


# STOP Sign Detection Using Python Programming

Sîrbu Cătălina, Macovei Dragoș, Rusu Dan Andrei,  
Grigore Alexandru, and Bogdan Grănescu<sup>(✉)</sup> 

University Politehnica of Bucharest, 313, Splaiul Independenței,  
060042 Bucharest, Romania  
bogdan.gramescu@upb.ro

**Abstract.** Autonomous driving systems are constantly evolving, in order to increase safety. One of the key elements is the recognition of traffic signs. The paper presents a solution in which a small scale demonstrator car is able to recognize the stop signs met on the road using Python libraries like OpenCV and NumPy in order to perform colors operations.

**Keywords:** Autonomous driving · Computer Vision · Traffic sign

## 1 Introduction

The autonomous car could be defined like a vehicle able of sensing its environment and operating without human involvement. A human passenger can is not required to take control of the vehicle at any time. He is not required to be present in the vehicle at all. An autonomous vehicle is able to go anywhere a traditional car goes [1].

The theme of the Bosch Future Mobility Challenge [2] proposes exactly this aspect only on a car (see Fig. 1) at a smaller scale (1:10). The ideal of the competition is the implementation of ideas that mimic real traffic situations like maintaining between lanes, parallel parking, recognition of pedestrians and traffic signs, bypassing obstacles, and reaction to traffic lights. The car has the following components: Raspberry Pi 4 board [3], Nucleo F401RE controller [4], Pi Camera [3], motor driver, LiPo Battery, servomotor, housing, chassis, and DC/DC converters for supplying the components.

Further, we will approach the image recognition process for the stop sign. The rest of the commands have already been implemented.



**Fig. 1.** Small scale demonstrator car (1:10), with housing (left) and without housing (right).

## 2 Fields of Use

Considering the high number of cars on the street, different problems appeared, such as decreasing the level of walkability and livability, traffic congestion, transportation costs (fuel, infrastructure), crowded or insufficient parking spaces, increasing urban CO<sub>2</sub> emissions.

In order to reduce and eliminate this type of problem, in the automotive industry have been created several levels of driving automation from Level 0 - No automation (Manual Control) to Level 5 - Full automation (No human interaction required). At this moment, fully autonomous cars are not available to the general public, but they are continuously tested in specialized centers.

Increasing the safety of autonomous cars depends on the level of intelligence that these mechatronic systems integrate. A key element is the recognition of traffic signs.

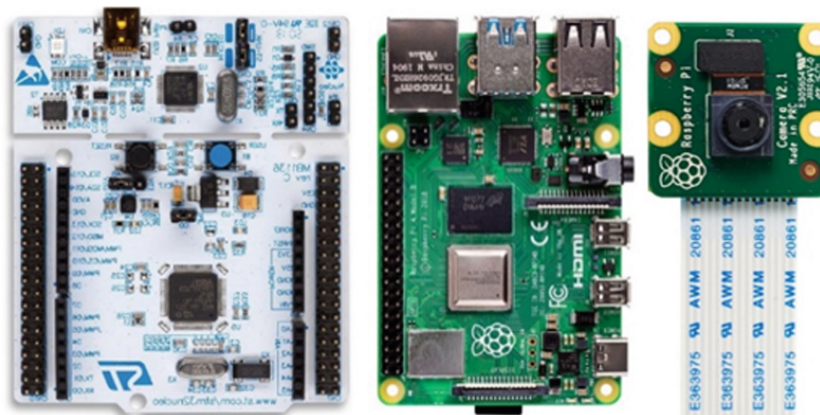
## 3 Analysis of Existing Solutions

The digital branch of photography became commercially available in 1990. Later, the issue of how computers can gain information from digital images or videos arose.

There are various online applications with which it is possible to detect objects, shapes, colors, textures, whether it is day or night and so on. Some of those applications are based on concepts such as Computer Vision [5], Machine Learning and Artificial Intelligence.

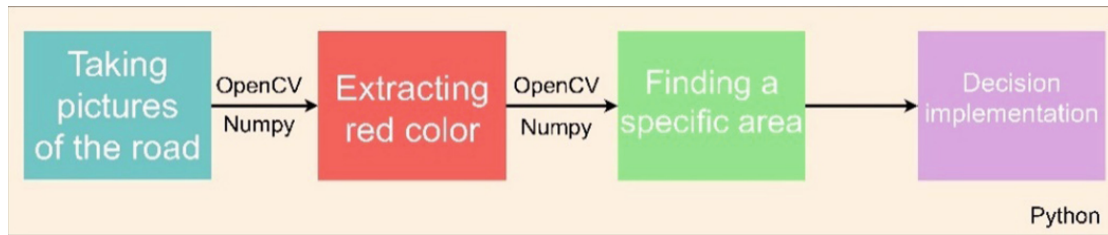
In a real case in which the driver meets the stop sign, he processes the information, realizes the meaning of the traffic sign, and makes a decision. The goal of computer vision is to succeed in accomplishing the same task at the same time or faster. These algorithms are implemented on computers.

In addition to the software part, the hardware part is also required. A real-life autonomous car needs a radar, a lidar, several cameras but the competition car consists of a motor controller, an image processing controller, a compatible camera (see Fig. 2).



**Fig. 2.** Hardware required: Nucleo F401RE (left), Raspberry Pi 4 model B (middle), Pi Camera (right)

## 4 Own Solution



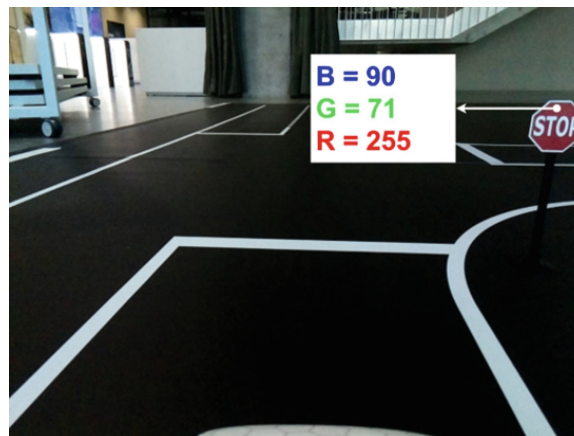
**Fig. 3.** Block diagram

Video processing is performed using the NumPy (1.18.1 version) and OpenCV (4.2.0 version) software modules, these being programmed in Python (3.8.1 version) (Fig. 3).

```
import cv2
import numpy as np
```

To get started, this program uses the Raspberry Pi camera to take pictures of the route in real time. To recognize the red color, we need a range of BGR colors (minimum and maximum values) because the shade of red may differ depending on the light intensity. We used a special function to transform the color space from BGR to HSV. Thus, we positioned the sign at different angles to see all the possibilities of values for the interval. This implementation gives us a wider range for sign recognition. An example of a group of BGR values can be seen in Fig. 4. To exemplify the implementation, we used a sample image with the stop sign on the road, more precisely from the intersection.

```
hsv_frame = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
low_red = np.array([76, 53, 228])
high_red = np.array([114, 92, 255])
```



**Fig. 4.** BGR values for red

The next step is to add a black mask over the original image to highlight only the color range to be processed (see Fig. 5).

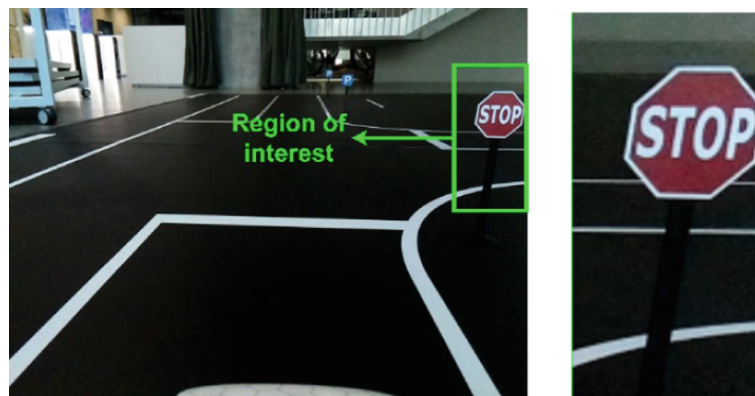
```
red_mask = cv2.inRange(hsv_frame, low_red, high_red)
red = cv2.bitwise_and(img, img, mask = red_mask)
```



**Fig. 5.** Black mask

Given the fact that we can meet other red objects that fall within the chosen range, we have implemented a region of interest (ROI) (see Fig. 5) on the right side of the image because in the case of the contest only there can be the stop sign. For this implementation, the width and length of the image must be known. For sketching the ROI, the upper left corner is represented by the coordinates (x1, y1) and the lower right corner is represented by the coordinates (x2, y2). In order to be able to use these coordinates, they must be placed in a certain order as follows:

```
ROI = img[y1:y2, x1:x2]
```



**Fig. 6.** Region of interest

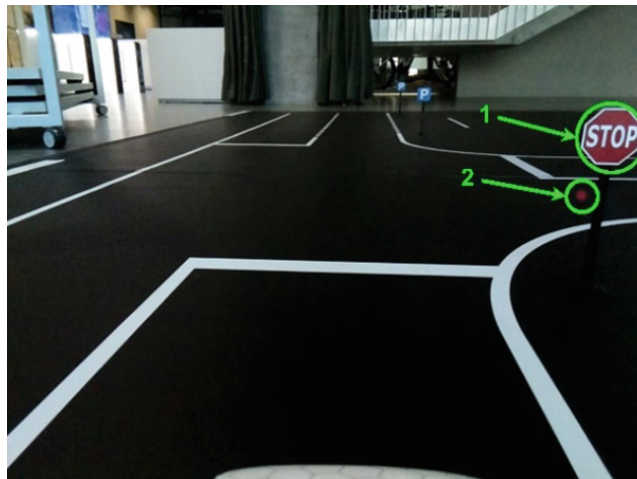
Another precaution to avoid confusion with other objects of the same color is to check the area of our color. The stop sign is a fixed object that does not change its size, so its area will remain the same in most cases, obviously excluding situations in which it is slightly inclined. To calculate the area, we first need the contour of the object. To

do this, *moments* is used because it skips searching in the contours array, computes the area based on moments and it is much faster.

```
Contours = cv2.findContours(filter, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE) ...
moments = cv2.moments(contours[0], True)
area = moments["m00"]
```

Given that the image is dynamic, it is necessary to have a working interval because the fixed values are not suitable for any situation encountered on the route. As can be seen in Fig. 6, zone 1 represents a red area with a value that falls within the chosen range and in zone 2 there is a much smaller area than the ideal one (Fig. 7).

```
if lowerAreaThreshold < area < upperAreaThreshold:
print("Area checked")
```



**Fig. 7.** Ideal (1) and small (2) red areas

When the object encountered has a certain color, a certain area and it is in the region of interest chosen earlier, it means that we have met the stop sign and further we have to give the command to the engine to make the decision to stop.

All this processing is possible thanks to the Raspberry Pi 4, a low cost, credit card sized computer, that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. This little device enables people to explore computing, and to learn how to program in languages like Scratch and Python. It is capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games [3].

Through serial communication the Raspberry PI sends the processed information to the Nucleo. This board is a microcontroller, and in this situation a personal library is used. With its help you can change the speed and angle of the wheels through the

servomotor. It also sends the command to the motor via the PWM signal and a motor driver containing the H-bridge.

```
import SerialHandler
SerialHandler.driveWithConstPower(2000)
SerialHandler.setAngle(f.normalize(float(speed)))
```

## 5 Conclusion

Although this project took place around a small-scale demonstrator car, the ideas can be implemented and used in real life.

In carrying out this project, all the measures that make the working method and material resources more efficient were taken into account.

Even if only the recognition of the stop sign was presented, the algorithm can be implemented for the recognition of parking signs, pedestrian crossings, priority roads and much more.

## References

1. Synopsys Page. <https://www.synopsys.com/automotive/what-is-autonomous-car.html>. Accessed 28 May 2020
2. Bosch Future Mobility Challenge Homepage. <https://www.boschfuturemobility.com/>. Accessed 28 May 2020
3. Raspberry PI Product Page. <https://www.raspberrypi.org/>. Accessed 28 May 2020
4. STM32 Product Page. [https://www.st.com/content/st\\_com/en.html](https://www.st.com/content/st_com/en.html). Accessed 28 May 2020
5. Computer Vision Wikipedia Page. [https://en.wikipedia.org/wiki/Computer\\_vision](https://en.wikipedia.org/wiki/Computer_vision). Accessed 28 May 2020