

# **CAMERA ASSISTANT FOR ANDROID**

**LORENZO CAVAZZI  
754986**

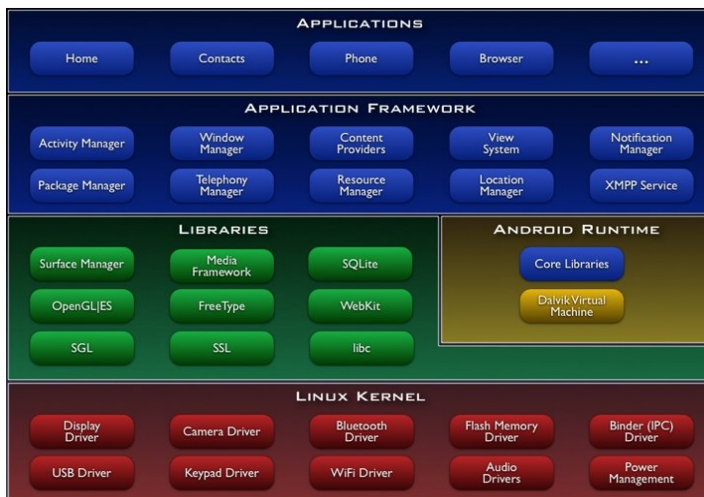
**POLITECNICO DI MILANO  
IMAGE ANALYSIS COURSE - AA 2011  
VINCENZO CAGLIOTI - ALESSANDRO GIUSTI**



Project coordinator: Alessandro Giusti  
Target platform: Android (2.3+)  
External libraries: openCV 2.3.1

# SMARTPHONE CAMERA ENHANCEMENT

Nowadays smartphones are becoming more and more powerful, and their operative systems are quite similar to laptop environments. Thanks to these innovations a great amount of applications are available to be installed, because now developing for mobile devices is more similar to developing for normal pc (similar programming languages, developing environment, ...).



*Android is based on linux kernel, and has a lot of affinities with laptop linux distributions*

Smartphones have some advantages: they are portable, can be used with sensors to interact with external world (gps, camera, accelerometers, ...), and everybody always carry them around. In fact it's a very powerful item, but using all this technology could be difficult for someone; why don't use computational power combined with already available tools to enhance user experience? Of course we

consider computer vision aspects of this idea, and in particular we focus on making the use of native camera simpler.

We should consider that costs of video cameras have dropped down in last years, and now almost every portable device has got an onboard camera, while the quality of cheap hardware and its software are sometimes arguable. Dedicated devices use a lot of software tricks in order help inexperienced photographers to avoid blurring or others undesired effects and also modern smartphones implement some of these functions but they usually lack of instruments to assist people to take good photos.

We have decided to try using features detection in Android environment to show on-screen tips in order to take good photos. This "augmented reality" approach (meaning enrich real time camera images in the way that it gives extra informations that could be useful to users) has recently become popular thanks to automotive industry that has decided to use features detection in order to enable the recognition of road signs or dangerous obstacles and alert driver.

We have focused mainly on the performance on smartphone in terms of frame per second to evaluate possibility of real time use of actual available image manipulation tools for Android, underlining positive aspects but also problems and limitations.

# OPENCV FOR ANDROID AND HAAR ALGORITHM

There is a moderate amount of tools for computer vision and diffusion of mobile devices has made them available to operative systems such as Android and iOS. We decided to privilege a library that is complete, well documented, popular and, last but not least, open source.

OpenCV is the winner: it's open, surely popular, and last year was ported to Android, giving the possibility to use all functions thanks to Java Native Interface; it allows C++ native code to run under Java, even if it negatively influences performance. Anyway a group of volunteers has started to translate all function in native Java code, in order to improve their performance.

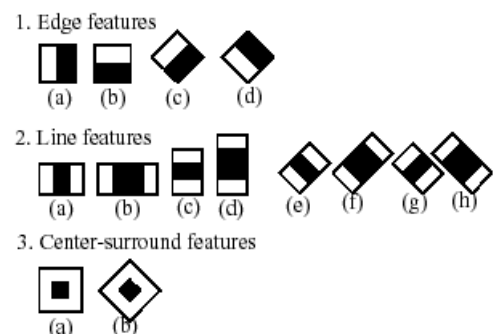
OpenCV libraries integrate an haar-like feature detection algorithm and a list of ready-to-use xml cascade files that contains information about bodies or parts of them (like eyes, mouth, nose, ...).

Cascade files are created by training, using hundreds of images to identify specific items (faces, eyes, but also cars or dogs). Images are translated to gray-level, then divided into rectangles and compared to set of haar features to identify lines and edges. Finally they are re-scaled to a common size, and the useful information is stored in xml cascade file.

Usually it is used also a set of “negative examples”, which helps to distinguishing between false positive confirmations by creating a list of details that should not appear in order to recognise the desired feature.

When detection function is used on image, it is analyzed several times to detect different object sizes, according to the rescale factor and the minimum feature size parameters. Performance go down if images are very big, or if the minimum feature size is small and the rescale factor high.

We focus in particular on face and eyes recognition, because we are interested in common scenarios with smartphone, for example the fact of taking a good photo of a person or a group of people that should be close-up and centred on the image. The work could easily be extend to other features by using other cascade classifiers.



*example of haar-features*

# IMPLEMENTATION

Now we focus on the realization of test application, for which we have chosen a development environment made by Eclipse + ADT plugin, with Android SDK installed on system (API level 8, 10, 12) and openCV libraries imported into project (source code and binaries are attached to this report, see readme file for deeper informations).

The project has been developed to work under Android 2.3+, and it's written in Java (C++ native are not called directly). It should work with every Android based phone with an onboard camera, one should just allow "unknown content application" to run.

Starting a New Project requires to create an object that extends Activity class, the access point of any application. Activity allows to instantiate other objects and manipulate the basic application variables such as menu creation and application dimensions.

```
public void onCreate(Bundle savedInstanceState) {  
    [...]  
  
    // remove application title and system bar  
    requestWindowFeature(Window.FEATURE_NO_TITLE);  
    this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,  
    WindowManager.LayoutParams.FLAG_FULLSCREEN);  
  
    // acquire display height and width  
    DisplayMetrics displaymetrics = new DisplayMetrics();  
    getWindowManager().getDefaultDisplay().getMetrics(displaymetrics);  
    dispHeight = displaymetrics.heightPixels;  
    dispWidth = displaymetrics.widthPixels;  
    [...]  
}
```

For our purpose it is sufficient to create a simple menu which allows to choose among available settings in order to try different features recognitions, parameters and some other options, then we need to display frame (eventually manipulated) taken from camera. To do this I have created a new class "camera\_view.java" that is started immediately, extends SurfaceView, and implements SurfaceHolder and Runnable, so that it can synchronize to camera and display frames when data from the source are ready.

```
public void onCreate(Bundle savedInstanceState) {  
    [...]  
  
    // set view  
    setContentView(new camera_view(this));  
}
```

```

public class camera_view extends SurfaceView implements SurfaceHolder.Callback,
Runnable {

    [...]

    public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {
        [...]

        synchronized (this) {
            // only if camera is correctly opened, set width and height
            if (camera != null && camera.isOpened()) {
                [...]
            }
        }
    }

    [...]
}

```

The object of the class `camera_view` first tries to recover cascade xml files (they must be included in project sources) and to read them in order to assign their content to `CascadeClassifiers` objects, so they are ready to be used to analyze frames. Then native camera raw contents are bound to openCV object `VideoCapture`, and if no error happens thread can starts.

```

// initialize surface with camera binding and start thread
public void surfaceCreated(SurfaceHolder holder) {
    Log.i("surfaceView", "SurfaceCreated in esecuzione...");
    // initialize camera using openCV function
    camera = new VideoCapture(Highgui.CV_CAP_ANDROID);
    // if camera succesfully open, strat thread
    if (camera.isOpened()) {
        Log.i("surfaceView_surfaceCreated", "Camera aperta, run start");
        (new Thread(this)).start();
        [...]
    }
}

```

The run function synchronizes data from raw camera and, when a frame is ready, image is recovered by `takeFrame` function, converted from `Bitmap` to `Canvas`, extra informations that should appears onscreen are drawn over the canvas, and finally it's showed in `surfaceHolder`.

```

public void run() {
    [...]

    synchronized (this) {
        // check if camera is set to null and eventually break, to avoid crash
when        [...]
        // take new frame from camera (see takeFrame function for more details)
        cameraBitmap = takeFrame(camera);
    }

    // create canvas to draw image from camera and display it
    Canvas canvas = sholder.lockCanvas();
}

```

```

        canvas.drawBitmap(cameraBitmap, 0, 0, null);

        [...]
        // FPS CODE: draw fps infos
        canvas.drawText("fps: " + (int)afps, 20, sheight - 80, paint);

        // display infos about selected parameters
        canvas.drawText(infoString(), 20, sheight - 40, paint);

        // eventually display extra infos
        if (Camera_assistant_754986Activity.helperParam) {
            canvas.drawText(extraInfos, positionInfos, 40, paintInfos);
        }

        // display canvas
        sholder.unlockCanvasAndPost(canvas);

        [...]
    }

```

The takeFrame function is the core of the program: it retrieves data from camera raw and convert them to RGB image and Grayscale image stored in 2 matrix objects Mat from openCV. The RGB image will be modified, converted to Bitmap and returned, while the Grayscale image will be used to call detectMultiScale function on cascade classifier loaded before. This function is by far the most computationally intense and in fact it's the bottleneck of application. Later there will be shown examples of performance impact of calling more than one times detectMultiScale, or of changing its input parameters.

This function returns a list of openCV Rect objects that contains useful information about detected features, that are used to draw what has been stored on RGB image and to extract data to understand features relative positions and give onscreen advices.

```

// take a frame from native camera using openCV functions and return a bitmap
private Bitmap takeFrame(VideoCapture camera) {
    // retrieve color frame, used to display preview
    camera.retrieve(colorMat, Highgui.CV_CAP_ANDROID_COLOR_FRAME_RGBA);
    // retrieve gray frame, used for features recognition
    camera.retrieve(grayMat, Highgui.CV_CAP_ANDROID_GREY_FRAME);

    // initialize bitmap
    Bitmap tmpBitmap = Bitmap.createBitmap(colorMat.cols(), colorMat.rows(),
    Bitmap.Config.ARGB_8888);

    // detect faces
    List<Rect> foundFaces = new LinkedList<Rect>();
    if (Camera_assistant_754986Activity.faceParam != 0) {
        // find faces
        faces.detectMultiScale(grayMat, foundFaces, 2, 2, 2, new
    Size(sheight*0.2*Camera_assistant_754986Activity.faceParam,
    sheight*0.2*Camera_assistant_754986Activity.faceParam));

        [...]
    }

```

```

        // draw detected faces
        for (Rect tmpRect:foundFaces) {
            Core.rectangle(colorMat, tmpRect.tl(), tmpRect.br(), new Scalar(0,
255, 255, 0), 2);
        }

        [...]

        // call function to understand position relative to preview sizes
        position = findFacePosition(mediumSxFaces, mediumDxFaces, numFaces);

        // set infos to display
        setDisplayInfos(position);

        [...]

    }

    // convert Mat to bitmap and return
    if (Utils.matToBitmap(colorMat, tmpBitmap)) {
        return tmpBitmap;
    }

    [...]
}

```

Depending on how much features we have found and their absolute and relative positions, it's possible to give advice or to create “augment” images. Here is the function that calculates if faces are centred on the screen.

```

// find position of the face
// return 0 if central, negative number if too sx, positive number if too dx
private short findFacePosition(int sx, int dx, int num) {
    // find medium distance from left border
    sx = sx/num;
    // find medium distance from right border
    dx = dx/num;
    dx = swidth - dx;
    // find relative difference
    int difference = sx - dx;

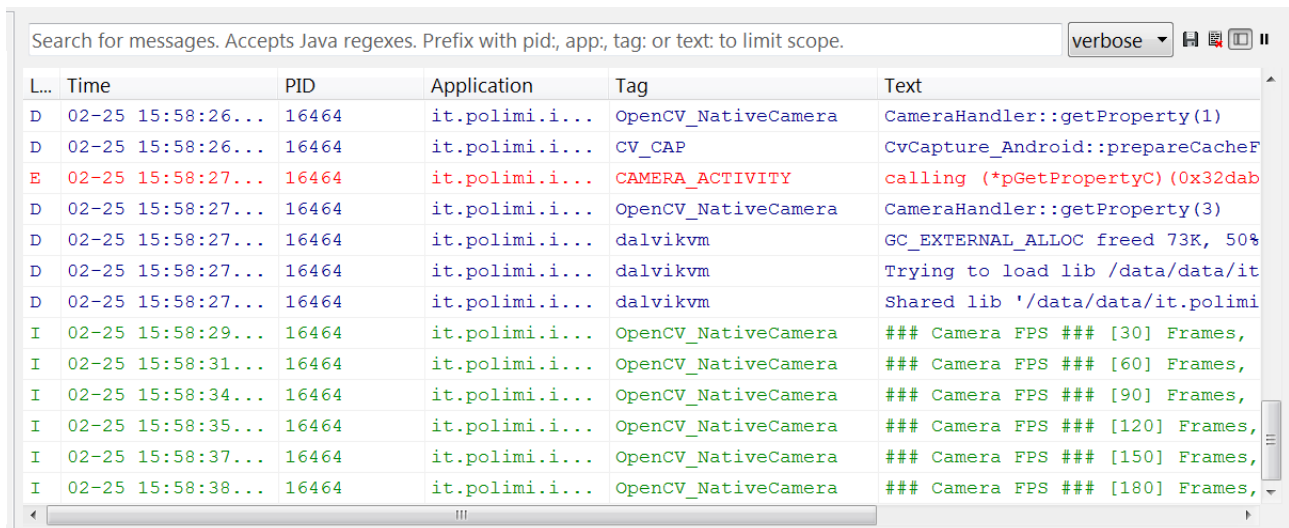
    // decide how much faces are centred
    int unit = swidth / 9;
    if(difference > 2*unit) {
        return -2;
    } else if ((difference <= 2*unit)&&(difference > unit)) {
        return -1;
    } else if ((difference <= unit)&&(difference > -unit)) {
        return 0;
    } else if ((difference <= -unit)&&(difference > -2*unit)) {
        return 1;
    } else {
        return 2;
    }
}

```

Other similar functions could be implemented easily; for further information on possible “assistant” functions see next chapters considerations on usability of

implemented algorithm at real time and 2<sup>nd</sup> video attached at this report, which shows examples of using face + eyes identification to understand if the pose is backlight.

In order to debug, a lot of Log functions have been added to code, and LogCat from DDMS has been used to capture this information while running code. They can be useful also to eventually produce other code in the future in order to extend the project.



L...	Time	PID	Application	Tag	Text
D	02-25 15:58:26...	16464	it.polimi.i...	OpenCV_NativeCamera	CameraHandler::getProperty(1)
D	02-25 15:58:26...	16464	it.polimi.i...	CV_CAP	CvCapture_Android::prepareCacheF
E	02-25 15:58:27...	16464	it.polimi.i...	CAMERA_ACTIVITY	calling (*pGetPropertyC) (0x32dab
D	02-25 15:58:27...	16464	it.polimi.i...	OpenCV_NativeCamera	CameraHandler::getProperty(3)
D	02-25 15:58:27...	16464	it.polimi.i...	dalvikvm	GC_EXTERNAL_ALLOC freed 73K, 50%
D	02-25 15:58:27...	16464	it.polimi.i...	dalvikvm	Trying to load lib /data/data/it
D	02-25 15:58:27...	16464	it.polimi.i...	dalvikvm	Shared lib '/data/data/it.polimi
I	02-25 15:58:29...	16464	it.polimi.i...	OpenCV_NativeCamera	### Camera FPS ### [30] Frames,
I	02-25 15:58:31...	16464	it.polimi.i...	OpenCV_NativeCamera	### Camera FPS ### [60] Frames,
I	02-25 15:58:34...	16464	it.polimi.i...	OpenCV_NativeCamera	### Camera FPS ### [90] Frames,
I	02-25 15:58:35...	16464	it.polimi.i...	OpenCV_NativeCamera	### Camera FPS ### [120] Frames,
I	02-25 15:58:37...	16464	it.polimi.i...	OpenCV_NativeCamera	### Camera FPS ### [150] Frames,
I	02-25 15:58:38...	16464	it.polimi.i...	OpenCV_NativeCamera	### Camera FPS ### [180] Frames,

*example of debug output in LogCat*



## CASE STUDIES AND RESULTS

We have focused on preparing some case studies to evaluate real time performance and usability, and application has been tested under different conditions to evaluate successes and failures. The first part is proposed also in first of two attached video, that should be also a brief tutorial on how to use the application, while the second one explores particular conditions in which detection can fail.

As said before the bottleneck is by far feature detection function, especially number of invocations and minimum detection size. When the application starts only faces are detected, and they must be close-up. In such a condition detection works well, also with non optimal illumination, and even if fps it's not very high, it's enough for a real time preview (6-7 fps).



*face detection: note fps and onscreen "OK - SCATTA"*



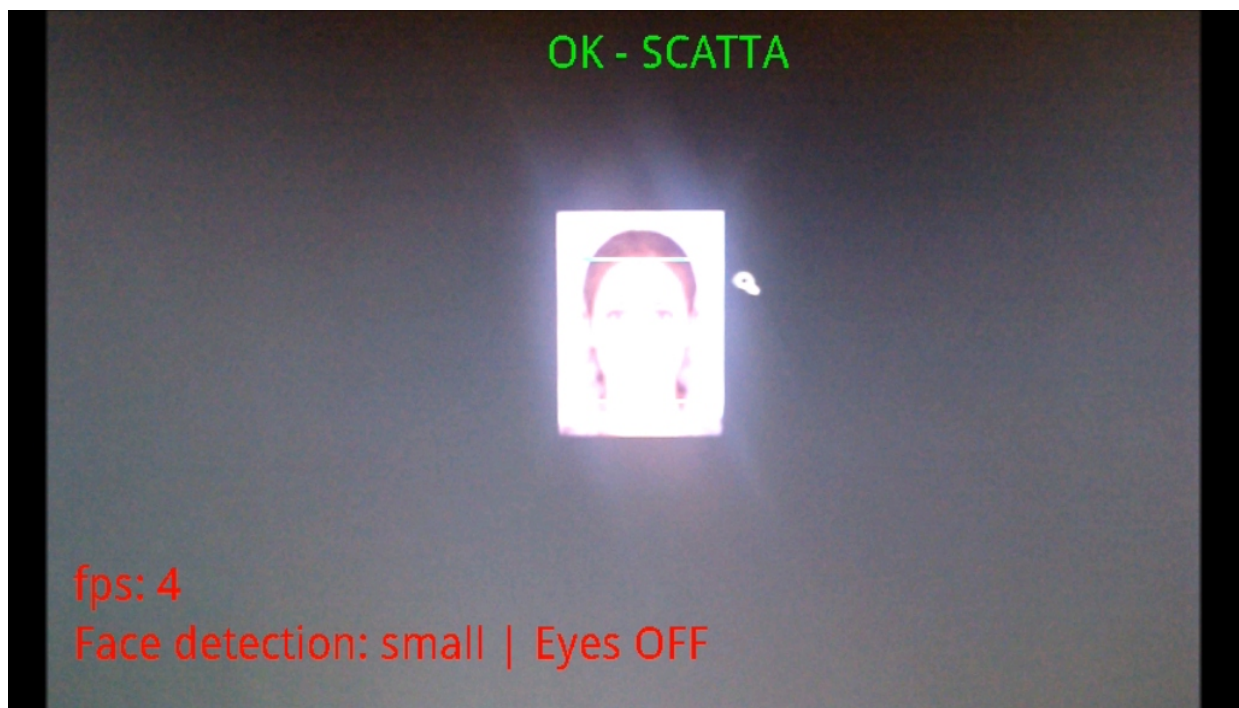
*now image is not centred, and helper says to move -->*

If we turn on eyes detection option, the detectMultiScale function is called another time in order to try to find also eyes, but fps go down to 3, and images become “jerky” (screenshots from Android aren't well synchronized with SurfaceHolder, so images seems to be overlying different frame but it's only a screencapture effect)



*notice eyes revealed and low fps*

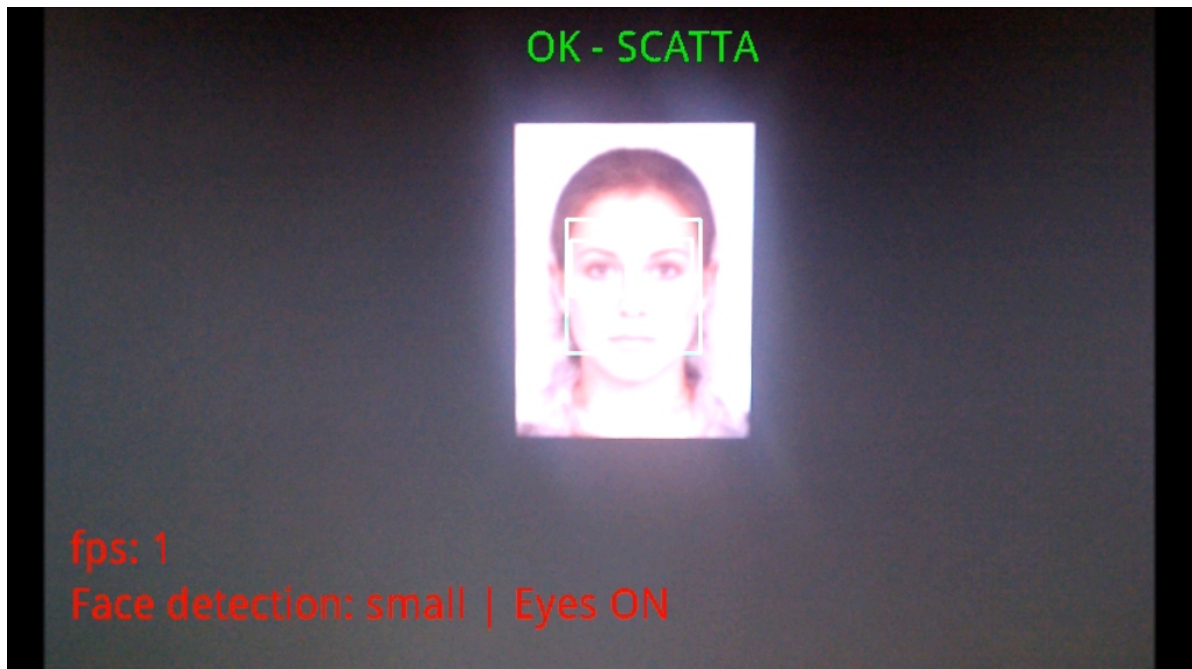
We obtain bad results also when we try to find far features. If we select “small detection” option any faces should be detected, unless they are really small, but fps doesn't exceed 3-4.



*small features detection enabled, detecting only faces*

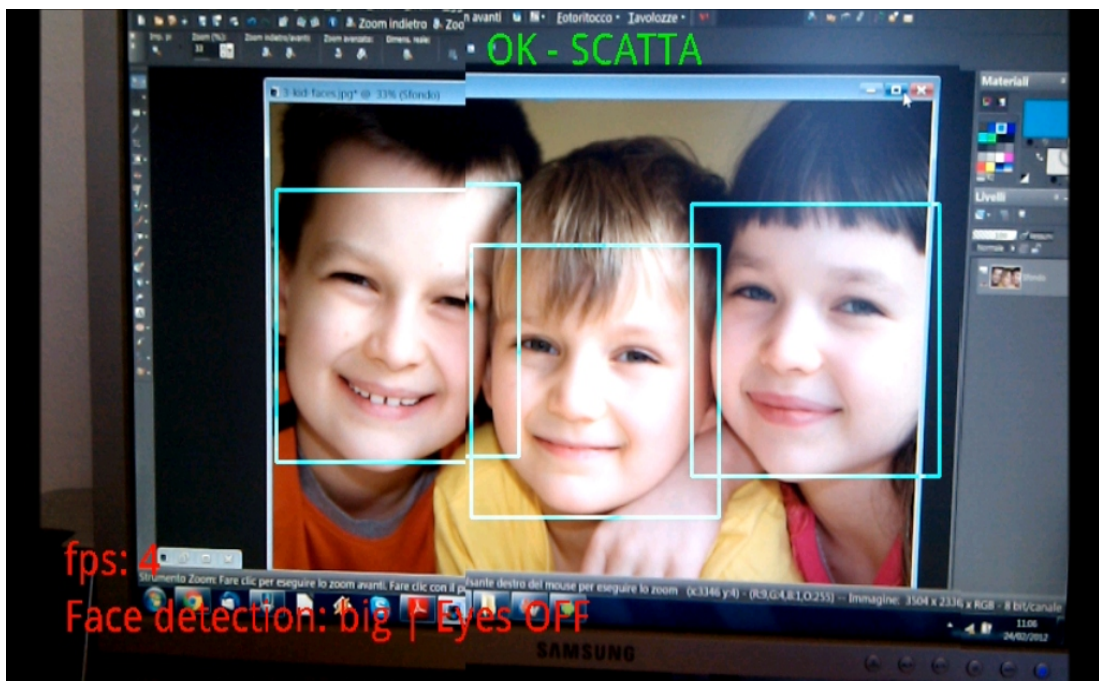


Finally we try to turn on both small features and eyes detection, and fps are now between 1 and 2. It's completely unusable for real time computation.



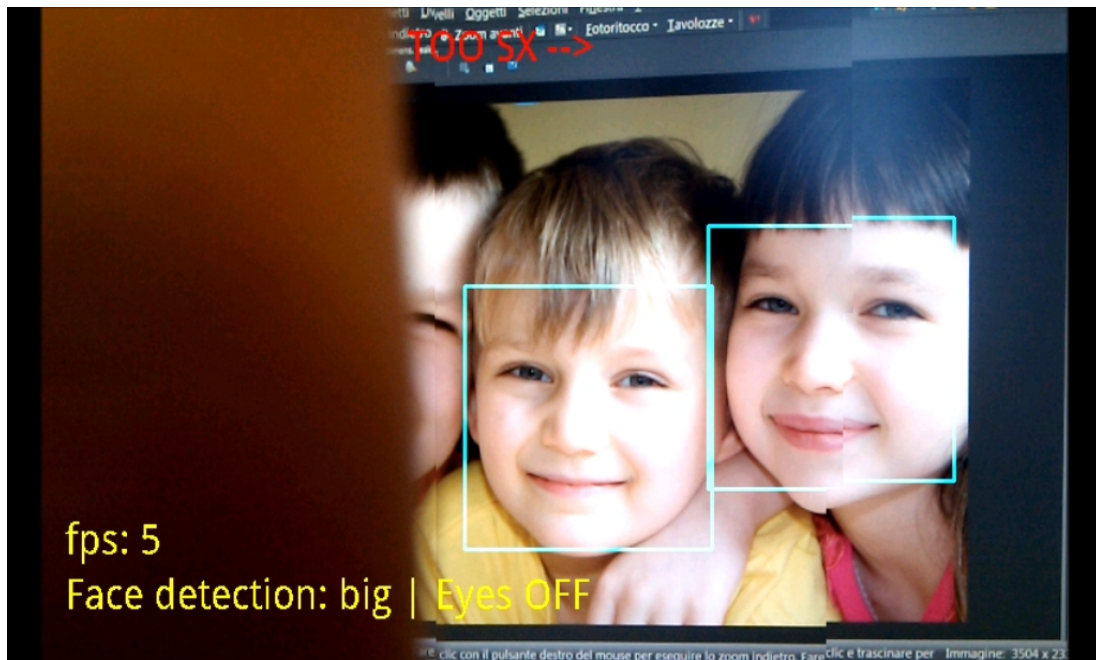
*small features and eyes detection both enabled, but fps are unacceptable*

The application works also with multiple faces in the scene. In this case a “weighed center” is calculated like a center of gravity of rectangles around faces, and onscreen helper tries to put this point in the center of the image. Notice than also the number of faces influences performance (in the same conditions we obtained 6 fps with only one face).



*multiple faces, only big features detection enabled*

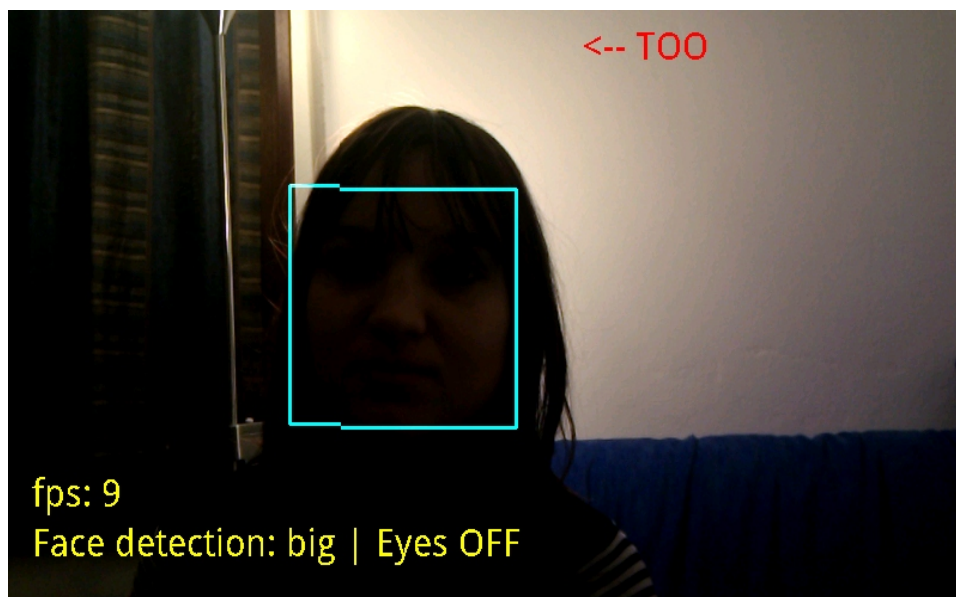
If the number of faces suddenly changes, center of gravity is re-calculated and a new indication appears onscreen.



*with 2 faces indication become immediately -->*

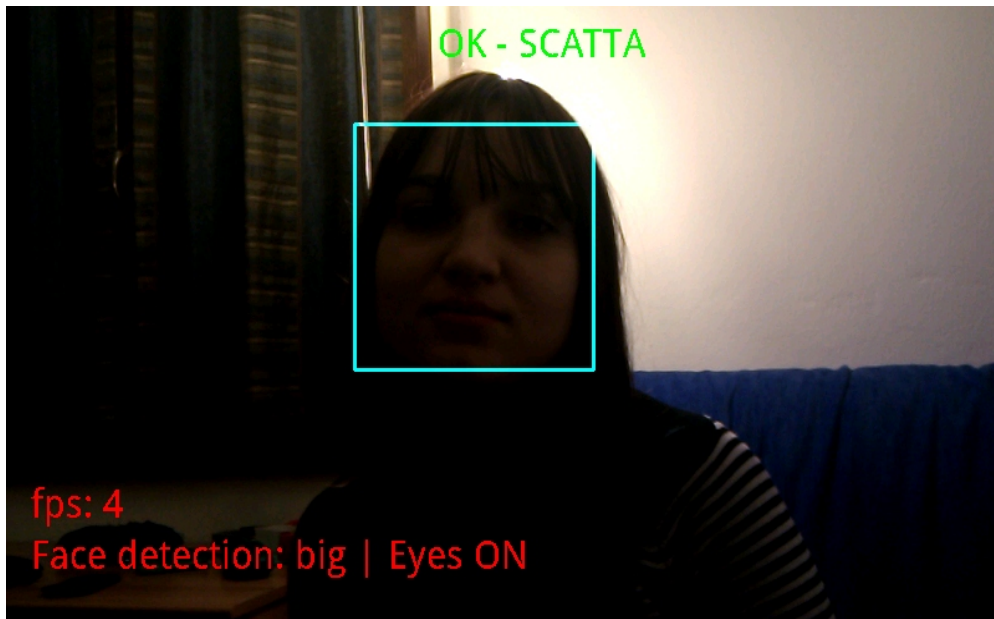
Application has been tested also in real conditions when illumination could be non optimal, for example with a soft or medium backlight, or with poor light (of course extreme conditions are excluded because they are meaningless).

Face detection is quite good in these situations, but eyes detection become inaccurate and not usable. In fact this is not a bad thing: we can use this information to alert user if illumination is good (everything detected) or if it is actually bad (face detected + eyes undetected means backlight, face detected + eyes unstable means poor illumination).

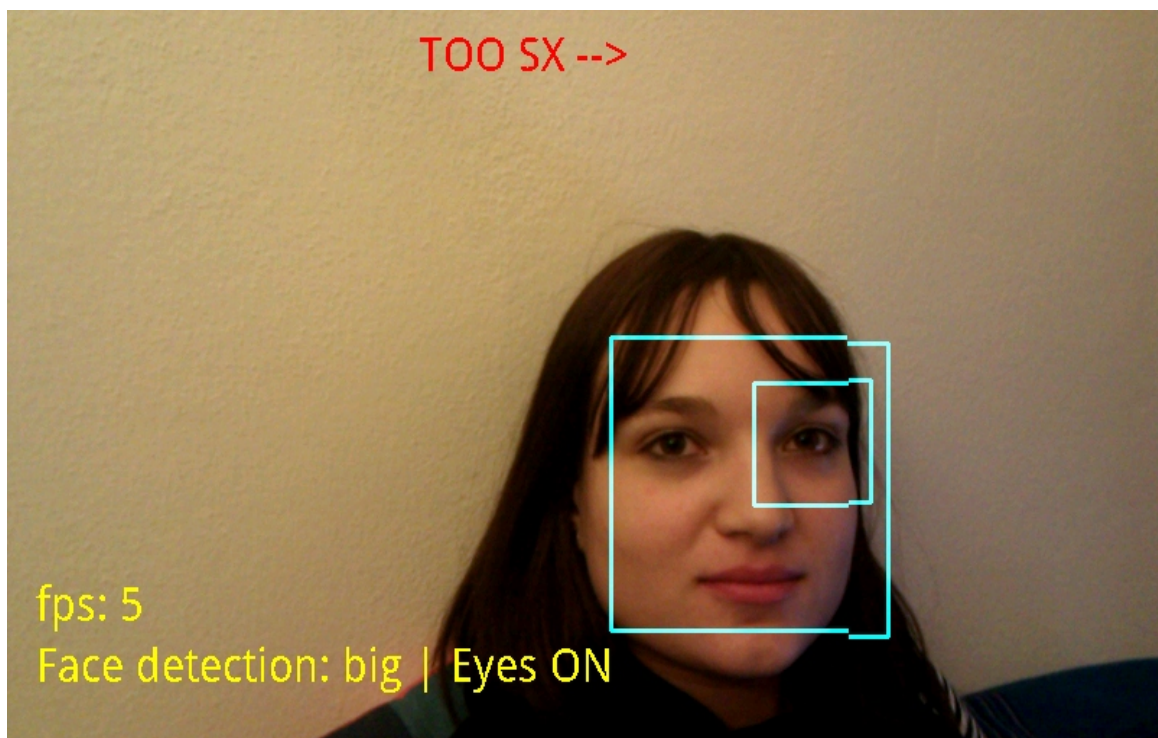


*backlight, only face detection*





*backlight, eyes detection on but only face is detected*



*poor light, now it's better but sometimes only one eye is detected*

## CONCLUSIONS AND FUTURE DEVELOPMENT

Features recognition has been proved to be a very powerful item to improve camera usability, and openCV implementation is quite accurate even if problems occurred with poor illumination.

However the real problem for real time use is the computation intensity of the feature detection function, that is clearly too heavy for smartphone used in testing operations (HTC Nexus One). According to official openCV for Android discussion group, better performance can be obtained by using a C++ cross compiler while compiling in eclipse environment and directly invoking C++ native function from original openCV libraries. This would be an interesting test to be carried out, but java methods have been preferred because their easiness of use, supposing that convenience of using external libraries is lost in the latter case.

We have to consider that openCV libraries are being translated to java without the use of Java Native Interface, so we do expect to have strongly better performance with new releases.

We have to consider also that the testing device is a 2 years old smartphone, and that new models could probably reach better fps rating, even if this should not resolve problem of heavy load conditions (lots of faces, high distance from camera, multiple features detection).

An interesting development could be trying to better understand, by testing and searching in available literature, how to use information from feature detection to alert user on particular environment condition (for example, as described before, face and eyes detection could be used together in order to understand if backlight is in the scene).