D4.1.1	
Version	1.0
Author	Visual Tools
Dissemination	PU
Date	15/01/2015
Status	Final

D4.4.1: Video Content Analysis cloud elements V1.0

Project acronym:	NUBOMEDIA
Project title:	NUBOMEDIA: an elastic Platform as a Service (PaaS) cloud for interactive social multimedia
Project duration:	2014-02-01 to 2016-09-30
Project type:	STREP
Project reference:	610576
Project web page:	http://www.nubomedia.eu
Work package	WP4: Video Content Analysis cloud elements.
WP leader	Fco. Javier Lopez (Naevatec)
Deliverable nature:	Report
Lead editor:	Victor Hidalgo (Visual Tools)
Planned delivery date	10/2015
Actual delivery date	31/01/2015
Keywords	Computer Vision, Video Content Analysis, VCA Filters

The research leading to these results has been funded by the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement nº 610576

FP7 ICT-2013.1.6. Connected and Social Media

Contributors:

Víctor Manuel Hidalgo (Visual – Tools)

Internal Reviewer(s):

Francisco Javier Lopez (Naevatec)
Luis Lopez (URJC)
Patricia Galvan (Zed)

Version History

Version	Date	Authors	Sections Affected
0.1	31/08/2014	Victor Hidalgo	Structure of the Content and first input
0.2	26/09/2014	Victor Hidalgo	Algorithms description (R2)
0.3	30/09/2014	Fco. Javier Lopez	Review (Feedback)
0.4	14/10/2014	Victor Hidalgo	Minor changes
0.5	07/01/2014	Victor Hidalgo	Algorithms description (R3)
0.6	11/01/2015	Luis Lopez	Review (Feedback)
0.7	15/01/2015	Victor Hidalgo	New structure of the document and minor changes.
1.0	20/01/2015	Patricia Galvan	Review (Feedback)

Table of contents

List of Figures:

List of Table:

Acronyms and abbreviations:

Acronyms	Definition
VCA	Video Content Analysis
OpenCV	Open Source Computer Vision
ME	Media Element
CV	Computer Vison

1 Executive summary

This document contains a description of the video content analysis media elements developed on the NUBOMEDIA project. The structure of this document is as follows.

- Introduction, in this section we will see a short state of the art of cloud computing applied to computer vision and some interesting concepts that we are going to need throughout this document
- Objectives, this section will explain the main objectives and goals related to this task and reflected on this document
- Strategy, this section shows the resulting list of requirements obtained in WP2: "*User centric analysis of user scenarios and requirements*", through which we can select the algorithms to develop on this task. The different factors taken into account to select an algorithm will also be explained.
- Architecture, we will see a high-level architecture explaining the main modules used to develop in general all the VCA filters.
- Software Analysis, the VCA algorithms developed on this task. For every algorithm, we will see details about the implementation, the inputs and outputs of the filter and the API of the filter.
- Implementation status, In this section, it is described the license of the different algorithms, how to access the source code and install all the modules developed in the associated task.

2 Introduction

Computer Vision (CV) is a rapidly growing field devoted to analyzing, modifying, and high-level understanding of images. Its objective is to determine what is happening in front of a camera and use that understanding to control a computer or robotic system, to provide people with new images that are more informative or to index content enabling richer and more efficient searching. Application areas for CV technology include video surveillance, biometrics, automotive, photography, movie production, Web search, medicine, augmented reality, gaming, new user interfaces and many more. However, CV is computationally expensive: it tends to consume prohibitive amount of resources including memory and CPU. This is probably the main problem avoiding CV mass-scale usage.

For this reason, the emergence of cloud technologies may enable CV to be applied in scenarios where it was not possible. Running in the cloud CV algorithms that can interoperate with each other and that are accessible through comprehensive and simple APIs can open new horizons to CV applications. Some of the benefits that cloud technology offers to computer vision are:

- Access to on-demand computational power and storage.
- Access to on-demand CV algorithms or services.
- Access to simple to use APIs for creating CV applications.
- Pay per-use models both for computational resources and algorithm or other intellectual property access.

As it was described on D2.2 “State of the art revision document”, there is a big **lack of information about computer vision** over cloud computing. Due to this gap, The NUBOMEDIA project emerges with the aim of creating a cloud platform specifically designed for hosting real-time interactive multimedia services incorporating VCA and Augmented Reality.

Before starting to see in more detail this deliverable, we will explain some concepts that are important for understanding the rest of the deliverable:

- **Media Element**: media elements are monolithic and abstractions elements which send and receive media streams and data through different protocols. The goal of these elements is to carry out specific processing of the media stream or the data
- **OpenCV** (Open Source Computer Vision) is a library of programming functions mainly aimed at real time computer vision.
- **Bounding Box:** in this deliverable we understand a bounding box as a rectangle defined by a point (x,y) within in the image, its height and weight. Bounding box serves to establish a specific interest in a certain area of the image. Bounding boxes will be used to represent information of interest that the different media elements can exchange between them.

3 Objectives

The **main goal** of this task is the **creation** of different **video computer vision algorithms** that will be **integrated** in the **NUBOMEDIA** framework in order to provide different VCA filters. These filters will enable the creation of cloud elements to extract semantic information from one or more uncompressed media streams. The VCA filter output will be delivered as a continuous stream of information, which will describe high level items detected in the media and information to associate the detected items to the corresponding images (frames) of the input stream or to a region or position within the image. The following elements are an example of the result of applying a VCA filter:

- The descriptor of an object detected in the image (e.g. bounding box) together with the coordinates of the object in the image (x,y)
- The descriptor of a face detected and the position (x,y) of the face.
- A license plate descriptor (e.g. characters, confidence level) and its position.
- Depth information associated to the image.

These VCA filters will be used in the vertical demonstrator implemented in Task 6.2. Therefore, the algorithms that develop in this task will be selected from the resulting list of the requirements from WP2. This list shows the services and functionalities that solve the customer problems or needs.

This document is organized as follows. The second section shows the resulting list of the D2.2, which will be the basis to select the algorithms to develop. In addition, we explain in this section the work that we will carry out in every release of the project. The third section contains information about every VCA filter, where we explain the algorithm, some tests and so on.

4 Strategy

As it was explained on the previous section, the idea of these tasks is to create different VCA algorithms as a proof of concept to be integraged in the NUBOMEDIA platform. These algorithms will be selected based on the list obtained in the WP2 which shows the services and functionalities that solve the customer problems and needs. The list with the different services is given below.

Requirement	Priority	Use Case	Owner
Nose, mouth and ears detection.	High	UC-ZED-1/3	VTOOLS
Motion detection	Low	UC-VT-1, UC-TI	VTOOLS
Object tracking	Relevant	UC-VT-1/2 UC-LU, UC-NT, UC- TI	VTOOLS
Intrusion detection	Low	UC-VT-1/2, UC-TI	VTOOLS
3D – extracting depth info	Low	UC-VT-1/2, UC-TI, UC-ZED	VTOOLS
Statistics for retail sector	Low	UC-VT-2	VTOOLS
Automatic lip reading	Low	UC-LU	VTOOLS
Lip extraction and color characterization	Low	UC-NT	VTOOLS
Face extraction and color characterization	Low	UC-NT, UC- TI	VTOOLS
Breath frequency characterization from chest movements	Low	UC-NT	VTOOLS
Walking movement characterization	Low	UC-NT	VTOOLS
Skin rash characterization	Low	UC-NT	VTOOLS
Spasms characterization from body movements	Low	UC-NT	VTOOLS
Heartbeat characterization from skin color variations	Low	UC-NT	VTOOLS
The system to be able to trigger events, based on video's feature	Medium	UC-TI	VTOOLS
Face detection	High	UC-VT-1/2, UC-ZED-1/3	VTOOLS
Identify the place where the objects have been hidden	High	UC-ZED-2	VTOOLS
Detect 3d structure of the environment where the object is being hidden.	Medium	UC-ZED-2	VTOOLS

Table 1: Requirements from the D2.2.1

Due to the high number of requirements and the time it takes to develop these algorithms in a robust manner, we are forced to choose some of them based on the following:

- The Description of Work (DoW).
- Strategic decision, by which we try to develop algorithms that can serve us for new business opportunities.
- Based on the frequency that these algorithms are used.
- The difficulty of the algorithms. Since we need to cover several algorithms to be used by the different partners on task 6.2, we cannot invest an unreasonable time for a specific algorithm, because we need to cover a reasonable number of filters so that all the partners can use in their demonstrator a VCA filter if they wish.
- The more mature computer vision algorithms currently used in real applications, which are all of them well known

In the following subsection, we can observe what are the different algorithms developed in every release.

4.1 Release 2 (R2)

This second release covers from M5 (06/2014) to M8 (09/2014). During this release we have realized the following work.

- VT's team has begun to learn Kurento platform which will be used on NUBOMEDIA to generate the different media elements and pipelines. In this case, we have learnt how to develop the VCA plugins and how integrate them in the Kurento platform.
  The requirement motion and face detection have been developed, you can read more about this algorithms on section 3.

4.2 Release 3 (R3)

The third release covers from M9(10/2014) to M12(01/2015). During this release we have realized the following work.

VT's team has carried out the necessary changes in the algorithms already developed (motion and face detector) to adapt them to the new changes of the Kurento platform.

- The nose, mouth and ear requirement which can be split into three different algorithms (Nose, mouth and ear detector) has been developed, you can read more about this algorithm on section 3.
- All the algorithms developed up to now have been integrated on the first version of NUBOMEDIA's cloud platform.
- Finally, some examples of demonstrator which combine different VCA filters have been developed.

5 Architecture

In this section, we are going to see a high-level architecture explaining the general overview of the architecture to develop the different VCA algorithms. The following figure depicts the base architecture to develop the different VCA algorithms over the NUBOMEDIA infrastructure.

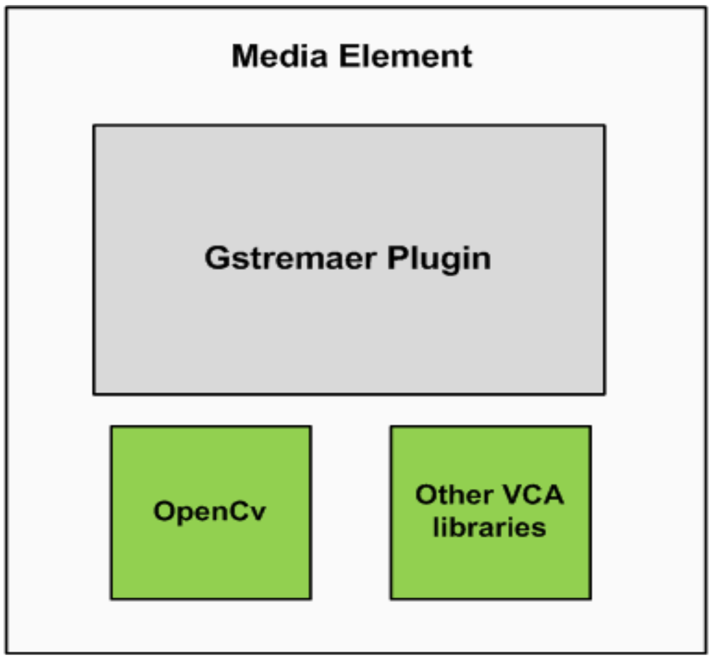

Figure 1: General VCA architecture

The different algorithms are integrated in NUBOMEDIA through the Media Elements (ME). These media elements are monolithic and abstractions elements which send and receive media streams and data through different protocols. The goal of these elements is to carry out specific processing of the media stream or the data. Specifically, for this task the media elements are focused on applying VCA algorithms in order to extract valuable semantic information. If you are interested on knowing more about Media Elements, please see the D2.4 "NuboMedia Architecture".

After seeing about which elements of the NUBOMEDIA architecture the VCA algorithms are integrated, we will see the modules used to integrate these algorithms and process the video and data. The Media Elements offer the possibility to develop a specific algorithm through a GStreamer plugin. These plugins offer to the developers the video frame by frame in a specific function. Therefore, the user can perform the processing of the frame in this function. With the aim to process every frame the developers can use different libraries. On this project a high percentage of the algorithms are going to be developed using the OpenCV library. On the other hand, an important percentage of the algorithms are going to be developed using other libraries.

Finally, once the algorithm has been developed and installed, the developers can use the filter through the Kurento platform using the Java or JavaScript API.

6 Software Analysis

In this section, we will explain the different algorithms developed up to now, and therefore exposed to the community of NUBOMEDIA as ME. For every algorithm, we will see details about the implementation, the inputs and outputs of the filter and the API of the filter.

6.1 Nose, Mouth, ear and face detection

First of all it is important to highlight that all this algorithms are included in the same section, since their basic technology is based on the same algorithm.

These algorithms have been selected to be included on the NUBOMEDIA platform for the following reasons:

- They are an algorithm which is widely used in industrial applications, and we believe that it could be of great use for the NUBOMEDIA partners and possible future users of the platform.
- Two of the partners in the project are interested in it.
- Furthermore, this requirement has a high priority for them.
- There is lot information about the technology in nose, mouth, ear and face detection which allows us to develop them in reasonable time.
- We are interested in developing them for a possible industrial exploitation.

We are going to introduce the common algorithm in which the four algorithms are based. After that, we will see the particularities of each algorithm.

6.1.1 Details of the basic algorithm

For this algorithm, we are using the **OpenCV library**. Specially, we take advantage of the **Haar featured-based Cascade classifier** included in this library.

This Object Detection method is an effective algorithm proposed by Paul Viola and Michael Jones in their paper, “Rapid Object Detection using a Boosted Cascade of Simple Features” [1] . The three main characteristics of this detection algorithm are:

1. A new image representation call ***integral image*** through which the image can be computed from an image using a few operations per pixels.
2. Build a classifier using a small number of important features using AdaBoost (adaptive boosting, a machine learning meta-algorithm). This **classifier** deletes the majority of the features and it is focused on a small set of critical features.
3. Combine successively more complex classifiers in a cascade (known as **cascade of classifiers**) structure which increases a lot the speed of the detector.

The features, which we have named on the previous list, are small region of the image with a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangles. You can see an example in the following image, in this image we can find the three kinds of features that this algorithm uses. The first feature is two rectangle feature showed in (a), the second one is a three-rectangle feature showed in (b) and the third one is a four-rectangle feature showed in (c).

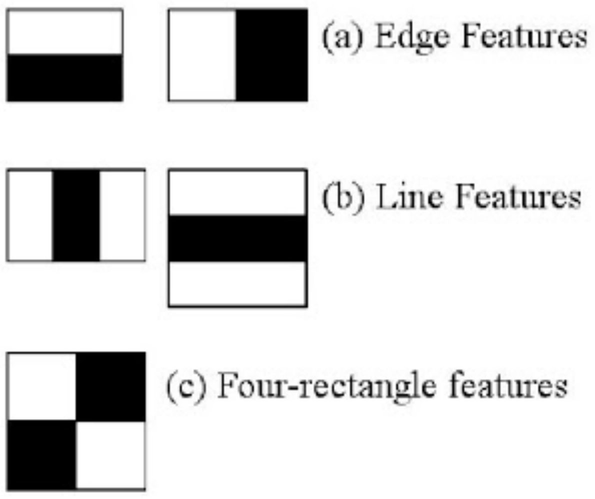

Figure 2: Features of the Haar cascade classifier

Integral image

All possible sizes and locations of each part of the image are used to calculate plenty of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find sum of pixels under white and black rectangles. To solve this, they introduced the integral image, which simplifies the calculation of sum of pixels, involving just four pixels. This makes the calculations super-fast. In more detail, the integral image at location *x,y*, contains the sum of the pixels above and to the left of *x,y,* inclusive.

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y')$$

Where ii(x,y) is the integral image and i(x,y) is the original image. For example the sum of the pixels within the rectangle D (see the following image) computed with four array reference. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value of location 2 is A + B, at location 3 is A + C, and at location 4 is A + B + C + D. Therefore, using the integral image any rectangle can be computed in four array references.

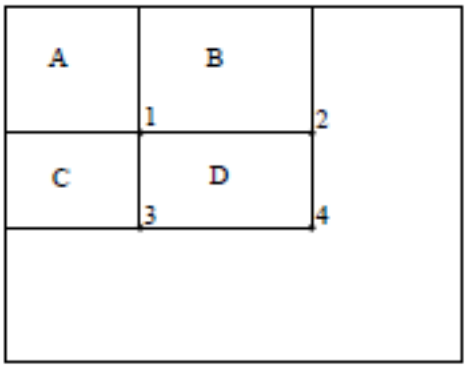

Figure 3: Face detection integral image

Learning classification functions

In this stage, the algorithm is trained from a lot of positive (images of faces) and negative images (images without faces). The algorithm looks for the best rectangle feature which separates the positive and negative images.

For example, in the case of face detection, the best features are:

1. The first feature measures the difference in intensity between the region of the eyes and a region across the upper cheeks.
2. The second feature compares the intensities in the eye regions to the intensity across the bridge of the nose.

In the following figure, we can see a graphical representation of these two areas.

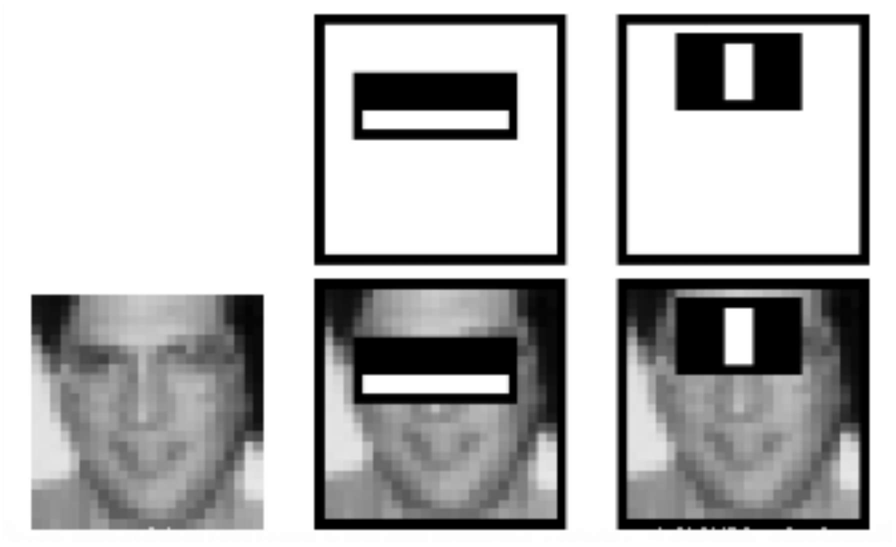
Figure 4: Best features for Face recognition

Cascade of Classifiers

The following figure illustrates the cascade of classifiers. The first classifier is applied to every sub-window. This first classifier eliminates a large number of negative examples with very little processing. Those sub-windows which are not rejected by this classifier are processed by a sequence of classifiers, each slightly more complex than the last. If any classifier rejects the sub-window, no further processing is performed to this sub-window.

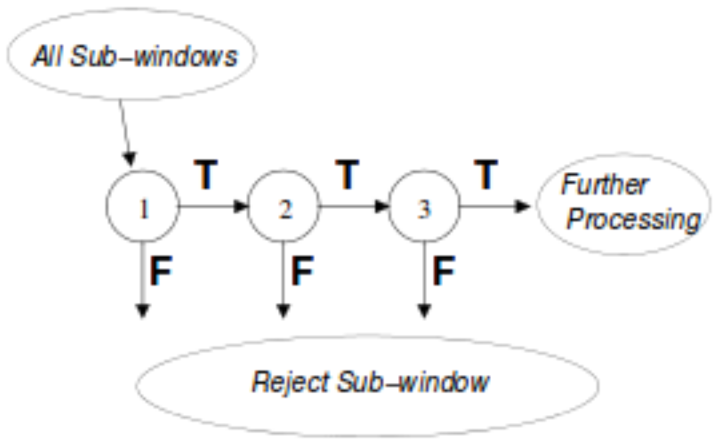

Figure 5: Cascade of classifiers

6.1.2 Inputs, outputs and API of the face detector filter

As all the rest of VCA filters, this one will receive a stream of images as input. The output of the filter will be a collection of bounding box. Each bounding box represents the position of each face in the image. A bounding box is an area defined by two points. It is very important to highlight that this algorithm **only detects frontal faces**. Therefore, all the faces that are laterally focused will not be detected.

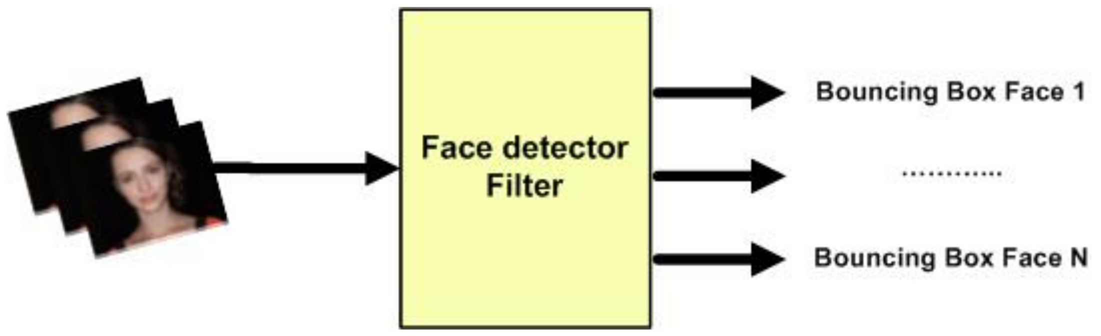

Figure 6: Face Detector Filter Input and Outputs.

Once we have seen the inputs and outputs of the filter, developers can use it creating a media pipeline which includes an instance of this filter. After the application starts to work, the filter will start to receive the video and produce the corresponding outputs. In addition, developers can use the filter's API for its setup. Then, the API is shown:

Function	Description
void showFaces **(int);**	To show or hide the bounding boxes of the detected faces within the image. Parameter's value: - 0 (default), the bounding boxes will not be shown. - > 0, the bounding boxes will be drawn in the frame.
void detectByEvent (int);	To indicate to the algorithm if it must process all the images or only when it receives a specific event such as motion detection. Parameter's value: - 0 (default) , process all the frames;

	- 1, process a frame when it receives a specific event
void sendMetaData (int);	To send the bounding boxes of the faces detected to another ME as a metadata. Parameter's value: - 0 (default), metadata are not sent. - 1, metadata are sent.

Table 2: Face detector API

6.1.3 Inputs, outputs and API of the nose detector filter

This filter will also receive a stream of images as input. The output of the filter will be a collection of bounding box. Each bounding box represents the position of each nose found in the image.

But this algorithm needs to detect previously the different faces included on the image. It can detect the faces by its own or can receive the bounding box of the faces included on the image as an input. Therefore, if the filter receives the bounding box of the faces as an input, it will not be necessary to detect the faces again. As a consequence, the processing time of the filter will be reduced. Once the faces have been detected, it will proceed to detect a nose in those parts of the image where there is a face. Moreover, all the bounding boxes representing the faces can be reduced by 20 to 30 percent the height of said regions as both the top and the bottom of the bounding box. In this way, we remove not only information which is not useful for detecting noses as the chin and forehead, but also we can reduce processing time and improve the likelihood of success of the algorithm.

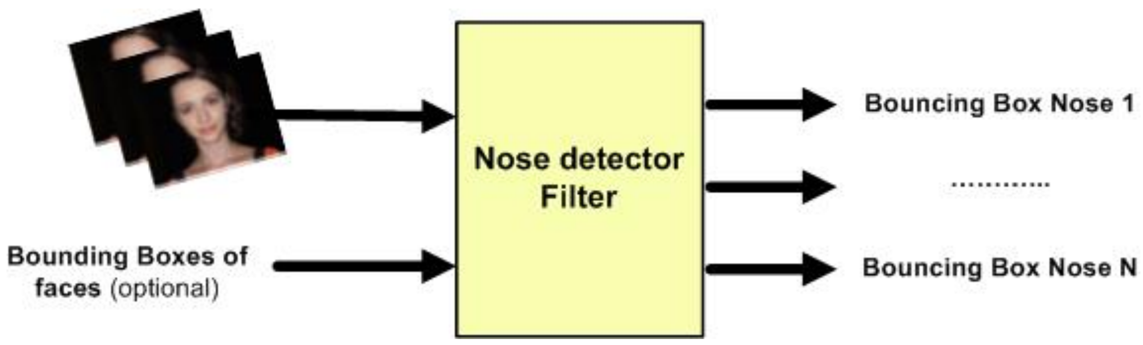

Figure 7: Nose Detector Filter Input and Outputs.

Developers can use the filter's API for its setup. Then, the API is shown:

Function	Description
void viewNoses **(int);**	To show or hide the bounding boxes of the detected noses within the image. Parameter's value: - 0 (default), the bounding boxes will not be shown. - > 0, the bounding boxes will be drawn in the frame.
void detectByEvent (int);	To indicate to the algorithm if it must process all the images or only when it receives a specific event such as face detection. Parameter's value: - 0 (default) , process all the frames;

	- 1, process a frame when it receives a specific event
void sendMetaData (int);	To send the bounding boxes of the noses detected to another ME as a metadata. Parameter's value: - 0 (default), metadata are not sent. - 1, metadata are sent.

Table 3: Nose detector API

6.1.4 Inputs, outputs and API of the mouth detector filter

This filter will also receive a stream of images as input. The output of the filter will be a collection of bounding box. Each bounding box represents the position of each mouth found in the image.

In the same way as the nose detector algorithm, this algorithm needs to detect previously the different faces included on the image. This filter can detect the faces by its own or can receive the bounding box of the faces included on the image as an input. Therefore, if the filter receives the bounding box of the faces as an input, it will not be necessary to detect the faces again. As a consequence, the processing time of the filter will be reduced. Once the faces have been detected, it will proceed to detect a mouth in those parts of the image where there is a face. Moreover, all the bounding boxes representing the faces can be reduced by approximately 50 percent the height of said regions to the top. In this way, we remove not only information which is not useful for detecting mouths as forehead and part of the nose, but also we can reduce processing time and improve the likelihood of success of the algorithm.

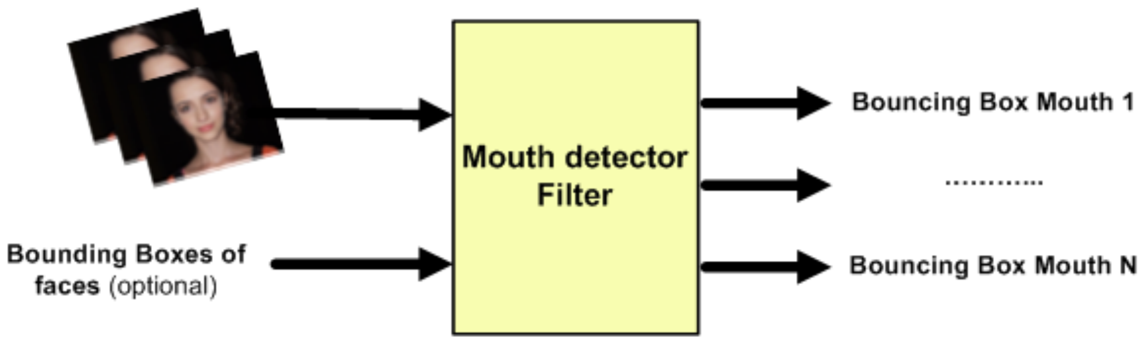

Figure 8: Mouth detector filter Inputs and Outputs

Developers can use the filter's API for its setup. Then, the API is shown:

Function	Description
void viewMouths **(int);**	To show or hide the bounding boxes of the detected mouths within the image. Parameter's value: - 0 (default), the bounding boxes will not be shown. - > 0, the bounding boxes will be drawn in the frame.
void detectByEvent (int);	To indicate to the algorithm if it must process all the images or only when it receives a specific event such as face detection. Parameter's value:

	- 0 (default) , process all the frames; - 1, process a frame when it receives a specific event
void sendMetaData (int);	To send the bounding boxes of the mouths detected to another ME as a metadata. Parameter's value: - 0 (default), metadata are not sent. - 1, metadata are sent.

Table 4: Mouth detector API

6.1.5 Inputs, outputs and API of the ear detector filter

This filter will also receive a stream of images as input. The output of the filter will be a collection of bounding box. Each bounding box represents the position of each ear found in the image.

In the same way as the nose and mouth detector algorithm, this algorithm needs to detect previously the different faces included on the image. But in this case, the faces find in the image must be profile faces and not front ones. Therefore, this filter need to detect profile faces by its own, since there is not going to be any filter which detects profile faces. Once the profile faces have been detected, it will proceed to detect an ear in those parts of the image where there is a profile face. Moreover, all the bounding boxes representing the profile faces can be reduced by approximately 30 percent the height of said regions as both the top and the bottom of the bounding box. In this way, we remove not only information which is not useful for detecting ears as chin and forehead, but also we can reduce processing time and improve the likelihood of success of the algorithm.

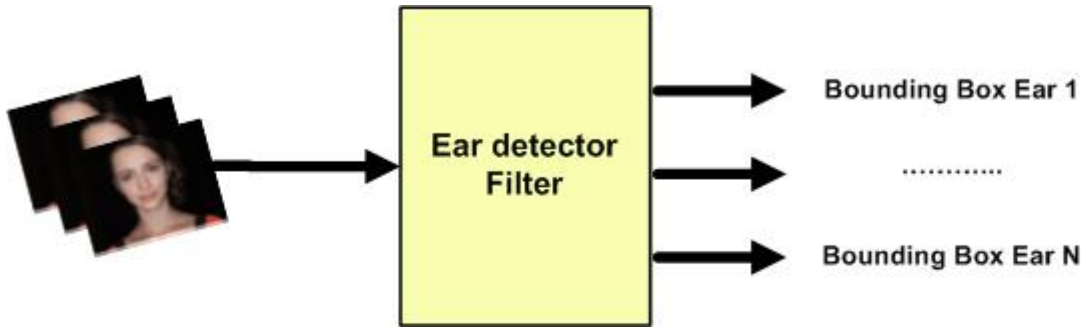

Figure 9: Face Detection Filter Input and Outputs.

Developers can use the filter's API for its setup. Then, the API is shown:

Function	Description
void viewEars**(int);**	To show or hide the bounding boxes of the detected ears within the image. Parameter's value: - 0 (default), the bounding boxes will not be shown. - > 0, the bounding boxes will be drawn in the frame.
void detectByEvent (int);	To indicate to the algorithm if it must process all the

	images or only when it receives a specific event such as face detection. Parameter's value: - 0 (default) , process all the frames; - 1, process a frame when it receives a specific event
void sendMetaData (int);	To send the bounding boxes of the mouths detected to another ME as a metadata. Parameter's value: - 0 (default), metadata are not sent. - 1, metadata are sent.

Table 5: Ear detector API

6.2 Motion detection

This algorithm has been selected to be included on the NUBOMEDIA platform for the following reasons:

- The main reason to include this algorithm on NUBOMEDIA project is a strategic decision. Since this is a typical functionality demanded by our customer, we are interested in using this algorithm on a cloud infrastructure where the filter can be combined with others to explore the benefits this algorithm can bring to customers.
- The core of the technology has already been developed, and only some minor changes have been added to the filter. This has allowed us to be agile in the generation of the filter

6.2.1 Details of the algorithm

For this algorithm, we are using a Visual Tools' proprietary software. Therefore, many details of the algorithm are not going to be reflected on this document.

This filter splits the images in a matrix of 10x8 blocks and weight up the changes that happen in each block between two consecutive frames. At the end of the processing, if the number of blocks where has been detected movement is higher than a specific threshold, the conclusion of the algorithm is that there has been movement. This threshold indicates the minimum number of blocks where motion has been detected.

Apart from this threshold, it is also configurable the blocks of the matrix which are going to be processed. If you look at the following image, you will see an image with the 10x8 blocks corresponding to the matrix. You can also observe some white points in some specific blocks. These points indicate the blocks where the algorithm will carry out the processing.

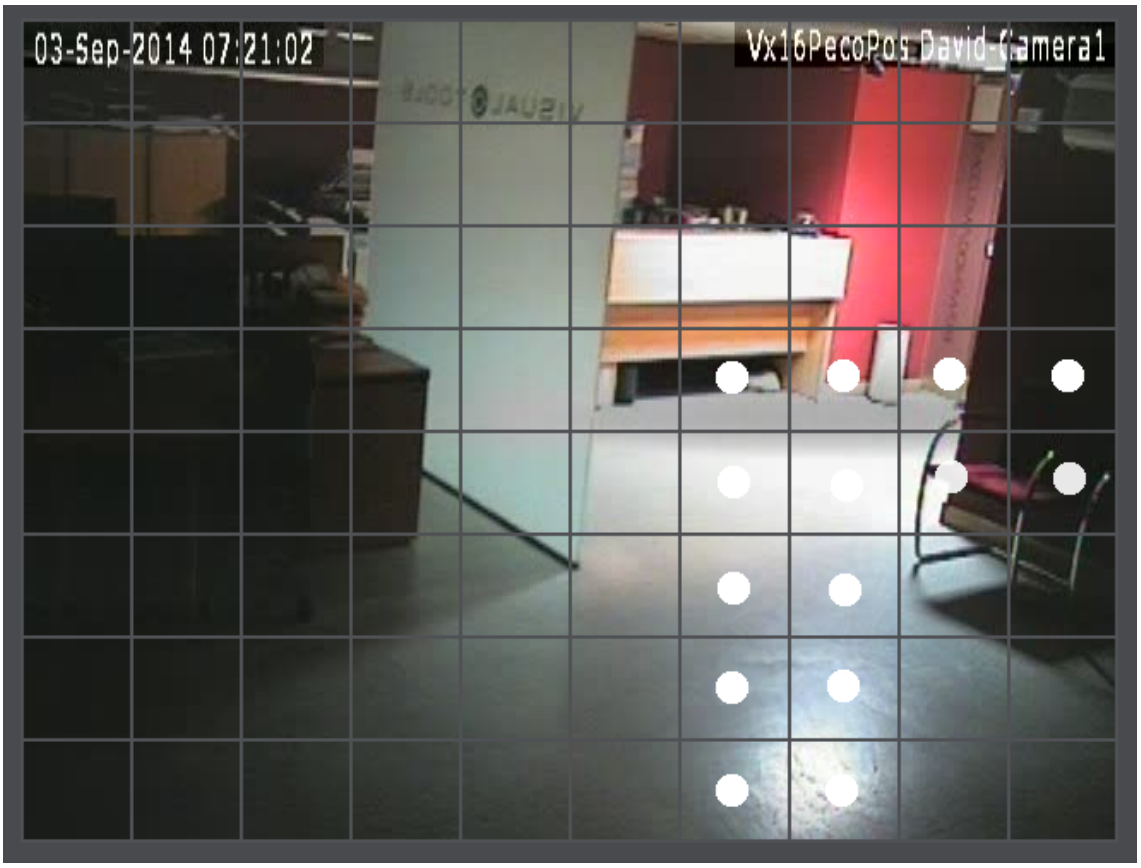

Figure 10: 10x8 matris and activated blocks

The detection of motion is based on a corner detector or high curvature points algorithm [2]. A corner can be defined as the intersection of two edges, or in other words corners are the features of a digital image that correspond to the points with high spatial gradient and high curvature. Once the algorithm detects a corner in a frame, it tries to find exactly the same point in the previous frame in order to detect whether there has been movement or not. To conclude that there has been movement in a given block, the number of corners where there has been movement must exceed a certain threshold.

6.2.2 Input and outputs of the filter

This filter will receive a stream of images as input. The output of the filter will be a 1 or a 0. The number 1 indicates that there has been motion detection. Otherwise, if there has not detected any motion, the output of the filter will be 0. It is very important to highlight that this algorithm works very well for **indoors**, while outdoors, the result may be inconsistent.

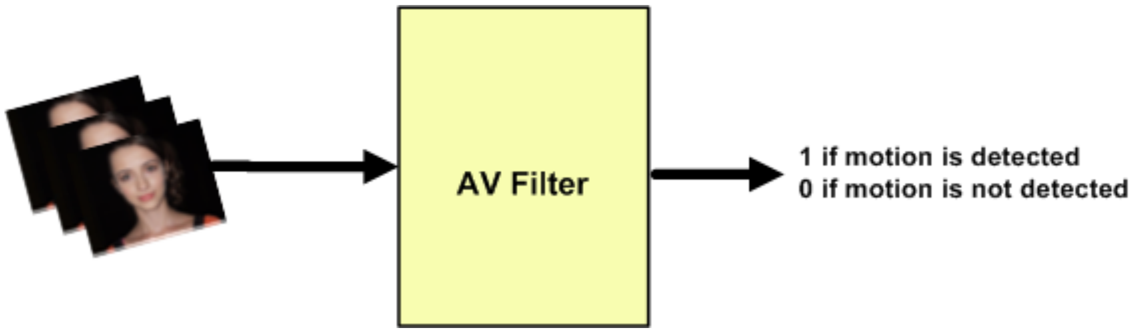

Figure 11: Motion detection Filter Input and Outputs.

Figure 12: Face Detection Filter Input and Outputs.

Developers can use the filter's API for its setup. Then, the API is shown:

Function	Description
void setMinNumBlocks**(String);**	To set the minimum number the blocks of the grid where the algorithm must detect movement to conclude that there has been motion in the image. Parameter's value: - "high", detect movement in 1 block at least - "medium", detect movement in 3 blocks at least - "low", detect movement in 9 blocks at least
void setGrid **(String);**	To set the areas (grid) where the algorithm will detect motion. Parameter's value: - The string will be composed by 80 characters. Each character is matched to a block of the grid. The value of character can be 0 (It does not matter to detect motion at this block) or 1 (it matters to detect motion at this block).
void applyConf **();**	After set up the minimum number of blocks and set the grid, we need to call this function to make the changes effective
void sendMetaData (int);	To send and event to indicate to another ME that it has been detected movement in the image. Parameter's value: - 0 (default), metadata are not sent. - 1, metadata are sent.

Table 6: Motion detector API

7 Implementation status

In this section, it is described the license of the different algorithms, how to access the source code and install all the modules developed in the associated task.

7.1 License

Algorithm	License
Face detector	LGPL v2.1
Nose detector	LGPL v2.1
Mouth Detector	LGPL v2.1
Ear Detector	LGPL v2.1
Motion detector	Proprietary Software

Table 7: VCA algorithms license

7.2 Obtaining the source code

All the algorithms described in this document can be obtained in the NUBOMEDIA Git repository accessible at the following URL:

- http://80.96.122.50/victor.hidalgo/vca_media_elements.git

In the repository you can find the algorithms developed up to know in the following links

Algorithm	Git link
NuboFaceDetector	http://80.96.122.50/victor.hidalgo/vca_media_elements/tree/master/modules/nubo_face
NuboMouthDetector	http://80.96.122.50/victor.hidalgo/vca_media_elements/tree/master/modules/nubo_mouth
NuboNoseDetector	http://80.96.122.50/victor.hidalgo/vca_media_elements/tree/master/modules/nubo_nose
NuboEarDetector	http://80.96.122.50/victor.hidalgo/vca_media_elements/tree/master/modules/nubo_ear
NuboMotionDetector	Proprietary Software. No source code provided. This software will be only used with the consent of the owner, in this case Visual Tools. The binaries will be provided to the project consortia for the duration of the project.

Table 8: Repository of the VCA modules

In addition, you can find the demos developed up to now which use the different media elements in the following links:

Demo	Description and Link
motionFaceJava	*Description:* pipeline created by a motion and Face detector http://80.96.122.50/victor.hidalgo/vca_media_elements/tree/master/apps/motionFaceJava
NuboFaceJava	*Description*: pipeline created by a face detector http://80.96.122.50/victor.hidalgo/vca_media_elements/tree/master/apps/NuboFaceJava
NuboMouthJava	*Description*: pipeline created by a mouth detector

	http://80.96.122.50/victor.hidalgo/vca_media_elements/tree/master/apps/NuboMouthJava
NuboNoseJava	*Description*: pipeline created by a nose detector http://80.96.122.50/victor.hidalgo/vca_media_elements/tree/master/apps/NuboNoseJava
NuboEarJava	*Description*: pipeline created by an ear detector http://80.96.122.50/victor.hidalgo/vca_media_elements/tree/master/apps/NuboEarJava
NuboFaceProfileJava	Description: pipeline created by a face, mouth and nose detector http://80.96.122.50/victor.hidalgo/vca_media_elements/tree/master/apps/NuboFaceProfileJava

Table 9: Repository of the VCA demos

7.3 Install and test the filters.

Now it is described the steps performed to install the VCA filters and demos developed in the images that will deploy on the NUBOMEDIA cloud. It is important to highlight that the images that are going to be deployed in the cloud include filters developed in the associated task and those filters that already exist in the Kurento infrastructure.

- **Install Kurento Media Server and its dependencies.**

See the deliverable D4.1.1 which provides an installation guide for Kurento Media Server

- **Install NUBOMEDIA VCA filters.**

```
cd /tmp/
wget http://vps.dnsvideo.net:88/nubomedia_install_vca_elements.sh
chmod +x nubomedia_install_vca_elements.sh
sudo sh nubomedia_install_vca_elements.sh
```

After finishing the installation and deploy an image on the NUBOMEDIA cloud, if you want to test the different demos you need to access the following urls (where ip-machine-deployed is the ip of the machine deployed in the NUBOMEDIA cloud):

- Face Detector =>	http://ip-machine-deployed:8100
- Motion + Face Detector =>	http://ip-machine-deployed:8101
- Nose Detector =>	http://ip-machine-deployed:8102
- Mouth Detector =>	http://ip-machine-deployed:8103
- Ear Detector =>	http://ip-machine-deployed:8104
- Face + Nose + Mouth Detector =>	http://ip-machine-deployed:8105

The following figures show an example of the pipeline created by the Face, Nose and Mouth detector demo, and the proper demo running.

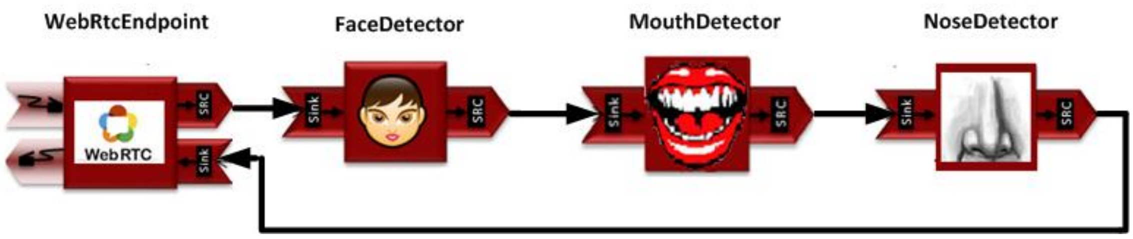

Figure 13. Face Mouth and Nose detector pipeline

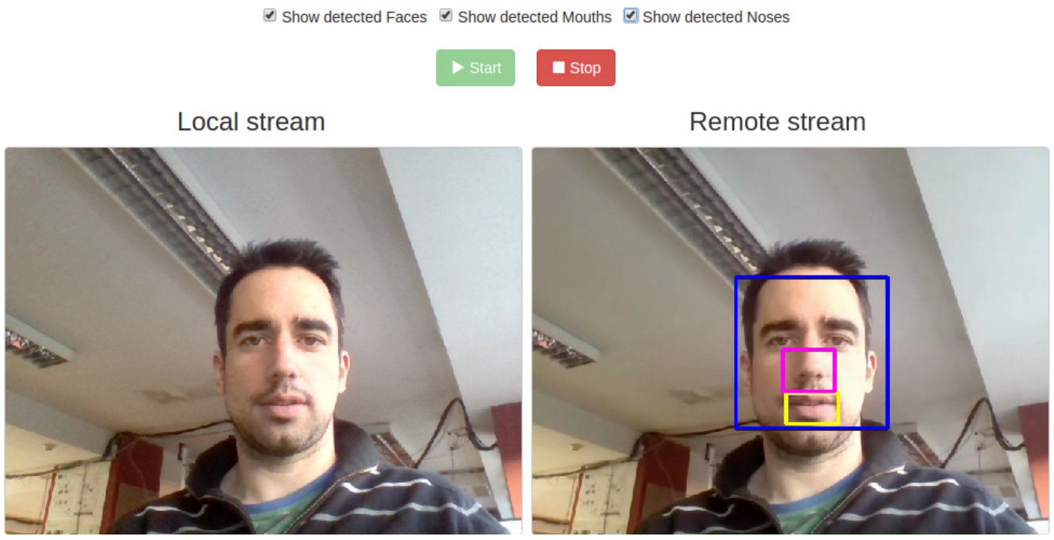

Figure 14: Face, Mouth and Nose detector running

References

[1] Rapid Object Detection using a Boosted Cascade of Simple Features. Paul Viola and Michael Jones. https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

[2] A combined corner and edge detector, Chris Harris & Mike Stephens. http://courses.daiict.ac.in/pluginfile.php/13002/mod_resource/content/0/References/harris1988.pdf

[3] OpenCv library, http://opencv.org/.