Faculty of Applied Sciences
University of Liege
September 9, 2019

Ben Mariem	Chiodo	Derroitte	Girineza	Testouri
Sami	Adrien	Natan	Guy	Mehdi

Computer Vision :

Task 2: Motion and person detection, panoramic back- ground, performance assessment

1 Abstract

This second part of the project aims to implement 3 new modules : the motion detection, the person detection and the enhanced panorama where the moving object have been removed.
The motion detection module was implemented using a homemade "frame differencing" method.
Then, this module will be used to improve the panorama. Indeed, the motion detection module will allow us to remove all the *moving parts* of the panorama.
Two approaches were implemented for the person detection module, the first one using the HOG method from *OpenCV* and the second one the tensorflow object detection API which gave significantly better results.
For each of the different module, a precise performance assessment have been implemented. Often based on the *missing pixels* and *pixels in excess* in the result, this computation of this error is based on the reference detection and mask manually created.

2 Camera motion estimation : performance assessment

$0 \leq p \leq 1$ During the first part of this project, the quality of the implemented motion detection algorithm had already been evaluated. This section therefore includes the experiences and results obtained previously.

In order to correctly estimate the quality of our module, the calculation of accuracy is based on the computation of the camera rotation angle. After a rotation of an angle x, the algorithm should return this x.
The first test was based on the image sequences recorded in our database. When it was taken, the panorama was supposed to correspond to a 180° rotation.
The accuracy then found was 8° in the worst case. However, these sequences were quickly found to be inadequate for the accurate calculation of the error made on the camera's motion detection. Indeed, when these sequences were shot, the end condition of the videos was that 1500 frames were acquired and not that

the angle reached 180°. Therefore, these sequences most certainly present experimental errors coming from the lack of accuracy of the rotation angle (180°).

A second method implemented consisted in recording a new sequence. This was taken by paying particular attention to the angle of rotation achieved by the camera. The accuracy of the angle did not deviate from 4°.

Finally, the calculation of the angle was then carried out for sequences making "round trips". Therefore, the expected final angle was 0°. Using this method, the worst case went up to 5° of error although the average was lower.

The quality of the results remains to be discussed: an angle of 5° at worst was considered acceptable and therefore no modifications were made to the camera's motion detection module.

3 New module explanation

3.1 Motion detection

The main thing that could lead to a great improvement of the generated panoramic image would be to detect and remove moving objects. To be able to do this, a motion detection algorithm has to be used. The motion detection module will decide wether or not there's movement in the frame being considered and will generate the corresponding mask.

3.1.1 Description of the algorithm

In order to perform the motion detection, the most obvious solution would be to take the reference background and to compute the *moving* part by simply doing a difference between the reference and the frame being considered.
Unfortunately, in most of the application, the reference background is not available. Thus, an other solution needs to be designed.
The motion detection algorithm will work as follow:

1. Compute the *common* or the *overlapping* part of two consecutive frames of the video stream
2. Consider the oldest frame as the *background* and compute movement between both.

Even if the algorithm seems elegant and functional, the implementation that is provided is not matching the requirements that are needed. Indeed, the computation time is too high and the results are not accurate enough

3.1.2 Implementation

The algorithm will be implemented as follow :

- For two consecutive frames :
    1. Project the frames into the cylindrical plane (cfr. Part 1)
    2. Compute the transformation between the two frames (cfr. Part 1)
    3. Compute the intersecting part of the two frames
    4. Project the two intersecting parts back into the cartesian plane
- Then, compute the difference between two frames and tune the different parameters to get the most accurate moving part.

3.1.3 Performance assessment methodology

In order to quantify the error achieved by the motion detection alogrithm, it is able to generate a mask of moving objects. Therefore, the calculations will be based on the comparison between the mask thus generated and the one manually generated when annotating images.

In order to determine an error measurement, two types of errors will be created and put together:

1. **Error on missing pixels**: these correspond to pixels qualified as objects in our reference masks (in white) that are not white in the generated mask.

2. **Error on pixels in excess**: pixels considered as moving objects in the generated mask and not in the reference image.

In order to implement these two errors, it is essential to first determine the number of missing pixels and the number of pixels in excess.
This can be done using `numpy` functions or by doing an exhaustive search by simply comparing the images pixel by pixel. In both cases, this step is not complicated.

Missing pixels : A measurement of the error corresponding to the missing pixels can simply be made by taking the percentage of missing pixels.

$$e_1 = \frac{p_m * 100}{p_{w,ref}}$$

where p_m is the number of *missing pixels* as defined before and $p_{w,ref}$ is the total number of white pixel in the reference mask.
The error will therefore go from 0% when the masks match perfectly and 100% when the masks are completely reversed.

Error on pixels in excess : For this error, two approaches were considered. After their definition, they will be discussed in order to choose the one that best suits our task.

The first is simply to take the ratio between the number of excess pixels and the number of pixels that should not be considered important. Therefore, this error is defined as:

$$e_2 = \frac{p_e * 100}{p_{b,ref}}$$

where p_e is the number of *pixels in excess* as defined before and $p_{b,ref}$ is the total number of black pixel in the reference mask.
Again, this error will go from 0% when there is no pixels in excess to 100% when the whole image is considered as object.

However, this error is not entirely satisfactory. Indeed, it is based on the size of the image and not on the size of the objects to be detected. Thus, if a small object contains a large error with respect to its size, the error created could be very small if the size of the image is very large compared to the size of the object.

An error that takes into account the size of the object and not the size of the image is then considered. This is defined as follows:

$$e_2^* = min\left(100.0, \frac{p_e * 100}{p_{w,ref}}\right)$$

This new error has a different functioning. It now goes from 0% when the masks are identical to 100% when the number of excess pixels is greater or equal than the number of pixels of the object. It therefore takes better account of the size of the object being tested.

Linking the two errors : In order to put together the two errors, the choice to calculate a root mean square of the values was made:

$$e_{rms} = \sqrt{\frac{1}{n}\sum_{i=0}^{n} e_i^2}$$

This error punishes big mistakes and this is why it was choosen over a simple mean error. If the error of missing pixels is 100% but the error of excess pixels is 0 (the case where the generated mask is all black), it would be more consistent if the total error was greater than 50%.

Introducing tolerance : The reference masks, created manually, are unfortunately not perfect. Human errors resulting from the lack of precision when creating masks must be taken into account.
In order to consider this error on the reference masks, a tolerance can be added.
Thus, a pixel will be considered missing or in excess if the pixels surrounding it in the reference do not match the color of the pixel in the generated mask. Otherwise, the pixel will be considered acceptable. This helps to limit errors due to the lack of accuracy of the reference mask.

This addition goes hand in hand with a loss of temporal performance of the algorithm. This can be explained simply by comparing the complexities with and without this tolerance.

- Without tolerance : $\mathcal{O}(n \cdot m)$
- With tolerance : $\mathcal{O}(n \cdot m \cdot tol^2)$

where n, m are the width and height of the image and tol the number of pixels of tolerance.
The calculated complexity is considered for the case of exhaustive research. The use of the `numpy` functions used in the algorithm reduces this complexity and therefore the actual complexities are lower. It is therefore interesting to compare the difference between these two complexities but not the complexity itself.

Error comparison and discussion The different types of errors will now be compared. To do this, the following images are considered:

Figure 1: Reference mask *(left)* and inverse mask *(right)*

Generated masks used	e_2	e_2^*	e_2 with tolerance	e_2^* with tolerance
Mask itself	0.0	0.0	0.0	0.0
Inverse mask	98.83	98.83	82.31	82.80
White image	70.71	70.13	70.71	70.71
Black image	70.71	70.13	70.71	70.71

Table 1: Comparison of the different results for different generated masks with respect to the reference mask

In view of all that has been explained above, the error relating to objects with a tolerance was chosen. Note that the error with the invert masks is not exactly 100%. This comes from *openCV* rounding transition pixels (between white and black) to greyish pixels.

3.1.4 Results

It is now possible to calculate the total error for all reference frames. The results for the sequence outside and inside our group are shown below:

	Total error
Inside	69.61
Outside	70.43

The error is great because our generated masks have two main defects:

- they have a lot of noise,
- the objects themselves are poorly detected.

These two disadvantages are very largely penalised by our error measurement as explained above.

3.2 Enhanced panoramic image

This section aims at providing the different approaches considered to enhance the generated panoramic image. This sections aims at improving the panorama computed over a video sequence containing moving objects.

3.2.1 Improvement of the last module

The first thing that has been done is to improve the computation time of the panorama. Indeed, computing the panorama is one thing, but computing the motion detection between each frames is something that can be very heavy in terms of computation time.
Thus, the computation time of the basic panorama has also been improved greatly.

In order to further improve the quality of the panorama created, blending has been added.

Blending
The first implementation of the blending has been attempted with the blending pyramid technique. Theoretically, the technique begins to calculate the Gaussian pyramid of the two images. This allows repeat filtering and subsampling as shown on the figure 2. This pyramid is compute thanks to the *pyDown* function of *OpenCV*.

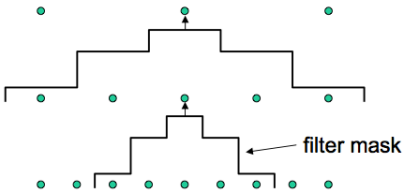

Figure 2: Representation of a Gaussian pyramid [3]

After establishing the Gaussian pyramids, the next step is to calculate the Laplacian pyramids. To do this, it is necessary to subtract (level n-1) of the pyramid from the (level n) which must first be resize to the same size as n-1 using the function *pyUp* of *OpenCV*. Then thanks to a mask, we weight and collapse the pyramid in order to get the final blended image.

Due to the fact that the frames are constantly growing, the length (in pixel) of the side of the panorama is not always fully divided. The only solution found was to add a column of zeros if the length was not even. And so, the length was only divided by 2 and the pyramid blending was perform on only 2 levels. Due to this limitation, the result of the pyramid blending was not really good and so another techniques has to be found.

The second implementation found was the alpha-blending. This technique gives a weight for each pixel of each frame that one wants to blend. This algorithm follow the equation below :

$$g(x) = (1 - \alpha)f_0(x) + \alpha f_1(x)$$

As shown on the figure the alpha-blending gave good result. Indeed, the vertical black lines are attenuated and the panorama seems smoother, more continuous. But the disadvantage of blending is that it makes the picture blurrier because of the differences between the two frames that have been blended.

Figure 3: Panorama without blending *(up)* and Panorama with blending *(down*

3.2.2 Description of the algorithm

The computation of the enhanced panoramic image is done in two steps:

1. Compute the mask representing the motion detected over two frames (cfr. Motion Detection)
2. Perform moving object subtraction using the previously mentioned mask.

Unfortunately, the results were really poor. This is mainly due to the poor results obtained while computing the motion detection and the corresponding mask. Thus, when performing moving object subtractions, the resulting frame will not be valid.

3.2.3 Results

- The results obtained for the computation of the enhanced panorama are mentioned below in figure 4. It can clearly be see that the "enhanced" panorama is far from being a good estimation of the reference panorama. This is totally due to the default in the motion detection module that is used.
Thus, the main improvement that should be made in this panorama would be to improve the motion detection module.

3.3 Person detection

This aspect of the project was implemented in two different ways. The first way is implemented using the *OpenCV* `cv2` library while the second way uses Tensorflow. The Tensorflow method is clearly the best one among them.

Figure 4: Reference Panorama *(up* and "Enhanced" Panorama *(down)*

3.3.1 Description of the algorithm

OpenCV

To make a person detection algorithm using *OpenCV* on python, we use the cv2 library. This library contains the well-known function `HOGDescriptor()` for the features description. This function implements an Histogram of Oriented Gradients. `HOGDescriptor()` can be associated to `setSVMDetector()` which implements a linear image classification algorithm such as a SVM (Support Vector Machine).

Let's briefly explain how a HOG function works. A feature descriptor is the representation of an image in which all important information has been stored. These important information are features and in this case, these features are the distribution of the gradients in the x and y directions. The advantage of these gradients is that they can describe the appearance and shape of an object using the distribution of intensity gradients.

Each image is divided into 8 cells. In each cell, for each pixel, the gradient is computed and this gradient has a magnitude and a direction. The histogram of each cell can be represented as an array of 9 values. Each of this nine value represent an angle belonging to [0,20,40,...,140,160]. The descriptor is then represented by the concatenation of the histogram of each cell.

Let's explain why using HOG + SVM with *OpenCV* do not give very good results. First of all, it is important to say that the models used are pre-trained. This model works well when humans are from the front or back. However, as soon as they are in profile, they perform very poorly.
This model also tends to frame non-human objects. Missed detection and false detection are common with this model, a trade-off can be found by adjusting the threshold. There are also duplicate detection that can be resolved with the function `non_max_suppression()` that merge boxes. In two consecutive frames, boxes around people can have different sizes. It is not necessarily constant. It is also possible that a person detected in one frame is not going to be detected in the next frame. Nevertheless, these pre-trained models consume less computation power and are very easy to use as they are already implemented in several libraries. This is why another person detection method has been implemented using the Tensorflow library and enhanced.

Tensorflow Object Detection API

As said before, the method mentioned above severely lack accuracy and as a result, a more accurate method using the neural networks was required.
However, training such algorithms is a very difficult task since it requires a lot of labelled data and also

require a lot of parameters and hyper parameters tuning.
After some research, the team found already pretrained model that can be used to perform the person detection such as the Tensorflow Object Detection API [1] which was chosen for this project.

The model itself consist in a **convolutional neural network** pretrained on the COCO dataset [2] and the API provides various implementations for different purposes, mainly for accuracy and speed requirements.

The main advantage of this model is the accuracy which is significantly better than the one obtained with the *OpenCV* method.
The main drawback is that the model can be slow mainly if the GPU acceleration is not available. Fortunately, the Jetson TX2 is compatible with CUDA and can thus provide GPU acceleration which will help improving the performance.

3.3.2 Implementation

OpenCV

The implementation consist in a *Humandetector* class which need a path to the file containing the frames. First, the `HOGDescriptor()` and the `setSVMDetector()` are initialized. And then for each frame, the function `detectMultiScaler()` is used to find the boxes around people. The parameters of this function have to be tuned, in the purpose to have the right balance between speed and accuracy. Finally, the `non_max_suppression()` is used in order to avoid to duplicate the boxes for a single person.

Tensorflow Object Detection API

The implementation consists in a *HumanDetector* class which need a path to the model file to be initialised, it will create a tensorflow session which will be used to perform the computation. This object provide a *detect* method that takes as argument the image to analyse as well as the threshold and return the location of the boxes around the detected humans in the image.

In this project, the model will have to run on an embedded system with real-time constraints, there will thus be speed requirement for the model. In order to select the best model, the performance in terms of speed and accuracy were measured on the already fast model provided by the API and specifically designed to be run on mobile devices as well as a slower model which is more accurate. Here below are the results for the indoor sequence:

Model	Computation time per detection	Total error
ssd_lite_mobilenet	39 ms	68.73
ssd_mobilenet	55 ms	68.24
ssd_inception	65 ms	53.01
faster_rcnn	155 ms	12.77

How the error value is computed will be developed in the next subsection.
As it can be seen, the faster the model, the lower the accuracy since the error increases. Given the results, the team decided to use the model **ssd_lite_mobilenet** provided by the Tensorflow API since it provides the best speed/accuracy trade-off.

Once the model has been chosen, the detection threshold has to be tuned. The model has been run on the dataset with different threshold values. Here below are the results :

Threshold	Total Error
0.6	68.24
0.7	68.24
0.8	68.24

Once again, a detail explanation on how the error is computed can be found in the next subsection.
From those results, it can be seen that the tested values didn't influenced the error. A threshold of 0.8 was selected to ensure the greater confidence of the model on the predictions.

3.3.3 Performances assessment methodology

Once again, the quality of the results will be based on the comparison between the generated rectangles and those of reference, manually calculated.

The final calculated error will also be composed of the same parts as previously introduced when assessing the quality of motion detection: **missing pixels** and **excess pixels**.
This time however, the computation of these pixels will no longer be done by an image comparison but rather by computing the area of the missing rectangles within the reference frame (*missing pixels*) and rectangles exceeding this frame (*pixels in excess*).

Missing pixels : As explained previously, the number of missing pixels is computed by computing the area of the missing rectangles inside the reference frame. Once this value is acquired, the definition of e_1, the error on missing pixels, follows the definition made in section 3.1.3.
Nevertheless, a significant change was introduced. Indeed, this error is mitigated when less than 50% of the missing pixels do not belong to a human. Beyond this value, it is considered that the missing part is too large and does not deserve mitigation.

The calculation of this attenuation is as follows: if less than 50% of the missing pixels do not belong to a human, then the number of missing pixels is reduced such that :

$$p_m = p_m - \frac{(p_m - p_w)}{2}$$

where p_m is the number of pixel missing in the frame, *ie* the area of the rectangle missing and p_w the number of white pixel in this rectangle missing.
p_w is computed using the reference masks introduced before. A little bias can be introduced if the missing rectangle also contains a moving object. The assumption that it doesn't, was still considered as acceptable. The worst that can happen is that a image will need benefice from a attenuation where it should be which has not a important impact on the final result.

Pixels in excess : Here, the selected error simply corresponds to the e_2^* error entered in the 3.1.3 section. In the case of image detection, the question between choosing an error depending on the size of the image or the size of the frame containing the human does not arise. Indeed, the rectangles created should never be far from the reference ones and little noise is possible.

As done in the section 3.1.3, the two errors introduced were linked together using a RMS to punish severe errors. A tolerance was also introduced : if the two rectangles are different by less of 3 pixels, no error will be detected.

3.3.4 Results

With the error evaluation now clearly defined, it is possible to compare the results on the image sequences. To do this, the total error for all images of our indoor and outdoor sequences is calculated and the two tensorflow models are compared. To get the total error, an average is simply realized.

	Tensorflow quick model	Tensorflow slow model
Inside	68.24	12.77
Outside	72.03	7.20

Table 2: Error comparison for the two algorithm

It is apparent that, as expected, the error using the slower tensorflow algorithm is much lower.

4 Contribution

In this section, it will be described how the work was divided among the group members.

Unlike the first part of the project where all group members had worked on almost all modules, the workload was more clearly divided for this part. The reason for this change must be linked to our understanding of the tasks to be performed. Having a better overview of the whole project, it was easier to make a clear division for this part while it seemed easier to work together in the first part.
However, this division remains an abstract idea of how the modules were handled. Nevertheless, some members of the group have also contributed to parts other than their own.
Thus, this delimitation should not be seen as a major decomposition of the work but rather on who were the main contributors of each part.

- **Enhanced panoramic image** : Ben Mariem Sami and Chiodo Adrien
- **Person detection**: Girimeza Guy and Testouri Mehdi
- **Performance assessment**: Derroitte Natan
- **Image annotation, report and poster**: Everyone

As said during the first task, being used to work together, the distribution of the workload was not a problem between us, on the contrary: everyone was able to contribute in a relevant way to this project.

5 Improvements

The next steps that should be taken to improve our results is to focus mainly on the panorama created.

First of all, the blending can be improve. Indeed, the resulting panorama obtain with the alpha-blending technique is blurry. An improvement can be done by sharpen the image in order to remove the blur.
Secondly,

In addition to that, the motion detection could be highly improve. Indeed, it is providing really bad results as the parameters are not tuned correctly.

6 Conclusion

The evolution of this project was marked by the implementation of the various modules of the final application.

In the first part, the video stream acquisition module was first implemented. From there, it was possible to estimate the camera's movement by calculating the angle of rotation it underwent. Then, a first panorama on sequences not containing moving objects was made.
In this second part, three new modules were introduced: the detection of moving objects, the detection of people and a better panorama. The latter can be created from any sequence and create a panorama from which the moving objects have been removed. Finally, particular attention was paid to the assessment of our results, developing in detail how the error on the results is calculated.
As a result, results could be introduced for the different modules.

While the work accomplished leaves room for many improvements, we remain proud of the work accomplished.

Finally, this project was particularly interesting for students in our field. A significant amount of research was required to achieve satisfactory results, allowing us to learn a lot about computer vision.

References

[1] https://github.com/tensorflow/models/tree/master/research/object_detection

[2] http://cocodataset.org/#home

[3] http://eric-yuan.me/image-pyramids/

[4] https://www.pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/