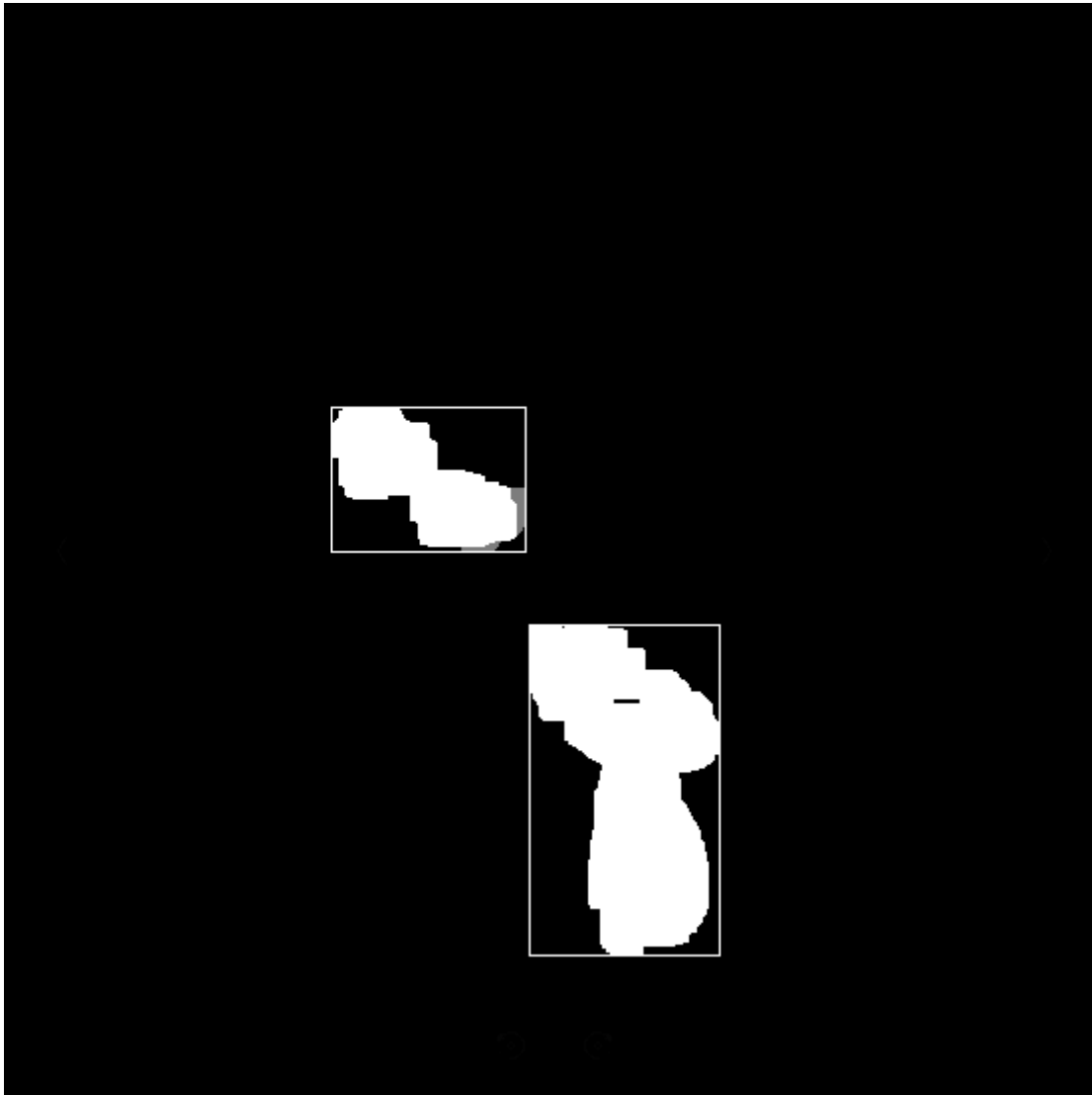


Jeremy VONG  
Kevin QUACH

Université Gustave Eiffel 2023-2024  
Master 2 Sciences de l'Image

# Person Detector



# Contents

<b>Presentation of the app</b>	<b>2</b>
<b>Implementations</b>	<b>2</b>
<b>Computing resources</b>	<b>3</b>
<b>Profiling</b>	<b>3</b>
<b>Optimization choices</b>	<b>4</b>
<b>Evaluations</b>	<b>5</b>
<b>Conclusion</b>	<b>5</b>

## Presentation of the app

This program is a person detector developed during our Master 2 Sciences de l'Image studies. The program reads a sequence of images forming a scene and will return the same amount of images in which the persons will be framed. These sequences are located in an adjacent folder 'dataset' to the 'Project' folder, with each sequence in their own sub-folder.

To use the program, create a "build" folder in the 'Project' folder.

Build the project with: `cmake .; make` .

This will create the executable PersonDetector.

To use it: `./PersonDetector [Number of thread to use]`

## Implementations

To read the images, we first start by gathering the path to the image files that we store in a set of paths. Using those paths, we will then read each image using openCV's `imread` function and store them in a vector.

After gathering our images, we will proceed to transform the image to remove the irrelevant parts and only keep the relevant part. To that end, we first equalize the histogram of our images using the `equalizeHist` function in order to improve contrast.

We then want to remove the background. After some research, we came across openCV's `createBackgroundSubtractorMOG2` function which does a decent job at removing backgrounds. The caveat here is that, we first need to give the BackgroundSubtractor a few images before it can start detecting the background

through comparison with previous images, thus, the first few images won't have their background removed. Moreover, if there are some inconsistencies between images, i.e. part of the background suddenly lit up, the BackgroundSubtractor won't consider it as part of the background and won't remove it. It also implies that something that is moving but then suddenly stops, will be considered as part of the background.

The program not being able to detect the background at first shouldn't be a problem in the case of a constant stream of images focused on a single scene, i.e. a surveillance camera.

Despite using the BackgroundSubtractor, it is still possible that some noise remains on the image. Morphological transformation is used to remove them. Once this is done, the only thing that remains are moving components (or background anomalies). Finally, we pinpoint the connected components whose area is bigger than a certain threshold and frame them. Those are what we consider to be people.

## Computing resources

When it comes to computing and comparing the performance, we are using OpenMP's [omp\\_get\\_wtime](#) function. The function returns the number of seconds since the initial value of the operating system real-time clock with a double precision. For each part whose performance we want to compute, we get the time before and after the execution of that specific part and we just have to subtract the time after execution to the time before execution.

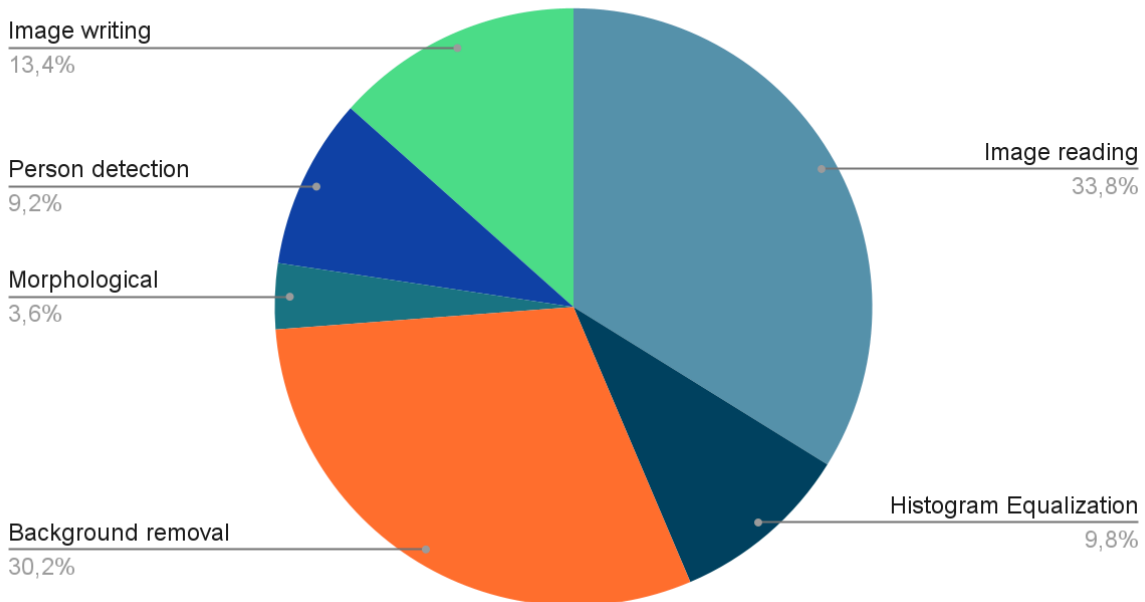
## Profiling

The program is composed of 6 major parts.

	Task	Comment
1	Image/Video reading	Done with OpenCV imread()
2	Images histogram equalization	Done with OpenCV equalizeHist()
3	Background subtraction	Done with OpenCV BackgroundSubtractor

4	Noise removal	Done with morphological transformations
5	Person detection	
6	Saving the processed image	Réalisé avec OpenCV imwrite()

## Execution time repartition



## Optimization choices

As most of the steps in the image processing uses for loops to iterate over the list of images, we can parallelize most of them. Thus, we used OpenMP's instruction to parallelize said loops.

The parts that we managed to parallelize are: the image reading, the histogram equalization for each image, the noise removal, the person detection and the image writing.

The only step that we couldn't parallelize is the background removal given the fact that the BackgroundSubtractor needs the previous image to remove it.

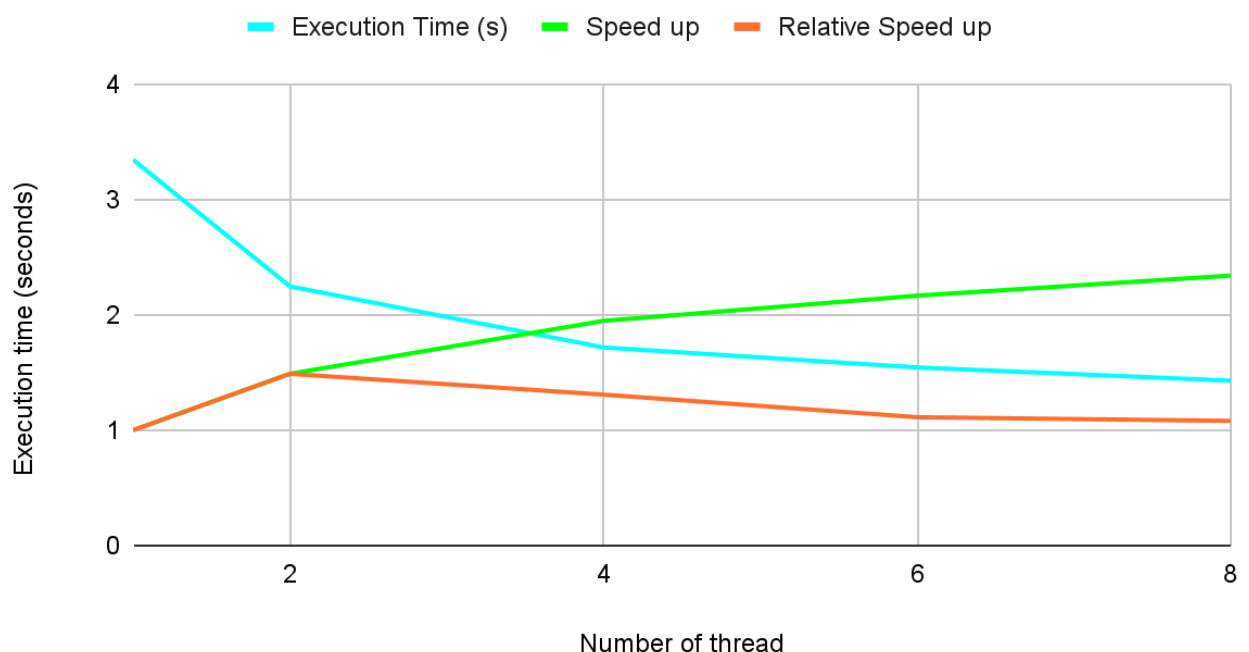
As one would expect, the parallelization did improve execution time quite significantly, depending on the number of threads used.

We've experimented further with the scheduler, using static, dynamic and guided schedulers with varying chunk sizes. However, none of our tests managed to

decrease the execution time by a significant amount. We can surmise that, since the iterations in each of those steps takes about the same amount of time and there isn't a big gap in execution time between them, we're not gaining much in terms of performance. There is also the fact that, when using scheduler, the thread must stop to receive the next chunk which also takes time. In this context, the drawbacks might even out the benefits which ends up not giving much of a performance boost, if at all.

## Evaluations

### Execution time evolution by number of threads



We can see on the graph that, the more threads we have, the faster the program is. However looking at the relative speed up, it is clear that this performance boost decreases proportionally to the number of threads.

In terms of results, the PersonDetector is decent, as moving humans were accurately found. Though some problems remain: when people stay still, the BackgroundSubtractor blends them in with the background. Another noticeable problem is that, when two people are close enough, the detector considers them as one single person. Finally, as mentioned above, some anomaly with the background can occur (flashing lights for example) which then result in big components not being removed by the BackgroundSubtractor, which can in turn, be interpreted as people.

## Conclusion

This project allowed us to have a first peek at how to process images to detect moving components on a stream of images. We managed to broaden our knowledge in the OpenCV library through the development of the program, as well as OpenMP through optimization through parallelization. We've analyzed which part of the code can be parallelized and which part cannot and witnessed the performance boost given by the parallelization. Through that, we deepened our understanding of parallel programming.