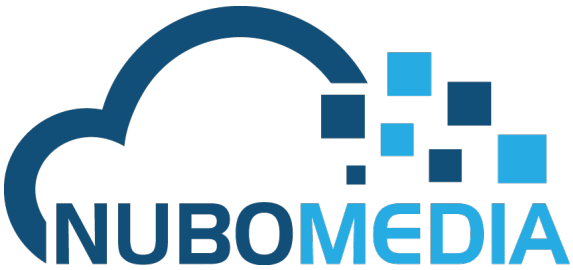
D4.5.1

Version	1.0
Author	VTT
Dissemination	PU/
Date	27/01/2015
Status	FINAL



D4.5.1.: Augmented Reality media element prototypes v1

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: NUBOMEDIA live multimedia cloud elements
WP leader	NAEVATEC/Javier López
Deliverable nature:	Prototype
Lead editor:	Petri Honkamaa
Planned delivery date	01/2015
Actual delivery date	27/01/2015
Keywords	Augmented reality

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:

Petri Honkamaa (VTT),
Satu-Marja Mäkelä (VTT),
Markus Ylikerälä (VTT),
Javier Lopez Fernandez (Naevatec)
Ivan Gracia(Naevatec)

Internal Reviewer(s):

Luis Lopez (URJC)

Version History

Version	Date	Authors	Comments
0.1	20.9.2014	Petri Honkamaa	Based on the 1st integration of the ALVAR
0.2	22.09.2014	Satu-Marja Mäkelä	Executive summary/first page
0.3	8.12.2014	Satu-Marja Mäkelä	Details/draft for R3
0.4	15.12.2014	Petri Honkamaa	Updated text to match with ar-markerdetector 0.0.3
0.5	17.12.2014	Markus Ylikerälä	Integration of 3D rendering
0.6	17.12.2014	Javier Lopez Fernandez	Face Augmentation Filter
0.7.	17.12.2014	Satu-Marja Mäkelä	Editing/executive summary
0.8.	8.1.2015	Petri Honkamaa	Changed structure
0.81	11.1.2015	Satu-Marja Mäkelä	Changed structure/Comment answering
0.9	18.1.2015	Ivan Gracia	Architecture for Face Overlay
1.0	20.1.2015	Satu-Marja Mäkelä	Finalising

Table of contents

List of Figures:

Acronyms and abbreviations:

AR augmented reality
FBO frame buffer object

1 Executive summary

NUBOMEDIA task 4.5 for Augmented Reality (AR) cloud elements concentrates on developing and integrating augmented reality capabilities to the NUBOMEDIA platform server (Kurento Media Server). This document introduces two custom modules for NUBOMEDIA: 1) Augmented reality tools containing marker detection, rendering 2D content on the marker position. This is based on ALVAR Desktop which has been developed by the VTT Technical Research Centre of Finland. Also an extension to the 3D content rendering with 3rd party open source library is integrated to Kurento to support further development of 3D content for AR. 2) Face overlay filter that augments an image on top of every face detected in video frames. The deliverable describes the situation after first project year (M12). The final API's and functionalities for AR filters will be released in M20 in D4.5.2

2 Module for marker Augmented Reality: the ar-markerdetector filter

2.1 Background

2.1.1 ALVAR desktop library

ALVAR is a software library for creating virtual and augmented reality (AR) applications. ALVAR has been developed by the VTT Technical Research Centre of Finland. ALVAR is released under the terms of the GNU Lesser General Public License, version 2.1, or (at your option) any later version.

ALVAR is designed to be as flexible as possible. It offers high-level tools and methods for creating augmented reality applications with just a few lines of code. The library also includes interfaces for all of the low-level tools and methods, which makes it possible for the user to develop their own solutions using alternative approaches or completely new algorithms.

ALVAR is currently provided on Windows and Linux operating systems and only depends on one third party library (OpenCV). ALVAR is independent of any graphical libraries and can be easily integrated into existing applications.

Supported features:

- Marker based tracking
  - accurate marker pose estimation
  - two types of square matrix markers
  - future marker types are easy to add
- Using multiple markers for pose detection
  - with fixed multi-marker setup we can track also when some of the markers are occluded
  - the marker setup coordinates can be set manually
  - or they can be automatically deduced by autocalibration
- Other
  - hiding markers from view
  - tools for calibrating cameras

- several methods for tracking optical flow
- distoring/undistorting points, projecting points
- finding exterior orientation using point-sets
- Kalman library and several other filters

Note, initially only part of these features (mainly Marker based tracking) will be supported on NUBOMEDIA platform.

2.2 Objectives

The main objective of the AR marker detection filter is provide tools for marker based augmentation of the 2D/3D content. The ar-marker detection filter will support detection of marker and rendering 2D content on the marker in real time.

2.3 Strategy

2.3.1 ar-markerdetector filter

As part of this deliverable, we have created a marker based augmented reality custom module for Kurento Media Server. This module has been named *ar-markerdetector and* is based on the ALVAR open source library. It detects an ALVAR marker (see Figure 1) from the video image and it can overlay the detected marker pose either with text or image.

General information of ALVAR can be found from http://virtual.vtt.fi/virtual/proj2/multimedia/alvar/index.html . The ALVAR version used here can be found from NUBOMEDIA git http://git.nubomedia.eu/Markus.Ylikerala/vtt-alvar. Currently the latest version for ALVAR is 0.0.3.

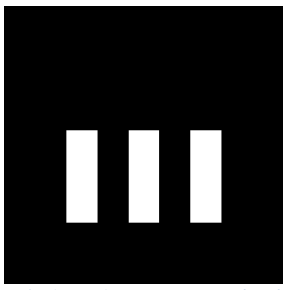


Figure 1. ALVAR Marker with id 251

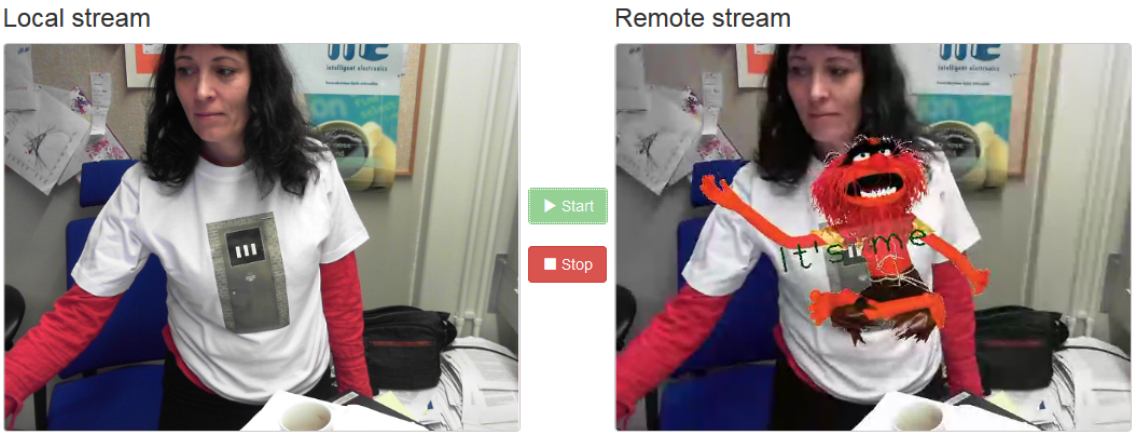


Figure 2. Example how ar-markerdetector can overlay the marker on video stream with text/image

Note, that currently we visualize the detected marker pose by overlaying it with text/image. Example is in figure 2, where content is rendered over the marker in T-Shirt. In practice it would make more sense to send the detected marker pose to client (e.g. as events), and the client can make the augmentation. This way we can effectively use display driver on the client side. However, in addition to this we plan to implement also basic 3D rendering capabilities for the server (see Section 2.6.4 for more details).

2.4 Architecture

Ar-markerdetector is developed as Kurento Media server modules. The ar-markerdetector is based on an example opencv-filter. The basic code for the implementation is generated automatically by Kurento, but we have the basic augmented reality processing in a separate `ArProcess` class. This way the core implementation is safe, even if interfaces change and we need to re-generate the code. `ArProcess` uses `alvar::MarkerDetector` to detect the marker and OpenCV to draw the augmentation. See the class diagram below for the key parts.

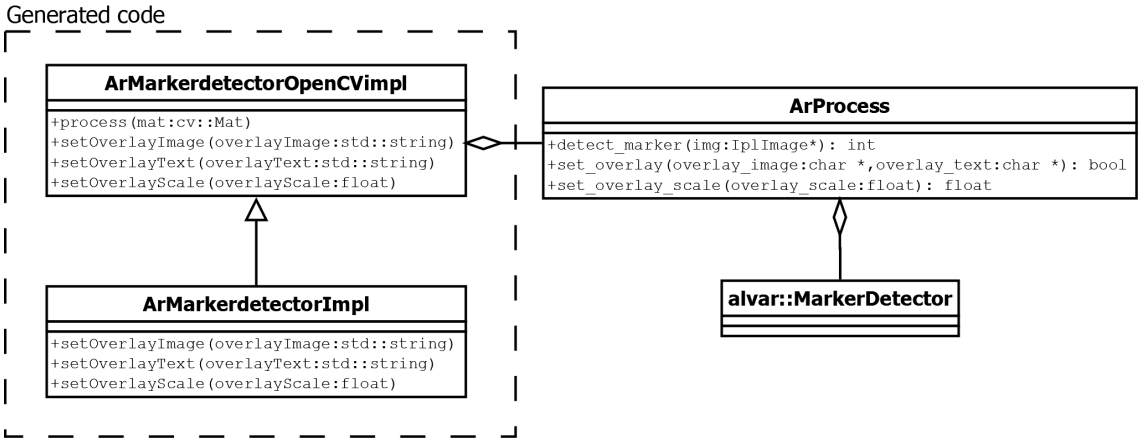


Figure 3. The relations of key components in ar-markerdetector.

The ar-markerdetector module is using on ALVAR Desktop (See **¡Error! No se encuentra el origen de la referencia.**), which again uses the OpenCV computer vision library. See the dependency diagram below.

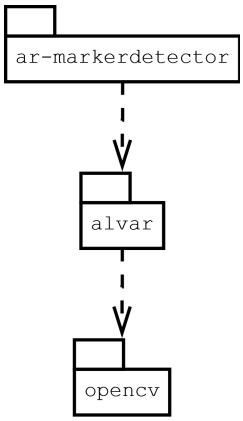


Figure 4. ar-markerdetector module dependencies

See the Kurento documentation (http://kurento.com/) for a more complete view of the Kurento architecture.

2.5 Implementation status: ar-markerdetector

2.5.1 Licensing

ALVAR Desktop is under GNU Lesser General Public License, version 2.1. For more details see the ALVAR Desktop home page:

http://virtual.vtt.fi/virtual/proj2/multimedia/alvar/index.html

Irrlicht library is free open source 3D engine and is used in the future for 3D rendering. The license is based on zlib/libpng license text. For more details see the Irrlicht home page:

http://irrlicht.sourceforge.net/license/

2.5.2 Getting source code and documentation

The ar-markerdetector binaries (including alvar source/binaries) can be downloaded from Nubomedia git:

http://git.nubomedia.eu/Markus.Ylikerala/vtt-armarkerdetector/tree/master
http://git.nubomedia.eu/Markus.Ylikerala/vtt-alvar

2.5.3 Installing dependencies

The *ar-markerdetector* is used in Kurento Media Server and internally it uses ALVAR augmented reality library. Both of these need to be installed before using this module. Instructions for installing Kurento Media Server can be found from NUBOMEDIA deliverable D4.1.1 or check the latest instructions from http://www.kurento.org/
We have been updating and upgrading always to the latest Kurento version available at the time of development.

The source for ALVAR augmented reality library is available from the following git repository, which also contains instructions for cloning:
http://git.nubomedia.eu/Markus.Ylikerala/vtt-alvar
git clone http://80.96.122.50/Markus.Ylikerala/vtt-alvar.git

A binary release is also available. These are the instruction to fetch and utilize it:

```
wget http://ssi.vtt.fi/ar-markerdetector-binaries/alvar-2.0.0-sdk-linux64-gcc44.tar.gz
tar xvfz alvar-2.0.0-sdk-linux64-gcc44
cd alvar-2.0.0-sdk-linux64-gcc44/bin
sudo cp libalvar200.so /usr/lib
```

2.6 Installing ar-markerdetector in Kurento Media Server

Developers guide chapter in this document describes how to get the latest version of ar-markerdetector filter source codes. These are the instructions how to utilize a binary version of ar-markerdetector module. Thus, once the above mentioned dependencies are setup, you can install the binary version of ar-markerdetector module from a debian packet.

```
wget http://ssi.vtt.fi/ar-markerdetector-binaries/ar-markerdetector-
dev_0.0.3~rc1_amd64.deb
dpkg -i ar-markerdetector-dev_0.0.3~rc1_amd64.deb
sudo /etc/init.d/kurento-media-server restart
```

As all custom Kurento Media Server modules, the KMD IDL compilation generates proxy classes for Java and JavaScript than can be used by developers for creating applications consuming ar-markerdetector capabilities.

For using ar-markerdetector from a Java Kurento Client, you need to obtain the appropriate libraries. You can do it using the following commands.

```
wget http://ssi.vtt.fi/ar-markerdetector-binaries/ar-
markerdetector_0.0.3~rc1_java/armarkerdetector-0.0.3-SNAPSHOT.jar
wget http://ssi.vtt.fi/ar-markerdetector-binaries/ar-
markerdetector_0.0.3~rc1_java/pom.xml
mvn install:install-file -Dfile=armarkerdetector-0.0.3-SNAPSHOT.jar -DpomFile=
pom.xml
```

2.6.1 AR marker detector API

Full description of the arMarkerdetector API is in Annex1. Following parameters can be found from the JSON description See the above example how these are changed.

Parameter	Description
show-debug-info	Boolean flag indicating if we will show some debug information from the marker detector.
overlay-text	String indicating a text string to be overlaid on the marker.
overlay-imagefile	Path to an image file on the server, which should be overlaid on the marker. This can also be an URL for remote file.
overlay-scale	Currently both the augmented text string and augmented image are scaled to match the marker size. This parameter will allow the content to be scaled either larger or smaller compared to the marker.
rtId	Integer to give a runtime-id to each client e.g. for developing purposes
width	Integer to tell the width of the input video e.g. for initializing the

	AR data structures
height	Integer to tell the height of the input video e.g. for initializing the AR data structures

Event	MarkerCount	
Properties	markerID	ID number of detected marker
	markerCount	Number of visible markers with the specified id
	markerCountDiff	Delta of markercount compared to previous event

2.6.2 Trying ar-markerdetector by modifying magic-mirror example

You can try the markerdetector out by modifying the kurento magic-mirror example.

```
> git clone https://github.com/Kurento/kurento-tutorial-java.git
> cd kurento-tutorial-java/kurento-magic-mirror
> gvim pom.xml # Add dependency

        <dependency>
                <groupId>org.kurento.module</groupId>
                <artifactId>armarkerdetector</artifactId>
                <version>0.0.3-SNAPSHOT</version>
        </dependency>

> gvim src/main/java/org/kurento/tutorial/magicmirror/MagicMirrorHandler.java

        // Add the needed additional imports
        import org.kurento.module.armarkerdetector.ArMarkerdetector;
        import org.kurento.module.armarkerdetector.MarkerCountEvent;
        import org.kurento.client.*;

                ArMarkerdetector arMarkerdetector = new
ArMarkerdetector.Builder(pipeline).build();
        arMarkerdetector.setShowDebugLevel(0);
        arMarkerdetector.setOverlayText("Hello World!");
        arMarkerdetector.setOverlayImage("http://ssi.vtt.fi/hawaii-shirt.png");
        arMarkerdetector.addMarkerCountListener(new EventListener<MarkerCountEvent>() {
                @Override
                public void onEvent(MarkerCountEvent event) {
                  String result = String.format("Marker %d count:%d (diff:%d): {}",
event.getMarkerId(), event.getMarkerCount(), event.getMarkerCountDiff());
                  log.debug(result, event);
                }
            });

> mvn compile exec:java # Execute the example
```

Try it out in the web browser: http://localhost:8080/

2.6.3 Developers guide

Instead of using debian package (see chapter 2.6), you can also get the latest *ar-markerdetector* sources directly from our git repository and compile them yourself. This way you can always get up-to-date version and make your own modifications.

The source for *ar-markerdetector* is available from the following git repository, which also contains instructions for cloning.

```
sudo apt-get install git
```

```
git clone http://80.96.122.50/Markus.Ylikerala/vtt-armarkerdetector.git
cd vtt-armarkerdetector/ar-markerdetector
mkdir build
cd build
cmake .. -DGENERATE_JAVA_CLIENT_PROJECT=TRUE
```

To use the compiled results in Kurento, you can call "`make install`" and then restart kurento-media-server. However, for developing purposes it can be more feasible to use directly the versions you compile into your working directory.

```
> sudo gvim /etc/default/kurento-media-server

        export KURENTO_MODULES_PATH=/home/alvar/kurento/ar-markerdetector/build
        export GST_PLUGIN_PATH=/home/alvar/kurento/ar-markerdetector/build
        export LD_LIBRARY_PATH=/home/alvar/kurento/alvar-2.0.0-sdk-linux64-gcc44/bin/
```

This way, each time you change the ar-mediaserver, it is enough to compile the sources and restart the kurento-media-server:

```
> make
> make java_install
> sudo /etc/init.d/kurento-media-server restart
```

The compiled *ar-markerdetector* module can be used exactly like the one installed from debian package (see Chapter 2.6.2).

Annex 1 Notes on how the ar-markerdetector module was made and Annex 2 Troubleshooting gives more insight and more detailed matters on the implementation.

2.6.4 Integration of 3D rendering

After ar-markerdetector filter version 0.0.3, development of version 0.0.4 has started. The new functionality enables rendering with OpenGL a 3D model over a video ie AR-rendering.
Irrlicht, a crossplatform real-time 3D engine, is currently chosen http://irrlicht.sourceforge.net/
The license of Irrlicht is very permissive and suitable http://irrlicht.sourceforge.net/license/ and the features enables feasible implementation.

Irrlicht code can be checked from:

```
> svn checkout http://svn.code.sf.net/p/irrlicht/code/trunk irrlicht-code
```

If you want to tweak Irrlicht, just modify the Irrlicht configuration header before compiling. The process for compiling is for example:

```
cd irrlicht-code/src/Irrlicht
make sharedlib NDEBUG=1
sudo make install
sudo ldconfig
```

AR-rendering is implemented as a component that is injected into the previous ar-markerdetector filter video processing flow. The filter functionality of AR-renderer is implemented as render to texture (RTT) with OpenGL. On Linux environment rendering with OpenGL finally utilizes XServer. Thus, Irrlicht needs XServer and DISPLAY environment variable.

```
sudo apt-get install xinit
startx
export DISPLAY=:0
```

Figure 3 shows an example of the development version 0.0.4 implementation where the end-user utilized a web browser to see a local web camera video stream unprocessed, on the left side and a 3D faerie model rendered over a video stream with ar-markerdetector, on the right side.
In detail the filtering steps for the process is following:

1. a video frame as OpenCV image is passed to the AR-renderer
2. the frame is converted to a Irrlicht image
3. the image is rendered to the Irrlicht render target texture
4. a 3D model is drawn to the render target texture
5. image data is taken from the render target texture
6. AR-renderer passes the image data back
7. the image data is substituted to the OpenCV video frame

During the development of AR-rendering it was observed that the resolution of the video frames that are passed to the filter can vary at run-time. The current solution resizes the frames based on the initial video input resolution but also re-initialization of Irrlicht based on the new resolution was considered. It is not yet known if change of the resolution causes some incidents to marker recognition. Note that the marker shown in the Figure 3 is for illustration purposes only. Thus, rendering 3D content on the marker position is going to be implemented in the upcoming version based on the previous 2D solution.

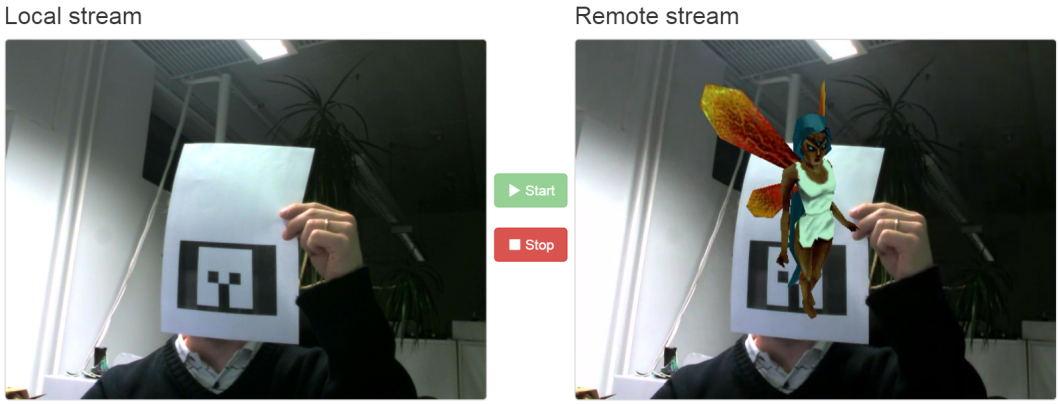


Figure 5 Ar-markerdetector example of rendering 3D model over marker on video stream

3 Implementation status: face overlay filter

3.1 General approach and background

The face overlay filter is a native MediaElementKurento Media Server module that is distributed off the shelf with it. It that superposes an image on top of every face detected in video frames. The filter dynamically adjusts the size and position of the image based oin the location and size of the detected face. It uses OpenCV as VCA engine for face detection and an alpha blending filter that overlays the image on top of the video frame.

This filter does not have any practical use, apart of showing Kurento capabilities and become a template for AR Filter development.

The following image depicts the hierarchy of the filter.

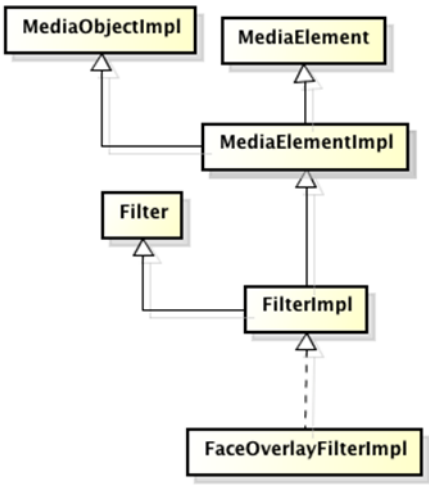


Figure 6: FaceOverlayFilter UML diagram

It is internally composed by two GStreamer elements: KmsFaceDetector, and KmsImageOverlay. The former is in charge of detecting faces, and sending its positions to the next filter. The latter is responsible for actually overlaying the image in the indicated position. In Figure 7, the internal GStreamer structure can be appreciated.

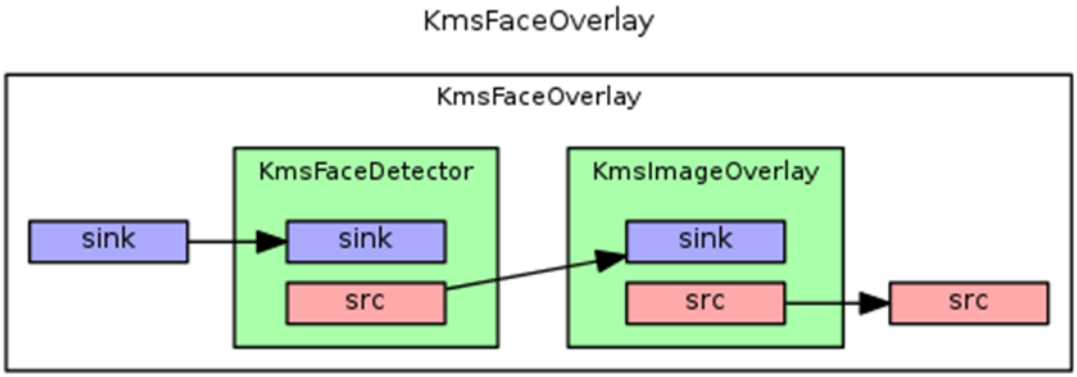


Figure 7 FaceOverlayFilter UML diagram

The face overlay filter is a native MediaElementKurento Media Server module that is distributed off the self with it. It that superposes an image on top of every face detected in video frames. The filter dynamically adjusts the size and position of the image based oin the location and size of the detected face. It uses OpenCV as VCA engine for face detection and an alpha blending filter that overlays the image on top of the video frame. This filter does not have any practical use, apart of showing Kurento capabilities and become a template for AR Filter development.

3.2 Build and install the Face overlay filter

The Face overlay filter is a native component of Kurento Media Server and it is build together with the reset of the platform.
Source code can be found in following repository

- https://github.com/Kurento/kms-filters

In order to build this filter from source code, follow the instructions to compile Kurento Media Server, that can be found in the link below

- https://github.com/Kurento/kurento-media-server/blob/develop/README.md

This filter can be also installed from Kurento's binary repository. In order to do that it is required to follow the installation instructions documented in the link below

- http://www.kurento.org/docs/current/installation_guide.html

3.3 Use face overlay filter with Kurento

As any other module, the Face overlay filter provides a control API enabling applications to insert and control this filter into Kurento media pipelines. Complete documentation on how to use Kurento API can be found in link below

- http://www.kurento.org/docs/current/mastering/kurento_API.html

The reference guide for the Face overlay filter is documented in links below

- Java client: http://www.kurento.org/docs/current/langdoc/javadoc/index.html
- JS client: http://www.kurento.org/docs/current/langdoc/jsdoc/kurento-client-js/module-filters.FaceOverlayFilter.html

A demonstrator for this filter is available in Kurento tutorials that are placed in following source code repository:

- https://github.com/Kurento/kurento-tutorial-java/tree/master/kurento-magic-mirror

A binary version of this demonstrator can be downloaded from:

- http://builds.kurento.org/dev/latest/tutorials/kurento-magic-mirror.zip

3.4 Virtual Machine for Face Overlay

Below are the instructions how install and configure a virtual machine for face overlay

3.4.1 General info

- Repository
- Jenkins builder

3.4.2 Install packages

Install Kurento Media Server

```
add-apt-repository ppa:kurento/kurento -y
apt-get install kurento-media-server -y
```

Install Naevatec/URJC Kurento Media Server Modules

```
apt-get install kms-markerdetector -y
```

Install JDK

```
apt-get install default-jdk -y
```

Install maven 3

```
apt-get install maven -y
```

Install apache web server

```
apt-get install apache2
```

Install Naevatec/URJC JS Tutorials

```
# Install JS Tutorials
rm -rf /var/www/html/*
rm -rf /var/www/html/.*
cd /var/www/html
git clone https://github.com/Kurento/kurento-tutorial-js.git .
chown -R ubuntu.ubuntu /var/www/html
for dir in $(sudo ls -d */) ; do
   cd /var/www/html/$dir
   sudo -H -u ubuntu bash -c 'yes n | bower install'
done
```

Install Naevatec/URJC Java Tutorials

```
mkdir -p /tmp/kurento-magic-mirror
cd /tmp/kurento-magic-mirror
wget http://builds.kurento.org/dev/latest/tutorials/kurento-magic-mirror.zip
unzip kurento-magic-mirror.zip
./install.sh
```

3.5 Configurations

Disable IPV6

```
sed -i '$ a\net.ipv6.conf.all.disable_ipv6 =
 1\nnet.ipv6.conf.default.disable_ipv6 = 1\nnet.ipv6.conf.lo.disable_ipv6 = 1'
/etc/sysctl.conf
sysctl -p
```

Replace file /etc/kurento/kurento.conf.json with content

```
{
"mediaServer" : {
       "net" : {
              "websocket": {
                     "port": 8888,
                     "path": "kurento",
                     "threads": 10
              }
```

```
        }
    },
    "modules": {
        "kurento": {
            "SdpEndpoint" : {
                "sdpPattern" : "sdp_pattern.txt"
            },
            "WebRtcEndpoint" : {
                "stunServerAddress" : "77.72.174.167",
                "stunServerPort" : 3478
            }
        }
      }
    }
```

3.6 Services

Enable autostart of Kurento Media Server on boot

```
update-rc.d kurento-media-server defaults
```

Enable autostart of Demonstrators

```
update-rc.d kurento-magic-mirror defaults
```

4 Future Roadmap

In the future development the first task is to support fully 3D rendering with AR which requires integrating the 3D rendering to the ar-markerdetection filter. We cca also support markerless augmented tracking via VTT's proprietary tracking technology. This will be implemented on the server side if there is need for this in the requirements.

5 Conclusions

The document presents the status of the task 4.5 Augmented Reality cloud elements. Two augmented reality filters, Face Overlay filter and arMarker detection, have been implemented in NUBOMEDIA for Kurento Media server. Also support for 3D rendering with Irrlicht have been implemented to the Kurento. The development of the AR filters is still on going and the work will be finalized in next 8 months.

References

Annex 1

```
{
  "remoteClasses": [
    {
      "name": "ArMarkerdetector",
      "extends": "OpenCVFilter",
```

```
      "doc": "ArMarkerdetector interface. Documentation about the
module",
      "constructor": {
        "doc": "Create an element",
        "params": [
          {
            "name": "mediaPipeline",
            "doc": "the parent :rom:cls:`MediaPipeline`",
            "type": "MediaPipeline",
            "final": true
          }
        ]
      },
      "methods": [
        {
          "name": "setRtId",
          "doc": "TODO",
          "params": [
            {
              "name": "setRtId",
              "doc": "set run-time id for developin purposes",
              "type": "int"
            }
          ]
        },
        {
          "name": "setWidth",
          "doc": "TODO",
          "params": [
            {
              "name": "setWidth",
              "doc": "tell width of camera image",
              "type": "int"
            }
          ]
        },
        {
          "name": "setHeight",
          "doc": "TODO",
          "params": [
            {
              "name": "setHeight",
              "doc": "tell height of camera image",
              "type": "int"
            }
          ]
        },
        {
          "name": "setShowDebugLevel",
          "doc": "TODO",
          "params": [
            {
              "name": "showDebugLevel",
              "doc": "show debug info on image",
              "type": "int"
            }
          ]
        },
        {
          "name": "getShowDebugLevel",
          "doc": "TODO",
          "params": [],
          "return": {
```

```
      "name": "showDebugLevel",
      "doc": "show debug info on image",
      "type": "int"
    }
  },
  {
    "name": "setOverlayImage",
    "doc": "TODO",
    "params": [
      {
        "name": "overlayImage",
        "doc": "TODO",
        "type": "String"
      }
    ]
  },
  {
    "name": "getOverlayImage",
    "doc": "TODO",
    "params": [],
    "return": {
      "name": "overlayImage",
      "doc": "TODO",
      "type": "String"
    }
  },
  {
    "name": "setOverlayText",
    "doc": "TODO",
    "params": [
      {
        "name": "overlayText",
        "doc": "TODO",
        "type": "String"
      }
    ]
  },
  {
    "name": "getOverlayText",
    "doc": "TODO",
    "params": [],
    "return": {
      "name": "overlayText",
      "doc": "TODO",
      "type": "String"
    }
  },
  {
    "name": "setOverlayScale",
    "doc": "TODO",
    "params": [
      {
        "name": "overlayScale",
        "doc": "TODO",
        "type": "float"
      }
    ]
  },
  {
    "name": "getOverlayScale",
    "doc": "TODO",
    "params": [],
    "return": {
```

```
            "name": "overlayScale",
            "doc": "TODO",
            "type": "float"
          }
        }

  "events": [
    {
      "name": "MarkerCount",
      "extends": "Media",
      "doc": "An event that is sent when the number of visible markers
is changed. Tracking coordinates for the markers is going to be sent
with some      ],
      "events": [
        "MarkerCount"
      ]
    }
  ],
other approach.",
      "properties": [
        {
          "name": "markerId",
          "doc": "marker id",
          "type": "int"
        },
        {
          "name": "markerCount",
          "doc": "Number of visible markers with the specified id",
          "type": "int"
        },
        {
          "name": "markerCountDiff",
          "doc": "How much the markerCount was changed from the
previous situation",
          "type": "int"
        }
      ]
    }
  ]
}
```

Annex 2 Notes on how the ar-markedetectro module was made

Generate module based on opencv-filter and describe the interface in armarkerdetector.ArMarkerdetector.kmd.json.

```
> kurento-module-scaffold.sh ArMarkerdetector . huuhaa
> cd ar-markerdetector
> gvim src/server/interface/armarkerdetector.ArMarkerdetector.kmd.json
```

Every time interface is changed you need to regenerate the related codes.

```
> mv src/server/implementation src/server/implementation.backup
> mkdir build
> cd build
> rm -rf *
> cmake .. -DGENERATE_JAVA_CLIENT_PROJECT=TRUE
> cd ..
```

Your code should be implemented into ArMarkerdetectorOpenCVImpl.* (Check out the FIXME parts).

As you might need to regenrate these later on it makes sense to make most of the actual implementation in separate files (e.g. Process.*). These separate files and lib debendencies need to be added in src/server/CMakeLists.txt

```
> cd src/server/implementation/objects
> gvim ArMarkerdetectorOpenCVImpl.cpp Process.cpp Process.h
> gvim ../../CMakeLists.txt
```

```
# Generate code
include (CodeGenerator)

# Possible parameters
# set (MULTI_VALUE_PARAMS
#   MODELS
#   INTERFACE_LIB_EXTRA_SOURCES
#   INTERFACE_LIB_EXTRA_HEADERS
#   INTERFACE_LIB_EXTRA_INCLUDE_DIRS
#   INTERFACE_LIB_EXTRA_LIBRARIES
#   SERVER_IMPL_LIB_EXTRA_SOURCES
#   SERVER_IMPL_LIB_EXTRA_HEADERS
#   SERVER_IMPL_LIB_EXTRA_INCLUDE_DIRS
#   SERVER_IMPL_LIB_EXTRA_LIBRARIES
#   MODULE_EXTRA_INCLUDE_DIRS
#   MODULE_EXTRA_LIBRARIES
#   SERVER_IMPL_LIB_FIND_CMAKE_EXTRA_LIBRARIES
# )

generate_code (
    MODELS ${CMAKE_CURRENT_SOURCE_DIR}/interface
    INTERFACE_LIB_EXTRA_INCLUDE_DIRS ${ALVAR_INC}
    SERVER_IMPL_LIB_EXTRA_SOURCES implementation/objects/Process.cpp
    SERVER_IMPL_LIB_EXTRA_HEADERS implementation/objects/Process.h
    SERVER_IMPL_LIB_EXTRA_INCLUDE_DIRS ${ALVAR_INC} ${SOUP_INCLUDE_DIRS}
    SERVER_IMPL_LIB_EXTRA_LIBRARIES ${ALVAR_LIB} ${SOUP_LIBRARIES}
    SERVER_STUB_DESTINATION ${CMAKE_CURRENT_SOURCE_DIR}/implementation/objects
)
```

Annex3 Troubleshooting

You can check was your module loaded correctly using -v flag (for some reason this does not work always?). Other option is to check out the log: /var/log/kurento-media-server/media-server.log

```
> kurento-media-server -v
  Version: 5.0.5~2.gc9ad968
  Found modules:
       Module: 'armarkerdetector' version '0.0.1~.g8e73efd'
       Module: 'core' version '5.0.5~1.g00c5165'
       Module: 'elements' version '5.0.5~1.gff86cba'
       Module: 'example' version '0.0.1~0.gbcfaae0'
       Module: 'filters' version '5.0.5~1.g15b6740'
       Module: 'sampleplugin' version '0.0.1~2.ga524493'
```

If you are missing something or you are having some version issues you can try some of the following things:

Make sure you have the kurento development repository:

```
> sudo apt-add-repository http://ubuntu.kurento.org
```

```
> wget -O - http://ubuntu.kurento.org/kurento.gpg.key | sudo apt-key add -
```

Check out versions of installed pacakges

```
> dpkg -l kms-core
> dpkg -l kms-core-dev
> dpkg -l kms-elements
> dpkg -l kms-elements-dev
> dpkg -l kms-filters
> dpkg -l kms-filters-dev
> dpkg -l kurento-media-server
```

Install latest versions

```
> sudo apt-get update
> sudo apt-get install kms-core-dev
> sudo apt-get install kms-elements-dev
> sudo apt-get install kms-filters-dev
> sudo apt-get install kurento-media-server
```

At least for kurento-media-server it sometimes does not install the latest version unless you force it.

```
> sudo apt-get remove kurento-media-server
> sudo apt-get install kurento-media-server
```

One potential problem is that if you have binary-incompatible modules somewhere. For example if you have compiled some packages yourself and they are installed e.g. on somewhere in /usr/local/? However, if you have the kurento-media-server path settings as described in chapter 2.6.3 it should not be a problem.