
Electronic, Telecomunication Engineering and Computer Engineering Department

Image Processing

# Motion Detection for Object Tracking

*Professor*
Pedro Mendes Jorge

*Student*
Ferro Enrico
Number 44566

Academic Year 2017/2018

# Contents

# Chapter 1

# Introduction

The computer vision application, described in this report, is able to detect, classify and track objects that appear in a video sequence typical from a surveillance environment.
It is supposed to have a fixed camera that acquires images in a parking and a real-time software can detect movements and can recognize veichles and people and to do this the background subtraction method is performed.
Furthermore a matches matrix is used in the case of vehicles or people, with this matrix the software can also tracks the movement and show how the object moves in the consecutives frames, labeling the rectangle surrounding it with a unique name.
The application was written in Python and was used the specific image processing library called OpenCV.

# Chapter 2

# Software description

## 2.1 User's use

To use this software the user needs python 3.5 with two libraries installed: openCV_3.3 (an image processing library) and Numpy (fundamental package for scientific computing with Python). The algorithm is incapsulated in a single function with the following interface: *outVideo* = **MotionDetection**(*inVideo*, *firstFrame*, *lastFrame*), where *inVideo* is a string with a file name containing the input video stream, *firstFrame* and *lastFrame* are int value that indicates respectively the first and the last frame to be processed, and *outVideo* is a string with the file name of the output video containing the results of the algorithm. Furthermore, if the user inserts the incorrect values about the frames, the software will show an error.
After executing the function, the software will produce the processed video starting from *firstFrame* up to *lastFrame* and this video will be in AVI format.

## 2.2 Additional functions

The previous section describes the basic use that the user can do of this application, but if the user open the code he can see how there are some commented rows. If he removes the hashtag from these, he can see the data and images that describe the step-by-step algorithm. These features are described in detail in the next chapter.

# Chapter 3

# Implemented algorithm

This chapter describes the project choices made and the various steps that the software performs to recognize people and veichles in the video.
Fundamentally the software finds the movements in the environment through a background-subtraction, then the various regions are detected and classified and finally the movement tracking is performed.

## 3.1 Estimation of background image

In this application, it is essential to perform a subtraction of the background in the best way, for this reason, below, three different approaches tested are proposed, but we want to point out to the reader how only the third is used in the software.

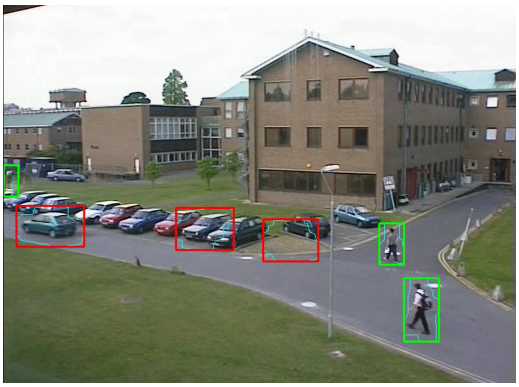
### 3.1.1 Background with the first frame of the video

The first solution adopted was to use the first frame as a background and then each successive frame was subtracted from this.
This is the simplest solution from an implementation point of view, but it turns out to be the worst from the point of view of results as it presents mainly a critical problem. With this solution, if an object that was present in the frame selected as a background moves from the initial position, the system will continue to detect at this point a pixel difference and then there would be active pixels as you can see in figure 3.1. The solution for this problem is represented by the background update. This technique allows you to consider changes in the environment and to change the background in relation to these.

### 3.1.2 Multi-Frame Average Method

The multi-frame average method [1] is based on *cv2.accumulateWeighted* function [2][3] and it's possible to describe as follows. For an image sequence $B_i$ i=1,...,n $B_n = \frac{\sum_{i=1}^{n} B_i}{n}$.
Indeed, the function offers the possibility of using a weighted average and not just an arithmetic average, but the basic concept is not very different, except that it is possible to vary the weight that each frame takes, with the consequence that this frame will affect the final result based on how much weight it takes. In figure 3.2 you can see, the same frame, but with 2 different values of average weight. The difference is quite clear: in figure 3.2(a), the movement "remains in memory" for a shorter time, but this leads to a limited detection of the active pixels; in figure (b) instead, there is more memory and the detection of the movement is better, but for a few

(a)



(b)

Figure 3.1: *Frame chosen as a background(a). In figure b you can see an example of how although there is no movement, active pixels are detected in the parking where the blue car was.*

seconds it may happen that areas where there is no change (the parking space left empty for example) are identified as active pixels. This method have 2 problems:

(a)

(b)

Figure 3.2: *Average weight 0.05 (a) and 0.005 (b)*

- if the number of frames, $n$, is large, a lot of memory is required to store the image sequence $B_i$.
- it is difficult to determine the threshold value as this needs to be regulated, depending on the degree of environmental change.

### 3.1.3 BackgroundSubtractorMOG2

Mixture of Gaussians is a widely used approach for background modeling to detect moving objects from static cameras[4]. It uses a method to model each background pixel by a mixture of K Gaussian distributions. The weights of the mixture represent the time proportions that those colours stay in the scene. The probable background colours are the ones which stay longer and more static.

In OpenCV 3.3, the *cv2.createBackgroundSubtractorMOG2* function [5] implements this algorithm which solves at least one of the problems of the average multi-frame method because the memory usage is less and the computation is optimized. The function selects the appropriate number of Gaussian distribution for each pixel and this allows to work more easily with a fixed threshold value since in any case the function will perform a sort of automatic correction. The threshold value chose in this software is 40. The value has been chosen deliberately low because the vehicles can be dark colored, just as it is possible for people to wear dark clothes. The reader notes that this value is suitable for the video used as a test but it is not said that it is the best one for any environment, and therefore it will have to be evaluated case by case which can be the optimal value to be used.
The OpenCV's function offers the possibility to detect the shadows through the *detectShadows* parameter, but in this case the value is set to *False* because it wasn't interesting.
Finally it's also possible to set the parameter *setHistory*, the number of last frames that affect the background model, that in this case it is set at 1000, but also for this parameter the value is set according to the test video used.

## 3.2 Active pixels detections and noise removal

With the background image is found, to determine the active pixels, it is necessary to subtract the background with the frame being analyzed. If the *cv2.createBackgroundSubtractorMOG2()* function is used, to perform the subtraction, it will be enough to apply the *apply()* method to the object of interest, where the parameter to insert will be the frame of interest, an example of such operation is visible in figure 3.3. If instead you decide to acquire the background with one of the first two methodologies illustrated, it is necessary to use the *cv2.absdiff()* function [6] which allows to obtain the active pixels, as in the previous case.
Looking the figure 3.3, it can be seen that there is some noise as there are some white dots

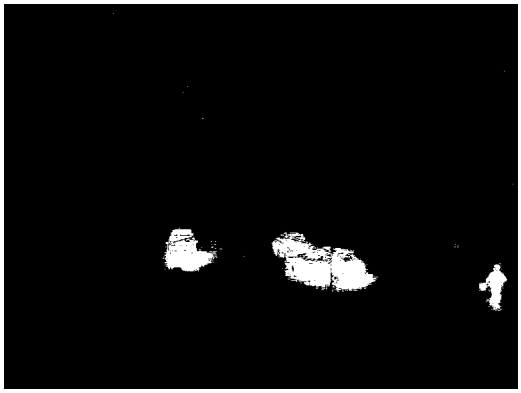
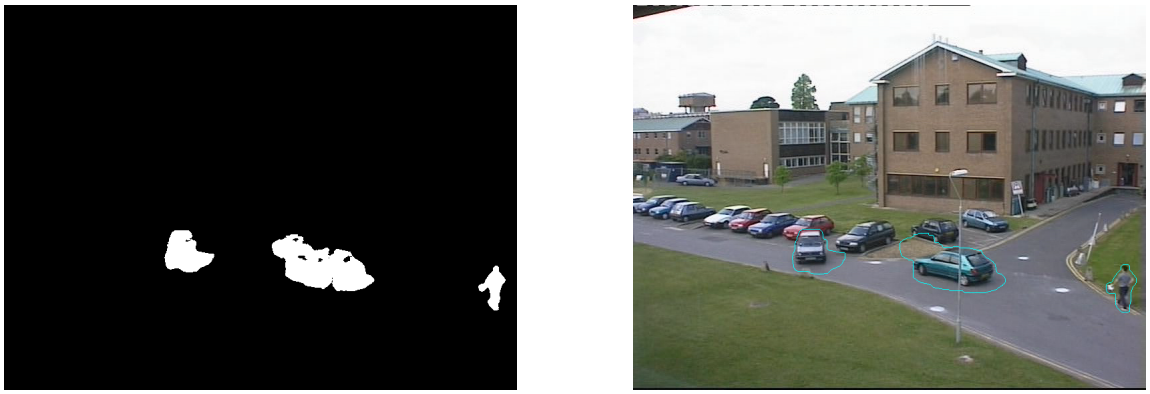


Figure 3.3: *Example of background subtraction*

present. To remove this effect, which could cause false detection of movement, it is necessary to apply a filter and in this case we used *cv2.medianBlur()* [7], where the value of the parameter used to define the size of the filter it will depend on the type of video that you intend to process.

## 3.3 Application of morphological operators and active regions detection

Once the background has been subtracted, it is necessary to identify the various regions formed by the active pixels in such a way as to be able to classify them later. For each mask created with the background subtraction two morphological operations are then applied: *dilate* and *erode* [8]. Both morphological operations are performed with a rectangular structuring element as then vehicles and people will be considered as regions within a rectangle.
After obtaining the regions, it is possible to apply the function *cv2.findContours* [9] which allows to find the contour of a region, a fundamental step to proceed with the classification.
In the figures 3.4 it is possible to observe an example of these two operations: the same frame of the figure 3.3 after the application of the median filter and of the morphological operations assumes the aspect of figure 3.4 (a) and from this the contours are found and then draw in Figure 3.4 (b).



(a) (b)

Figure 3.4: *Regions after morphological operations (a) and contour detection (b).*

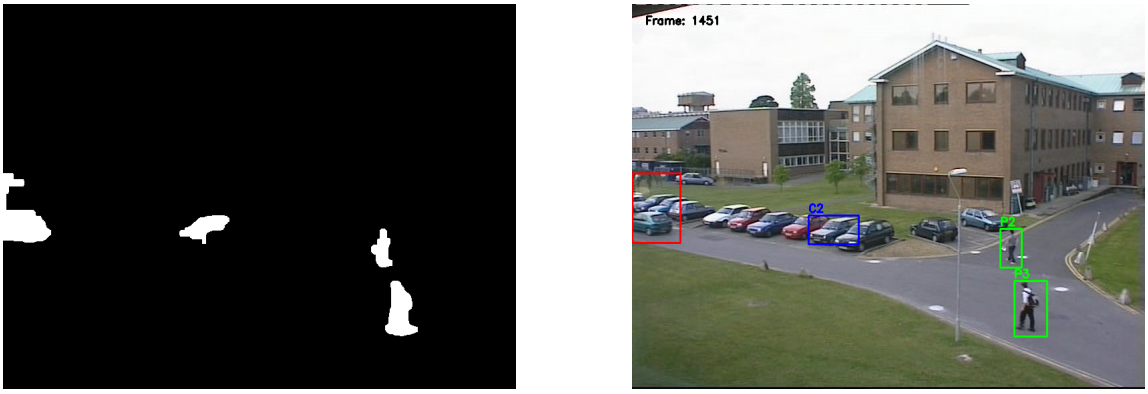
## 3.4 Classification

This section presents one of the crucial parts of the algorithm: the classification of the various movements. This phase is essential to recognize if the moving object is a vehicle, a person, or if there is another type of movement. Please note that the choices made for classification apply to the test video used and may not be valid for other scenarios.
To classify the movements, the algorithm analyzes one by one the previously detected regions and for each of them, through the function *cv2.boundingRect()* [10], creates the parameters that will allow then, thanks to the function *cv2.rectangle()*, to draw the appropriate rectangle around each of them.

### 3.4.1 50 pixels elimination on the right side

First of all it was decided to classify the movements, distinguishing them in vehicles, people and other, excluding all the regions that had the X coordinate of their centroid less than 50, in fact all sectors having centroid with X between 0 and 50 are detected but are always classified as "other" and then surrounded by a red rectangle.

The reason may be noted in Figures 3.5: in the left side of the image there is a branch of a tree which inevitably oscillates. This movement is obviously detected producing active pixels, which if in that moment come into contact with active pixels corresponding to another movement (in the example the car that is coming out of the scene), a single region is formed by creating a region that would be classified incorrectly.




(a) (b)

Figure 3.5: *Problem created by branch oscillation. In figure (a) it is possible to observe how a single region of pixels active between branch and vehicle is created.*

### 3.4.2 Distinction between vehicles, people and other movements

The distinction between vehicles and people is essentially based on a very simple concept: if a region of active pixels has the height greater than the width then it will be classified as a person, otherwise as a vehicle.
This assumption would however lead to classifying all movements as persons or vehicles, thus excluding "unidentified movements". Indeed, the software classifies the people if, in addition to the requirement to have the height greater than the width, they also have a value of these parameters above 15 and analogous speech is made for vehicles, where the threshold is 20.
The choice of the two thresholds was made again based on the test video and trying out various values, which could be invalid in the case of a different environment.

## 3.5 Movement tracking: matching regions detected in consecutive frames

Movement tracking is a technique in surveillance applications that allows you to recognize a region of active pixels in consecutive frames and that then allows different applications such as the design of the trajectory of the moving object or for example, in this case, the assignment of a unique name for each type of object.
The basic idea is the fact that a region moving within the environment, in two consecutive frames, will move only in a few pixels.
From the software point of view this idea is essentially realized with a matrix that memorizes the positions of the moving objects present in the environment and updates each frame so as to memorize the current position of the considered object [11].

In the code there are two distinct matrices: one for vehicles and one for people. However, their structure and functioning is equal, so the following applies to both.

### 3.5.1 Matrix structure

When the matrix is initialized it is composed of a single row and 6 columns and while the number of columns remains fixed, the number of rows is updated as the objects enter and leave the visual field of the camera, what is then obtained is a 6 columns table in which the various moving objects are saved.
The 6 columns represent respectively:

- X coordinate of the centroid of the rectangle surrounding the region, int value
- Y coordinate of the centroid of the rectangle surrounding the region, int value
- name of the region, int value
- number of frames in which the region appears
- column that identifies whether a name has already been assigned to the region or not. The parameter inside this column is an int used however as a boolean as it can only have values 0 and 1.
- last frame in which the region appears

### 3.5.2 Adding objects to the matrix

Updating the matrix data is the most important part regarding the movement tracking process and basically the update can be divided into 3 different conditions: the region is already in the matrix, the region is not yet in the matrix , the region is in the matrix but does not appear in the image.
For each region, to check if it is in the matrix or not, the whole matrix is scanned starting from the last row. We chose to scan the matrix starting from its last element as it is much more probable that the elements present in the image have been recently added in it and therefore they are in the last lines. A region is already present if the coordinates of its centroids are close to those of one of the rows of the matrix, to determine this, for each pair of coordinates (region that is being analyzed and matrix row), an absolute value subtraction is carried out and if the result is lower than a threshold, then it can be said that the analyzed region was present in the previous frame. Two different thresholds have been chosen to check the matrix membership: the value for the matrix referred to the vehicles is higher because a car moves faster than a person and therefore between a frame and another its centroid may have been moved a greater number of pixels. If a region is already present in the matrix the software updates, in the row of interest, the coordinates (columns 1 and 2), the current frame in which it is present (column 6) and it increase the number of frames in which it appears (column 4). If the region is found, the software checks for how many frames this region has been present in the environment, this check is to avoid that values are generated for movements erroneously detected by the application, such as the one shown in figure 3.6. If the region is present by a number of frames greater than the set threshold, the fifth column of the matrix is checked, which determines if a region has already been assigned a name or not, if this has not been done the system proceed with the assignment of a new one that will be unique. The assignment of the name in this case is trivial in that it increase to each new person or vehicle, a special variable that acts as a counter and take this value (there is a counter for people and one for vehicles). Finally, if the region is

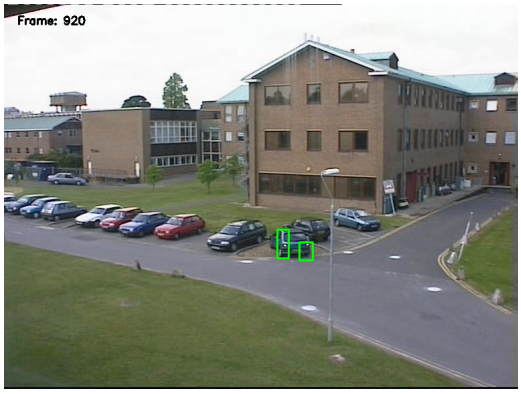



Figure 3.6: *In the image you can see how the car in this frame is detected as two detached regions that the system classifies as people: without the appropriate countermeasures this could lead to errors.*

present in the matrix, the scan of the latter is interrupted and the boolean variable *find* is set to the value *True*.
If, on the other hand, no correspondence is found in the matrix, it means that a new object has entered the visual field of the camera and then the application proceeds by adding a new row to the corresponding matrix.

### 3.5.3 Deleting objects to the matrix

Finally, it is necessary to add a mechanism for deleting data from the matrix to prevent new objects entering the scene from taking the name of objects that went out from the point where the new object is entered. The problem especially concerns vehicles, as they enter and exit only from two points in the test environment. In fact without a mechanism that can take into account this fact, if for example a car leaves the scene through the road to the left of the camera and shortly after another car enters the environment from the same point, the latter, will take the same name of the exit car, as the last coordinates saved on the matrix referring to the first vehicle will be "close" to the first detected ones of the new vehicle.
To avoid this, at each frame, the two matrices are scanned and all values where the difference between the current frame and the last frame in which a given region appears is greater than a threshold, are eliminated.

# Chapter 4

# Software limitations and results

As repeated several times in the description of the project, the software in question obtains good results when it is used with the video test, but can not guarantee the same for other environments. The following are the main application limits and the results obtained in the test video.

## 4.1 Values of the thresholds

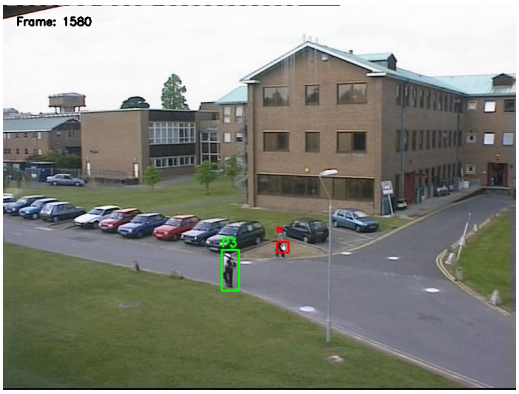
The main problem that could be had by using the software in a different environment is due to the fact that all the various thresholds set in the various conditional choice points are not consistent with what the camera is observing. In particular:

- the binarization threshold depends strongly on the environment and on what is presumed will move in it (see the following section for more details);
- as regards the minimum size of people and vehicles, as this depends on how the camera is positioned and the distance at which the various movements are generally detected (see the section 4.3 for more details);
- maximum distance in pixels that there can be in two consecutive frames for the same object and this depends on how fast the objects in the scene can generally move. In this context it is a parking and therefore people generally walk and vehicles keep a moderate speed, but it is obvious that for example if you were monitoring an highway the vehicles would pass at a much higher speed;
- a similar reasoning to the previous applies to the threshold that determines whether a data have to be deleted from the matrix or not. In the current software a value of 150 frames is set (6 seconds since the video is at 25 fps), but it is normal that in different contexts this value may not provide good results.

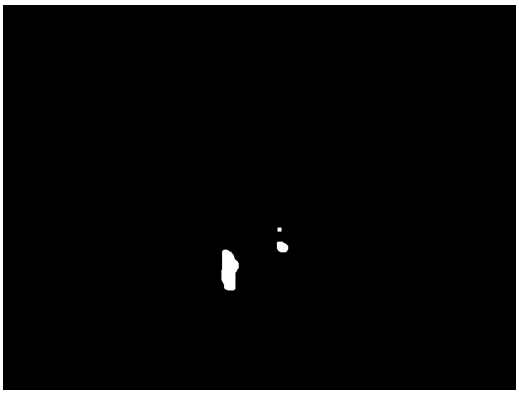
## 4.2 Binarization threshold and environment lights

The chosen binarization threshold is fundamental for the detection of active pixels. In this case the chosen value is very low as vehicles can be painted in dark colors and people can wear dark clothes. Nevertheless it can happen that in some frames the binarization fails to acquire exactly all the movements and this could lead for a few seconds to the creation of wrong rectangles as in the example of figure 4.1 (a), where the person is not correctly detected. Also for this reason it is essential to keep a fairly high value as regards the threshold that in the

tracking process determines if a region is already present in the matrix or not, since it may happen that for some frames there is a sort of "loss of signal" for a particular region. Obviously




(a)



(b)

Figure 4.1: *Example of a frame in which the binarization does not detect all active pixels and generate unwanted rectangles.*

regarding the choice of the binarization threshold it is very important also to consider the lights of the environment. In this case this is not considered, as there is no drastic change in light in the test video, but, especially in the case of outdoor environments, this is a factor that should not be neglected, as climate change and difference between day and night create big differences regarding the lights.

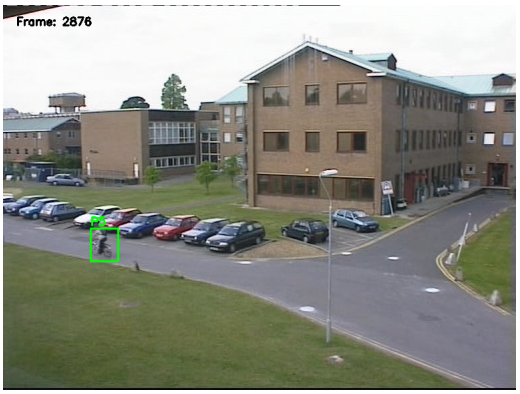
## 4.3 Camera position

Another very important aspect is the position of the camera. The aspects discussed in the section 4.1 depend mainly on this, in fact it is very important to take into account how the objects vary their size and their speed in relation to the distance they have from the camera, due to the perspective.
An example of how this can lead to problems is given by the guy who rides his bike in the test video. When he enters in the scene he is detected, by the software, as a person because the surrounding rectangle is taller than wide, although it would not be strictly correct to classify him as a person. To solve this, one might think of limiting the height and width of a person to a certain value, but indeed the problem would be repeated when the boy is further from the camera, as he moves away, his sizes decrease, as can be seen in figures 4.2. One could also think to use the fact that the guy on a bike is theoretically faster than a person on foot, but even in this case, moving away from the camera the speed, that the camera could detect, decreases and would no longer be a reliable value.

## 4.4 Intersection between different objects

Finally the last important consideration to make regarding this software concerns the case in which two objects cross each other. As can be seen in Figure 4.3 (b) if this happens it is inevitable that one creates a single region of active pixels that is nothing else the intersection between the two regions. The biggest problem, however, is represented by the fact that the tracking of the car that is initially called *C1*, after the intersection of the regions, becomes *C4*.

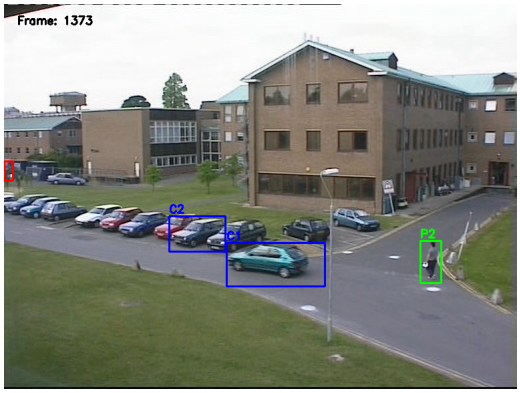

(a)



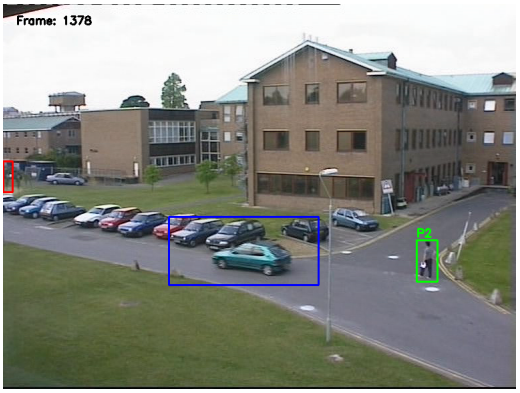

(b)

Figure 4.2: *In the figures it is possible to notice how, due to the effect of the perspective, the dimensions of the cyclist decrease as he moves away from the camera.*

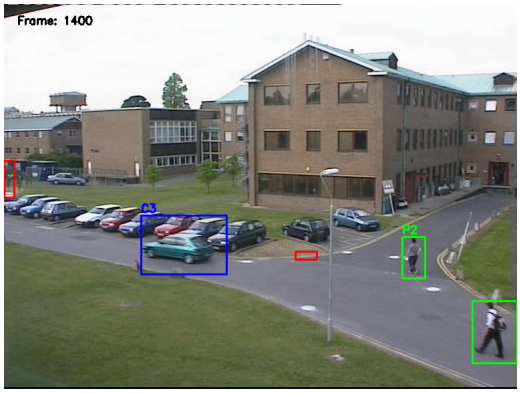
This is due to the fact that the separation of the two regions brings the blue car back to being a single region when it is too far from the last point where was called as *C1* and therefore no match is found in the matrix for tracking (which is not done for *C2*). In addition, the region created when the two cars are close is erroneously cataloged *C3* and this would be avoidable if you enter the maximum measures that a vehicle can reach, but could create other problems when a car changes its size quickly just like the blue car when it comes out of the parking. In this case, in fact, the active pixels considered are many because the frequency of updating the background is not high enough to perceive the change of direction.



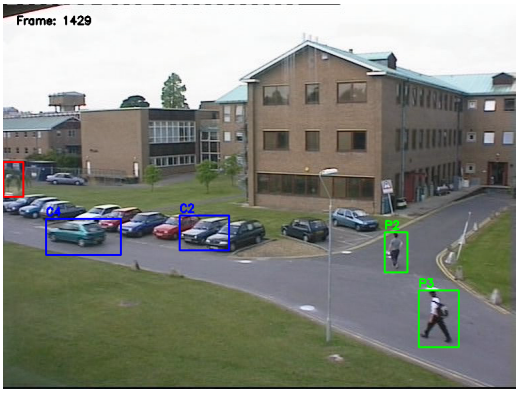

(a)




(b)




(c)




(d)

Figure 4.3: *Frame sequence when two vehicles cross each other*

# Chapter 5

# Conclusions

This project proposes a real-time detection method of moving objects that combine background subtraction, background updating, motion object classification and motion tracking that could be ideal for a surveillance system.
As is continually repeated in the report, the system offers good results for the environment in which the simulation of use is performed, but can not guarantee the same for different environments. However, the software offers a good starting point for adapting the algorithm to different environments and situations provided that the various parameters are set correctly and the necessary considerations are made.

# Chapter 6

# References


1. "A Detection System for Human Abnormal Behavior", *IEEE International Conference on Intelligent Robot Systems*, Xinyu Wu, Yongsheng Ou, Huihuan Qian and Yangsheng Xu.
2. `https://docs.opencv.org/2.4/modules/imgproc/doc/motion_analysis_and_object_tracking.html`
3. "BackGround Extraction using Running Average", Abid Rahman K.
4. "Adaptive background mixture models for real-time tracking", *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on*, Chris Stauffer, W. Eric L. Grimson, 1999.
5. `https://docs.opencv.org/3.1.0/db/d5c/tutorial_py_bg_subtraction.html`
6. `https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html`
7. `https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html`
8. `https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html`
9. `https://docs.opencv.org/3.3.1/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf`
10. `https://docs.opencv.org/3.1.0/d3/dc0/group__imgproc__shape.html#gacb413ddce8e48ff3ca61e`
11. "People tracking in surveillance applications", *Proceedings 2nd IEEE Int. Workshop on PETS, Kauai, Hawaii, USA*, Luis M. Fuentes and Sergio A. Velastin, December 9 2001


You can also see:


- Pedro Mendes Jorge, "Image Processing transparence", academic year 2017/2018.
- L. Shapiro, G. Stockman, "Computer Vision", 2001.