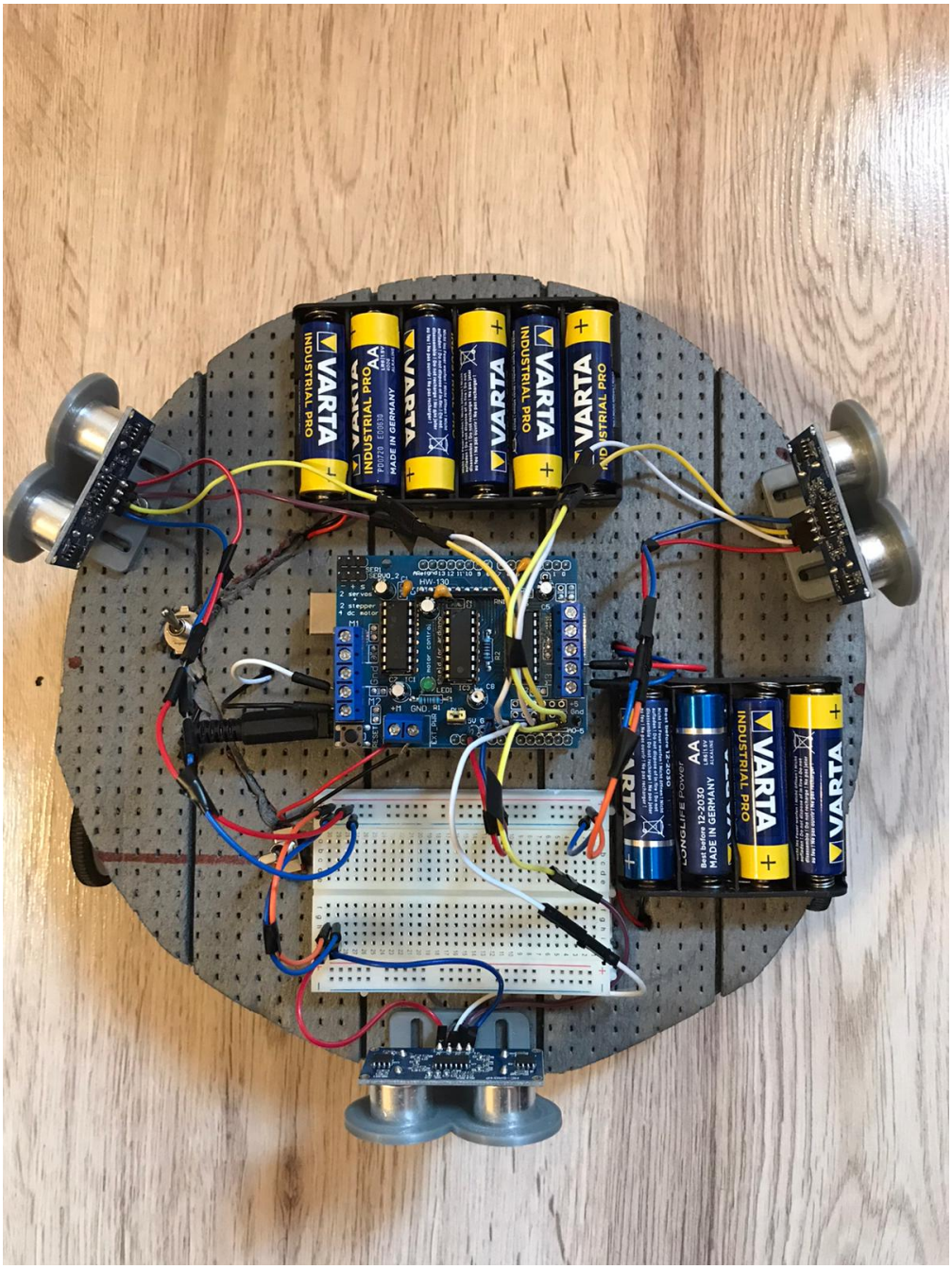
Lista de piese

- 1 x Arduino UNO R3
- 1 x L293D Motor Driver
- 1 x Breadboard 400
- 1 x Suport carcasa baterii 6xAA
- 1 x Suport carcasa baterii 4xAA
- 10 x Baterii AA Varta
- 1 x Ball Caster

- 3 x Senzor ultrasonic HC-SR04
- 3 x Suport pentru senzor ultrasonic HC-SR04
- 2 x Switch cu 2 pozitii
- 2 x Micro motor cu reducție - RPM : 60
- 2 x Roata din plastic/cauciuc - Culoare : Alb
- 2 x Suport micro-motor

Schema hardware:

Rezultat final:



Software Design

Mediu de dezvoltare

- Arduino IDE 2.0.0 - beta3 (are auto code completion)

Librarii 3rd-party

- Adafruit-Motor-Shield-library (https://github.com/adafruit/Adafruit-Motor-Shield-library)

Surse

- **smart_vacuum_cleaner.ino**

```
#include <AFMotor.h>

static constexpr int startupSleepTime = 5000;
static constexpr int loopSleepTime = 10;

// MOTOR DECLARATIONS
static AF_DCMotor motors[2] = {
  AF_DCMotor(2, MOTOR12_1KHZ),
  AF_DCMotor(3, MOTOR12_1KHZ)
};
static constexpr int motorSpeed = 200;
static constexpr int LRTurnDelay = 1200;

// SENSOR DECLARATIONS
static constexpr int triggers[3] = {A0, A1, A2}, echoes[3] = {A3, A4, A5};
static constexpr float speedOfSound = 0.0343;
static float distances[3] = {0};
static float minDistance = 5.0;

enum SensorPosition {
  FRONT_LEFT = 0,
  FRONT_RIGHT = 2,
  BACK = 1
};

// MOTOR FUNCTIONS
void initMotors() {
  motors[0].setSpeed(motorSpeed);
  motors[1].setSpeed(motorSpeed);
}
```

```
void goForward() {
  motors[0].run(FORWARD);
  motors[1].run(BACKWARD);
}

void goBackward() {
  motors[0].run(BACKWARD);
  motors[1].run(FORWARD);
}

void turnLeft() {
  motors[0].run(BACKWARD);
  motors[1].run(BACKWARD);
  delay(LRTurnDelay);
}

void turnRight() {
  motors[0].run(FORWARD);
  motors[1].run(FORWARD);
  delay(LRTurnDelay);
}

void brake() {
  motors[0].run(RELEASE);
  motors[1].run(RELEASE);
}

// SENSOR FUNCTIONS
void initSensors() {
  for (int i = 0; i < 3; i++) {
    pinMode(triggers[i], OUTPUT);
    pinMode(echoes[i], INPUT);
  }
}

void sendSensorsImpulse() {
  for (int i = 0; i < 3; i++) {
    // Trigger sensor
    digitalWrite(triggers[i], LOW);
    delayMicroseconds(2);
    digitalWrite(triggers[i], HIGH);
    delayMicroseconds(10);
    digitalWrite(triggers[i], LOW);

    // Compute distance
    float durations = pulseIn(echoes[i], HIGH);
    distances[i] = (durations / 2) * speedOfSound;
  }
}
```

```
bool isCollision(SensorPosition pos) {
  return distances[pos] <= minDistance;
}

void setup() {
  delay(startupSleepTime);

  // Enable serial debugging
  Serial.begin(9600);

  initMotors();
  initSensors();
}

void loop() {
  sendSensorsImpulse();

  // Check for collisions:
  // Left and right side collision
  if(isCollision(SensorPosition::FRONT_LEFT) && isCollision(SensorPosition::
FRONT_RIGHT)) {
    goBackward();
    delay(LRTurnDelay);
  }
  // Left side collision
  else if(isCollision(SensorPosition::FRONT_LEFT)) {
    turnRight();
    goForward();
  }
  // Right side collision
  else if(isCollision(SensorPosition::FRONT_RIGHT)) {
    turnLeft();
    goForward();
  }

  // Back side collision
  if(isCollision(SensorPosition::BACK)) {
    goForward();
  }

  delay(loopSleepTime);
}
```

Detalii despre implementare

- Se porneste una din cele doua surse de alimentare (utilizand switchu-rile de ON/OFF)
- Se asteapta 5 secunde, iar pentru a porni rutina aspiratorului se atinge unul din cei trei senzori

- Daca unul din senzori este la distanta de cel mult 5cm de un obstacol, se calculeaza noua directie astfel:
  - Senzorul stanga-fata (FL) ⇒ Viraj dreapta
  - Senzorul dreapta-fata (FR) ⇒ Viraj stanga
  - Ambii senzori fata (FL && FR)⇒ Mers in spate
  - Senzorul spate (B) ⇒ Mers in fata

Rezultate Obţinute

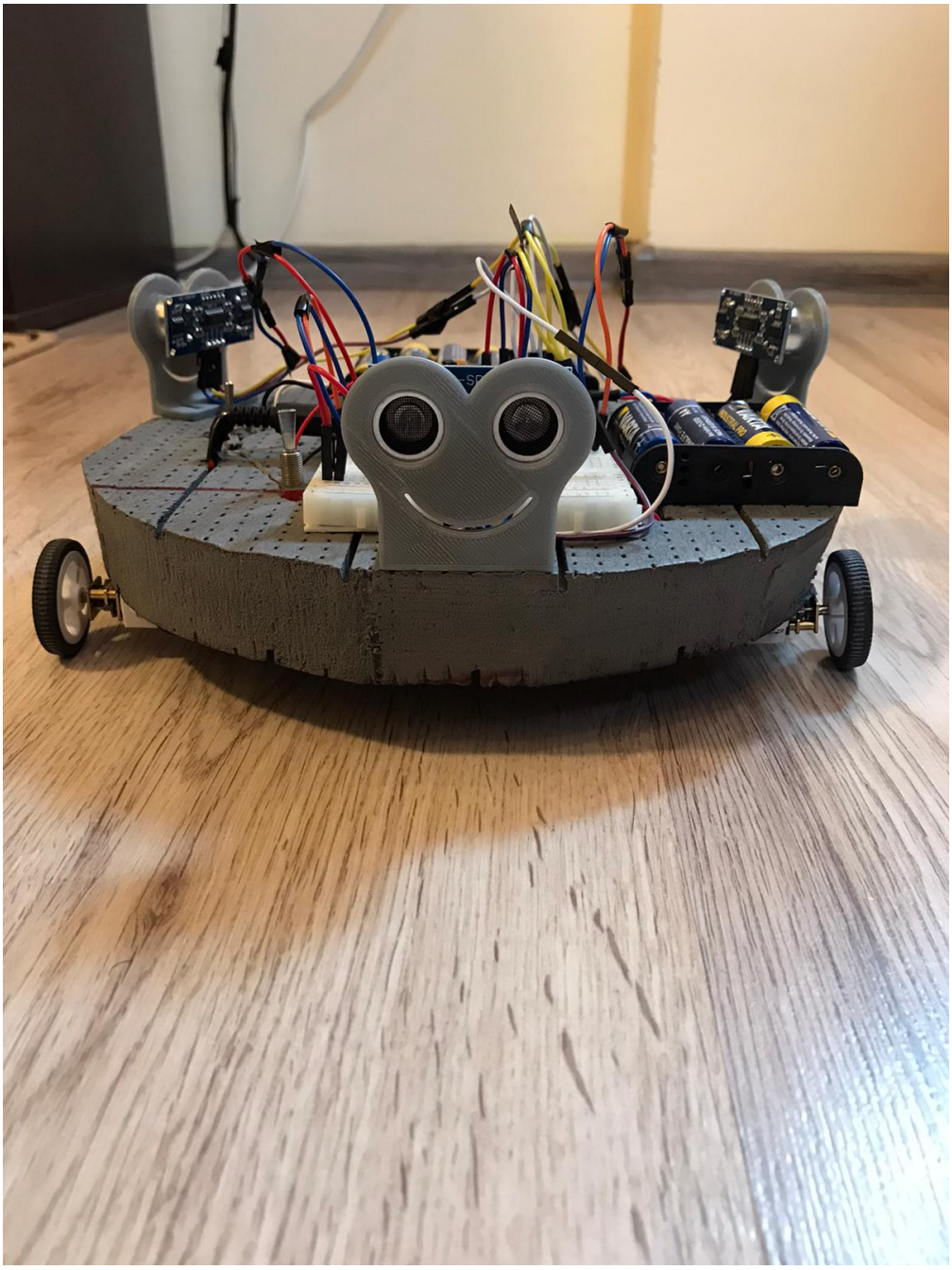
Ce nu am reusit sa obtin:

- Pompa de aspirat
- Virtual area map
- Modul bluetooth

Ce am resit sa obtin:

- Partea mecanica (destul de complexa)
- Detectie obstacole si miscari simple

Videoclip demonstrativ:



Concluzii

Partea mecanica a fost mult mai complexa decat am crezut initial. Acest proiect m-a facut sa vreau pe viitor sa creez si alte proiecte utilizand Arduino

Download

smart-vacuum-cleaner.zip

Bibliografie/Resurse

https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/using-dc-motors

https://create.arduino.cc/projecthub/abdularbi17/ultrasonic-sensor-hc-sr04-with-arduino-tutorial-327ff6

Export to PDF