Vehicle Counter

Ruben Teimas, ***m47753***
Universidade de Évora

February 3, 2021

1 Introduction

For the second assignment of *Sistemas Multimodais* we were asked to develop a project with real world applications using at least one of the unconventional form of interaction approached in the second half of the semester.

The project chosen to develop was a simple *vehicle counter* to appreciate the traffic volume on the road. Computer Vision techniques were used in order to process the images and detected the vehicles.

2 Approach

When thinking about a vehicle counter, one of the first problems that comes to mind is: how to identify a vehicle on the road.

2.1 Haar Cascade

There are many techniques to complete this task but the first approach that came to mind was by using a *Haar Cascade*. This technique was used in class for simple tasks, like identify a person's face and eyes using the sample cascades provided by *OpenCV*. For these simple tasks, the results were really good, so it seemed like a good idea to try it for vehicles.

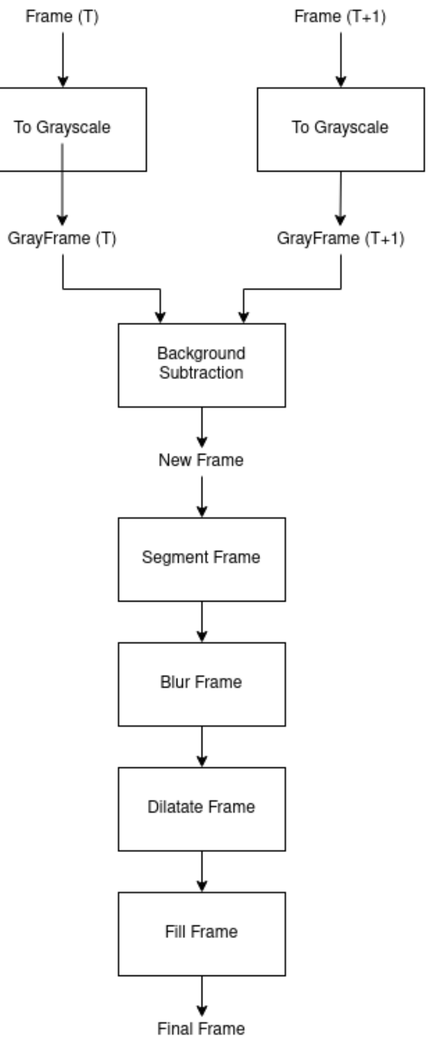
With this goal in mind I tried to create an *Haar Cascade* for cars, using *OpenCV*'s provided functions *opencv_createsamples* and *opencv_traincascade*. In order to use these functions I had to download an *OpenCV*'s version bellow 4 (as they are no longer provided in the most recent version) and build it from source.

After that, i gathered a set of positive and negative images (1000 entries for each) and used the previous commands to create and train the cascade with 10 stages.
The resulting *Haar Cascade* was not very good, as it had a loot of difficulty to recognize vehicles. To improve the training, I tried to train the same cascade with 15 stages, but the training process kept crashing at later stages. Eventually, I ended giving up on this approach.

2.2 Frame Difference

The second approach was to use frame difference along with frame processing. This approach is, in my opinion, inferior to the previous one, simply by the fact that frame difference essentially applies a background subtraction, which cleans the image except for the moving objects. Therefore, this method does not identify a vehicle, instead it identifies a moving object.

Figure 1: Pre-processing pipeline



In ***Figure 1*** it is described the pipeline of transformations for a frame. First, we pick 2 subsequent frame and turn them into grayscale. This transformation is useful because we get rid of colors as they are not useful for for image processing, by eliminating them, we remove noise from our frames. Then, we make the difference between both frames, which will resulting in a background subtraction.

After having only the moving objects, we will try to make them stand-out as much as possible. To do this we apply various techniques, since thresholding (to segment the frame), bluring (to smooth the frame), dilatation (to expand the frame) and filled (to fill the remaining spaces).

It's time to detect the vehicles by finding their contours, making a rectangle around the contoured area and getting its center. I opted to create a minimum size for the height and the width of the rectangle to ensure that the area detected is not just noise.

Finally, to count the detected vehicles, a line was placed at each frame. If the detected vehicle crosses the line a vehicle will be counted.

3 Technical Documentation

To test the system, *OpenCV* (version 4.*) and *Python* (version 3.*) are needed. The project can be executed by using: *python3 main.py* at the root directory.

The original videos, as well as the processed videos, can be found at the following `https://drive.google.com/drive/folders/1qmKJaTbivQ8mXZjUtpsnQHyA4htyU17x?usp=sharing`

4 Considerations

This project should not, by any means, be used as a *traffic counter*, at least, without some tuning. It has a lot of limitations which hold it back from performing as well as it would be needed.

For the first video, it performs quite well but still has some flaws since it counts only 15 vehicles in 19 vehicles available. At first, I thought it failed to count the cars because it was very unlikely to detect the car at and exact pixel (thickness of the line), so, I increased the delta(thickness). That way, I was able to count cars in a certain interval of pixels, but that still didn't solve all the problems.

The count is not that good for the second video. This video is very different as it has both lanes. First of all, the system has some difficulty to recognize cars that come from the right line, mainly, because when they enter the field of vision, they are already very close to the line.

For the left lane, the system performs better, but still, it has some problems. One of them is the fact that some vehicles are counted twice. This happens because, after the pre-processing, some "vehicles" have discontinuations and the system assumes that they are two different vehicles, instead of one.

These are just some of the flaws that can be spotted, but there are others that are not as clear, for instance, the line's position. It is static, which means that the line will always be on the same spot, independently of it being a point of interest in a different video or real-time footage. The position can be changed, but is has to be manually done inside the code.

The project was partially completed with success, but I have to admit that it did not fulfill my expectations, there was a lot more to be done. It is still an interesting work from a *Computer Vision* perspective, but it lacks from other unconventional interaction modules and intelligent agents.