# SPM Project - Video motion detector

Geremia Pompei (MAT. 638432)

August 22, 2022

## 1 Introduction

The aim of this project is detect movements inside a video located in the file system. This is done comparing pixels of a background with the current frame of the video. Before the comparison frames are transformed applying them a grayscale and smoothing transformation. Given in input a parameter $k$ the program is able to count the number of frames that have $k\%$ different pixels from background image.

## 2 Process detail

### 2.1 Sequential implementation

At the beginning to solve this task it's decided to scroll video frames in a sequential way. Before the scrolling it's processed and set as background the first frame. During the scrolling is applied the sequential process showed on Figure 1. The composition of the following functions is able to understand if a frame is equal or not with respect the background image.

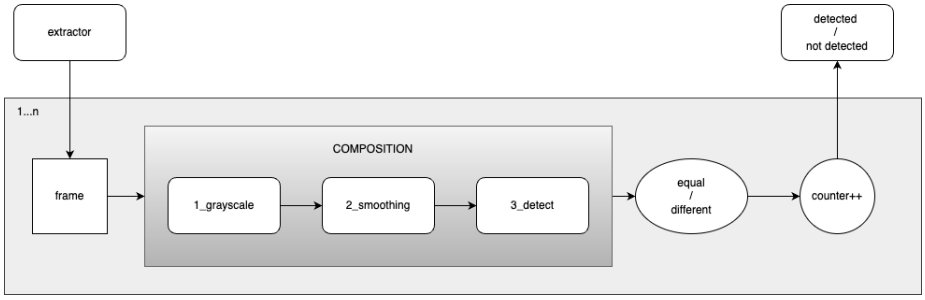


Figure 1: Sequential process

### 2.1.1 GrayScale

Method of detector that takes a colored image and transforms this in its black and white version. This is done scrolling the matrix of a frame by rows and columns and for each pixel computing the average of color channels. The result of the average is set as value for each color channel of the same pixel.

### 2.1.2 Smoothing

Smoothing method takes the frame transformed by *GrayScale* and convolves this with a kernel. The kernel used in this project is a 3x3 matrix of $\frac{1}{9}$ values so an average kernel. For the convolution is scrolled the entire image by rows and columns and for each pixel is multiplied the current and the neighbours pixels by the kernel. The results of these multiplications are summed and putted inside the color channels of the current pixel. The image that is transformed here has a blur effect that change in intensity with respect the size and the type of filter. Applying this kind of average filter the effect is soft. Before applying the smoothing is cloned the original frame and is used this during the computation to transform the original one. It's used the clone frame to avoid to change neighbour pixels during smoothing process.

### 2.1.3 DetectDifference

Having a background frame this method is able to understand if a given smoothed frame is similar or not. Scrolling together the smoothed current frame and the background one by rows and columns it's counted the number of different pixels. If this number is bigger than the given threshold $k$ the method return true because the current frame is detected as different otherwise return false.

## 2.2 Parallel implementation

As shown in Figure 2 to parallelize this task is used a threadpool. Each of it's thread run the same composition of function listed in the previous section. To optimize the sequential implementation is decided to use a farm of composition of function, so the normal form. There are two point of synchronization before and after the execution of the composition function. In this way the implementation remain simple and efficient.

## 2.3 Brief view on code structure

The code is kept simple using a main file that switch the execution with respect the implementation decided in input using a string. For each implementation there is a specific file with his own implementation.

**SEQUENTIAL** In this file there is a class able to construct a `Sequential` object that apply the composition of `gray`, `smoothing` and `detect` functions to

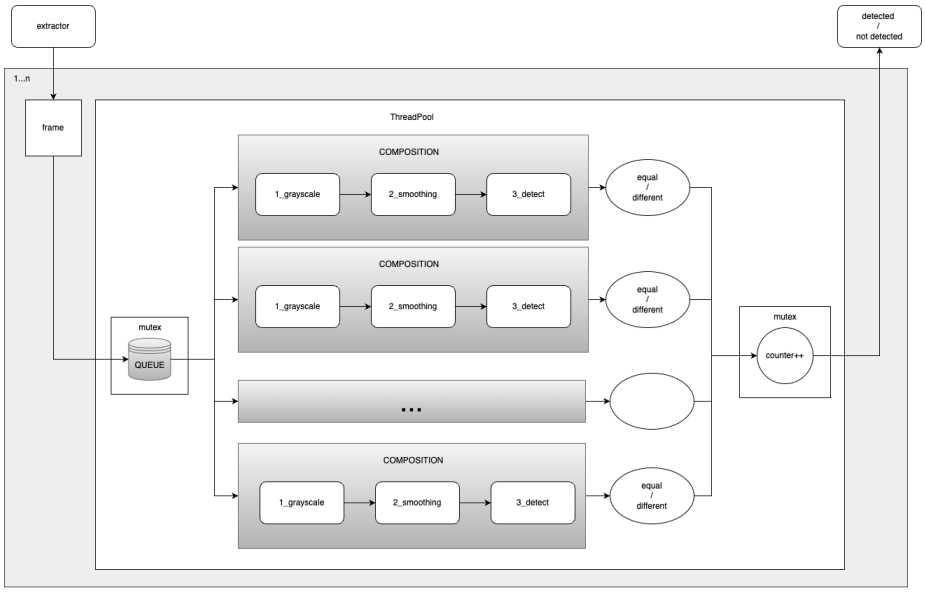


Figure 2: Parallel process

each frame. The different pictures are extracted from the video source one after another.

**NATIVE_THREADS** In this file there is a class able to construct a `NativeThread` object that like before applies the three functions one after another to each frame. Differently from before is that here is used another object called `ThreadPool`. It allows to push frames inside a queue and simultaneously execute a function to each frame in different threads. When the threadpool object is built it runs $n$ threads with a particular function that can be described in few passages. The function inside each thread:

1. tries to access in a mutual exclusive way to the head of the queue and removing his head
2. if the thread is able to extract the head of the queue, that is a frame, it applies the composition of the three functions to it

If an image is detected as different from the background using a mutex is increased a counter.

**FASTFLOW** In this file there is a class able to construct a `Fastflow` object that is able to build a threadpool implementation with fastflow library. In particular is used a `ff_Farm` object with:

- *Emitter*: able to extract the different frames of a video and filling a queue

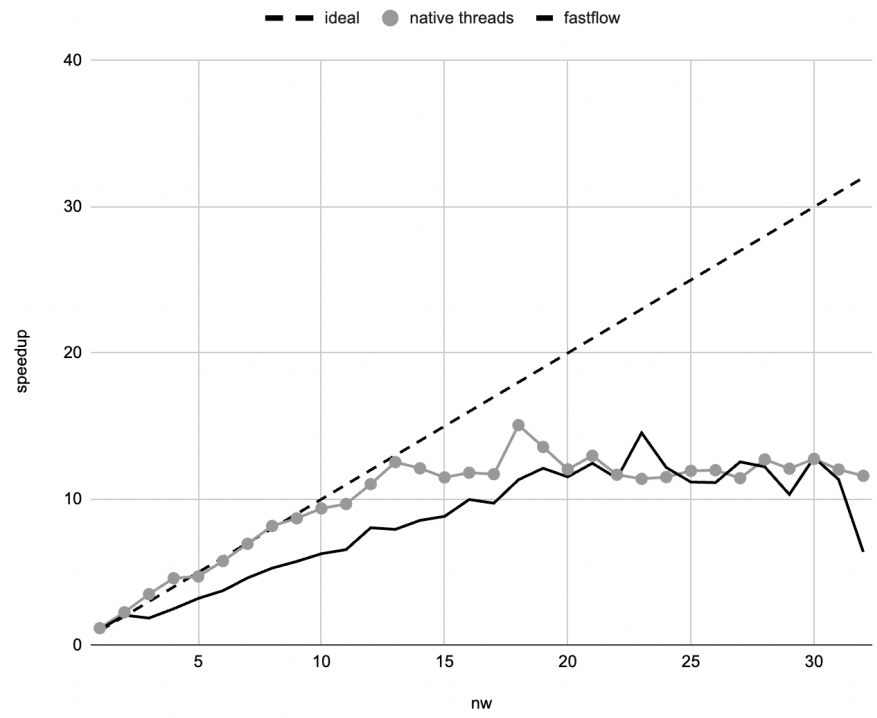


Figure 3: Speedup of native thread and fastflow parallelization with respect ideal

- *Worker*: able to compute the composition function to the frame extracted from the head of the queue. The frame extraction is mutual exclusive.

# 3 Experiments

Experiments are done on a video of one minute and 1799 frames and the threshold is set to 0.4. During compilation time the code is vectorized using the g++ parameter -O3. Using this trick the performances three times better.
In Figure 3 is plotted the speedup related to native thread and fastflow implementation with respect the ideal one.
In Figure 4 is plotted the efficiency related to native thread and fastflow implementation with respect the ideal one.

# 4 Analysis

As seen in experiments using parallelization there is an optimization with respect sequential case. Of course this acceleration is not equal to the ideal cases for

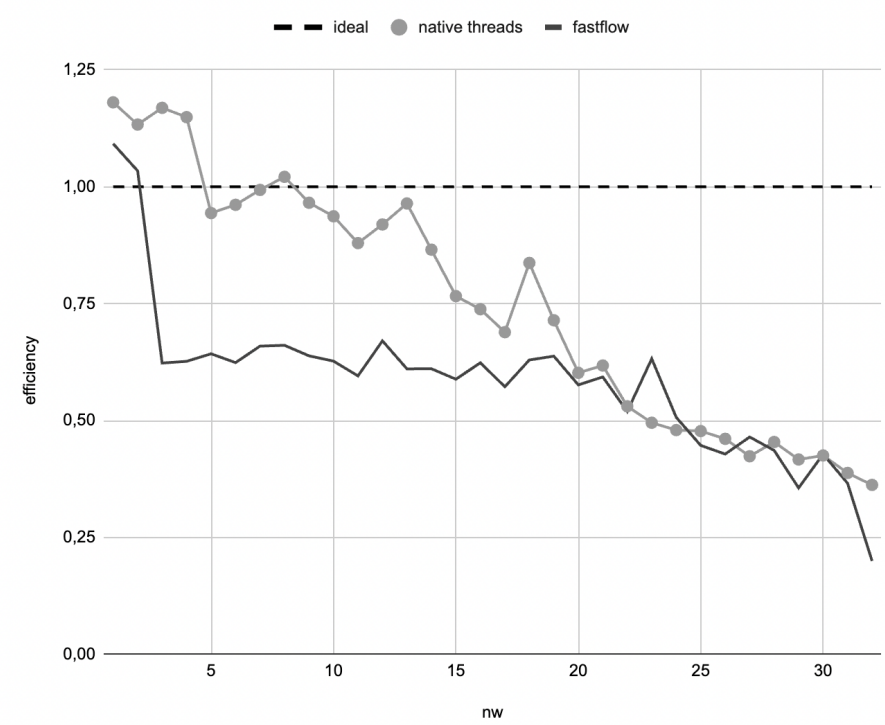


Figure 4: Efficiency of native thread and fastflow parallelization with respect ideal

the overhead that is mainly related to the synchronization points.

We can notice that the native thread implementation is able to reach better results in the first part of plots. With $1 \leq nw \leq 13$ this implementation is able to have a performance very similar to the ideal time while then it tends to decrease. We can see that in Figure 3 after $nw = 13$ the speedup remains flat. In Figure 4 the native thread efficiency keeps a regular decreasing effect with more or less the same slope.

The fastflow implementation instead has a different shape. It is worse in the first part of plot while is able to reach the same native thread results in the second part. After different experiments it's noticed that from $nw = 3$ to $nw = 20$ the performances are not so good as in native thread implementation and are different from the ideal while then are better to mitigate the decreasing effect. In Figure 3 the shape is less close to the ideal line and native thread implementation while then is overlapped with the other perallel implementation. In term of efficiency Figure 4 shows a quick decreasing effect from $nw = 2$ to $nw = 3$ and from there to $20^{th}$ nw the behavior is more or less linear whle then decreases.

# 5 Instructions

There are two script files in the main folder of project to compile and run the code on remote machine. To build the program there is the `./build.sh` file that is able to construct the executable of the code in `./src` directory. There is also a `./test.sh` file to run a simulation of the program with some alternative configurations. The basic commands to do this without script files are

```
g++-10 src/main.cpp -O3 -o build/main ‘pkg-config --cflags
 opencv4‘ ‘pkg-config --libs opencv4‘ -pthread -std=c++17
```

for compiling. To run it there are different parameters to use:

1. `videoPath`: path of video to detect. There are some video for testing on directory `./media`
2. `k`: float parameter that lives between 0 and 1 to give the percentage of pixels that is used as threshold to detect if a frame is similar or not with respect the background. If $k = 0.2$ a frame with less than 20% of pixels different with respect the background one is considered similar to it.
3. `paralelizationType`: string that could be `SEQUENTIAL`, `PARALLEL` or `FASTFLOW` with respect the implementation that is decided to use to optimize the code
4. `nw`: number of workers used to parallelize the program
5. `formatter`: string that if is set to `CSV` print log results in a csv formatted way (used for statistics). Otherwise if the string is not present or different log are more readable

To run the program the formatted command is:

```
./build/main [videoPath] [k] [paralelizationType] [nw]
                        [formatter]
```

An example of this could be:

```
./build/main ./media/test_1m.mp4 0.4 FASTFLOW 4
```