Self-Driving Car Model based on the Image Processing Algorithm

Mukhit Yelemes, Azamat Kaibaldiyev
Departments of Robotics and Mechatronics, and Computer Science
School of Science and Technology
Nazarbayev University
Nur-Sultan, Kazakhstan
azamat.kaibaldiyev@nu.edu.kz
mukhit.yelemes.nu.edu.kz

***Abstract* — The paper proposes navigation and obstacle avoidance methods based on bitwise and technique. The RC car was used as the main vehicle for testing with a video streaming smartphone mounted on it. The computer processed the data and via Arduino sent commands to RC controller. Results of testing the car in the artificially constructed environment by A4 papers demonstrated the satisfactory performance of the proposed algorithm.**

Keywords *– Self-driving car, Raspberry Pi, Arduino, Image Processing*

I. Introduction

The current state of self-driving car technology is on the verge of a breakthrough. More and more companies are getting involved in the design of their own self-driving car concept: Tesla, Renault, Uber, Lyft, Google, Apple, Yandex, Nvidia and even Nazarbayev University in cooperation with KAMAZ and VistRobotics.

Therefore, there have been a lot of projects related to creating their own self-driving car models. For example, authors in [1], for his project of the self-driving car, use ultrasonic sensors along with the camera for obstacle, signs and line detection and following. Furthermore, they apply a neural network for decision making of the car's behavior. The Raspberry Pi sends data from camera and ultrasonic sensor to the computer via Wi-Fi. The computer receives the data, processes it using OpenCV and neural network, and sends to Arduino which receives the instructions and controls the car movement. Furthermore, in [2], the authors also use RC car, Arduino and Raspberry Pi for building their testing car model and apply convolutional neural network for vehicle navigation.

This paper proposes bitwise and method both for line following and obstacle maneuvering without the implementation of the neural network.

II. Setup

The RC car was taken as the main vehicle for design and testing. The smartphone with a camera was mounted on an RC car (Figure 1) for streaming the video to the computer.

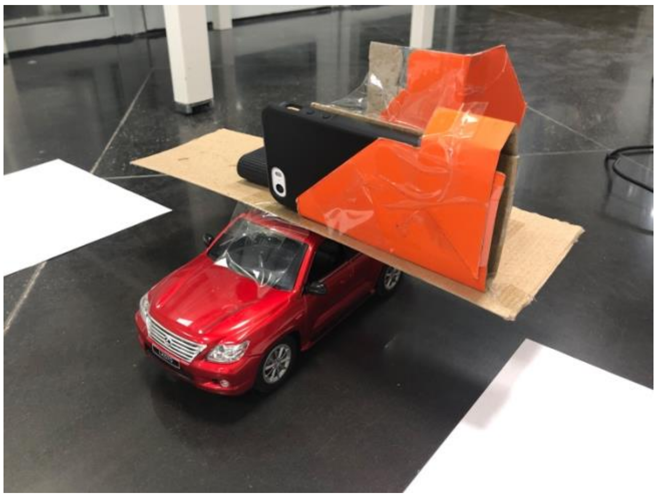


Fig.1: Smartphone with a camera mounted on an RC car

The computer connected Arduino was connected to a controller of RC car through breadboard and transistors. (Figure 2)

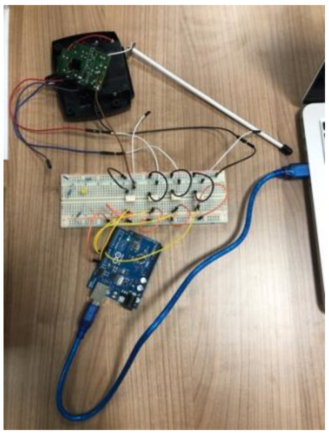


Fig.2: The Arduino - controller connection setup

The video was streamed via smartphone to the computer by using the streaming app EpocCam. The output video was exhibited as a window on the monitor. The python took that window as an input for processing and sent it to Arduino. The input and output of processed video were displayed on the monitor for observation. Arduino based on that input sends commands to RC controller thus defining the movement of the RC car. The proposed model is shown in figure 3.

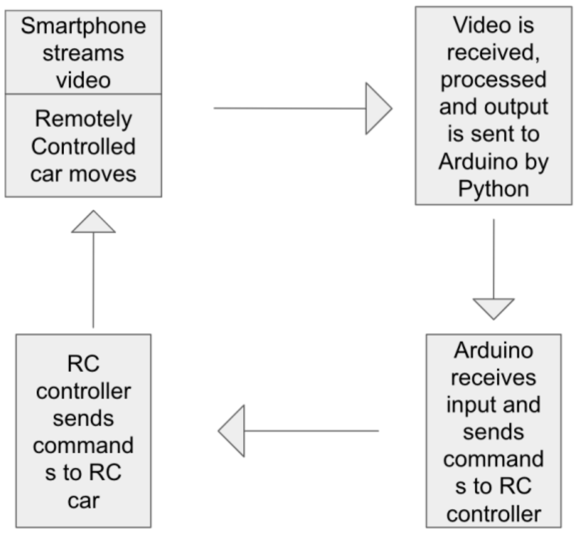

Smartphone streams video
Remotely Controlled car moves
Video is received, processed and output is sent to Arduino by Python
Arduino receives input and sends command s to RC controller
RC controller sends command s to RC car

Fig.3: The proposed setup model

III. Approach

A. Pre-processing

The initial image from the camera was too complex and included a lot of unnecessary details.

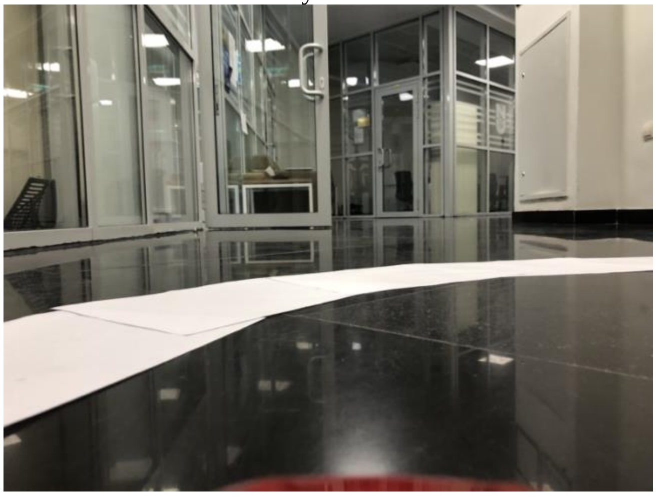


Fig.4: Original image from camera

For example, the road terrain, the object around and near the road. Therefore, it was decided to crop the image vertically, so that the part close to the road horizon in front of the car will be at the top. Also, the image was transformed from RGB to Grayscale, as individual colours were not important and intensity value was the main priority.

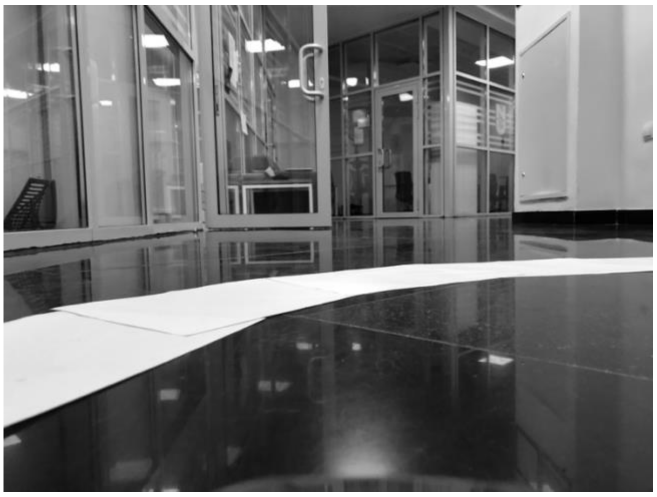


Fig.5: Original image converted to grayscale

In order to simplify the process of road edges detection, there were used white papers on both sides of the road. It also allowed to extract them by using thresholding, as the white paper will have very high-intensity value in the image. Also, the test location was changed to a place with a very dark floor, to increase the contrast with the white road edges. The intensity value for threshold was selected by testing as a one which saved almost all road edges but removed any other features.

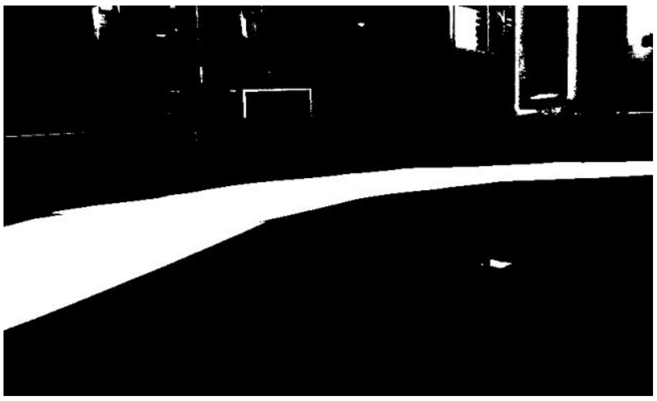


Fig.6: The result of thresholding with intensity value of 175

Nevertheless, there remained some silhouettes of objects around after thresholding. Moreover, a shadow from one paper could fall on part of another paper and create a low-intensity region. On the image after thresholding, it was seen as gaps among road edges. To solve this, there was used morphological operations. Actually, the image was processed with erosion, followed by several dilations, both with 3*3 box filter. This process also made the edges smoother.

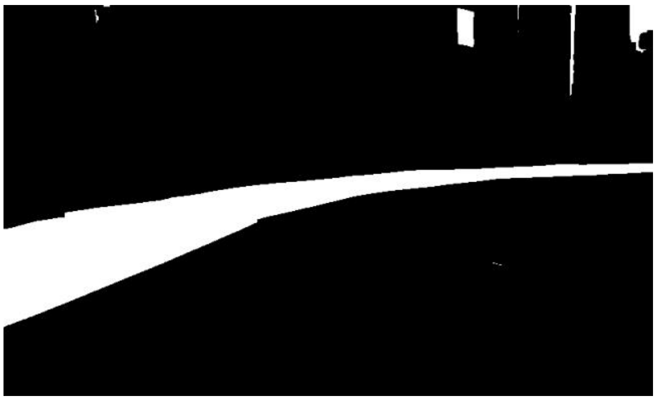


Fig.7: Result of applying erosion 1 time with 3*3 box filter

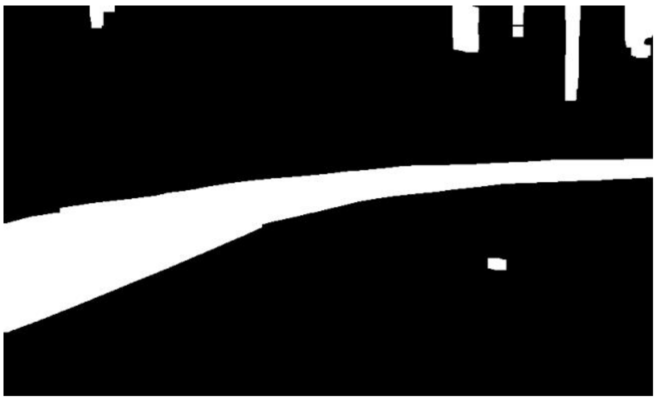


Fig.8: Result of applying dilation 4 times with 3*3 box filter

However, during the testing, there was found a more serious. The light from ceiling lighting was reflected from the tiled floor and seen in the image as a white region on the road. There was no way to remove them by thresholding because its intensity was close to the paper's intensity value. Here comes the next step of pre-processing, which is finding connected regions. The binary image after thresholding contained at most two regions of reading edges and other ones were unwanted. So, after the identification of different connected regions, they were filtered by the number of pixels they include. By other several tests, it was found that 3000 pixels are the required threshold for that. The regions with smaller pixel amount were removed by the creation of a special mask with opposite pixel values for those regions.

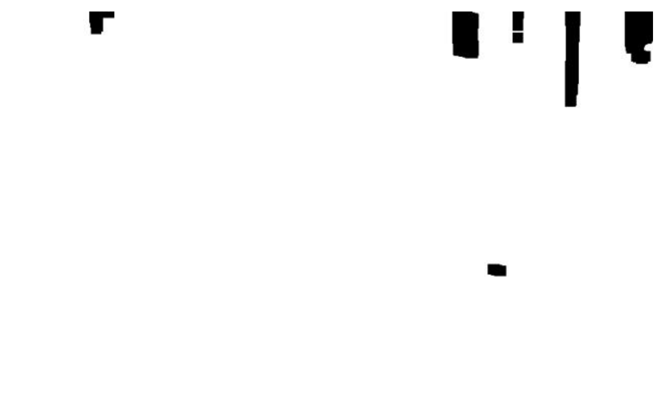


Fig.9: The resultant mask, that contains small connected regions as black contours

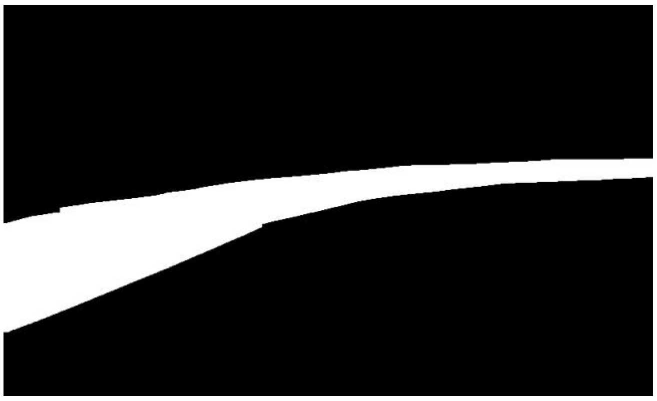


Fig.10: The final image after pre-processing with removed small regions

The image and mask were passed through "bitwise and" operation, which computes "and" of corresponding pixels on two images.

B. Image-processing

The main idea of navigation is based on the usage of masks for movement in different directions. Actually, there are 5 masks, 1 for forward, 2 for a turn to left and 2 to right.

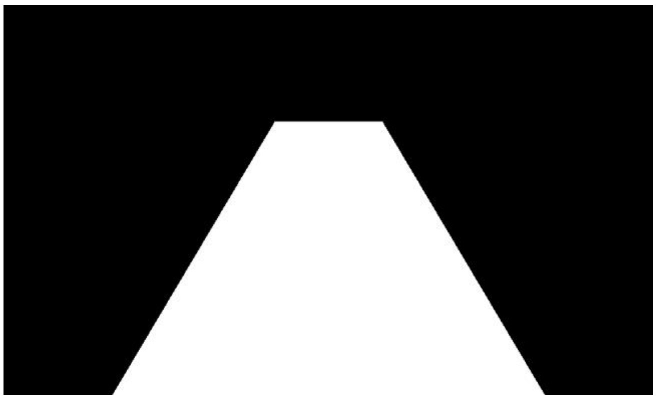


Fig.11: Forward movement mask

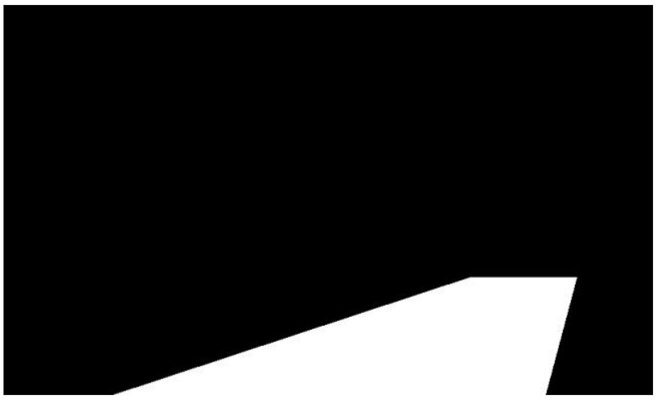


Fig.12: Turn right mask

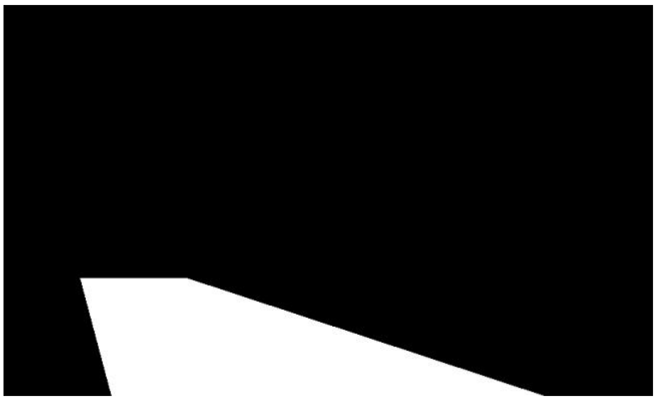


Fig.13: Turn left mask

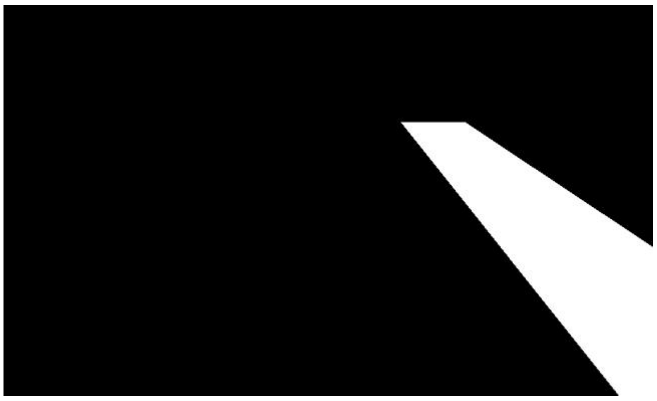


Fig.14: Right line mask

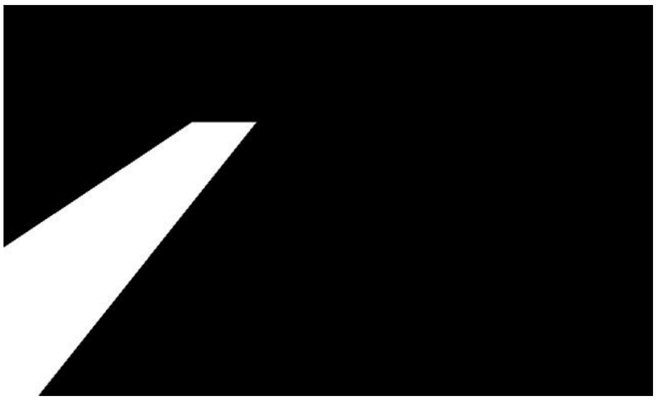


Fig.15: Left lane mask

The goal of these masks is to check if there are some obstacles on the way or if the car is going to get off the road. For each frame obtained from the pre-processing stage, there will be executed “bitwise and” with each mask. Then there will be counted the number of non-zero pixels on resulting image. If it is equal or very close to 0, then there is no problem with moving to the direction of the corresponding mask.

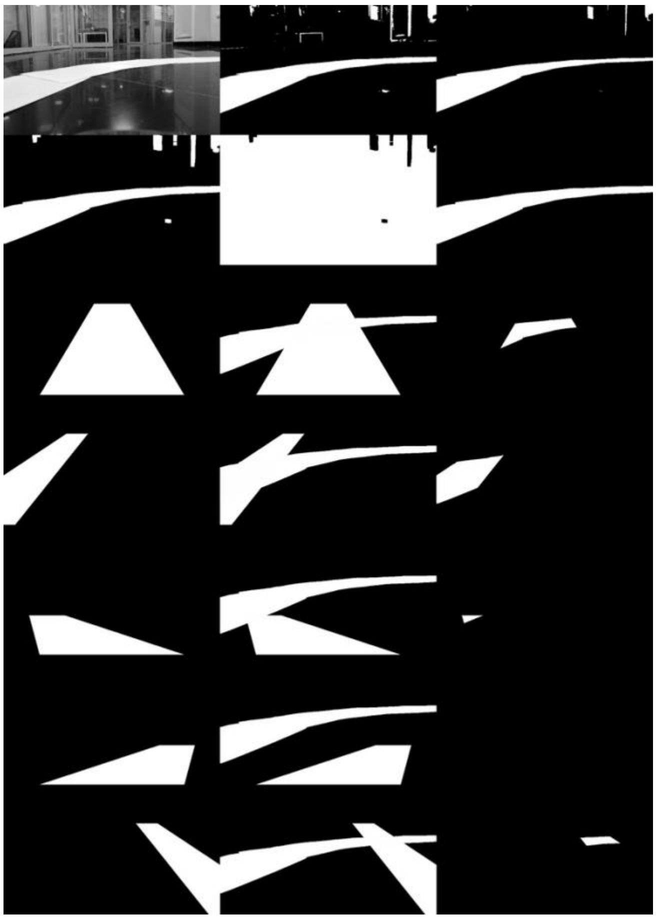


Fig.16: The collage of all images

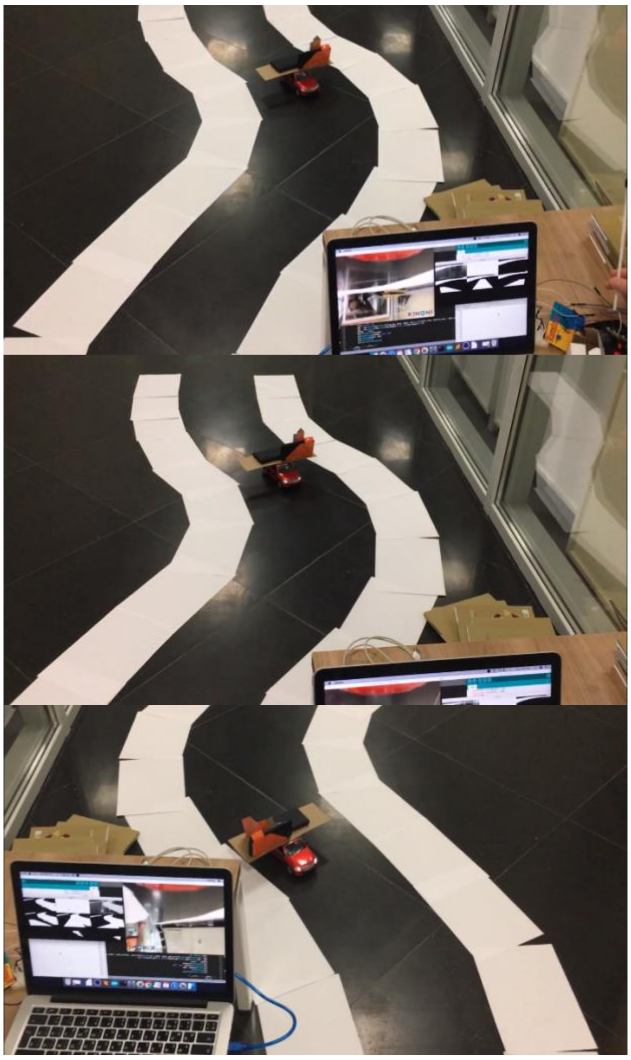
Next, the algorithm will be explained step-by-step.
First of all, it uses a forward mask for non-zero pixels after “bitwise and” operation. If there is a small amount of them, it passes forward movement as output to Arduino.
Else if there are more non-zero pixels, it checks “left lane” or “right lane” mask for a low amount of non-zero. So, it assures that the problem with moving forward is not just an obstacle at the road, because if it is the obstacle, it can get around it.
Then, if it is not an obstacle, the algorithm checks “turn right” and “turn left” masks for low non-zero pixels amount. For either of them, it will write the corresponding command. Last, if all of the upper cases failed, it will stop the car.

C. Naviagation

The work is nod done with just sending a command to Arduino because it is necessary to control the car’s movement in an accurate way. During the testing, there was found that if there would be sent non-stop signal to move forward, the car will keep accelerating. However, at high speed, it is impossible to react to the situations in front of it quickly enough. Also, the manoeuvrability of a car would become very low. Therefore, it was decided to control the car by series of a long move and shortstop signals, so that it would not be able to accelerate. All of these codes were written in Arduino and for each of movement types, like forwarding, turn somewhere or change the lane, there were chosen series of signals with specific delays. The value of delay was set by trial and error method after lots of experiments.

IV. Testing and Results

The testing environment was set by white A4 papers constituting road as well as obstacles in front of an RC car. For line following, the road was set as a straight line with curvature turn in the middle of the road.



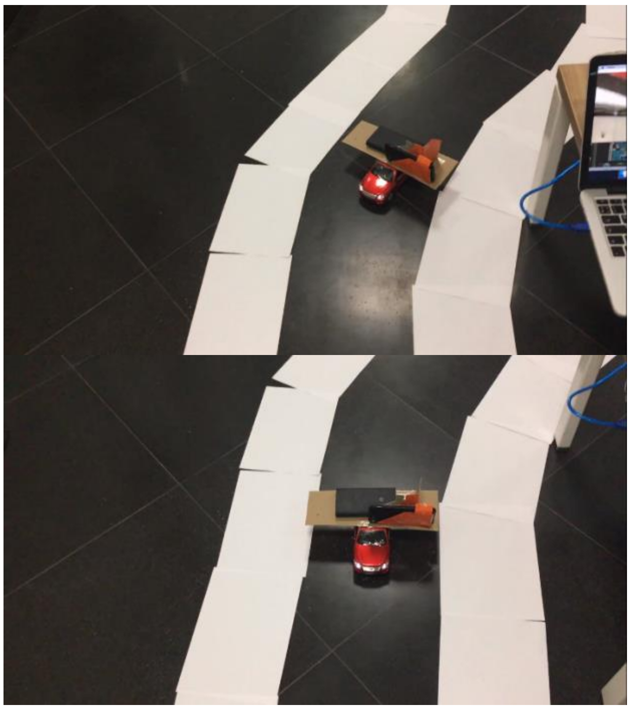
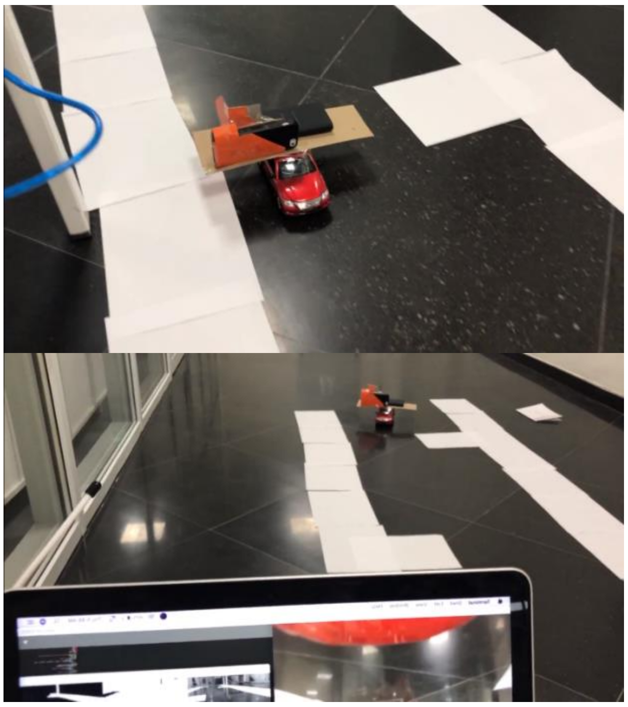


Fig.17: Curvature road following

For obstacle detection and avoidance straight, wide road was used. The obstacles were chosen as white A4 regions in different locations on the road.



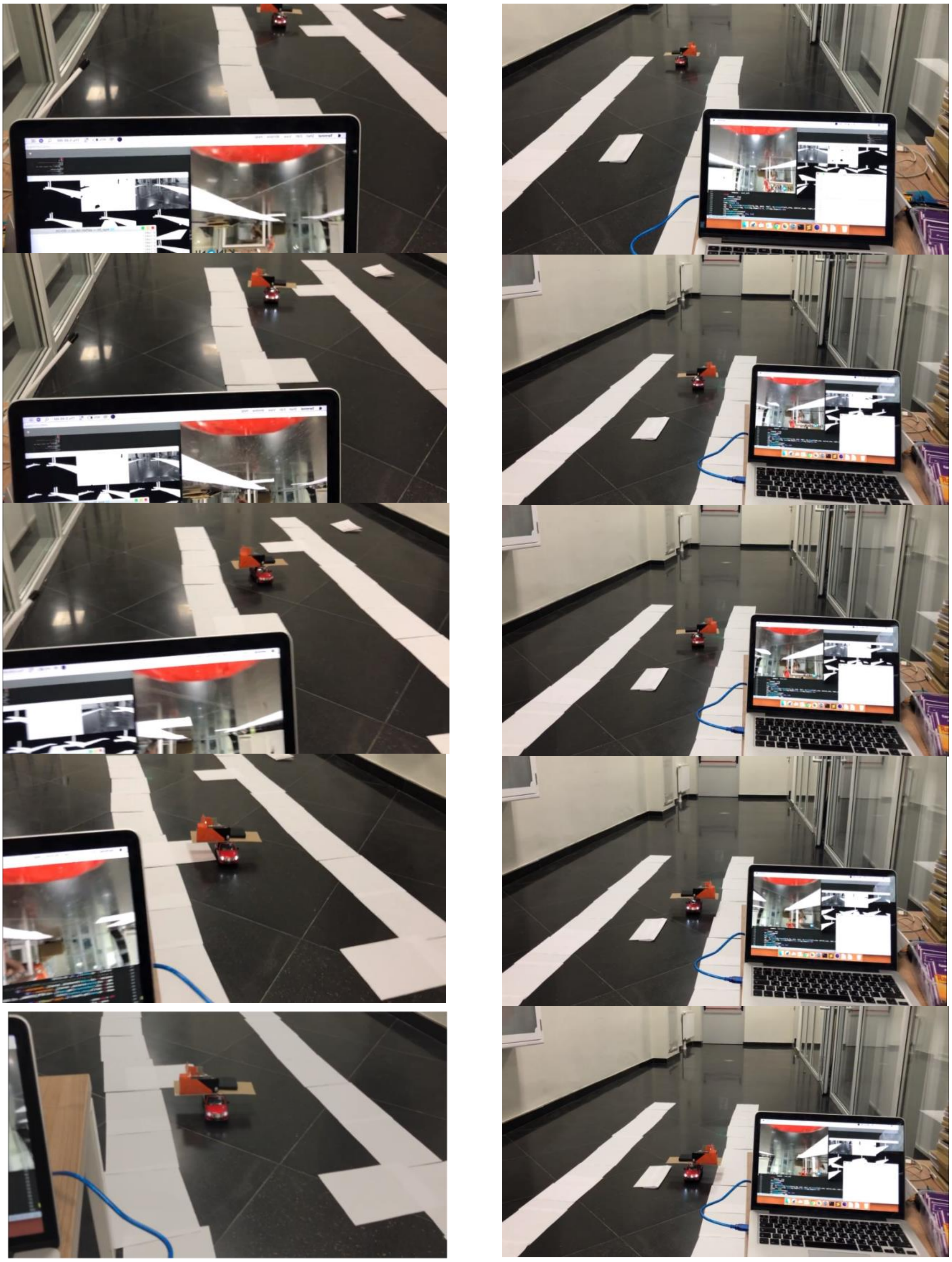


Fig.18. Avoiding obstacles at the edge of road

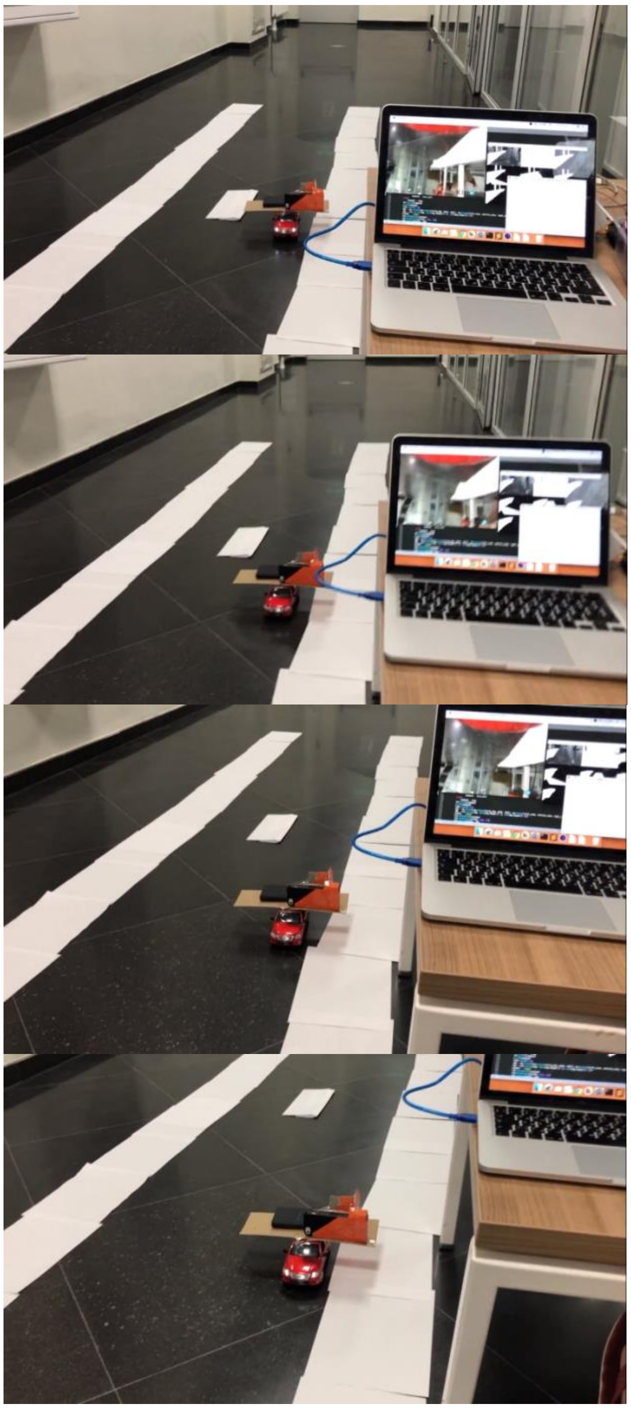


Fig.19. Avoiding obstacle at the center of road

Video:
https://drive.google.com/open?id=16xnFExv5c21L1J431Hw..

V. Conclusion

In conclusion, the offered algorithm works well with all cases of possible situations at the tested road. The limitation was computation speed of processing an image in real-time and it forced to lower the speed of RC car. This can be solved with using more powerful hardware and faster video transmission methods. Furthemore, The reason for usage of only built-in functions is that the algorithm works in real-time in order to react to new video frames and control RC car's movements. It was tested with some of the built-in functions replaced and there was observed significant reduce in computation speed, so that commands were sended too late. As built-in functions in Python are written in highly optimized C++ format, their analogue fully written in Python should be much slower. Also, this algorithm can be improved at the future with adding Machine Learning, so that it will better adapt to car's maneuvering and speed properties to make the total movement faster. In addition, here can be integrated the feature of detecting road signs and traffic lights.

References

[1] Zheng Wang. (2019). Self Driving RC Car. [online] Available at: https://zhengludwig.wordpress.com/projects/self-driving-rc-car/ [Accessed 15 Apr. 2019].

[2] A. K. Jain, "Working model of Self-driving car using Convolutional Neural Network, Raspberry Pi and Arduino," *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, Coimbatore, 2018, pp. 1630-1635.