Multimedia Computing Project [V0.91]

(Description revised: 28/03, 20:00)

Important dates

- Specification and state of the art study (related work) - max 4 pages: **7/04/2018.** Send to nmc@fct.unl.pt. With this deliverable the team members should be identified and should not change after this date.
- Source code and final report:**2/6/2018**, by email or using a service such as Google Drive.
  Send the code to nmc@fct.unl.pt in zip format. The file should have the student numbers separated with "-". Example: 11111-2222.zip. It should include all the files required to compile and run the program. The **final report** should also be included. If large video files are needed please use a service such as Dropbox or Google Drive. Visualizations will take place after. Discussions will be scheduled if necessary.

The project and report requirements are presented next. There are some open options and **additional or alternative features are encouraged (e.g, audio support both for playing and event detection)**.

(Semi) Automatic Infinite Entertainment System

In this project the goal is to continuously provide video content based on context information such as if anyone is watching the screen or how many people are paying attention. Also, the video content will be selected based on its characteristics such as topic, color, texture, rhythm or keywords. The project will have two main parts: one for sensing the environment, count and detect people and another to extract characteristics of the video streams and use them to select what is playing. The system should also include transitions between videos (e.g., fade and dissolve).

There will be two mains modes, (1) display the resulting video in a public screen and (2) setup and configure the system. The first mode only displays the selected video whereas the second mode allows to see the video library and what the camera is processing.

Video Library

This view shows the collection of videos available to play. Each video should be represented by a frame or moving icon. Moving icons should be implemented based on cut detection. It should also be possible to see the metadata extracted for each of the videos.

Player

The player plays the sequence of video clips. Play/stop/pause/resume control (using a keyboard) should be included. There will be an option for playing each video (by the operator to setup) and a player for the final user watching the public screen.

Context Sensing

Sensing the environment will be done through a video camera. It should be possible to detect, at least, whether there is motion, if there are people watching and how many. Recognizing an object or color pattern should trigger related video. Audio could also be used to detect activity [*extra*].

Metadata

The metadata for each video clip is the following and it is used to decide which video will be shown next:

- Tags
- Luminance
- Color - based on first moment
- Edge distribution
- Number of times a specific object (input as an image) appears in the video frame
- Number of faces appearing in the video
- Texture characteristics (only for first frame)
- Rhythm
- Audio amplitude [*extra*]

This metadata can be stored in an XML file, so that it is only necessary to process each video once.

Development and Library Support

In order to support access to media content, openFrameworks is the suggested framework. The **videoPlayer** and **dirListExample** are good starting points for the project. The **opencvHaarFinderExample** and the **cameraLensOffsetExample** show how to integrate **OpenCV with openFrameworks** and also use face detection. To integrate XML files (for example to store/retrieve metadata) the **ofxXmlSettings** and the sample application could be used. In order to build the User Interface an addon such as **ofxGui** can be used. This addon is included in the distribution but there are others in the openFrameworks site (addons).

Using OpenCV with OF Video Capture and OpenCV addon

```
// set ofxCvColorImage colorImg
colorImg.setFromPixels(myGrabber.getPixels().getData(), camWidth, camHeight );

// create Mat
Mat colorm(colorImg.getCvImage()) ;

// create grayscale Mat
cvtColor(colorm, graym, COLOR_BGR2GRAY);

// load image (ofImage) and create Mat in grayscale as an example. markImg is a ofxCvColorImage
loader.loadImage("test.jpg");
markImg.allocate(loader.getWidth(),loader.getHeight());
markImg.setFromPixels(loader.getPixels().getData(), loader.getWidth(), loader.getHeight());
cvtColor(Mat(markImg.getCvImage()), markm, COLOR_BGR2GRAY);


// or convert an OfImage to OpenCV format
ofImage ofimg;
ofimg.load("someimage.png");
cv::Mat img = ofxCv::toCv(ofimg);
```

Edge distribution

Edge distribution can be obtained by using one or more edge filters (see slides) and counting or averaging the results. An edge histogram is a good way to represent the edge histogram.

Number of times a specific object (input as an image) appears in the video frame

To compare images, keypoint based methods usually give good results in image matching and OpenCV includes some of these methods (features2D). There are several examples using this framework in OpenCV. The examples matchersimple.cpp, descriptor_extractor_matcher.cpp and generic_descriptor_match.cpp include the different processing stages: 1) keypoint detection; 2) descriptor extraction; 3)descriptor matching. The matchersimple example and description is a good starting point to understand the process. There are different ways to detect the keypoints and extract the descriptors, including SIFT, SURF and FAST, that lead to different results and processing needs.

Addtionally, OpenCV provides more than one way to use these descriptors. For example, the SURF keypoint detector can be created using `SurfFeatureDetector detector(400);` or using a generic interface `cv::Ptr<FeatureDetector> detector = cv::FeatureDetector::create("SURF");`. As additional examples, the following sequence includes the three main steps:

```
cv::FeatureDetector::create("FAST"); DescriptorExtractor::create("BRIEF"); DescriptorMatcher::create("BruteForce-Hamming");
```

The same could be done with SIFT descriptors:

```
//handle non free (at least in Windows/Visual Studio)
#include "opencv2/nonfree/nonfree.hpp"

cv::initModule_nonfree();
cv::FeatureDetector::create("SIFT"); DescriptorExtractor::create("SIFT"); DescriptorMatcher::create("BruteForce");
```

Texture characteristics

Texture can be evaluated using Gabor filters. Currenty, Gabor filters are supported by OpenCV. The kernels (filters) can be generated with **getGaborKernel** and the filters are applied with **filter2D**.

Further information and sample code (if needed) is available here. There are options for the parameters depending on the desired result. Several orientations could be used and frequencies/wavelengths resulting in a bank with multiple filters (for example 6 orientations and 4 frequencies).

Rhythm

Rhythm can be approximated using measures based on image difference. As a suggestion, histogram differences in grayscale images can be used (see slides).

Specification and State of the Art Study

This is the initial specification of the system and a study of related work. Enhancing multimedia information with metadata to improve browse and search operations was explored by other authors in different situations and contexts. Some of these systems use advanced image and video processing algorithms. The main goal of this study is to provide an overview of the state of the art in this area. The results should be summarized and discussed in a short paper (max 4 pages), that should include:

- Introduction: this section should briefly describe the context and the project.
- Related work: should include an overview of related systems. Most of the information is available in scientific papers, such as the ones available from the ACM Digital Library (access is free at FCT/UNL). Searching this library with and "video interfaces", "video features", "video similarity" as keywords results in multiple relevant results such as:
  - VFerret: content-based similarity search tool for continuous archived video
  - Advanced user interfaces for dynamic video browsing
  - Video exploration: from multimedia content analysis to interactive visualization
- The specification should include an interface sketch/storyboard as first approach to the interface, considering the requirements and features described above. User interface elements should be identified and described. The sketch can be done using several tools (e.g., pen based interfaces or even digitized drawings) and should be included in the document. This type of low fidelity prototypes provides an initial specification of the interface elements and the sequence of actions.
- References: references to websites and papers that are relevant to the project. Last access dates should be included for the websites.

There is a Word template and a LaTeX template available to write this first the state or the art study and initial specification.

Final Report

The following structure is suggested for the final report (up to 6 pages), including part of the content from the initial specification and related work:

- Introduction
- Related work [based on the state of the art study done previously]
- Algorithms and techniques (description of the processing algorithms and visualization techniques)
- Class description
- Application implementation
- Conclusions
- References

The report should also include as appendix:

- Full class diagram, if not included in the report.
- User manual (1-2 pages).