CAMERA-BASED DRIVER ASSISTANCE AND CAR SECURITY SYSTEM

EVIO ABAZI

BACHELOR OF SCIENCE IN COMPUTER ENGINEERING

FACULTY OF ARCHITECTURE AND ENGINEERING

EPOKA UNIVERSITY

JUNE 2015

CAMERA-BASED DRIVER ASSISTANCE AND CAR SECURITY SYSTEM

EVIO ABAZI

Thesis submitted to the Faculty of Architecture and Engineering,

Epoka University,

In Fulfilment of the Requirements for the Degree of Bachelor Studies

June 2015

To my parents for their endless love, support and sacrifice.

ABSTRACT

Faculty of Architecture and Engineering

Advisor: Igli Hakrama

Visual sensors are going to have a very important role in the driver assistance systems. Nowadays several car models equipped with cameras have been presented with the aim of facilitating the process of parking. Certainly, these cameras could be used for other tasks with the objective to offer a better assistance to the drivers. This paper describes a system that is able to recognize, in real-time, people and traffic lights, warning the driver of their presence. In addition with the rotation of the camera towards the inside of the car, it is implemented a security system that tracks the motions and informs the driver of suspicious movements during his absence. The system is designed using simple standard hardware and the computer vision library OpenCV combined to libraries created by Processing Foundation.

Keywords: detection, pedestrian, system, traffic light, motion, security, OpenCV

ABSTRAKT

Sensorët vizualë kanë filluar të kenë një rol të rëndësishëm në sistemet e asistencës së drejtuesit të mjetit. Deri më tani janë prezantuar disa modele makinash të paisura me kamera me qëllimin e ndihmës në procesin e parkimit. Sigurisht që këto kamera mund të përdoren për qëllime të tjera me objektiv asistencën maksimale për drejtuesin e mjetit. Ky punim përshkruan një sistem i cili është në gjëndje të njohë, në kohë reale, njerëzit dhe dritat e trafikut, duke njoftuar drejtuesin e mjetit për prezencën e tyre. Gjithashtu me rrotullimin e kameres në drejtim të brendshëm të makinës , është implementuar një sistem sigurie që gjurmon lëvizjet brenda makinës dhe informon drejtuesin për lëvizje të dyshimta gjatë mungeses së tij . Ky sistem është projektuar duke përdorur harduerë standart dhe librarinë OpenCV të kombinuar me libraritë e krijuara nga Processing Foundation.

Fjalë kyqe: zbulim, kalimtar,sistem, drita trafiku, lëvizje , sigurie, OpenCV

ACKNOWLEDGMENTS

I would like to thank my supervisor Mr. Igli Hakrama for his enormous help and support, for inspiring and motivating me to do my best for this thesis. Also, I would like to express my gratitude to my department advisor Mr. Ibrahim Mesecan for his advices and support during these three years.

DECLARATION

I hereby declare that the thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Epoka University or other institutions.

Evio Abazi

25 June 2015

Table of Contents

CHAPTER 1

Introduction

In the recent decades, there has been a rapid development of the computer vision that is attempting to mimic the ability of a perceptual user to reconstruct the three-dimensional environment. New methods and algorithms in object recognition and reconstruction of scenes and 3D models are opening the way for new applications. In the industrial field, it is establishing the use of computational vision systems offering:

- quality control, in order to identify defects and ensure the respect tolerance;
- recognition of objects for classification;
- navigation and coordination of machines through visual enslavement;

In addition, with the ability to recognize objects, equally important applications are made in video surveillance, in robotics, in the development of artificial intelligence and virtual reality.

The wide variety of applications that is emerging relies on the fact that the capture of information from the environment is increasingly rapidly, efficiently and with a low cost. The increasing processing power of the processors of our PCs and the reduction of the cost of webcams makes it accessible to both researchers and students the instrumentation that was once the prerogative only for those who were specialized in the field. The student who is not yet expert in these techniques has also no difficulty in setting the instrumentation for the development of a vision system, so his efforts will be directed to the understanding of the mathematical aspects and implementation of the algorithms behind the computer vision.

To make it even more affordable for everyone the machine vision, the attention has shifted to the use of the software and its immediate understanding. In this direction are the development of the OpenCV libraries and other software as Image Processing Toolbox for Matlab that facilitate the approach to a subject that can be tricky because of the complexity of the algorithms and he deep geometric basics [1]. This software hides to the user the most difficult aspects and moves the commitment to the development of new applications. This project is entirely based on OpenCV and its secondary libraries; all implemented in Java Programming Language, offering a recognition system that is perfectly working in daily environment.

CHAPTER 2

Literature Review

In this chapter are presented the results from the research done in order to meet the requirements for the implementation of the system. Several motion detection and recognition systems have been researched and implemented by other researchers and as result there are many libraries of programming functions for implementing in real time computer vision. There are no limitations: these libraries offer high performance low-level routines for imaging, video, audio, signal processing and codecs.

It is also needed a trained detector based on HOG and SVM and both OpenCV and Matlab offer this feature, as well as other libraries of computer vision. The choice is wide, but there are some important features that differentiate these libraries: one of this is speed. The system will work in real time, speed performance is one of the most important requirements to have a fully functional system without delays and OpenCV is really fast, offering at least 20 frames analyzed per second while other libraries or Matlab can't reach the half of it. Another requirement to be considered are the resources needed, OpenCV require less than a hundred MB of RAM to run in real-time [1]. One last factor that made OpenCV to be the final choice is cost: it is totally free and, at the end, it reduces the total cost for the implementation and production of the system.

2.1 Computer Vision

Computer Vision is the set of processes that aims to create an approximate model of the real world (3D) starting from two-dimensional images (2D). The main purpose of artificial vision is to reproduce the human vision, when vision is not intended only as the acquisition

of a two-dimensional picture of a region, but especially as the interpretation of the content of that region. Information is intended in this case as something that implies an automatic decision. The purpose of computer vision is to program a computer, which can understand a scene or the characteristics of an image.

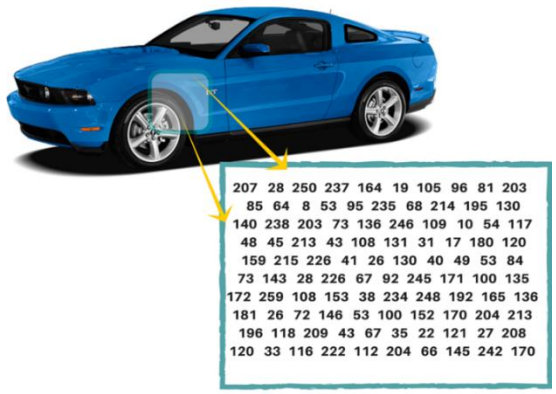


Figure 1: machine vision of objects.

A classic problem in computer vision is to determine whether the image contains or not certain objects (object recognition). The problem can be solved effectively and without any difficulty for specific objects in specific situations, such as the recognition of specific geometric objects, numbers or characters. Things get complicated in the case of arbitrary objects in arbitrary situations.

The difference between human capabilities and those of the machines is not in the number of information processed, but in the ability to integrate them. A modern system of artificial vision is now able to integrate them. A modern system of artificial vision is now able to distinguish a variety of objects in a scene with a fair degree of complexity, but is not able to put them in relation to each other and to compare them with other images seen

previously. The human brain correlates information at multiple levels; not only distinguishes color, location and form of a single object, but combines all information into a single scene from which extracts the information that it considers more important at that time.

2.2 Computer Vision Libraries

Most of the work related to the design of the software present on the market are done with Open Source libraries. The library allows the developer to obtain two important things: to reduce the time for the development and to simplify the conceptual design. Many algorithms used in image processing are heavily defined by mathematical models specific to some theory. Within this sector, there exists an infinite of mathematical conversions that are used for the reduction of the amount of data that must be treated. The duty to translate into mathematical algorithms such systems would claim a very long time writing and debugging.

Generally, it can be identified at least three categories of libraries used for different purposes:

- *Toolkits*, primitive libraries for creating graphical objects interface;
- *Multimedia and rendering libraries*, as DirectX and OpenGL, focused on maximum performance in creating vector or polygonal effects whose more common utilization is to obtain high performance graphics utilized in games or multimedia applications;
- *Libraries for graphic hardware management* as digitalization and frame grabber (OpenCV, BoofCV, OpenSURF, ImageJ, Pan-o-Matic, Reference, JavaSURF).

The system presented is done using one of the last category of libraries. The choice for OpenCV was made after a careful study performed by Peter Abeles ("Speeding UP SURF – 9th International Symposium on Visual Computing, 2013"), where after benchmarks OpenCV resulted the more stable for real-time detection configured also to use multi-threaded code.

Implementation	**Version**	**Programming Language**	**Threaded**
BoofCV-F	0.5	Java	No
BoofCV-M	0.5	Java	No
OpenSURF	27/05/2010	C++	No
Reference	1.09	C++	No
JOpenSURF	SVN r24	Java	No
OpenCV	2.3.1	C++	Yes
Pan-o-Matic	0.9.4	C++	No

2.3 OpenCV

OpenCV was launched in 1999 by Intel and it consists in a collection of C functions for artificial vision, whose main aim is to extract meaningful data from images and treatable in an automatic way. This field of study has its applications in robotics, in the systems of video surveillance or advance monitoring systems and security, as well as in any system of automatic archiving of visual information.

The library is mainly oriented to the real-time vision, using a set of routines with low overhead and high performance. For offering this, it is optimized for Intel architectures and extensively utilizes the MMX technology[1]. In fact, more than half of the functions are

[1] MMX is a Pentium microprocessor from Intel that is designed to run faster when playing multimedia applications

optimized for this purpose. It is compatible with IPL (Image Processing Library) of Intel, and Open Source library that perform low-level operations on digital images, OpenCV is more focused on high-level algorithms, such as calibration techniques, feature detection, object tracking (via optical flow), morphological analysis (geometry and contour analysis), motion analysis, 3D reconstruction and segmentation of objects.

These are some interesting features of OpenCV:

- it is optimized for Intel architectures and, although this may be seen as a limit to the generality of what will be developed based on the library, actually, considering the scope of future programs and the optimization level reached, it can be considered a point of force, given that the majority of machines are based on Intel architecture;
- It offers many high-level functions that simplify the development of complex applications, allowing the developer to focus on core issues and not on those standards that have already been solved effectively;
- it is actively developed and updated, this thesis is based on the last Beta Version of OpenCV 3.0;
- the user community is very large and active, and many common problems encountered have already been solved;
- the documentation offered is the best among of the open source libraries;
- There are a large number of examples, which cover the use of many basic functions.

For these reasons and the reasons discussed at the beginning of the chapter, OpenCV is often chosen as library for the development of other high-level applications.

2.4 Open Processing

Processing is an Open Source programming language created in 2001 and supported by an online community. It allows creating a system that integrates Java with the principal concepts of motion, interaction and visual form. With the born of Processing Foundation in 2012, it also offers a software sketchbook that can be used as development environment (prototyping and production) for any category of developer.

The main objectives of Processing language is to integrate the libraries functionality with the production tool and generate high-level programs containing image processing, models, events, sounds, 2d and 3D file formats. In this project are used Processing libraries combined with OpenCV, to have an easier implementation of pedestrian detection using HOG.

CHAPTER 3

Problem and solution

Driving in Albania is an adventure. Even the most experienced drivers can be distracted and not pay attention to the traffic light that is going to be red or to a pedestrian that suddenly crosses the road where he should not. In addition, that goes both ways, pedestrians also have to be in constant alert, they have to look everywhere in order to cross the road. However, in Albania it is not only bad drivers and pedestrians that you have to worry about, it is the actual roads themselves.

And these are only a part of the causes of accidents which according to INSTAT cause a total of 590 deaths per year ,where drivers themselves are the most endangered,273 deaths per year , followed by passengers with 162, pedestrians 90 deaths per year and other road users with 79 deaths per year.

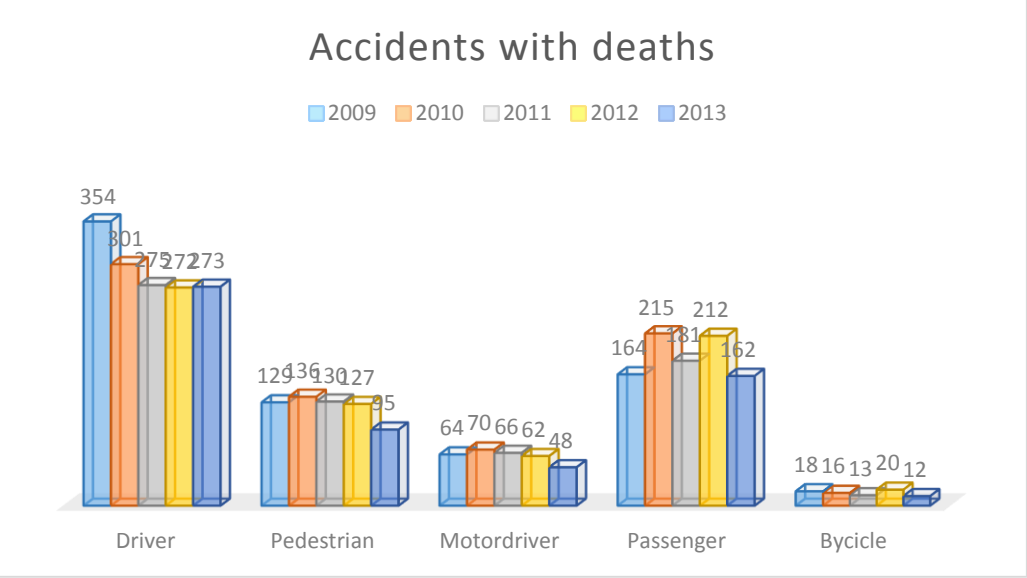


Figure 2: statistics of accidents [2]

These problems have been afflicting the Albanian Roads for years. Many solutions have been proposed such as the one adopted by the Transport Minister to increase disproportionately the fines for red light, does not work properly: the drivers try at the most, but the distractions are so many that it does not take much to break the rule and cause an accident.

However there is not much you can do about the counditions of the roads as that is under the control of the Ministry of Transport , but something can be done in assisting the driver. And this assistace can be in the form of detecting the tragic light and signals of the road, which is a very problematic situation especially in the cross roads, and also to detect the pedestrians that may cross the road at any time which may put not only themselves in danger but also the people around them .

The solution proposed is a system that with a simple HD camera, in real-time, tracks the traffic light and alerts the driver in base of the color of the traffic light avoiding any trespassing caused by distractions. In addition the pedestrian detection is done at a distance to allow the driver to curb and avoid any accident. This way we could help minimize the risks of any accident caused by distractions.

Another problem to which it have attempted to give a solution is the continuous reporting of thefts and damages to vehicles when parked and left unattended.

The proposed solution for this problem consists in the detection of motions inside the car and in case of risk report a photo to the owner of the captured by email.

CHAPTER 4

Implementation of the solution

The implementation is done in Java Programming Language. Kalman filter and Hungarian algorithm are used for motion detection: the position predictions of the objects are done by Hungarian algorithm, while the state estimation is performed using Kalman filter algorithm. As hardware is used a Raspnerry Pi B which offers the hardware requirements necessary to support and allows the system to operate optimally. The videos are captured using a normal HD camera with 30fps.

4.1 Traffic Light detection

A system that helps the driver to detect signal lights could be a method for improving safety driving. It is sufficient a minimum of inattention to create a car accident. Traffic light detection and signalization can be a good support for any driver, especially in urban environments where there are many obstacles can distract the driver , who has to pay attention to pedestrians, road signs, cars and other moving obstacles, all done in real time. The feature of the proposed system requires only a basic hardware: a simple HD camera (720p) mounted on a position to guarantee a wide view of the road.

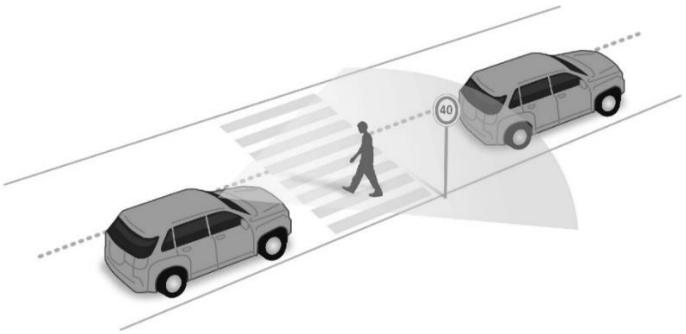


Figure 3: view of the camera

4.1.1 Implementation

- *Connect the camera*

The camera is connected to the system through an USB 2.0 port, no drivers will be needed and the device will be automatically recognized by the system. The quality of the camera, obviously, will affect the operation of the system, the best choice is a compromise between high resolution for better recognition and less load for the processor of the Raspberry Pi.

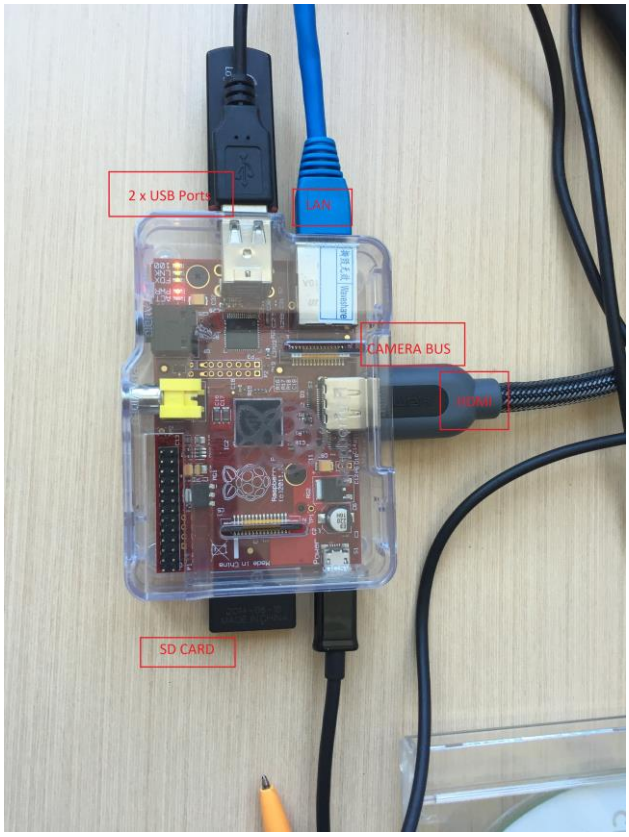


Figure 4: Raspberry Pi ports

- *Obtain frame*

  Code:

```
opencv_core.IplImage image;
```

Initialize an IPLImage using opencv_core.IplImage that will be used to store each frame of the video. Intel Image Processing Library has an image structure that stores all the information of the frame (depth, how many bits per channel, numChannels, width, height). In OpenCV it is used FrameGrabber extracts the frames from the video. Once a FrameGrabber object is initialized with the CameraNum (0), the video streaming will start and the read will be done frame by frame, in real time. Each frame will be assigned to the IPLImage previously declared.

Code:

```
FrameGrabber = capture = new OpenCVFrameGrabber(0);
capture.start();
```

To analyze in details how the system will recognize the traffic light color, it will be considered only one frame, but normally this procedure will be applied to each frame of the video offering a continuous recognition during the video streaming in real time.

Figure 5: frame of the video

It there is no need to specify the dimension of the frame grabbed because it is automatically determined in base of the capture device and stored in the image structure (in case of a webcam with 1.3 Mpx with an HD resolution – the dimensions will be 1280x720 pixels).

- *Convert color space from BGR to HSV format*

The default format of OpenCV is not Red, Green, Blue (RGB) but it is an inverted one: BGR. BGR indicates the amount of blue, green and red color that are present in an image. There is also HSV (Hue, Saturation, Value) color space that offers a better color recognition, which is similar to the human eye perception. For this reason, HSV is a better option for the traffic light color recognition, offering so better results. The frame grabber capture the image in BGR color space; it is required a conversion from BGR to HSV.

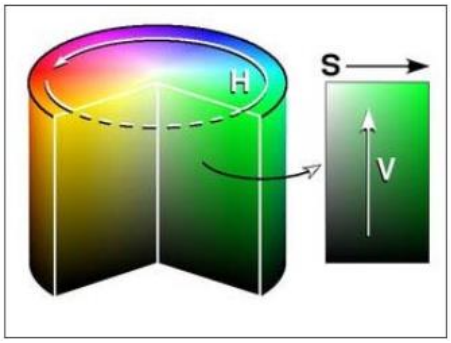


Figure 6: HSV color space representation [3]

Hue can be represented as a circle and it defines the color type, it has a range from 0 to 255. Saturation defines the vibrancy of the color, if it is more pastel or vivid. Value defines the brightness of the color, with Hue equal to 0 the image will be fully dark and 255 fully bright.

Before converting the image, it is necessary to create an IPLImage which will store the converted frame.

Code:

```
opencv_core.IplImage imgHSV = cvCreateImage(cvGetSize(Image), 8, 3);
```

There are added three parameters, size (width and height) which are the same as the size of the image that will be converted, bit depth (8) and number of channels (3). There can be 8-bit, 16-bit and 32-bit images, in case of 32-bit the image will not need to be converted because the HSV channels will have the same value of the BGR's. It is considered an 8-bit image so the conversion will be done using cvCvtColor from OpenCV which does it without any difficulties.

Code:

```
cvCvtColor(iplImage, imgHSV, CV_BGR2GSV);
```

The conversion starts dividing the values of B, G and R by 255 to change the range from 0 – 255 to 0 – 1. Then, once the correct range is established, Value = max (B, G, R), it will be equal to the maximum value from B, G, R (V = max (B, G, R)). After V is found, if it is equal to zero, also Saturation will be equal to zero (S = 0), otherwise Saturation will be equal to Value – the minimum of B, G or R (S = V – min (B, G, R). The result of Hue will be determined on what RGB channel has the max value, if R is the max then H = (G – B) / (V – min (R, G, B)), G is the max then H = (2 + (B-R) / (max – min)) or if B is the max then H = (4 + (R-B) / (max – min)).

- *Thresholding*

Once the frame is in HSV color space, it can proceed with the color detection. Also this feature is easy to implement using one function of OpenCV called cvInRangeS()[4]. In consist on a segmentation type: it will be done a differentiation of the pixels that the system wants to detect and exclude the rest. The differentiation is done setting a lower bound and an upper bound, these two values will set the range of pixels that will correspond to the Red color of the traffic light. It will proceed with a comparison of all pixels of the image and the pixels that are in range, its intensity will be set to 255 (white color) otherwise if it

is not in rage its intensity will be 0 (black color). The returned image will consist in a black and white and the Red color detected.

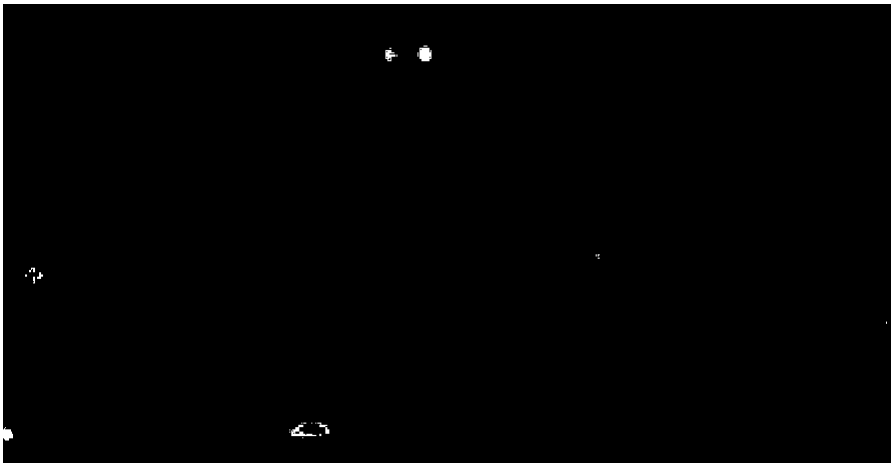


Figure 7: thresholding using red cvInRanges

Code:

```
cvInRangeS(imgHSV, cvScalar(H, S, V), cvScalar(H, S, V), filtered);
```

cvInRanges (imageInput, lowerBound(HueMin, SaturationMin, ValueMin), upperBound(HueMax, SaturationMax, ValueMax), image Output).

The range (60, 150, 100) – (200, 255, 255) corresponds exactly to the red color generated by led sources, as the traffic lights, back car lights, etc.

The same logic is applied also for the green color detection of the traffic light, the only change consists in changing the lower bound and the upper bound into (50, 160, 200) – (120, 255, 255) and it will be detected.

There is one problematic situation for the detection system: it is not rare and it can happen that other objects generate a coloration that is in range of the cvScalar declared. The solution consists in a combination of a strategic placement of the camera, normally in a static position, and an accurate selection of the objects that are detected.

Figure 8: part of the frame to be considered

To make the selection it is necessary firstly get the position of all the objects that are detected as red colors. The position is found using moments feature of OpenCV: it starts getting the area of all the white objects and then it divides the spartialMoment result with the area getting the X and Y coordinates of the center of the object.

Code:

```
double areaObj = cvGetCentralMoment(moments, 0, 0);
posX = (int) (cvGetSpatialMoment(moments, 1, 0) / areaObj);
posY = (int) (cvGetSpatialMoment(moments, 0, 1) / areaObj);
```

When that the values of the coordinates are returned, the system will check if it is found in a specific region of the image it will correspond to the traffic light otherwise the object detected will be discarded since it is a false positive.

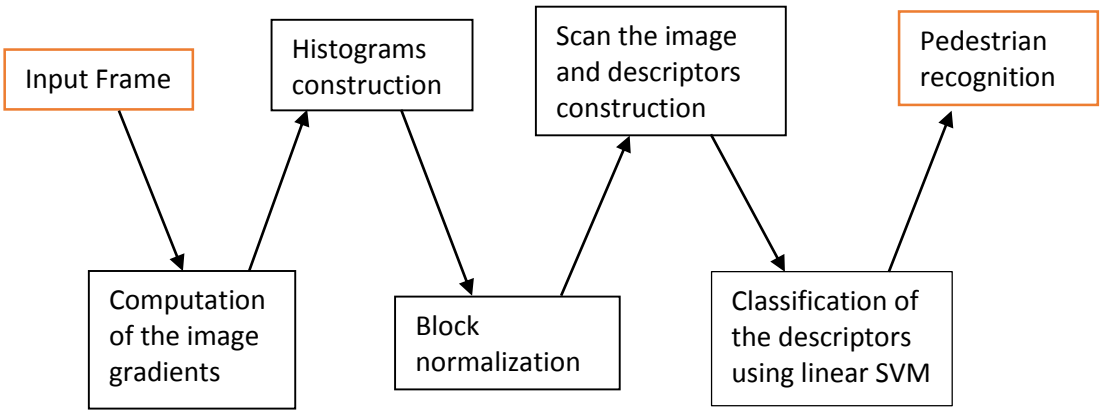
4.2 Pedestrian recognition using HOG

HOG, Histogram of Oriented Gradients, is a feature used in computer vision for the detection of objects. It is mostly used for detecting pedestrians in real-time video streaming since that it has a very low rate of false-negatives.

The fundamental idea behind the HOG descriptor is that the appearance and the local form of an object in an image can be described by the distribution of the intensity gradient or the direction of the contours. The computation of the individual histograms is obtained by dividing the image into a grid of cells and for each of them is processed a histogram for the gradients of the individual pixels. Then the cells are grouped into bigger regions called blocks. From the information received from each block, it is obtained a descriptor that will be used for the detection.

4.2.1 Method Description

HOG is a really important feature in the object recognition method and the main objective is to obtain the descriptors of a single image. The main steps of processing are the following:



4.2.1.1 Computing the gradient image

The gradient of an image can be obtained simply by filtering it with two one-dimensional filters: a vertical filter and a horizontal filter. This procedure puts in evidence the regions of the image where there is a greater variation in brightness. They are generally near the edge of an object and can therefore be used to evidence the silhouette of a person.

The filters that are used are the following:

1. Horizontal: $D_x = (-1, 0, 1)$

2. Vertical: $D_y = (-1, 0, 1)^T$

Given an image I, it is calculated the derivative with respect to the x axis and y axis using the convolution operations:

$$I_x = I * D_x$$

$$I_y = I * D_y$$

(a) **(b)** **(c)**

The figure shows an example of application of the filter for obtaining the gradient of the image. Figure (a) shows the original image, figure (b) shows the result of the application of the horizontal filter and figure (c) shows the result of the vertical filter. In case of a colored image, it is also needed a preliminary operation of conversion to gray scale; this is done to avoid the need to consider a contribution of different intensity for each color plane (RGB).

Gradients can be considered signed or unsigned. The latter case is justified by the fact that the direction of the contrast has no importance. In other words, it would have the same result analyzing a white object on a black background, or vice versa a black object on a white background.

The filter used for obtaining the gradients is one of the simplest but also one of the most effective; however, there exist other more complex masks that can be used to obtain the gradient of the image to be analyzed: one example is the Sobel filter.

4.2.1.2 Histogram construction

The next step of the procedure consists in the construction of the histograms on the basis of the gradient calculated at the previous step. First of all, the image is divided into cells. A cell is defined as a region of space that assumes a certain shape and size; they can be rectangular or circular and the bins of each histogram are distributed on 0° - 180° (gradients without sign).

For each cell is constructed a histogram accumulating in the bins the result of each gradient. If for example we want to build histograms distributed over 0° - 180° with a number of bins equal to four, the rate for the construction of the histogram is made as follows:

- All gradients of the cell with a range [0° - 45°) give their rate to the first channel;
- All gradients of the cell with a range [45° - 90°) give their rate to the second channel;
- All gradients of the cell with a range [90° - 135°) give their rate to the third channel;
- All gradients of the cell with a range [135° - 180°) give their rate to the fourth channel.

In this way, each histogram is calculated taking in consideration the importance of a gradient at a given point. This is justified by the fact that a gradient around an edge of an object is generally more meaningful that a point in a region that is uniform of the image. It is expected that more bins there are and more detailed will be the histograms. When all histograms are created, it can build the image descriptor concatenating all histograms in a single vector.

4.2.1.3 Blocks normalization

Before creating the descriptors it is required a step of standardization. It is needed because of the variations in brightness that there can be in an image.

The normalization of the histograms is made from groups of cells called blocks. For each block is calculated a normalization factor and all histograms are normalized based on this factor. The final descriptor is then represented by the vector of the components of all the cells after that they have been normalized by grouping them by blocks.

If the blocks have a square or rectangular shape it is called R-HOG. R-HOGs are composed of [n x m] cells of [m x m] pixels, each containing C bins, where n, m, and C are parameters.

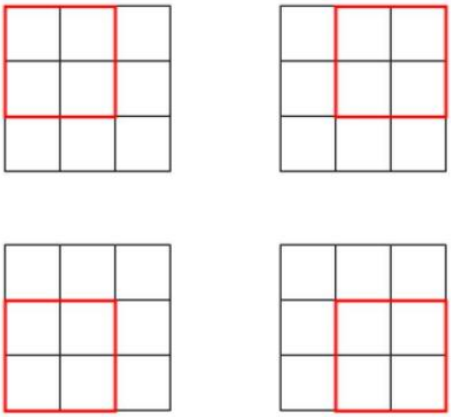


Figure 9: 2x2 cells example

4.2.1.4 Descriptors construction

The goal is now to scroll the image in multiple scales using windows of finite size. For each windows it obtains a descriptor that can be finally classified by a SVM linear classifier. The descriptors are obtained by scrolling the window up and down, left and write and appending the histograms of the individual cells to a final vector.

It is supposed:

- Window of size 64x128;
- Cells of 8x8 pixels (in total 8x16 cells)
- 9 bins histograms
- Block of 2x2 cells (in total 4x8 blocks)

It is obtained a final descriptor of dimension (4x8) x (2x2) x 9 = 1152, then the descriptors are classified using a linear SVM.

4.2.1.5 SVM (Support Vector Machine) Classifier

SVM is a classification algorithm through learning and supervision. To ensure that a classifier "learns" what to look for in the image it must be firstly trained giving a large number of positive and negative images (containing or not the object that should be detected).

OpenCV provides the default people detector that already had been trained with the images of "INRIA Person" [6] databases that consists of about a thousand positive pictures and two thousand negative pictures.

4.2.2 HOG Implementation

The implementation in Java of HOG [5] pedestrian detection does not require a great difficulty.

```
myMovie = new Capture(this, width, height);
myMovie.start();
```

The first step is to start the video streaming using start() function and store each frame of the video that will be grabbed during the streaming in real time.

Create a SVM classifier for People detector that will be used to detect the pedestrians on each frame.

Code:

```
hog = new HOGDescriptor();
hog.setSVMDetector(HOGDescriptor.getDefaultPeopleDetector());
```

Once HOG Descriptor.getDefaultPeopleDetector() is set, it is necessary to give the parameters win_size, block_size, block_stride, cell_size, bins_num, win_sigma, threshold, gamma_correction, levels_num and the system will recognize easily pedestrians.

Code:

```
hog.detectMultiScale(mat01, rect, weight, 0.35, new Size(8,8),
                        new Size(32,32), 1.10, 2, false);
```

One array will be added to store all the pedestrians that are detected on each frame. Each pedestrian detected will be surrounded will a green rectangle on the screen.

Code:

```
Rect [] peopleArray = rect.toArray();
```

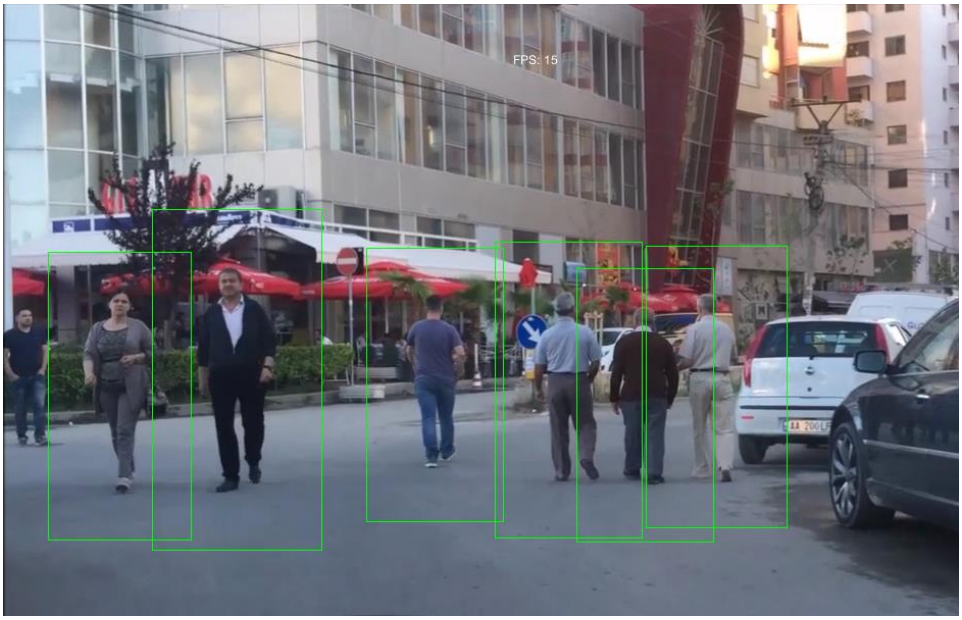
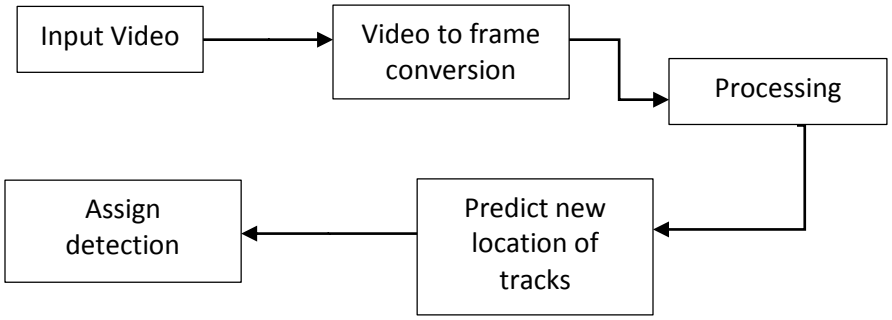


Figure 10: detection results of pedestrians

4.3 Motion Detector Security

The last feature of the system is motion detector security. The main idea of it is that once the driver leaves the car, if it is in a public parking in the street, it remains unattended and it is not rare that someone damages the car with the aim to steal equipment (radio, GPS, stereo, CD), damage the interior of the car for vandalism or even worse steal the car. The security system of the car consists in tracking every motion inside the car and when alarming, report it to the owner sending frames of the video by an e-mail.



4.3.1 Tracking using Kalman Filter and Hungarian Algorithm

The Kalman filter is not actually the filter which tries to estimate the state of a system from the data collected (tracking). The filtering is based in a two-step process: the first step consists in the prediction of the state of the system, and the second step consists in removing noisy disturbance to offer a better estimation of the system state. These two steps, prediction and correction, are done by the functions getPrediction(), update() and correction() implemented in Java and got from external sources.

The objects that are detected have to be assigned to tracks and for this reason it is combined with the Hungarian Algorithm, which uses matrix method to detect and assign the object to tracks.

It has a method that stores all the registered tracks, and on each frame determines which of the tracks are missing. The cost of matrix condition the cost of each tracklet; if the cost is low then the task is higher or the inverse situation if the cost is high the task is low.

4.3.2 Surveillance System

The first step is up to the driver; once he parked the car he must rotate the camera connected to the system and change the view from the road to the interior of the car, then activate the second feature, the security system.

Few tens of seconds later (the time for the driver to leave the car), the security system will be activated.

Figure 11: view of the inside of the camera

The security system is implemented using OpenCV library and it automatically detect and tracks moving object inside the car [7].

The motion-based detection and tracking of objects can be done in two steps:

a. Detect and track each object frame by frame

b. Tracking the same object over time

The detection and tracking of each object by its frames uses an algorithm on background subtraction, which is done by using Gaussian models.

Morphological reconstruction is used to extract shapes in the frame and marked objects using a generalization of flood-filling (marker) and removing the noise from the image. Then, it is use blob analysis that allow to detect pixels that are connected to each other, forming in this way groups that correspond to the moving objects. The motion of every object in every frame is determined using Kalman filter [8]; it tries to estimate and predict the location of the object in the next frame, and it determine the probability of every motion-detection of every object.

It can have two types of tracking objects: one type is called the unassigned whose motion is detected in few frames and that they are not important for the system, called so invisible tracks that will not be considered, and the other type are the assigned which are the motions generated by the objects that correspond to the system's target and who will be detected.

The basic solution to differentiate these two types of tracks is implemented in the system, which counts in how many frames the same motion is detected, and if it result a moving object in more than a specific number of frames it will be categorized as a moving object and so alerted.

Shadows of external vehicles, light changes, pedestrian passers near the car can be captured by the camera and can cause motion that will attract the system's attention but they are not inside the car so they should not be considered.

This is an example of invisible tracks; some pixel changes result in movement (caused by shadows or light direction changes) but they does not correspond to a risk object so they will be added to the background and they will be totally discarded.

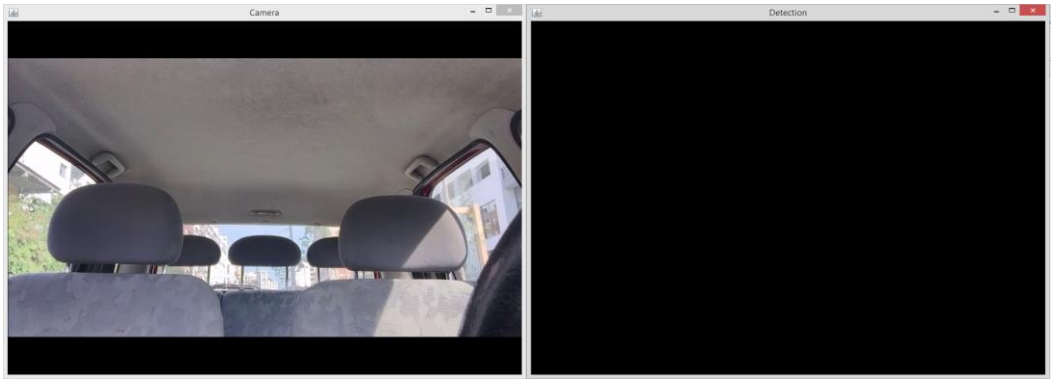


Figure 12: background subtraction

As invisible tracks, all of them are not considered and once detected they are added to the background, and the system continues the video streaming.

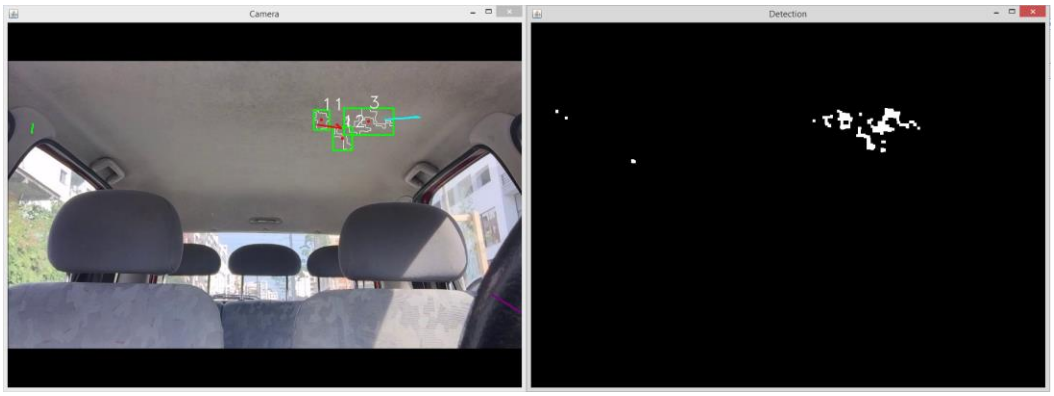


Figure 13: Light changing or shadows detection

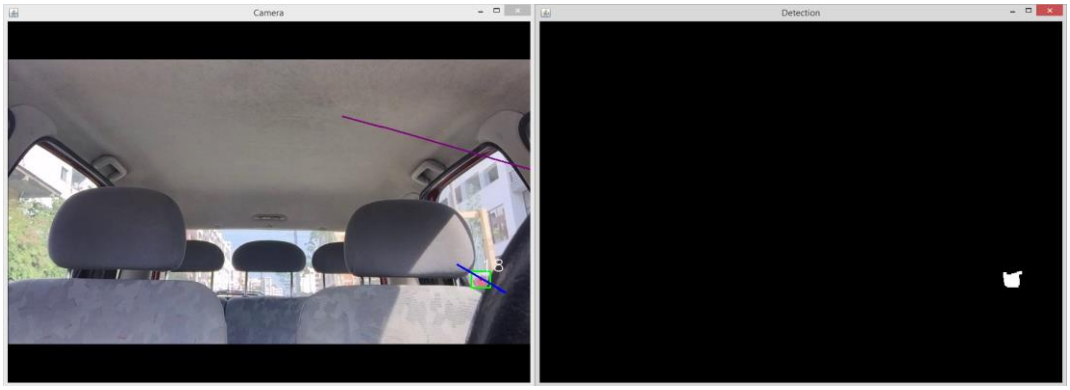


Figure 14: motions of other objects are not considered

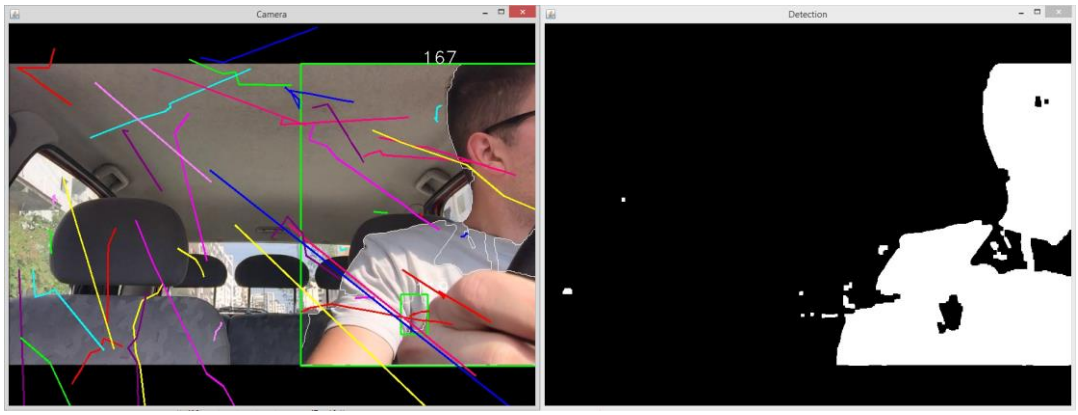


Figure 15: the motion captured

In Figure 15 it can be noticed that if motion is detected in more than a hundred frames and most of the motion is generated inside the car, it obviously represent a risk element that will be reported by the system.

4.3.3 Implementation in Java

The implementation of the classes of the Hungarian Algorithm and the Kalman filtering for simplification reasons are imported from external sources [9].

It will be described only how the use of them will allow the proper functioning of the system.

After that the video streaming started on each frame will be done a background substraction, OpenCV provides this feature simply calling BackGroundSubstractorMOG2.

Code:

```
BackgroundSubstractorMOG2 backgroundSubstraction =
                        Video.createBackgroundSubstractorMOG2();
```

And then apply it to the video:

Code:

```
Video(camera, frame, tframe, backgroundSubstraction);
```

To have a better distinction between the background and the moving object it is necessary to perform a binary thresholding where the background of the image will be set to 0 (black) and the foreground (the moving object) will be set to 255 (white).

Code:

```
Imgproc.threshhold(tframe, tframe, 127, 255, Imgproc.THRESH_BINARY);
```

Figure 16: image thresholding example

Now that the object is well distinguished from the background it is easy to detect its contours by using findContours() method and find the outlines of the object colored as white.

Code:

```
Imgproc.findContours(matJr, list, mat, Imgproc.RETR_LIST,
                        Imgproc.CHAIN_APPROX_SIMPLE);
```

The object is clearly defined on the frame, it remains to check each frame of the video for the object using the Tracker, and then the Kalman filter to estimate its next position.

Code:

```
Imgproc.line(imag, tracker.tracks.get(k).trace.get(movesT-1),
            tracker.tracks.get(k).trace.get(movesT),
            lines[tracker.tracks.get(k).track_id % 2], 2, 4, 0);
```

CHAPTER 5

CONCLUSIONS

In this paper was proposed the implementation of a system that can assist the driver recognizing in real time the traffic light and alerting him for the presence of pedestrians. It also was proposed a surveillance feature that tracks all the motions inside the car when it is parked, and, in case of alarm, report the presence of strangers to the owner by an e-mail.

Once the problems has been identified and the solutions have been proposed, it is necessary to verify the actual performance of the system in a real environment. Based on the video test presented, the system achieved a very reasonable result and precision rate: it works perfectly with a high percentage of recognition of traffic lights and pedestrians. The audio signal, that reports the presence of pedestrians and/or red traffic light, offers sufficient support to the driver. Also, the surveillance system reported every human motion that was detected in the car and the frames grabbed were successfully sent to the owner by e-mail, making the person traceable.

The hardware used for testing the sytem was low cost and with a high performance. A simple Raspberry Pi B and a normal HD webcamera allowed the normal operating of the functionalities. In future implementations, with an extra budget, the hardware can be upgraded using extra componets as GPS antenna - for reporting the location of the car along with the photo in the e mail or an LCD monitor to display the video captured or other hardware to offer the maximum support to the driver.

REFERENCES

[1] K.J. Thakkar "What is OpenCV? OpenCV vs. MATLAB", 2012

[2] Based on: http://www.instat.gov.al/al/figures/statistical-databases.aspx

[3] R.M. Thrun, "Converting from RGB to HSV", Journal of Field Robotics, 2004

[4] OpenCV documentation. Available:

http://docs.opencv.org/

[5] Dalal, N. and Triggs, B., "Histograms of Oriented Gradients for Human Detection," IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, San Diego, CA, USA.

[6] N. Dalal, "INRIA Person Dataset," http://pascal.inrialpes.fr/data/human/

[7] Motion detection using a webcam, Java, OpenCV and Differential Images, 2010

[8] M. Thomas "Kalman Filter Applications in Java", 2005

[9] Kalman Filter and Hungarian algorithm. Available :

http://algs4.cs.princeton.edu/65reductions/Hungarian.java.html

[10] Y. Ma, Pedestrian detection using HOG and oriented-LBP features, 2005

[11] G.Siogkas, Traffic Light Detection using Color, Symmetry and Spatiotemporal Information, 2009

[12] Sounds available:

http://www.moviewavs.com/Movies/Star_Wars/c3po.html

[13] MathWorks, Motion-Based Multiple Object Tracking. Available:

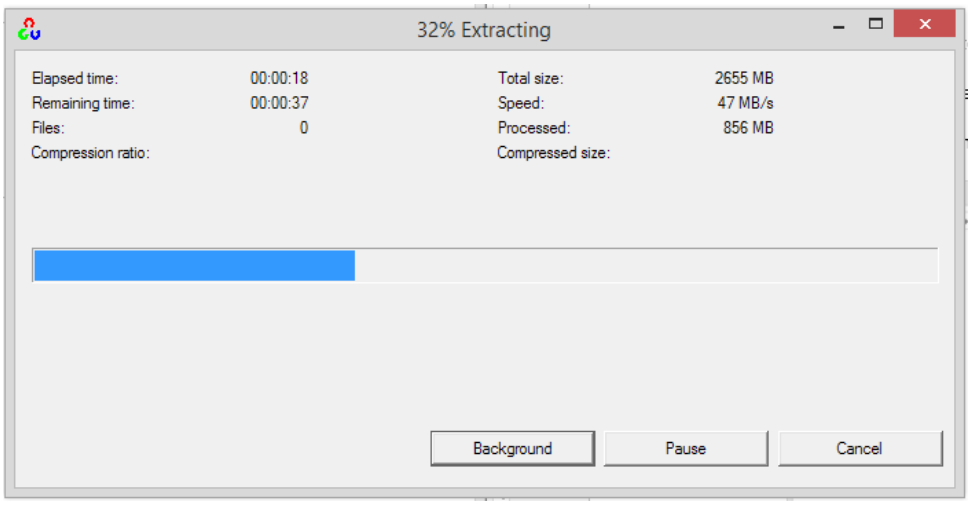
http://www.mathworks.com/help/vision/examples/motion-based-multiple-object-tracking.html

APPENDIX A

A1. OpenCV installation

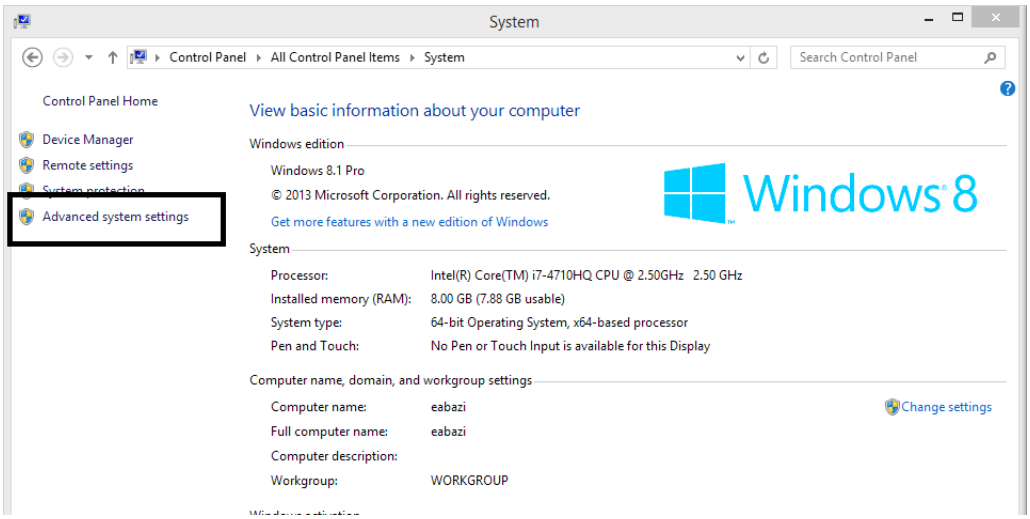
1. Download OpenCV 3.0 from the website: http://opencv.org/downloads.html

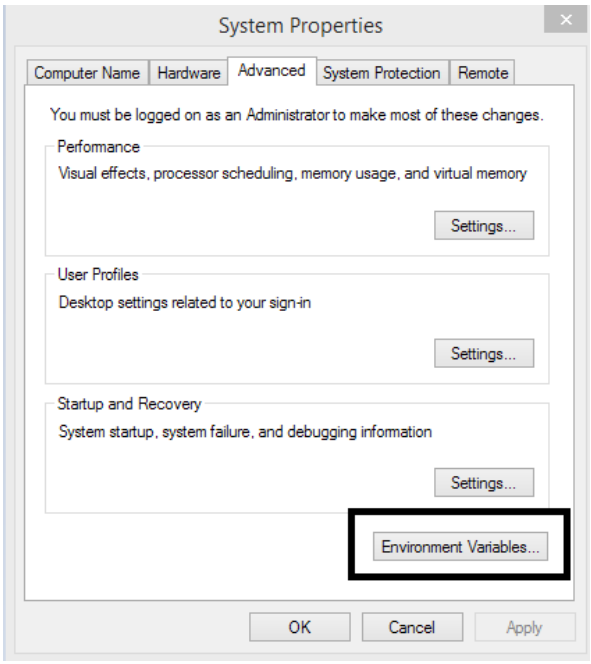
2. Extract the self-extracting archive to a location, for example C:\ProgramFiles\OpenCV



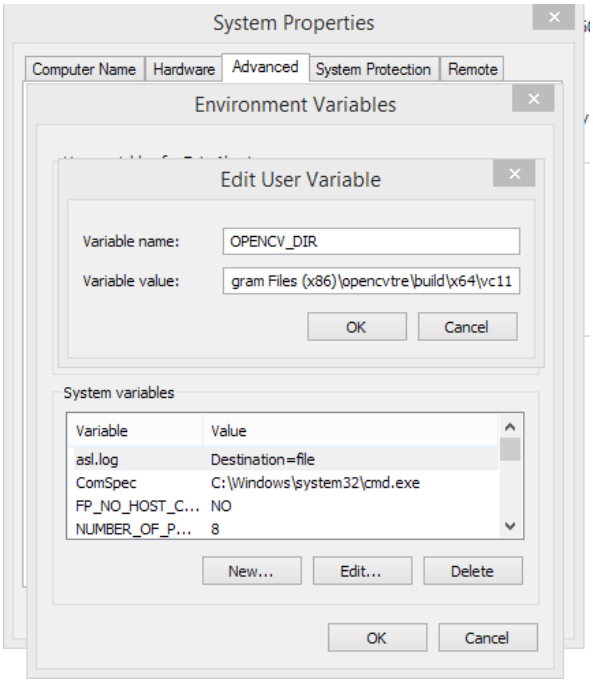
3. Set the system environment variables

a. Open Start Menu, then control Panel and open System. Click Advanced system settings and then Environment Variables





b. Click New to create a new system variable. Set the variable name OPENCV_DIR and the variable value the directory used for the OpenCV folder extraction\build\x86\vc11\



c. click ok edit, ok environment and again ok to confirm the system changes

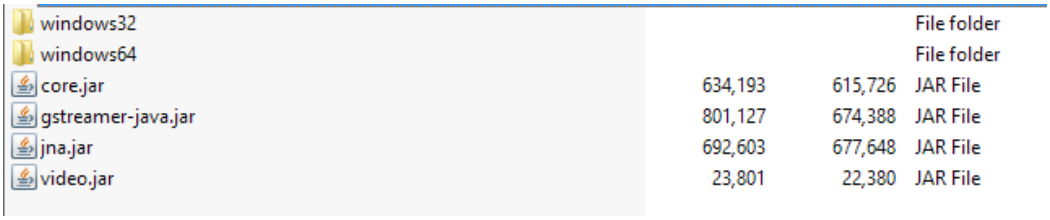
4. The installation is completed, import the OpenCV .jar file and the libraries will work.

A2. Processing 2.2.1 Installation

1. Download Processing from the website: http://processing.org/download

2. Extract the .zip file to a location, for example C:\Program Files\Processing\

3. Processing has its own Development Environment, but this system is implemented using only the Processing Libraries on a different IDE. Import the libraries necessary and import them to the \libs project folder. The libraries used are: video.jar, core.jar, jna.jar and gstreamer-java.jar. Also the .dll files located in the two folders windows32 and windows64 will be needed.



windows32			File folder
windows64			File folder
core.jar	634,193	615,726	JAR File
gstreamer-java.jar	801,127	674,388	JAR File
jna.jar	692,603	677,648	JAR File
video.jar	23,801	22,380	JAR File

4. Netbeans is used as DE. Create a new Java Project and then a new Java class.

5. Add “import processing.core.*;” and extend the class with PApplet.

6. The main method consists in the package name and the class name, for example:

PApplet.main(new String[]{*packageName.className*.class.getName()});

A4. Install OpenCV on the Raspberry Pi to run the system program

1. Install the JDK: the command line: *sudo apt-get update && sudo apt-get install openjdk-7-jdk* will install the openjdk 7.

2. Install dependencies with the command line: *sudo apt-get install ant* that will be used to install ant, in order to build the OpenCV .jar file.

3. Install the packages to build OpenCV. The command lines:

- *sudo apt-get install build-essential cmake pkg-config libpng12-0 libpng12-dev libpng++-dev libpng3*
- *sudo apt-get install libpnglite-dev zlib1g-dbg zlib1g zlib1g-dev pngtools libtiff4- dev*
- *sudo apt-get install libtiff4 libtiffxx0c2 libtiff-tools libjpeg8 libjpeg8-dev libjpeg8-dbg libjpeg-progs ffmpeg libavcodec-dev*
- *sudo apt-get install libavcodec53 libavformat53 libavformat-dev libgstreamer0.10-0-dbg libgstreamer0.10-0 libgstreamer0.10-dev*
- *sudo apt-get install libxine1-ffmpeg libxine-dev libxine1-bin libunicap2 libunicap2-dev libdc1394-22-dev libdc1394-22 libdc1394-utils*
- *sudo apt-get install swig libv4l-0 libv4l-dev*

4. Build OpenCV from the git repository

- *sudo apt-get install git*
- *git clone git://github.com/Itseez/opencv.git*
- *cd opencv*
- *git checkout 3.0*
- *mkdir build*
- *cd build*

5. Build OpenCV

- *cmake -D CMAKE_BUILD_TYPE=RELEASE –D CMAKE_INSTALL_PREFIX= /usr/local -D BUILD_EXAMPLES=OFF ..*

6. Make process

- *make*

7. The build will generate .jar file and a .so file: bin/libopencv_java3.0.so and bin/opencv-3.0.jar. Set the JAVA_HOME environment variables to the path of the JDK:

- *export JAVA_HOME=/usr/lib/jvm/jdkfolder*

8. Place the java3.0.so inside the jdk folder:

- */usr/lib/jvm/java-6-openjdk/jre/lib/arm/libopencv_java2xx.so*

9. OpenCV is installed