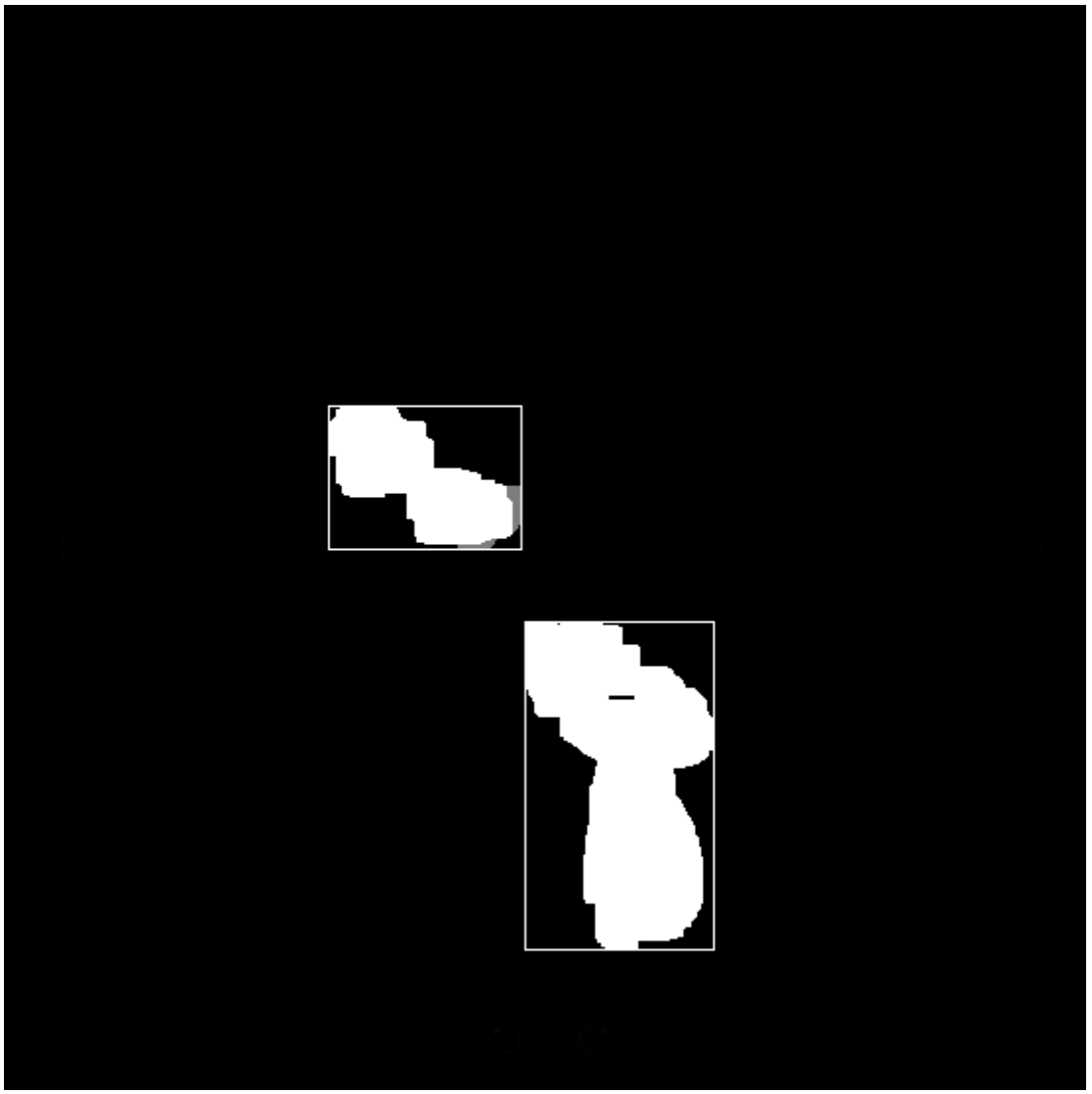
Jeremy VONG
Kevin QUACH

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

Person Detector



Contents

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.

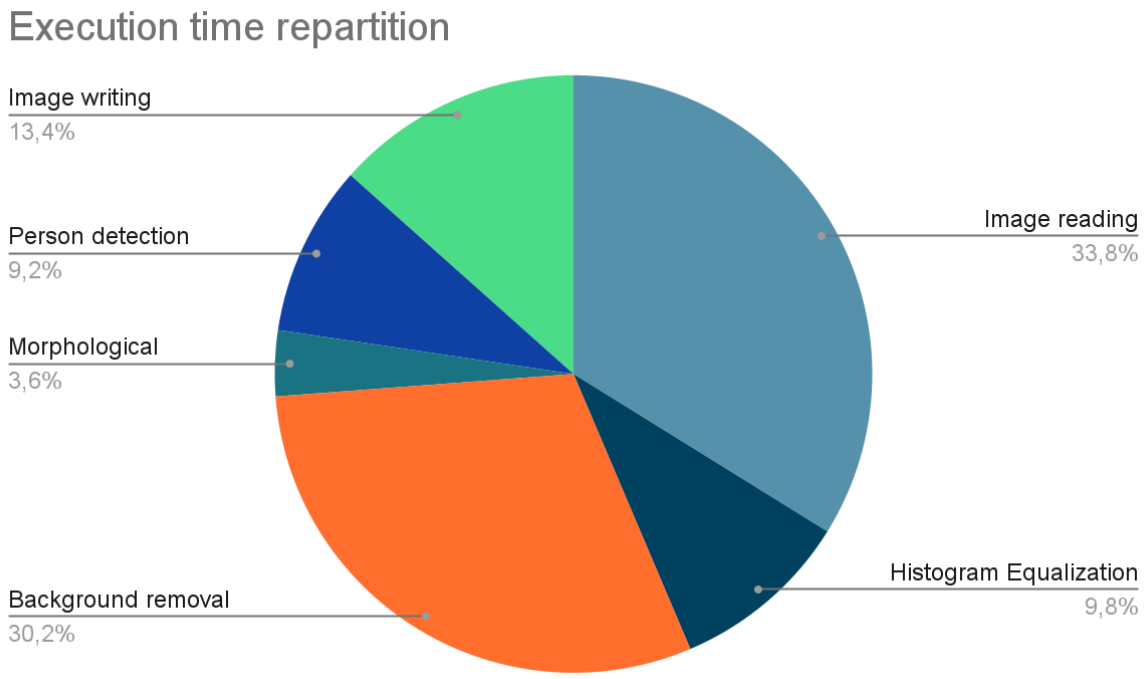
Computing resources

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()



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.

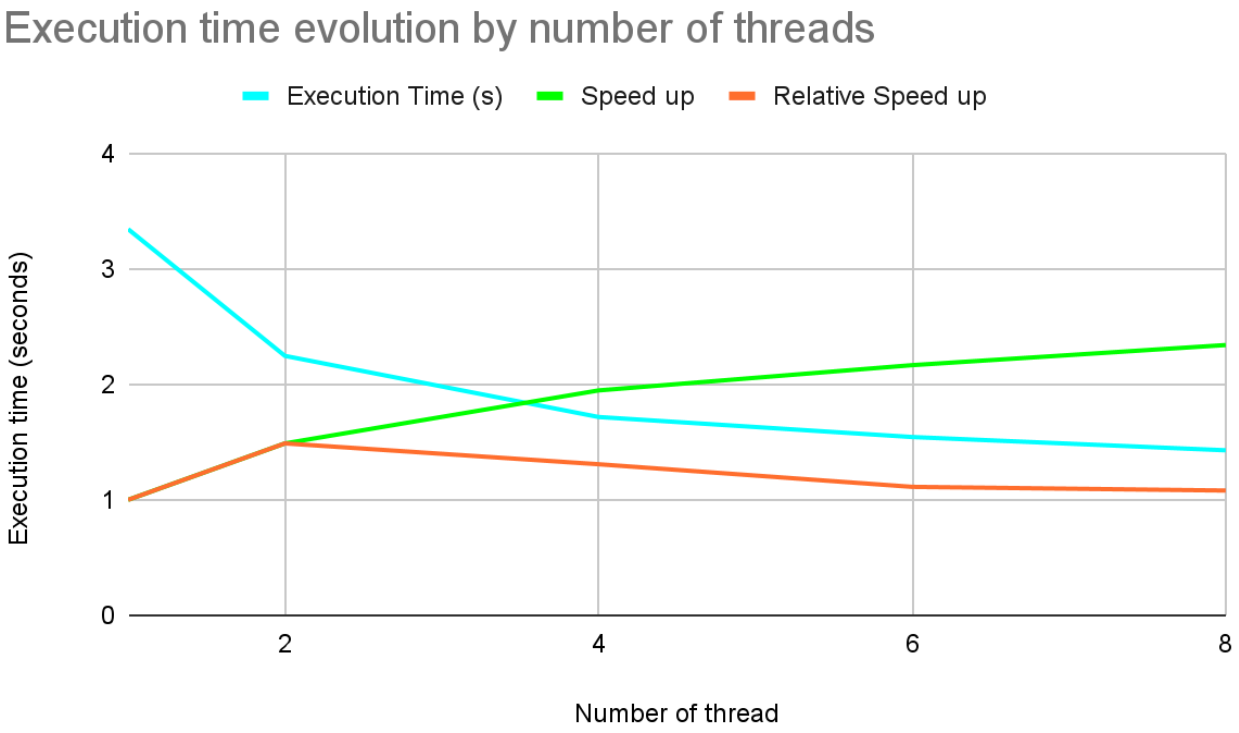
The parts that we managed to parallelize are: the image reading, the histogram equalization for each image, the noise removal, the person detection and the image writing.

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



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.

Annexe

Code PersonDetector.cpp

```
#include <omp.h>
#include <filesystem>
#include <iostream>
#include <map>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/video/background_segm.hpp>
#include <regex>
#include <set>
#include <string>
#include <utility>
using namespace cv;
namespace fs = std::filesystem;

/**
* @brief Retrieve the path of the images contained in the directory at
the path given and return them in a set
* @param path The path where to directory with subdirectories
containing image
*/
std::vector<fs::path> retrieveVideoImagePaths(std::string path) {
   std::vector<std::string> dataset_path;
   std::set<fs::path> sorted_by_name;
   for (const auto &entry : std::filesystem::directory_iterator{path})
{

       // Retrieve directory names
```

```
        std::string dirname = entry.path().filename();

        // Retrieve filepath in each directory
        for (const auto &file_entry :
std::filesystem::directory_iterator{entry.path()}) {
            sorted_by_name.emplace(file_entry.path());
        }
    }
    std::vector paths(sorted_by_name.begin(), sorted_by_name.end());
    return paths;
}

/**
* @brief Create directories in outputPath with the same name as the
ones in path
* @param path The path containing the directories
* @param outputPath The path where to create the directories
*/
void createOutputDirectory(std::string path, std::string outputPath){
    for (const auto &entry : std::filesystem::directory_iterator{path})
{
        std::string dirname = entry.path().filename();
        fs::create_directories(outputPath + dirname);
    }
}

/**
* @brief Remove small noise in the image through morphological
transformations using a structuring element
* @param img The image on which to perform the morphological
transformations
* @param element The structuring element to use for the morphological
transformations
*/
void removeNoise(Mat &img, Mat &element) {
    morphologyEx(img, img, MORPH_ERODE, element, Point(-1, -1), 5);
    morphologyEx(img, img, MORPH_DILATE, element, Point(-1, -1), 8);
}

/**
* @brief Detect connected components whose area is above a certain
threshold and frame them
* @param img The image that we want to process
```

```
* @param threshold The area threshold
*/
void spotPeople(Mat &img, int threshold) {
   Mat labels, stats, centroids;
   int nbcomp = connectedComponentsWithStats(img, labels, stats,
centroids, 4, CV_32S);
   Point pt1, pt2;
   int area;
   for (int i = 1; i < nbcomp; i++) {  // Start at 1 because 0 is the
background.
       area = stats.at<int>(i, CC_STAT_AREA);
       if (area > threshold) {
           pt1 = Point(stats.at<int>(i, CC_STAT_LEFT), stats.at<int>(i,
CC_STAT_TOP));
           pt2 = Point(pt1.x + stats.at<int>(i, CC_STAT_WIDTH), pt1.y +
stats.at<int>(i, CC_STAT_HEIGHT));
           rectangle(img, pt1, pt2, 255);
       }
   }
}

int main(int argc, char *argv[]) {
   int nbThreads = 1;
   if (argc >= 2) {
       nbThreads = std::stoi(argv[1]);
   }

   std::string inputPath = "../../dataset/";
   std::string outputPath = "../../outputDataset/";
   std::string outputDir = "outputDataset";

   omp_set_num_threads(nbThreads);  // Setting the number of threads.
   int nbChunks = 96; // Number of chunk for iteration distribution

   std::cout << "Nb threads : " << nbThreads << std::endl;

   double start_time_all = omp_get_wtime();
   double start_time, end_time;
   double total_time_spotPeople = 0, total_time_morphTransform = 0,
total_time_equalize = 0, total_time_applyBackgroundRemover = 0;
   double total_time_readImg = 0, total_time_writeImg = 0;

   // ------- Retrieve all images -------
```

```
    start_time = omp_get_wtime();
    std::vector<fs::path> sorted_by_name =
retrieveVideoImagePaths(inputPath);
    end_time = omp_get_wtime();
    std::cout << "Total time taken in retrieveVideoImagePaths() : " <<
end_time - start_time << "seconds " << std::endl;

    std::regex pattern("dataset");
    fs::path current_parent_path = "./";

    std::vector<Mat> imgs(sorted_by_name.size());
    std::map<int, bool> valid_imgs;
    fs::path filepath;
    std::string image_path;

    start_time = omp_get_wtime();

    #pragma omp parallel for private(image_path) schedule(dynamic,
nbChunks)
    for (size_t i = 0; i < sorted_by_name.size(); i++) {
        // ------- Read Image -------
        image_path = samples::findFile(sorted_by_name[i]);
        imgs[i] = imread(image_path, IMREAD_GRAYSCALE);
        if (imgs[i].empty()) {
            std::cout << "Could not read the image: " << image_path <<
std::endl;
            valid_imgs[i] = false;
            // return 1;
        } else {
            valid_imgs[i] = true;
        }
    }

    end_time = omp_get_wtime();
    total_time_readImg = (end_time - start_time);
    std::cout << "Total time taken for reading image : " <<
total_time_readImg << "seconds " << std::endl;


    // ------- Equalize Image -------

    start_time = omp_get_wtime();
```

```
    #pragma omp parallel for schedule(dynamic, nbChunks)
    for (size_t i = 0; i < sorted_by_name.size(); i++) {
        if (valid_imgs[i]) {
            equalizeHist(imgs[i], imgs[i]);
        }
    }

    end_time = omp_get_wtime();
    total_time_equalize = (end_time - start_time);
    std::cout << "Total time taken for equalizeHist() : " <<
total_time_equalize << "seconds " << std::endl;

    // ------- Apply background remover -------

    Ptr<BackgroundSubtractor> pBackSub;
    pBackSub = createBackgroundSubtractorMOG2();
    start_time = omp_get_wtime();

    for (size_t i = 0; i < sorted_by_name.size(); i++) {

        if (valid_imgs[i]) {
            const auto filepath = sorted_by_name[i];

            // Checking if it's still the same scene, if not, create a
new BGSubtractor
            fs::path parent_path = filepath.parent_path();
            if (current_parent_path.compare(parent_path) != 0) {
                // std::cout << filepath.parent_path() << std::endl;
                current_parent_path = parent_path;
                pBackSub = createBackgroundSubtractorMOG2();
            }
            pBackSub->apply(imgs[i], imgs[i]);
        }
    }
    end_time = omp_get_wtime();
    total_time_applyBackgroundRemover = (end_time - start_time);

    std::cout << "Total time taken for background Remover : " <<
total_time_applyBackgroundRemover << "seconds " << std::endl;

    // ------- Removing noise with morphological transformation -------
    Mat element = getStructuringElement(MORPH_RECT, Size(3, 3));
    start_time = omp_get_wtime();
```

```
    #pragma omp parallel for schedule(dynamic, nbChunks)
    for (size_t i = 0; i < sorted_by_name.size(); i++) {
        if (valid_imgs[i]) {

            removeNoise(imgs[i], element);
        }
    }

    end_time = omp_get_wtime();
    total_time_morphTransform = (end_time - start_time);
    std::cout << "Total time taken in  removeNoise() : " <<
total_time_morphTransform << "seconds " << std::endl;

    // ------- Looking for ppl among connected components -------
    start_time = omp_get_wtime();
    #pragma omp parallel for schedule(dynamic, nbChunks)
    for (size_t i = 0; i < sorted_by_name.size(); i++) {
        if (valid_imgs[i]) {
            spotPeople(imgs[i], 2000);
        }
    }
    end_time = omp_get_wtime();
    total_time_spotPeople = (end_time - start_time);
    std::cout << "Total time taken in  spotPeople() : " <<
total_time_spotPeople << "seconds " << std::endl;

    // ------- Read and Save Image ------- */

    // Creating the output directories
    createOutputDirectory(inputPath,  outputPath);

    start_time = omp_get_wtime();
    #pragma omp parallel for schedule(dynamic, nbChunks)
    for (size_t i = 0; i < sorted_by_name.size(); i++) {
        if (valid_imgs[i]) {

            // ------- Uncomment to display images -------

            // std::string filename = sorted_by_name[i].filename();
            // imshow("Display window", imgs[i]);
            // int k = waitKey(0); // Wait for a keystroke in the window
```

```
            // ----------------------------------------


imwrite(std::regex_replace(samples::findFile(sorted_by_name[i]),
pattern, outputDir), imgs[i]);

        }
    }
    end_time = omp_get_wtime();
    total_time_writeImg += (end_time - start_time);
    std::cout << "Total time taken for writing image : " <<
total_time_writeImg << "seconds " << std::endl;

    double end_time_all = omp_get_wtime();

    std::cout << "========================== \n Total time taken overall
: " << end_time_all - start_time_all << "seconds " << std::endl;

    return 0;
}
```