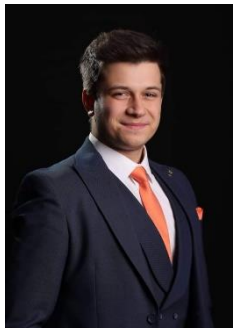
**Gheorghe Asachi Technical University of Iasi**
**Faculty of Automatic Control and Computer Engineering**

**Radu-Andrei Budeanu - 1309A**

**Cosmin-Constantin Cojocaru - 1309A**



**Raul Andrei Dima - 1306A**

**Alexandru-Gicu Melinte - 1306A**

**Mails:**

- radu-andrei.budeanu@student.tuiasi.ro
- cosmin-constantin.cojocaru@student.tuiasi.ro
- raul-andrei.dima@student.tuiasi.ro
- alexandru-gicu.melinte@student.tuiasi.ro

**Project:** Remote Controlled Car

**Git:** https://github.com/RaduAndrei99/AutonomousCar/tree/main

**Video: N/A**

# I Elevator Pitch

The project consists of a remote controlled car via a web browser. The car allows travel in all directions, this being monitored by the camera attached to the chassis.

# II Cover image

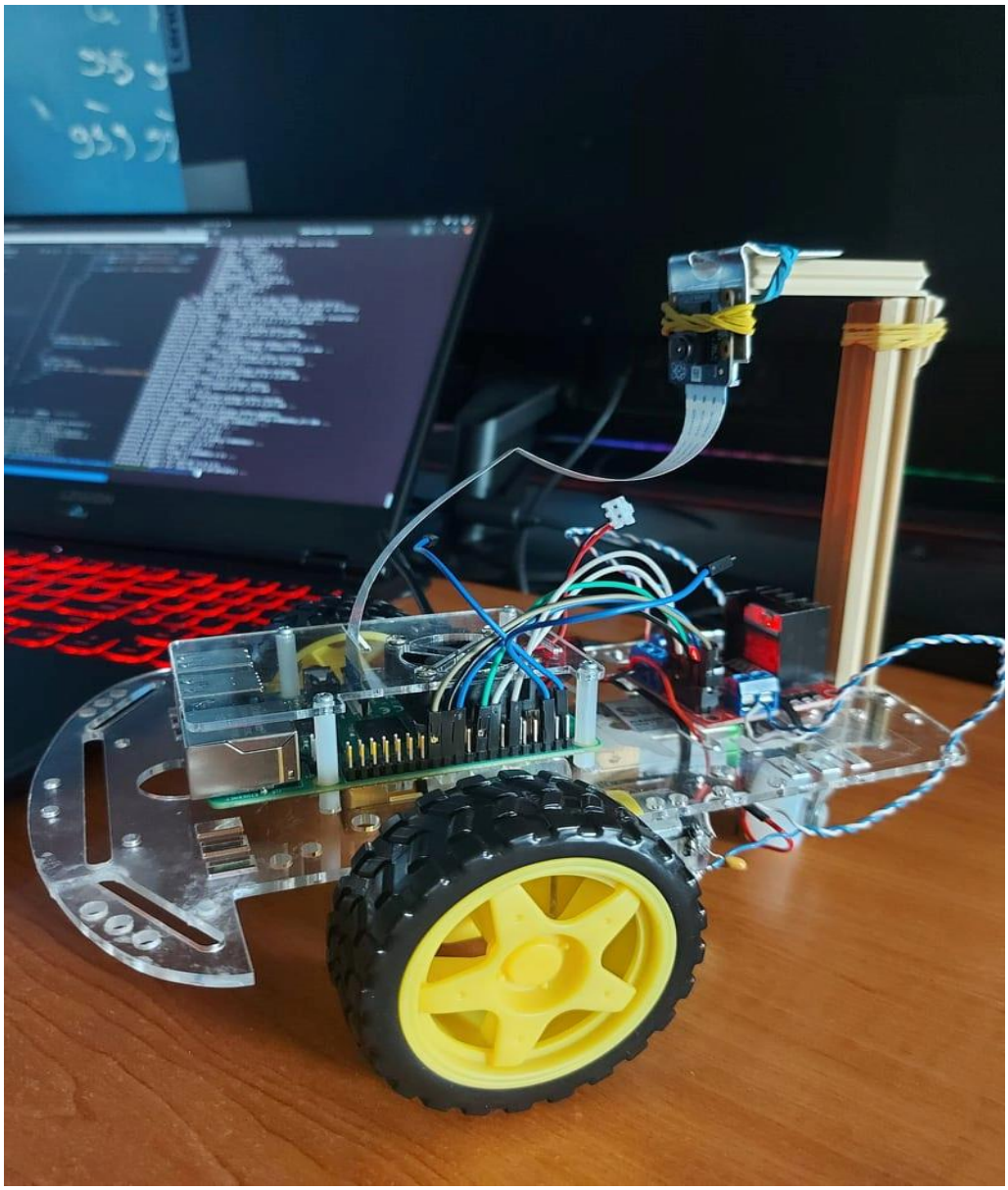


**Image 1 – The Remote Controlled Car**

## III Story

The Remote Controlled Car is a car controlled via a client remote page provided by a web server made in NodeJS. The server is providing a web interface for the user to control the car. The client page has the control buttons to move the car (forward, backward, left and right), on/off brake lights, on/off daylights, PWM (pulse with modulation) used to adjust the speed of motor, switch (manual to automatic), signal lights and also it shows the actions that the car is performing in real time (a log).

This project is made for educational purposes, entertaiment and also for testing our skills in hardware programming using Python and Raspberry Pi 4.

The NodeJS server is running on the Raspberry Pi 4 and it can be accesed in the Local Area Network. The user can also see the stream coming from the Raspberry Pi NOIR camera mounted on the car and can check the car actions log.

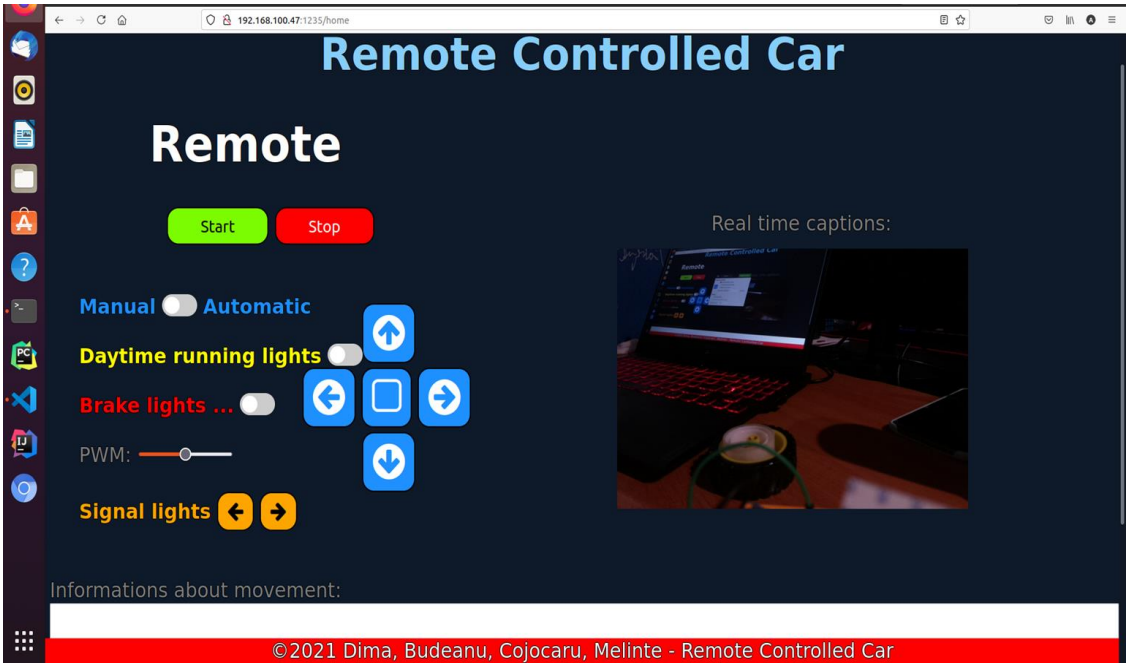



**Image 2 – The Remote Interface on the PC**

The page is responsive and it can also be used from a mobile phone. For that functionality we designed the interface using media queries.

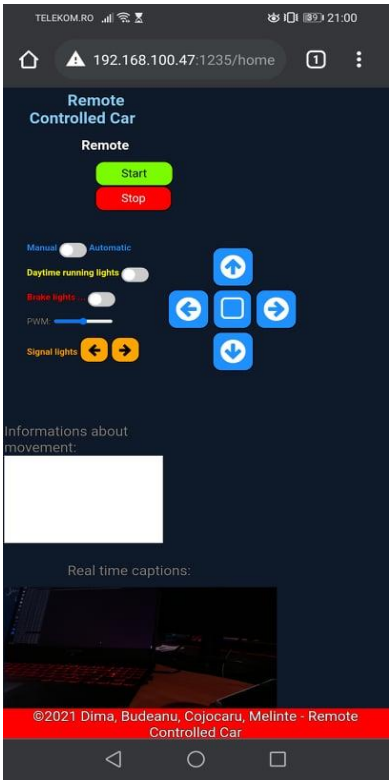



**Image 3 – The Remote Interface on a mobile phone**

In the initial stage, when the user connects to the server, the web interface is displayed. To start the car, the user must press the start button in the interface. This command sends a message to the server to notify it to run the script in Python that starts the car. Once the car is turned on, the user can perform several commands. These are: moving forward, backward, left and right. It can also turn on the brake lights and it can stream the live feed from the camera mounted on the top.

Every action is displayed real-time in the browser's webpage.

## How to setup your car

In order to make your own Remote Controlled Car, after the assembly of the car as it's shown in the schematics, follow these steps:

1 Download and install Raspberry OS on your Raspberry Pi 4 (https://www.raspberrypi.org/software/ ).

2 Make the board to connect to the local Wi-Fi using this tutorial (https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md ).

3 Connect to your Raspberry Pi using SSH (https://www.raspberrypi.org/documentation/remote-access/ssh/ ).

4 Enable the camera from the raspi-config interface using the following command on the terminal.

```
sudo raspi-config
```

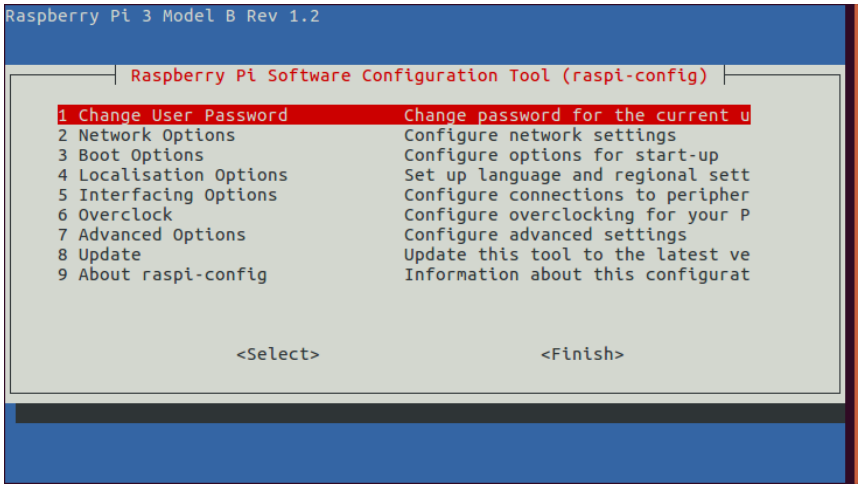



**Image 4 – The raspi-config interface**

Interfacing Options - > Enable the Camera and then reboot the board.

5 Install NodeJS on your Rasberry Pi (https://www.makersupplies.sg/blogs/tutorials/how-to-install-node-js-and-npm-on-the-raspberry-pi ).

6 Download the NodeJS server from the GitHub repository (https://github.com/RaduAndrei99/AutonomousCar/tree/andrei ).

7 Start the server with the following command:

```
node app.js
```

Your NodeJS server should start.

```
pi@raspberrypi:~/AutonomousCar $ node app.js
Serverul rulează la adresa http://localhost:1235/home
A user connected
child process close all stdio with code 0
key: s=> command: pressed
false
child process close all stdio with code 0
child process close all stdio with code 0
child process close all stdio with code 0
child process close all stdio with code 0
key: start=> command: start
false
child process close all stdio with code 0
child process close all stdio with code 0
key: s=> command: pressed
true
child process close all stdio with code 0
child process close all stdio with code 0
key: s=> command: released
false
child process close all stdio with code 0
child process close all stdio with code 0
child process close all stdio with code 0
child process close all stdio with code 0
child process close all stdio with code 0
child process close all stdio with code 0
child process close all stdio with code 0
```

**Image 5 – The output of the NodeJS server**

8 If everything all right, you should be able the control your car and see data from the camera.

## Possible improvements

One thing that we can improve is the frame rate of the video stream. Currently we are streaming at about 1-2 frames per second.

The second thing that we want to add is the control of the motor's speed by modifing the PWM.

The last thing we want to add is a Lane Keep Assist function.

## IV Hardware components

- ✓ RaspberryPi 4B x 1
- ✓ Camera Pi NoIR V2 x 1
- ✓ Female / Female jumper wires x 6
- ✓ Male / Female jumper wires x 24
- ✓ RC car chassis x 1
- ✓ L298N Motor Driver Module x 1
- ✓ Battery 9V x 2
- ✓ Power Bank x 1
- ✓ Red Led x 2
- ✓ White Led x 6
- ✓ Mini breadboard x 1
- ✓ Resistor 200 Ohm x 8

## V Software:

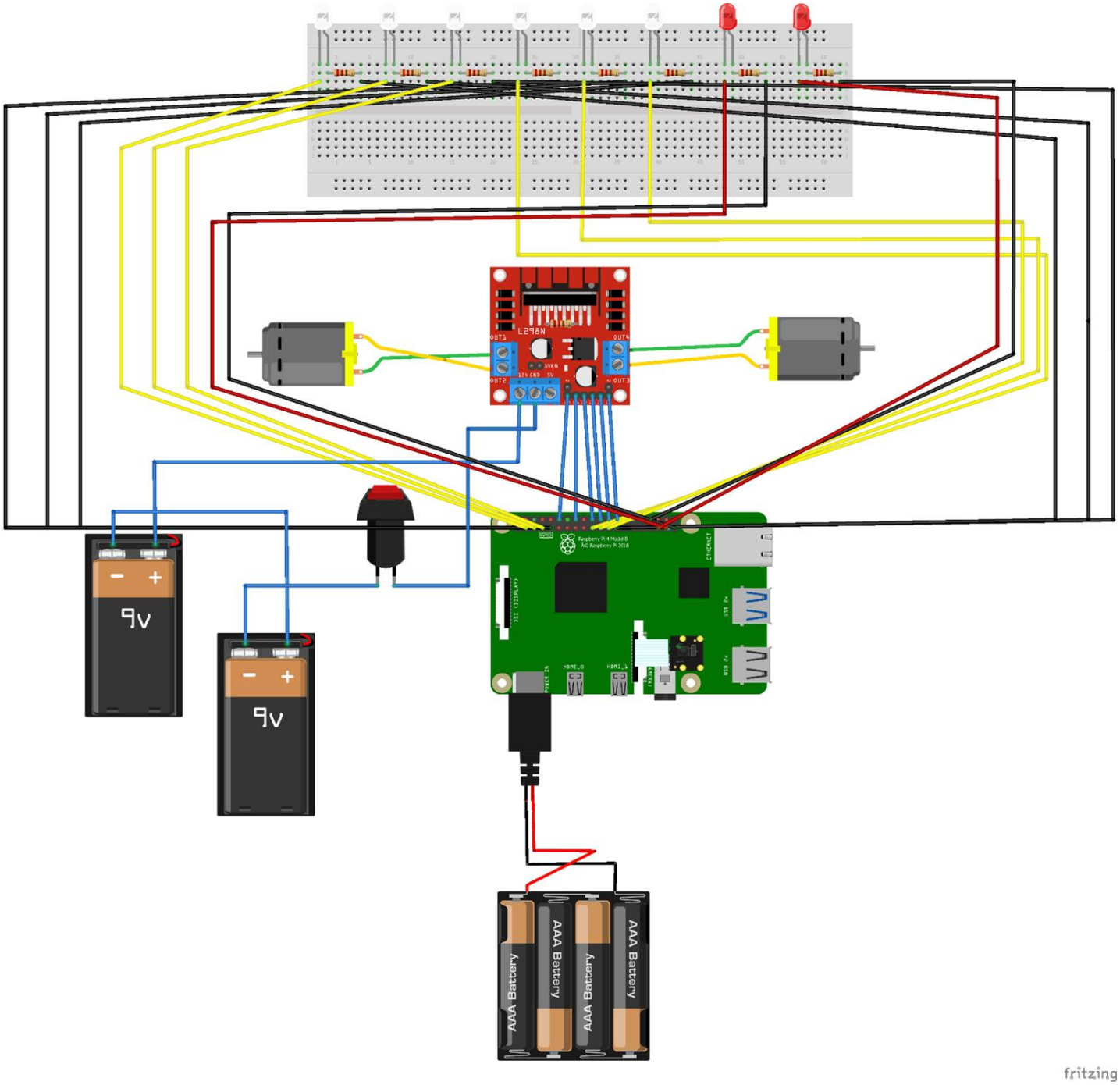
- ✓ Raspberry Pi OS
- ✓ Python3
- ✓ ssh
- ✓ NodeJs
- ✓ Express

- ✓ HTML
- ✓ CSS
- ✓ JavaScript
- ✓ Linux

## VI Schematics

**Image 6 – Schematics**

## VII Code




**functions.py**

```
import RPi.GPIO as GPIO
```

```
import sys
import threading
import traceback
import logging

HOST = '192.168.100.47'
PORT = 1234

motor_A_0 = 16
motor_A_1 = 18
motor_B_0 = 22
motor_B_1 = 24
pwm_motor_A = 12
pwm_motor_B = 26

GPIO.setmode(GPIO.BOARD)

GPIO.setup(pwm_motor_A, GPIO.OUT)
GPIO.setup(pwm_motor_B, GPIO.OUT)

pwm_A = GPIO.PWM(pwm_motor_A, 80)
pwm_B = GPIO.PWM(pwm_motor_B, 80)

def rear_lights_on():
    GPIO.output(37, 1)
    GPIO.output(35, 1)

def rear_lights_off():
    GPIO.output(37, 0)
    GPIO.output(35, 0)

def move_forward(settingPins, settingDutyCycle, speed):
    global motor_A_0
    global motor_A_1
    global motor_B_0
    global motor_B_1
    global pwm_A
    global pwm_B
    if settingPins:
        GPIO.output(motor_A_0, GPIO.HIGH)
        GPIO.output(motor_A_1, GPIO.LOW)
        GPIO.output(motor_B_0, GPIO.HIGH)
        GPIO.output(motor_B_1, GPIO.LOW)
    if settingDutyCycle:
        pwm_A.ChangeDutyCycle(speed)
        pwm_B.ChangeDutyCycle(speed)

def move_backward(settingPins, settingDutyCycle, speed):
    global motor_A_0
```

```
    global motor_A_1
    global motor_B_0
    global motor_B_1
    global pwm_A
    global pwm_B
    if settingPins:
        GPIO.output(motor_A_0, GPIO.LOW)
        GPIO.output(motor_A_1, GPIO.HIGH)
        GPIO.output(motor_B_0, GPIO.LOW)
        GPIO.output(motor_B_1, GPIO.HIGH)
    if settingDutyCycle:
        pwm_A.ChangeDutyCycle(speed)
        pwm_B.ChangeDutyCycle(speed)
    rear_lights_on()

def move_to_the_left_forward(settingPins, settingDutyCycle, speed):
    global motor_A_0
    global motor_A_1
    global motor_B_0
    global motor_B_1
    global pwm_A
    global pwm_B
    if settingPins:
        GPIO.output(motor_A_0, GPIO.HIGH)
        GPIO.output(motor_A_1, GPIO.LOW)
        GPIO.output(motor_B_0, GPIO.HIGH)
        GPIO.output(motor_B_1, GPIO.LOW)
    if  settingDutyCycle:
        pwm_A.ChangeDutyCycle(speed/2)
        pwm_B.ChangeDutyCycle(speed)

def move_to_the_right_forward(settingPins, settingDutyCycle, speed):
    global motor_A_0
    global motor_A_1
    global motor_B_0
    global motor_B_1
    global pwm_A
    global pwm_B
    if settingPins:
        GPIO.output(motor_A_0, GPIO.HIGH)
        GPIO.output(motor_A_1, GPIO.LOW)
        GPIO.output(motor_B_0, GPIO.HIGH)
        GPIO.output(motor_B_1, GPIO.LOW)
    if settingDutyCycle:
        pwm_A.ChangeDutyCycle(speed)
        pwm_B.ChangeDutyCycle(speed/2)

def move_to_the_left_backward(settingPins, settingDutyCycle, speed):
    global motor_A_0
```

```
    global motor_A_1
    global motor_B_0
    global motor_B_1
    global pwm_A
    global pwm_B
    if settingPins:
        GPIO.output(motor_A_0, GPIO.LOW)
        GPIO.output(motor_A_1, GPIO.HIGH)
        GPIO.output(motor_B_0, GPIO.LOW)
        GPIO.output(motor_B_1, GPIO.HIGH)
    if settingDutyCycle:
        pwm_A.ChangeDutyCycle(speed/2)
        pwm_B.ChangeDutyCycle(speed)

def move_to_the_right_backward(settingPins, settingDutyCycle, speed):
    global motor_A_0
    global motor_A_1
    global motor_B_0
    global motor_B_1
    global pwm_A
    global pwm_B
    if settingPins:
        GPIO.output(motor_A_0, GPIO.LOW)
        GPIO.output(motor_A_1, GPIO.HIGH)
        GPIO.output(motor_B_0, GPIO.LOW)
        GPIO.output(motor_B_1, GPIO.HIGH)
    if  settingDutyCycle:
        pwm_A.ChangeDutyCycle(speed)
        pwm_B.ChangeDutyCycle(speed/2)

def init():
    global motor_A_0
    global motor_A_1
    global motor_B_0
    global motor_B_1
    global pwm_motor_A
    global pwm_motor_B

    GPIO.setup(motor_A_0, GPIO.OUT)
    GPIO.setup(motor_A_1, GPIO.OUT)
    GPIO.setup(motor_B_0, GPIO.OUT)
    GPIO.setup(motor_B_1, GPIO.OUT)

    GPIO.setup(pwm_motor_A, GPIO.OUT)
    GPIO.setup(pwm_motor_B, GPIO.OUT)
    GPIO.setup(37, GPIO.OUT)
    GPIO.setup(35, GPIO.OUT)
    GPIO.setwarnings(False)
```

```
def clean():
    pwm_A.stop()
    pwm_B.stop()
    GPIO.cleanup()
    print("Cleanup pins..")

def stop_motors():
    global pwm_A
    global pwm_B

    pwm_A.ChangeDutyCycle(0)
    pwm_B.ChangeDutyCycle(0)

    print("Motors execution stopped")

def prepare():
    init()
    dc = 0                                  # set dc variable to 0 for 0%
    pwm_A.start(dc)                         # Start PWM with 0% duty cycle
    pwm_B.start(dc)

directions = set()

def main():
    prepare()
    # main loop of program
    # Print blank line before and after message.
    print("\nPress Ctrl C to quit \n")

    speed = 30

    try:
        while True:
            received_message = sys.stdin.readline()

            if not received_message or "esc" in received_message:
                break

            # DO SOMETHING WITH DATA
            print("Mesaj primit: " + received_message)

            command = received_message.split(':')
            key = command[0]
            state = command[1]

            if state == "pressed\n" and len(directions) <= 2:
                if not (('w' in directions and key == 's') or ('s' in directions and
key == 'w') or ('a' in directions and key == 'd') or ('d' in directions and key ==
'a')):
```

```
                directions.add(key)

            if state == "released\n" and key in directions:
                if key=="s":
                    rear_lights_off()
                directions.remove(key)

            print(directions)

            if "w" in directions and "a" in directions:
                move_to_the_left_forward(True, True, speed)

            if "w" in directions and "d" in directions:
                move_to_the_right_forward(True, True, speed)

            if "s" in directions and "a" in directions:
                move_to_the_left_backward(True, True, speed)

            if "s" in directions and "d" in directions:
                move_to_the_right_backward(True, True, speed)

            if "w" in directions and len(directions)==1:
                move_forward(True, True, speed)

            if "s" in directions and len(directions)==1:
                move_backward(True, True, speed)

            if len(directions) == 0:
                stop_motors()

    except Exception as ex:
        print(ex)
    finally:
        clean()

if __name__ == '__main__':
    main()
```

**cameraScript.py**

```
import sys
import time
from picamera import PiCamera

DEFAULT_IMAGE_LOCATION = './public/SavedImage/image.jpg'

if __name__ == '__main__':
      # start=time.time()
      #camera = PiCamera()
      with PiCamera() as camera:
```

```
            camera.resolution = (640, 480)
            camera.framerate = 80
            location = DEFAULT_IMAGE_LOCATION
            if len(sys.argv) > 1:
                  location = sys.argv[1]

            camera.capture(location)
      #end=time.time()
      #print(end-start)
```

**leds.py**

```
import RPi.GPIO as GPIO
from time import sleep
import threading
import sys

GPIO.setmode(GPIO.BOARD)
GPIO.setup(3, GPIO.OUT)
GPIO.setup(5, GPIO.OUT)
GPIO.setup(7, GPIO.OUT)
GPIO.setup(19, GPIO.OUT)
GPIO.setup(21, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)
GPIO.setup(37, GPIO.OUT)
GPIO.setup(35, GPIO.OUT)
GPIO.setwarnings(False)

def rear_lights_on():
    GPIO.output(37, 1)
    GPIO.output(35, 1)

def rear_lights_off():
    GPIO.output(37, 0)
    GPIO.output(35, 0)

def leds_on_right(stop):
    while True:
        for i in [7, 5, 3]:
            GPIO.output(i, 1)
            sleep(0.3)
        GPIO.setwarnings(False)
        for j in [7, 5, 3]:
            GPIO.output(j, 0)
        GPIO.setwarnings(False)
        sleep(0.3)
        if stop():
            break


def leds_on_left(stop):
    while True:
        for i in [19, 21, 23]:
```

```
            GPIO.output(i, 1)
            sleep(0.3)
        GPIO.setwarnings(False)
        for j in [19, 21, 23]:
            GPIO.output(j, 0)
        GPIO.setwarnings(False)
        sleep(0.3)
        if stop():
            break

def emergency_leds():
    while True:
        for i in [(7, 19), (5, 21), (3, 23)]:
            GPIO.output(i[0], 1)
            GPIO.output(i[1], 1)
            sleep(0.3)
        GPIO.setwarnings(False)
        for j in [(7, 19), (5, 21), (3, 23)]:
            GPIO.output(j[0], 0)
            GPIO.output(j[1], 0)
        GPIO.setwarnings(False)
        sleep(0.3)

def leds_on():
    for i in [(7, 19), (5, 21), (3, 23)]:
        GPIO.output(i[0], 1)
        GPIO.output(i[1], 1)
    GPIO.setwarnings(False)

def leds_off():
    for i in [(7, 19), (5, 21), (3, 23)]:
        GPIO.output(i[0], 0)
        GPIO.output(i[1], 0)
    GPIO.setwarnings(False)

def leds_off_left():
    for i in [7, 5, 3, ]:
        GPIO.output(i, 0)
    GPIO.setwarnings(False)
def leds_off_right():
    for i in [19, 21, 23]:
        GPIO.output(i, 0)
    GPIO.setwarnings(False)

def main():
    # main loop of program
    # Print blank line before and after message.
    print("\nPress Ctrl C to quit \n")
    thread_left_state = False
    thread_right_state = False

    thread_left = None
    thread_right = None
```

```
    try:
        while True:
            received_message = sys.stdin.readline()

            sys.stderr.write(received_message)
            if not received_message or "esc" in received_message:
                break

            # DO SOMETHING WITH DATA
            print("Mesaj primit: " + received_message)
            # received_message.strip()
            command = received_message.split(':')
            key = command[0]
            state = command[1]
            if key == "daytimeLights":
                if state == "on\n":
                    leds_on()
                elif state == "off\n":
                    leds_off()

            if key == "brakeLights":
                if state == "on\n":
                    rear_lights_on()
                elif state == "off\n":
                    rear_lights_off()

            if key == "leftSignal":
                if state == "on\n":
                    thread_left_state=False
                    thread_left = threading.Thread(target = leds_on_left, args
=(lambda : thread_left_state, ))
                    thread_left.start()
                elif state == "off\n":
                    thread_left_state=True
                    thread_left.join()
                    leds_off_left()

            if key == "rightSignal":
                if state == "on\n":
                    thread_right_state=False
                    thread_right = threading.Thread(target = leds_on_right, args
=(lambda: thread_right_state, ))
                    thread_right.start()
                elif state == "off\n":
                    thread_right_state=True
                    thread_right.join()
                    leds_off_right()

    except Exception as ex:
        print(ex)
    finally:
        thread_right_state=True
```

```
        thread_left_state=True
        leds_off()
        rear_lights_off()

if __name__ == '__main__':
    main()
```

## app.js (web server)

```
const express = require('express');
const expressLayouts = require('express-ejs-layouts');
const bodyParser = require('body-parser')
const child_process = require('child_process');
const app = express();
var dir = process.cwd();

const port = 1235;
const fs = require('fs');
const { Console } = require('console');
const { mainModule } = require('process');
const http = require('http');
const server = http.createServer(app);
const { Server } = require("socket.io");
const io = new Server(server);
var net = require('net');

var HOST = '192.168.100.47';
var PORT = 1234;
var clientPi = new net.Socket();

app.set('view engine', 'ejs');

app.use(expressLayouts);

app.use(express.static('public'))

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

app.get('/favicon.ico', (req, res) => {
      res.sendFile("/home/pi/AutonomousCar/public/images/favicon.ico");
});

app.get('/home', (req, res) => {
      res.render('home', {
            title: "Home",

      })
});

function GetCurrentLogTime() {
      var today = new Date();
```

```
	var date = today.getDate() + '-' + (today.getMonth() + 1) + '-' +
today.getFullYear();
	var time = today.getHours() + ":" + today.getMinutes() + ":" +
today.getSeconds();
	var dateTime = '[' + date + ' ' + time + ']';

	return dateTime
}

var carMovementProcess = null;
var carLightsProcess = null;
io.on('connection', (socket) => {
	console.log('A user connected');
	socket.send(GetCurrentLogTime() + " " + "Connection successfully")

	socket.on('lights', (msg) => {
		if (carLightsProcess != null) {
			carLightsProcess.stdin.write(msg);
			socket.send(GetCurrentLogTime() + " " + msg)
			if(msg=="esc")
				carLightsProcess =null
		}
		else{
			console.log("lightsProcess doesn't exist");
		}

	});

	socket.on('commands', (msg) => {
		var command = msg.split(':')
		console.log('key: ' + command[0] + '=> command: ' + command[1]);
		if (command[0] == "start") {
			carMovementProcess = child_process.spawn('python3',
['./PythonCode/functions.py']);
			carMovementProcess.stdin.setEncoding('utf-8');


			carMovementProcess.stdout.on('data', function (data) {
				console.log(`stdout:${data}`);
				//dataToSend = data.toString();
			});

			carMovementProcess.stderr.on('data', function (data) {
				console.log(`stderr:${data}`);
				//dataToSend += data.toString();
			});

			carMovementProcess.on('close', (code) => {
				console.log(`movement process close all stdio with code
${code}`);
				//console.log(data);
			});
```

```
                carLightsProcess = child_process.spawn('python3',
['./PythonCode/leds.py']);
                carLightsProcess.stdin.setEncoding('utf-8');

                carLightsProcess.stdout.on('data', function (data) {
                        console.log(`stdout:${data}`);
                        //dataToSend = data.toString();
                });

                carLightsProcess.stderr.on('data', function (data) {
                        console.log(`stderr:${data}`);
                        //dataToSend += data.toString();
                });

                carLightsProcess.on('close', (code) => {
                        console.log(`lights process close all stdio with code
${code}`);
                        //console.log(data);
                });

                socket.send(GetCurrentLogTime() + " " + "Python script
running...")
            }
            console.log(command[1] == "pressed" && carMovementProcess != null)
            if (command[1] == "pressed" && carMovementProcess != null) {

                if (command[0] == "w") {
                        carMovementProcess.stdin.write("w:pressed\n");
                        socket.send(GetCurrentLogTime() + " " + "Forward pressed")
                }
                else if (command[0] == "a") {
                        carMovementProcess.stdin.write("w:pressed\n");
                        carMovementProcess.stdin.write("a:pressed\n");
                        socket.send(GetCurrentLogTime() + " " + "Left pressed")
                } else if (command[0] == "s") {
                        carMovementProcess.stdin.write("s:pressed\n");
                        socket.send(GetCurrentLogTime() + " " + "Back pressed")
                } else if (command[0] == "d") {
                        carMovementProcess.stdin.write("w:pressed\n");
                        carMovementProcess.stdin.write("d:pressed\n");
                        socket.send(GetCurrentLogTime() + " " + "Right pressed")
                }
            }
            else if (command[1] == "released" && carMovementProcess != null) {

                if (command[0] == "w") {
                        carMovementProcess.stdin.write("w:released\n")
                        socket.send(GetCurrentLogTime() + " " + "Forward
released")
                } else if (command[0] == "a") {
                        carMovementProcess.stdin.write("a:released\n");
                        carMovementProcess.stdin.write("w:released\n");
                        socket.send(GetCurrentLogTime() + " " + "Left released")
```

```
                } else if (command[0] == "s") {
                        carMovementProcess.stdin.write("s:released\n");
                        socket.send(GetCurrentLogTime() + " " + "Back released")
                } else if (command[0] == "d") {
                        carMovementProcess.stdin.write("w:released\n");
                        carMovementProcess.stdin.write("d:released\n");
                        socket.send(GetCurrentLogTime() + " " + "Right released")
                }
        }
        if (command[0] == "stop") {
                if(carMovementProcess != null)
                {
                        carMovementProcess.stdin.write("esc");
                        carMovementProcess=null
                        socket.send(GetCurrentLogTime() + " " + "Turning off...")
                }
        }

    });
    socket.on('disconnect', () => {
        console.log('User disconnected');
    });
});

app.get('/live-feed', (req, res) => {

    var cameraPython = child_process.spawn('python3',
['./PythonCode/cameraScript.py']);
    cameraPython.stdout.on('data', function (data) {
        console.log(`stdout:${data}`);

    });

    cameraPython.stderr.on('data', function (data) {
        console.log(`stderr:${data}`);

    });

    cameraPython.on('close', (code) => {
        console.log(`child process close all stdio with code ${code}`);
        res.writeHead(200, { 'Content-Type': 'text/txt' });
        res.end();
    });
});

server.listen(port, () => console.log(`Serverul rulează la adresa
http://localhost:${port}/home`));
```

## VIII Contributions

*Radu-Andrei Budeanu:*

- ✓ Soldering, assembling the car, brake lights.

*Cosmin-Constantin Cojocaru:*

- ✓ Python scripts that receive commands and controls the movement of the car.

*Raul Andrei Dima:*

- ✓ NodeJS web server and client, camera stream and lights python scripts.

*Alexandru-Gicu Melinte:*

- ✓ Remote-client interface, documentation, schematics.