Autonomous Remote-Control Car

Authors – GRAMENI Alexia, MIHAI Rareș, ȚÎȘTEA Ștefan

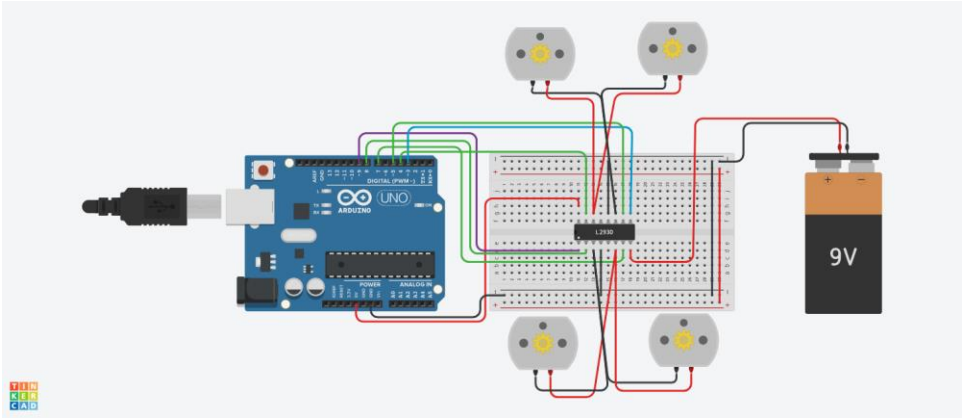
1. Introduction

The landscape of autonomous technologies is rapidly evolving, with a growing emphasis on making these innovations accessible for educational purposes. In our project, we introduce a small-scale autonomous RC car equipped with a camera module, strategically employing an Arduino Uno board as a user-friendly and educational tool. This approach not only contributes to the practical understanding of autonomous systems but also highlights the significance of democratizing access to cutting-edge technologies for enthusiasts and students alike.

Central to our project is the integration of a camera module that employs a learning visual detection algorithm for real-time road view analysis. The integration of a learning algorithm adds a layer of adaptability to our RC car's decision-making process, allowing it to evolve and improve its performance over time based on experience. Through the camera's lens, our system interprets and processes the surrounding environment, enabling the RC car to recognize and respond to traffic signs in real time.

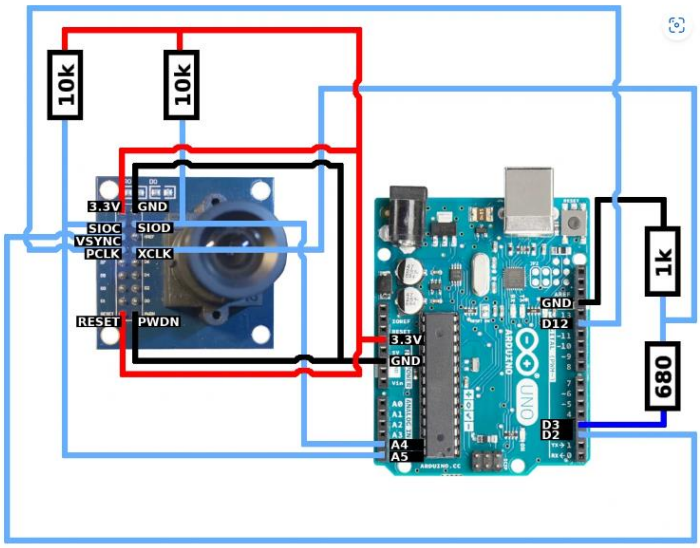
2. Hardware

Fig 1



This simulation (Fig 1) serves as a crucial step in designing the electrical circuit for our autonomous RC car. It illustrates the connections between key components, including the Arduino Uno R3 microcontroller, an L293D motor driver, four DC motors, and the battery. The Arduino Uno R3, acting as the central processing unit, manipulates input signals to regulate the vehicle's movements. The L293D motor driver facilitates control over the speed and direction of the DC motors based on signals from the Arduino Uno, ensuring precise movement control. Power is supplied by two 3.7V batteries, chosen for efficient energy consumption and reliable operation.

Fig 2



Additionally, the camera module (Fig 2) plays the most important role in our project. Mounted on the car's roof, it captures real-time images for traffic sign detection. These images are then processed by the Arduino Uno R3 microcontroller, which incorporates advanced algorithms for object detection and recognition. The camera module's ability to adapt to new traffic signs enhances it's utility in creating an autonomous RC car capable of navigating through complex environments.

Table 1

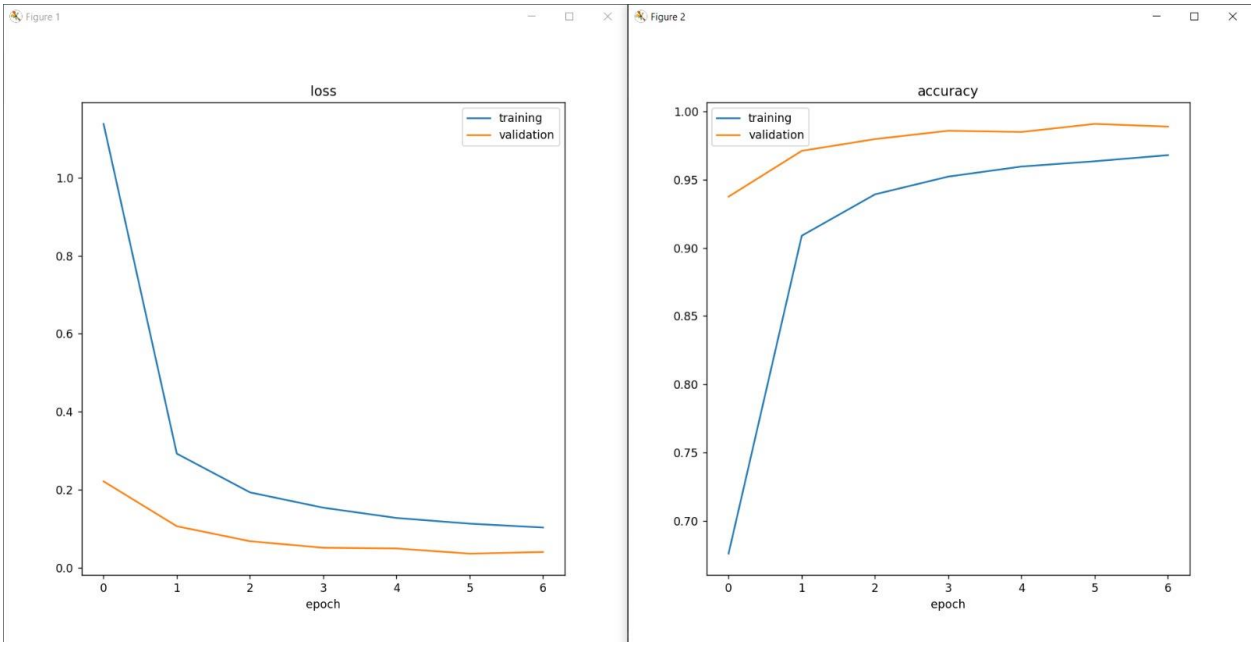
Quantity and price of the used components

Component	Quantity	Price (ron)
Arduino Uno R3 microcontroller board	1	30.76
Arduino L293D motor driver shield	1	15.46
Motor DC 3V-6V 7000 RPM	4	35.96
Camera module	1	20.83
3.7V batteries	2	59.14

3. Software

3.1 Python CNN

```python
import numpy as np
import cv2
import os
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
#from keras.utils.np_utils import to_categorical
from keras.utils import to_categorical
import tensorflow as tf
import pickle
```



This code implements a Convolutional Neural Network (CNN) for image classification using TensorFlow and Keras.The dataset is organized in folders, each representing a different class.

Libraries Used:

- NumPy: For numerical operations.
- OpenCV (cv2): For image processing.
- os: For interacting with the operating system.
- scikit-learn: For data splitting.
- Matplotlib: For data visualization.
- ImageDataGenerator: For data augmentation.

- TensorFlow and Keras: For building and training the CNN.

Dataset Preparation:
Images are loaded from the specified path and resized to (32, 32, 3).
Training and testing data are split using train_test_split.
Data is preprocessed by converting to grayscale, equalizing histogram, and normalizing pixel values.

Data Exploration:
Number of images in each class is visualized using a bar chart.

Data Augmentation:
ImageDataGenerator is used to perform data augmentation, including shifts, zoom, shear, and rotation.

Model Architecture:
The CNN model consists of convolutional layers, max-pooling, dropout, and fully connected layers.
Model summary is printed.

Model Training:
The model is trained using the training data with specified batch size, epochs, and steps per epoch.
Validation data is used for model evaluation during training.

Training Visualization:
Loss and accuracy curves are plotted for both training and validation sets.

Model Evaluation:
The trained model is evaluated on the test set, and the test score and accuracy are printed.

Model Saving:
The trained model is serialized using pickle and saved as "model_trained.p".

Conclusion:
The CNN is successfully trained for image classification, demonstrating good performance on the test set.
This code serves as a comprehensive example of building, training, and saving a CNN for image classification using TensorFlow and Keras.

3.2 Arduino Motor Test

Introduction:
The following Arduino code is a test script designed to validate the functionality of a robot's motor.
The code is intended for initial testing to ensure the robot can move forward, backward, and stop.

Libraries Used:
AFMotor.h: Library for controlling DC motors and stepper motors.
cppCopy code

```
#include <Servo.h>
#include <AFMotor.h>
```

Components Used:
Servo Motor (servo1)
DC Motors (M1, M2, M3, M4)
Stepper Motor (motor1)

Setup:
Motor speeds are set for DC motors M1 to M4.
cppCopy code

```
void setup() {
  M1.setSpeed(350);
  M2.setSpeed(350);
  M3.setSpeed(350);
  M4.setSpeed(350);
}
```

Loop:
The loop() function calls the gearmotor() function to execute the test sequence repeatedly.
cppCopy code

```
void loop() {
  gearmotor();
}
```

Motor Movement Test:
The gearmotor() function runs all four DC motors in the forward direction for 2 seconds.
Then, it runs them in the backward direction for 2 seconds.
Finally, it releases all motors and adds a delay of 500 milliseconds.
cppCopy code

```
void gearmotor() {
  M1.run(FORWARD);
  M2.run(FORWARD);
  M3.run(FORWARD);
  M4.run(FORWARD);
  delay(2000);
  M1.run(BACKWARD);
  M2.run(BACKWARD);
  M3.run(BACKWARD);
  M4.run(BACKWARD);
  delay(2000);
```

```
  M1.run(RELEASE);
  M2.run(RELEASE);
  M3.run(RELEASE);
  M4.run(RELEASE);
  delay(500);
}
```

Conclusion:
This test code is designed to verify the basic functionality of the robot's motor system.
Motor speeds and directions can be adjusted based on the robot's design and requirements.

3.3 Arduino Camera Test Code

Introduction:
The following Arduino code serves as a test script to verify the functionality of an OV7670 camera module. The code initializes the camera, captures an image, and displays the image on the Serial Monitor for verification.

Libraries Used:
Wire.h: Library for I2C communication.
Adafruit_OV7670.h: Library for interfacing with the OV7670 camera module.
cppCopy code

```
#include <Wire.h>
#include "Adafruit_OV7670.h"
// Adafruit_OV7670 camera;
```

Setup:
The code sets up the Serial communication and initializes the OV7670 camera.
If initialization fails, an error message is printed, and the program enters an infinite loop.
cppCopy code

```
void setup() {
  Serial.begin(115200);

  // Initialize OV7670 camera
  if (camera.begin() != OV7670_STATUS_OK) {
    Serial.println("Error initializing OV7670 camera!");
    while (1);
  }

  // Set the camera resolution (change if needed)
  camera.setSize(OV7670_SIZE_DIV4);
```

```
 // Capture an image
 camera.capture();

 // Display captured image on Serial Monitor
 displayImageOnSerialMonitor();
}
```

Display Image on Serial Monitor:

The displayImageOnSerialMonitor() function retrieves the image buffer from the camera and prints the captured image in hexadecimal format on the Serial Monitor.

The image width and height are used to iterate through the image buffer.

cppCopy code

```
void displayImageOnSerialMonitor() {
 uint16_t *imageBuffer = camera.getBuffer();
 uint16_t imageWidth = camera.width();
 uint16_t imageHeight = camera.height();

 Serial.println("Captured Image:");

 for (uint16_t y = 0; y < imageHeight; y++) {
  for (uint16_t x = 0; x < imageWidth; x++) {
   Serial.print(imageBuffer[y * imageWidth + x], HEX);
   Serial.print(' ');
  }
  Serial.println();
 }

 Serial.println("Image display complete.");
}
```

Conclusion:

This test code is essential for confirming the proper functioning of the OV7670 camera module.

Adjustments to the camera configuration, such as resolution, can be made based on specific requirements.