```cpp
/*
 * CvProcessor.h
 *
 *  Created on: 21 fÃ©vr. 2012
 *      Author: davidroussel
 */

#ifndef CVPROCESSOR_H_
#define CVPROCESSOR_H_

#include <string>
#include <map>
#include <iostream>
#include <ctime>    // for clock
using namespace std;

#include <opencv2/core/mat.hpp>
using namespace cv;

#include "CvProcessorException.h"
#include "MeanValue.h"

/**
 * Class to process a source image with OpenCV 2+
 */
class CvProcessor
{
    public:

        /**
         * Verbose level for error / warnings / notification messages
         */
        typedef enum
        {
            VERBOSE_NONE = 0,   //!< no messages are displayed
            VERBOSE_ERRORS,   //!< only error messages are displayed
            VERBOSE_WARNINGS,   //!< error & warning messages are displayed
            VERBOSE_NOTIFICATIONS, //!< error, warning and notifications messages are displayed
            VERBOSE_ACTIVITY, //!< all previouses + log messages
            NBVERBOSELEVEL
        } VerboseLevel;


        /**
         * Index of channels in OpenCV BGR or Gray images
         */
        typedef enum
        {
            BLUE = 0,//!< Blue component is first in BGR images
            GRAY = 0,//!< Gray component is first in gray images
            GREEN,   //!< Green component is second in BGR images
            RED,     //!< Red component is last in BGR images
            NBCHANNELS
        } Channels;

        /**
         * Mean/Std, min & max processing time type
         */
        typedef MeanValue<clock_t, double> ProcessTime;

    protected:
        /**
         * The source image: CV_8UC<nbChannels>
         */
        Mat * sourceImage;

        /**
         * Source image number of channels (generally 1 or 3)
         */
        int nbChannels;

        /**
         * Source image size (cols, rows)
         */
        Size size;

        /**
         * The source image type (generally CV_8UC<nbChannels>)
         */
        int type;

        /**
         * Map to store aditionnal images pointers by name
         */
        map<string, Mat*> images;

        /**
         * The verbose level for printed messages
         */
        VerboseLevel verboseLevel;
```



```cpp

        /**
         * Process time in ticks (~1e6 ticks/second)
         * @see clock_t for details on ticks
         */
        clock_t processTime;

        /**
         * Mean process time (averaged process times)
         */
        ProcessTime meanProcessTime;

        /**
         * Indicates if processing time is absolute or measured in ticks/feature
         * processed by this processor.
         * A feature can be any kind of things the processor has to detect or
         * create while processing an image.
         */
        bool timePerFeature;

    public:
        /**
         * OpenCV image processor constructor
         * @param sourceImage the source image
         * @param level verbose level for printed messages
         * @pre source image is not NULL
         */
        CvProcessor(Mat * sourceImage,
                    const VerboseLevel level = VERBOSE_NONE);

        /**
         * OpenCV image Processor destructor
         */
        virtual ~CvProcessor();

        /**
         * OpenCV image Processor abstract Update
         * @note this method should be implemented in sub classes
         */
        virtual void update() = 0;

        // ------------------------------------------------------------------
        // Images accessors
        // ------------------------------------------------------------------
        /**
         * Changes source image
         * @param sourceImage the new source image
         * @throw CvProcessorException#NULL_IMAGE when new source image is NULL
         * @note this method should NOT be directly reimplemented in sub classes
         * unless it is transformed into a QT slot
         */
        virtual void setSourceImage(Mat * sourceImage)
            throw (CvProcessorException);

        /**
         * Adds a named image to additionnal images
         * @param name the name of the image
         * @param image the image reference
         * @return true if image has been added to additionnal images map, false
         * if image key (the name) already exists in the addtitionnal images map.
         */
        bool addImage(const char * name, Mat * image);

        /**
         * Adds a named image to additionnal images
         * @param name the name of the image
         * @param image the image reference
         * @return true if image has been added to additionnal images map, false
         * if image key (the name) already exists in the addtitionnal images map.
         */
        bool addImage(const string & name, Mat * image);

//      /*
//       * Update named image in additionnal images.
//       * @param name the name of the image
//       * @param image the image reference
//       * @post the image located at key name is updated.
//       */
//      virtual void updateImage(const char * name, const Mat & image);
//
//      /*
//       * Update named image in additionnal images.
//       * @param name the name of the image
//       * @param image the image reference
//       * @post the image located at key name is updated.
//       */
//      virtual void updateImage(const string & name, const Mat & image);

        /**
         * Get image by name
```

```
         * @param name the name of the image we're looking for
         * @return the image registered by this name in the additionnal images
         * map
         * @throw CvProcessorException#INVALID_NAME is used name is not already
         * registerd in the images
         */
        const Mat & getImage(const char * name) const
            throw (CvProcessorException);

        /**
         * Get image by name
         * @param name the name of the image we're looking for
         * @return the image registered by this name in the additionnal images
         * map
         * @throw CvProcessorException#INVALID_NAME is used name is not already
         * registerd in the images
         */
        const Mat & getImage(const string & name) const
            throw (CvProcessorException);

        /**
         * Get image pointer by name
         * @param name the name of the image we're looking for
         * @return the image pointer registered by this name in the additionnal
         * images map
         * @throw CvProcessorException#INVALID_NAME is used name is not already
         * registerd in the images
         */
        Mat * getImagePtr(const char * name)
            throw (CvProcessorException);

        /**
         * Get image pointer by name
         * @param name the name of the image we're looking for
         * @return the image registered by this name in the additionnal images
         * map
         * @throw CvProcessorException#INVALID_NAME is used name is not already
         * registerd in the images
         */
        Mat * getImagePtr(const string & name)
            throw (CvProcessorException);
        // ------------------------------------------------------------------
        // Options settings and gettings
        // ------------------------------------------------------------------
        /**
         * Number of channels in source image
         * @return the number of channels of source image
         */
        int getNbChannels() const;

        /**
         * Type of the source image
         * @return the openCV type of the source image
         */
        int getType() const;

        /**
         * Get the current verbose level
         * @return the current verbose level
         */
        VerboseLevel getVerboseLevel() const;

        /**
         * Set new verbose level
         * @param level the new verobse level
         */
        virtual void setVerboseLevel(const VerboseLevel level);

        /**
         * Return processor processing time of step index [default implementation
         * returning only processTime, should be reimplemented in subclasses]
         * @param index index of the step which processing time is required,
         * 0 indicates all steps, and values above 0 indicates step #. If
         * required index is bigger than number of steps then all steps value
         * should be returned.
         * @return the processing time of step index.
         * @note should be reimplemented in subclasses in order to define
         * time/feature behaviour
         */
        virtual double getProcessTime(const size_t index = 0) const;

        /**
         * Return processor mean processing time of step index [default
         * implementation returning only processTime, should be reimplemented
         * in subclasses]
         * @param index index of the step which processing time is required,
         * 0 indicates all steps, and values above 0 indicates step #. If
         * required index is bigger than number of steps then all steps value
         * should be returned.
         * @return the mean processing time of step index.
```



```
         * @note should be reimplemented in subclasses in order to define
         * time/feature behaviour
         * @param index
         */
        virtual double getMeanProcessTime(const size_t index = 0) const;

        /**
         * Return processor processing time std of step index [default
         * implementation returning only processTime, should be reimplemented
         * in subclasses]
         * @param index index of the step which processing time is required,
         * 0 indicates all steps, and values above 0 indicates step #. If
         * required index is bigger than number of steps than all steps value
         * should be returned.
         * @return the mean processing time of step index.
         * @note should be reimplemented in subclasses in order to define
         * time/feature behaviour
         * @param index
         */
        virtual double getStdProcessTime(const size_t index = 0) const;

        /**
         * Return processor minimum processing time of step index [default
         * implementation returning only processTime, should be reimplemented
         * in subclasses]
         * @param index index of the step which processing time is required,
         * 0 indicates all steps, and values above 0 indicates step #. If
         * required index is bigger than number of steps than all steps value
         * should be returned.
         * @return the mean processing time of step index.
         * @note should be reimplemented in subclasses in order to define
         * time/feature behaviour
         * @param index
         */
        virtual clock_t getMinProcessTime(const size_t index = 0) const;

        /**
         * Return processor maximum processing time of step index [default
         * implementation returning only processTime, should be reimplemented
         * in subclasses]
         * @param index index of the step which processing time is required,
         * 0 indicates all steps, and values above 0 indicates step #. If
         * required index is bigger than number of steps than all steps value
         * should be returned.
         * @return the mean processing time of step index.
         * @note should be reimplemented in subclasses in order to define
         * time/feature behaviour
         * @param index
         */
        virtual clock_t getMaxProcessTime(const size_t index = 0) const;

        /**
         * Reset mean and std process time in order to re-start computing
         * new mean and std process time values.
         */
        virtual void resetMeanProcessTime();

        /**
         * Indicates if processing time is per feature processed in the current
         * image or absolute
         * @return
         */
        bool isTimePerFeature() const;

        /**
         * Sets Time per feature processing time unit
         * @param value the time per feature value (true or false)
         */
        virtual void setTimePerFeature(const bool value);

        /**
         * Send to stream (for showing processor attributes values)
         * @param out the stream to send to
         * @return a reference to the output stream
         */
        virtual ostream & toStream(ostream & out) const;

        /**
         * Send to any stream template
         * @tparam Stream the stream type
         * @param out the output stream
         * @return a reference to the output stream
         * @note this template method needs to be implemented in the header so
         * it could be available in any source (.cpp) file that need a specific
         * instantiation of this template method, for instance:
         * @code
         * template ostream & CvProcessor::toStream_Impl<ostream>(ostream &) const;
         * @endcode
         */
        template <typename Stream>
```

```
361         Stream & toStream_Impl(Stream & out) const
362         {
363             out << "Verbose Level = ";
364             switch (verboseLevel)
365             {
366                 case VERBOSE_NONE:
367                     out << "None";
368                     break;
369                 case VERBOSE_ERRORS:
370                     out << "Only error messages";
371                     break;
372                 case VERBOSE_WARNINGS:
373                     out << "Error & warning messages";
374                     break;
375                 case VERBOSE_NOTIFICATIONS:
376                     out << "Error + warning + notifications";
377                     break;
378                 case VERBOSE_ACTIVITY:
379                     out << "Error + warning + notifications + log";
380                     break;
381                 case NBVERBOSELEVEL:
382                 default:
383                     out << "Unkonwn";
384                     break;
385             }
386
387             out << '\n' << "Images = " << '\n';
388
389             map<string, Mat*>::const_iterator cit;
390
391             for (cit = images.begin(); cit ≠ images.end(); ++cit)
392             {
393                 Mat * currentImage = cit→second;
394
395                 out << '\t' << cit→first.c_str() << "(" << currentImage→cols << 'x'
396                     << currentImage→rows << 'x' << currentImage→channels() << ")[";
397                 switch (currentImage→depth())
398                 {
399                     case CV_8U:
400                         out << "8-bit unsigned integers]";
401                         break;
402                     case CV_8S:
403                         out << "8-bit signed integers]";
404                         break;
405                     case CV_16U:
406                         out << "16-bit unsigned integers]";
407                         break;
408                     case CV_16S:
409                         out << "16-bit signed integers]";
410                         break;
411                     case CV_32S:
412                         out << "32-bit signed integers]";
413                         break;
414                     case CV_32F:
415                         out << "32-bit floating-point numbers]";
416                         break;
417                     case CV_64F:
418                         out << "64-bit floating-point numbers]";
419                         break;
420                     default:
421                         out << "Unknwon number type]";
422                         break;
423                 }
424
425                 out << '\n';
426             }
427
428             out << "Time per feature = " << (timePerFeature ? "Yes" : "No")
429                 << '\n';
430
431             return out;
432         }
433
434     protected:
435         // ----------------------------------------------------------------------
436         // Setup and cleanup attributes
437         // ----------------------------------------------------------------------
438         /**
439          * Setup internal attributes according to source image
440          * @param sourceImage a new source image
441          * @param fullSetup full setup is needed when source image is changed
442          * @pre sourceImage is not NULL
443          * @note this method should be reimplemented in sub classes
444          */
445         virtual void setup(Mat * sourceImage, const bool fullSetup = true);
446
447         /**
448          * Clean up internal attributes before changing source image or
449          * cleaning up class before destruction
450          * @note this method should be reimplemented in sub classes
```



```
451          */
452         virtual void cleanup();
453 };
454
455 /**
456  * Send to output stream operator
457  * @param out the output stream to send to
458  * @param proc the processor to send to the output stream
459  * @return a reference to the output stream used
460  */
461 ostream & operator <<(ostream & out, const CvProcessor & proc);
462
463 /**
464  * Converts en enum element into its integral type.
465  * Iff the enum is defined as int as its base type
466  * @param e the enum item to be converted into its underlying type
467  */
468 template<typename E>
469 constexpr auto integral(const E e) → typename underlying_type<E>::type
470 {
471    return static_cast<typename underlying_type<E>::type>(e);
472 }
473
474 #endif /* CVPROCESSOR_H_ */
```

```
/*
 * CvProcessor.cpp
 *
 *  Created on: 21 fÃ©vr. 2012
 *      Author: davidroussel
 */


#include "CvProcessor.h"

/*
 * OpenCV image processor constructor
 * @param sourceImage the source image
 * @pre source image is not NULL
 */
CvProcessor::CvProcessor(Mat *sourceImage, const VerboseLevel level) :
    sourceImage(sourceImage),
    nbChannels(sourceImage->channels()),
    size(sourceImage->size()),
    type(sourceImage->type()),
    verboseLevel(level),
    processTime(0),
    meanProcessTime(clock_t(0)),
    timePerFeature(false)
{
    // No dynamic links in constructors, so this setup will always be
    // CvProcessor::setup
    setup(sourceImage, false);
}

/*
 * OpenCV image Processor destructor
 */
CvProcessor::~CvProcessor()
{
    // No Dynamic link in destructors ?
    cleanup();

    map<string, Mat*>::const_iterator cit;
    for (cit = images.begin(); cit ≠ images.end(); ++cit)
    {
        // Release handle to evt deallocate data
        /*
         * Since this is a pointer it should be necessary to release data
         */
        cit->second->release();
    }
    // Calls destructors on all elements
    images.clear();
}

/*
 * Setup internal attributes according to source image
 * @param sourceImage a new source image
 * @param fullSetup full setup is needed when source image is changed
 * @pre sourceimage is not NULL
 * @note this method should be reimplemented in sub classes
 */
void CvProcessor::setup(Mat *sourceImage, const bool fullSetup)
{
    if (verboseLevel ≥ VERBOSE_ACTIVITY)
    {
        clog << "CvProcessor::"<< (fullSetup ? "full " : "") <<"setup" << endl;
    }

    // Full setup starting point (==> previous cleanup)
    if (fullSetup)
    {
        this->sourceImage = sourceImage;
        nbChannels = sourceImage->channels();
        size = sourceImage->size();
        type = sourceImage->type();
    }

    // Partial setup starting point  (==> in any cases)
    processTime = (clock_t) 0;
    resetMeanProcessTime();
    addImage("source", this->sourceImage);
}

/*
 * Clean up internal attributes before changing source image or
 * cleaning up class before destruction
 * @note this method should be reimplemented in sub classes
 */
void CvProcessor::cleanup()
{
    if (verboseLevel ≥ VERBOSE_ACTIVITY)
    {
        clog << "CvProcessor::cleanup()" << endl;
```



```
    }

    // remove source pointer
    map<string, Mat*>::iterator it;
    for (it = images.begin(); it ≠ images.end(); ++it)
    {
        if (it->first ≡ "source")
        {
            images.erase(it);
            break;
        }
    }
}

/*
 * Changes source image
 * @param sourceImage the new source image
 * @throw CvProcessorException#NULL_IMAGE when new source image is NULL
 */
void CvProcessor::setSourceImage(Mat *sourceImage)
    throw (CvProcessorException)
{
    if (verboseLevel ≥ VERBOSE_NOTIFICATIONS)
    {
        clog << "CvProcessor::setSourceImage(" << (unsigned long) sourceImage
             << ")" << endl;
    }

    // clean up current attributes
    cleanup();

    if (sourceImage ≡ NULL)
    {
        clog << "CvProcessor::setSourceImage NULL sourceImage" << endl;
        throw CvProcessorException(CvProcessorException::NULL_IMAGE);
    }

    // setup attributes again
    setup(sourceImage);
}

/*
 * Adds a named image to additionnal images
 * @param name the name of the image
 * @param image the image reference
 * @return true if image has been added to additionnal images map, false
 * if image key (the name) already exists in the addtitionnal images map.
 */
bool CvProcessor::addImage(const char *name, Mat * image)
{
    string sname(name);

    return addImage(sname, image);
}

/*
 * Adds a named image to additionnal images
 * @param name the name of the image
 * @param image the image reference
 * @return true if image has been added to additionnal images map, false
 * if image key (the name) already exists in the addtitionnal images map.
 */
bool CvProcessor::addImage(const string & name, Mat * image)
{
    if (verboseLevel ≥ VERBOSE_ACTIVITY)
    {
        clog << "Adding image " << name << "@[" << (long)(image) << "] in" << endl;
        // Show map content before adding image
        map<string, Mat*>::const_iterator cit;
        for (cit = images.begin(); cit ≠ images.end(); ++cit)
        {
            clog  << "\t" << cit->first << "@["<< (long)(cit->second) << "]" << endl;
        }
    }

    pair<map<string,Mat*>::iterator,bool> ret;
    bool retValue;
    ret = images.insert(pair<string, Mat*>(name, image));

    if (ret.second ≡ false)
    {
        if (verboseLevel ≥ VERBOSE_WARNINGS)
        {
            cerr << "CvProcessor::addImage(\"" << name
                 << "\",...) : already added" << endl;
        }

        retValue = false;
    }
    else
```

```
181     {
182         retValue = true;
183     }
184 
185     return retValue;
186 }
187 /*
188  * Update named image in additionnal images.
189  * @param name the name of the image
190  * @param image the image reference
191  * @post the image located at key name is updated.
192  */
193 //void CvProcessor::updateImage(const char * name, Mat * image)
194 //{
195 //  // Search for this name in the map
196 //  map<string, Mat*>::iterator it;
197 //  for (it = images.begin(); it != images.end(); ++it)
198 //  {
199 //      if (it->first == name)
200 //      {
201 //          (it->second->release();
202 //          images.erase(it);
203 //      }
204 //  }
205 //
206 //  string sname(name);
207 //
208 //  updateImage(sname, image);
209 //}
210 
211 /*
212  * Update named image in additionnal images.
213  * @param name the name of the image
214  * @param image the image reference
215  * @post the image located at key name is updated.
216  */
217 //void CvProcessor::updateImage(const string & name, const Mat & image)
218 //{
219 //  clog << "update image " << name << " with " << (long) &image << endl;
220 //  images.erase(name);
221 //
222 //  addImage(name, image);
223 //}
224 
225 /*
226  * Get image by name
227  * @param name the name of the image we're looking for
228  * @return the image registered by this name in the additionnal images
229  * map
230  * @throw CvProcessorException#INVALID_NAME is used name is not already
231  * registerd in the images
232  */
233 const Mat & CvProcessor::getImage(const char *name) const
234     throw (CvProcessorException)
235 {
236     string sname(name);
237 
238     return getImage(sname);
239 }
240 
241 /*
242  * Get image pointer by name
243  * @param name the name of the image we're looking for
244  * @return the image pointer registered by this name in the additionnal
245  * images map
246  * @throw CvProcessorException#INVALID_NAME is used name is not already
247  * registerd in the images
248  */
249 const Mat & CvProcessor::getImage(const string & name) const
250     throw (CvProcessorException)
251 {
252     // Search for this name
253     map<string, Mat*>::const_iterator cit;
254     for (cit = images.begin(); cit ≠ images.end(); ++cit)
255     {
256         if (cit→first ≡ name)
257         {
258             if (cit→second→data ≡ NULL)
259             {
260                 // image contains no data
261                 throw CvProcessorException(CvProcessorException::NULL_DATA,
262                                            name.c_str());
263             }
264             return *(cit→second);
265         }
266     }
267 
268     // not found : throw exception
269     throw CvProcessorException(CvProcessorException::INVALID_NAME,
270                                name.c_str());
```



```
271 }
272 
273 /*
274  * Get image pointer by name
275  * @param name the name of the image we're looking for
276  * @return the image pointer registered by this name in the additionnal
277  * images map
278  * @throw CvProcessorException#INVALID_NAME is used name is not already
279  * registerd in the images
280  */
281 Mat * CvProcessor::getImagePtr(const char *name)
282     throw (CvProcessorException)
283 {
284     string sname(name);
285 
286     return getImagePtr(sname);
287 }
288 
289 /*
290  * Get image pointer by name
291  * @param name the name of the image we're looking for
292  * @return the image registered by this name in the additionnal images
293  * map
294  * @throw CvProcessorException#INVALID_NAME is used name is not already
295  * registerd in the images
296  */
297 Mat * CvProcessor::getImagePtr(const string & name)
298     throw (CvProcessorException)
299 {
300     // Search for this name
301     map<string, Mat*>::const_iterator cit;
302     for (cit = images.begin(); cit ≠ images.end(); ++cit)
303     {
304         if (cit→first ≡ name)
305         {
306             if (verboseLevel ≥ VERBOSE_ACTIVITY)
307             {
308                 clog << "getImagePtr(" << name << "): returning : "
309                      << (long) (cit→second) << endl;
310             }
311             return cit→second;
312         }
313     }
314 
315     // not found : throw exception
316     throw CvProcessorException(CvProcessorException::INVALID_NAME, name.c_str());
317 }
318 
319 /*
320  * Number of channels in source image
321  * @return the number of channels of source image
322  */
323 int CvProcessor::getNbChannels() const
324 {
325     return nbChannels;
326 }
327 
328 /*
329  * Type of the source image
330  * @return the openCV type of the source image
331  */
332 int CvProcessor::getType() const
333 {
334     return type;
335 }
336 
337 /*
338  * Get the current verbose level
339  * @return the current verbose level
340  */
341 CvProcessor::VerboseLevel CvProcessor::getVerboseLevel() const
342 {
343     return verboseLevel;
344 }
345 
346 /*
347  * Set new verbose level
348  * @param level the new verobse level
349  */
350 void CvProcessor::setVerboseLevel(const VerboseLevel level)
351 {
352     if ((level ≥ VERBOSE_NONE) ∧ (level < NBVERBOSELEVEL))
353     {
354         verboseLevel = level;
355     }
356 
357     cout << "Verbose level set to: ";
358     switch (verboseLevel)
359     {
360         case VERBOSE_NONE:
```

```
361                 cout << "no messages";
362                 break;
363             case VERBOSE_ERRORS:
364                 cout << "unrecoverable errors only";
365                 break;
366             case VERBOSE_WARNINGS:
367                 cout << "errors and warnings";
368                 break;
369             case VERBOSE_NOTIFICATIONS:
370                 cout << "errors, warnings and notifications";
371                 break;
372             case VERBOSE_ACTIVITY:
373                 cout << "All messages";
374                 break;
375             case NBVERBOSELEVEL:
376             default:
377                 cout << "Unknown verobse mode (unchanged)";
378                 break;
379         }
380         cout << endl;
381     }
382
383     /*
384      * Return processor processing time of step index [default implementation
385      * returning only processTime, should be reimplemented in subclasses]
386      * @param index index of the step which processing time is required,
387      * 0 indicates all steps, and values above 0 indicates step #. If
388      * required index is bigger than number of steps than all steps value
389      * should be returned.
390      * @return the processing time of step index.
391      * @note should be reimplemented in subclasses in order to define
392      * time/feature behaviour
393      */
394     double CvProcessor::getProcessTime(const size_t) const
395     {
396         return processTime;
397     }
398
399     /*
400      * Return processor mean processing time of step index [default
401      * implementation returning only processTime, should be reimplemented
402      * in subclasses]
403      * @param index index of the step which processing time is required,
404      * 0 indicates all steps, and values above 0 indicates step #. If
405      * required index is bigger than number of steps than all steps value
406      * should be returned.
407      * @return the mean processing time of step index.
408      * @note should be reimplemented in subclasses in order to define
409      * time/feature behaviour
410      * @param index
411      */
412     double CvProcessor::getMeanProcessTime(const size_t) const
413     {
414         return meanProcessTime.mean();
415     }
416
417     /*
418      * Return processor processing time std of step index [default
419      * implementation returning only processTime, should be reimplemented
420      * in subclasses]
421      * @param index index of the step which processing time is required,
422      * 0 indicates all steps, and values above 0 indicates step #. If
423      * required index is bigger than number of steps than all steps value
424      * should be returned.
425      * @return the mean processing time of step index.
426      * @note should be reimplemented in subclasses in order to define
427      * time/feature behaviour
428      * @param index
429      */
430     double CvProcessor::getStdProcessTime(const size_t) const
431     {
432         return meanProcessTime.std();
433     }
434
435     /*
436      * Return processor minimum processing time of step index [default
437      * implementation returning only processTime, should be reimplemented
438      * in subclasses]
439      * @param index index of the step which processing time is required,
440      * 0 indicates all steps, and values above 0 indicates step #. If
441      * required index is bigger than number of steps than all steps value
442      * should be returned.
443      * @return the mean processing time of step index.
444      * @note should be reimplemented in subclasses in order to define
445      * time/feature behaviour
446      * @param index
447      */
448     clock_t CvProcessor::getMinProcessTime(const size_t) const
449     {
450         return meanProcessTime.min();
```



```
451 }
452
453 /*
454  * Return processor maximum processing time of step index [default
455  * implementation returning only processTime, should be reimplemented
456  * in subclasses]
457  * @param index index of the step which processing time is required,
458  * 0 indicates all steps, and values above 0 indicates step #. If
459  * required index is bigger than number of steps than all steps value
460  * should be returned.
461  * @return the mean processing time of step index.
462  * @note should be reimplemented in subclasses in order to define
463  * time/feature behaviour
464  * @param index
465  */
466 clock_t CvProcessor::getMaxProcessTime(const size_t) const
467 {
468     return meanProcessTime.max();
469 }
470
471 /*
472  * Reset mean and std process time in order to re-start computing
473  * new mean and std process time values.
474  */
475 void CvProcessor::resetMeanProcessTime()
476 {
477     meanProcessTime.reset();
478 }
479
480
481 /*
482  * Indicates if processing time is per feature processed in the current
483  * image or absolute
484  * @return
485  */
486 bool CvProcessor::isTimePerFeature() const
487 {
488     return timePerFeature;
489 }
490
491 /*
492  * Sets Time per feature processing time unit
493  * @param value the time per feature value (true or false)
494  */
495 void CvProcessor::setTimePerFeature(const bool value)
496 {
497     timePerFeature = value;
498 }
499
500 /*
501  * Send to stream (for showing processor attributes values)
502  * @param out the stream to send to
503  * @return a reference to the output stream
504  */
505 ostream & CvProcessor::toStream(ostream & out) const
506 {
507     return toStream_Impl<ostream>(out);
508 }
509
510 /*
511  * Send to output stream operator
512  * @param out the output stream to send to
513  * @param proc the processor to send to the output stream
514  * @return a reference to the output stream used
515  */
516 ostream & operator <<(ostream & out, const CvProcessor & proc)
517 {
518     return proc.toStream(out);
519 }
520
521 /*
522  * Proto instantiation of CvProcessor template method
523  * Stream & CvProcessor::toStream_Impl<Stream>(Stream &) const with concrete
524  * type ostream
525  */
526 template ostream & CvProcessor::toStream_Impl<ostream>(ostream &) const;
```

```
#ifndef CVPROCESSOREXCEPTION_H_
#define CVPROCESSOREXCEPTION_H_

#include <iostream>     // for ostream
#include <string>       // for string
#include <exception>    // for std::exception base class
using namespace std;

/**
 * Exception class for CvProcessor.
 * Contains mainly exception reasons why an CvProcessor operation could not be
 * performed.
 */
class CvProcessorException : public exception
{
    public:
        /**
         * Matrices operation exception cases
         */
        typedef enum
        {
            /**
             * Null image.
             * Used when trying to add null image as source image of the
             * processor
             */
            NULL_IMAGE,
            /**
             * Null image data.
             * Used when trying to use image with NULL data
             */
            NULL_DATA,
            /**
             * Invalid name in image acces by name.
             * Used when searching for images by name which is not contained
             * in the already registered names
             */
            INVALID_NAME,
            /**
             * Invalid image type.
             * Some Processors needs specific images types
             */
            INVALID_IMAGE_TYPE,
            /**
             * Illegal data access (i.e. read/write access on read only data)
             */
            ILLEGAL_ACCESS,
            /**
             * Allocation failure on dynamically allocated elements
             */
            ALLOC_FAILURE,
            /**
             * Unable to read a file
             */
            FILE_READ_FAIL,
            /**
             * File parse error
             */
            FILE_PARSE_FAIL,
            /**
             * Unable to write file
             */
            FILE_WRITE_FAIL,
            /**
             * OpenCV exception
             */
            OPENCV_EXCEPTION
        } ExceptionCause;

        /**
         * CvProcessor exception constructor
         * @param e the chosen error case for this error
         * @see ExceptionCause
         */
        CvProcessorException(const CvProcessorException::ExceptionCause e);

        /**
         * CvProcessor exception constructor with exception message descriptor
         * @param e the chosen error case for this error
         * @param descr character string describing the message
         * @see ExceptionCause
         */
        CvProcessorException(const CvProcessorException::ExceptionCause e,
                             const char * descr);

        /**
         * CvProcessor exception from regular (typically OpenCV) exception
         * @param e the exception to relay
         */
        CvProcessorException(const exception & e, const char * descr = "");
```



```

        /**
         * CvProcessor exception destructor
         * @post message cleared
         */
        virtual ~CvProcessorException() throw ();

        /**
         * Explanation message of the exception
         * @return a C-style character string describing the general cause
         * of the current error.
         */
        virtual const char* what() const throw();

        /**
         * CvProcessorException cause
         * @return the cause enum of the exception
         */
        CvProcessorException::ExceptionCause getCause();

        /**
         * Source message of the exception
         * @return the message string of the exception
         */
        string getMessage();

        /*
         * Note output operators are not necessary since what() method is used
         * to explain the reason of the exception.
         * Example :
         * try
         * {
         *  ... do something which throws an std::exception
         * }
         * catch (exception & e)
         * {
         *  cerr << e.what() << endl;
         * }
         */

    private:
        /**
         * The current error case
         */
        CvProcessorException::ExceptionCause cause;

        /**
         * description message of the exception
         */
        string message;
};

#endif /*CVPROCESSOREXCEPTION_H_*/
```

```cpp
#include "CvProcessorException.h"
#include <iostream>      // for cerr et endl;
#include <string>        // for string
#include <sstream>       // for ostringstream
using namespace std;

/*
 * CvProcessor exception constructor
 * @param e the chosen error case for this error
 * @see ExceptionCause
 */
CvProcessorException::CvProcessorException(
    const CvProcessorException::ExceptionCause e) :
    exception(),
    cause(e),
    message("")
{
}

/*
 * CvProcessor exception constructor with message descriptor
 * @param e the chosen error case for this error
 * @param descr character string describing the message
 * @see ExceptionCause
 */
CvProcessorException::CvProcessorException(
    const CvProcessorException::ExceptionCause e, const char * descr) :
    exception(),
    cause(e),
    message(descr)
{
}

/*
 * CvProcessor exception from regular (typically OpenCV) exception
 * @param e the exception to relay
 */
CvProcessorException::CvProcessorException(const exception & e, const char * descr) :
    exception(e),
    cause(OPENCV_EXCEPTION),
    message(descr)
{
}


/*
 * CvProcessor exception destructor
 * @post message cleared
 */
CvProcessorException::~CvProcessorException() throw ()
{
    message.clear();
}

/*
 * Explanation message of the exception
 * @return a C-style character string describing the general cause
 * of the current error.
 */
const char * CvProcessorException::what() const throw()
{
    const char * initialWhat = exception::what();

    ostringstream output;

    output << initialWhat << ":";

    output << "CvProcessorException : ";

    if (message.length() > 0)
    {
        output << message << ":";
    }

    switch (cause) {
        case CvProcessorException::NULL_IMAGE:
            output << "NULL image" << endl ;
            break;
        case CvProcessorException::NULL_DATA:
            output << "NULL image data" << endl ;
            break;
        case CvProcessorException::INVALID_NAME:
            output << "Invalid name" << endl ;
            break;
        case CvProcessorException::INVALID_IMAGE_TYPE:
            output << "Invalid image type" << endl;
            break;
        case CvProcessorException::ILLEGAL_ACCESS:
            output << "Illegal access" << endl;
            break;
```



```cpp
        case CvProcessorException::ALLOC_FAILURE:
            output << "New element allocation failure" << endl;
            break;
        case CvProcessorException::FILE_READ_FAIL:
            output << "Unable to read file" << endl;
            break;
        case CvProcessorException::FILE_PARSE_FAIL:
            output << "File parse error" << endl;
            break;
        case CvProcessorException::FILE_WRITE_FAIL:
            output << "Unable to write file" << endl;
            break;
        default:
            output << "Unknown exception" << endl;
            break;
    }

    return output.str().c_str();
}


/*
 * CvProcessorException cause
 * @return the cause enum of the exception
 */
CvProcessorException::ExceptionCause CvProcessorException::getCause()
{
    return cause;
}

/*
 * Source message of the exception
 * @return the message string of the exception
 */
string CvProcessorException::getMessage()
{
    return message;
}
```

```
/*
 * QcvProcessor.h
 *
 *  Created on: 19 fÃ©vr. 2012
 *      Author: davidroussel
 */

#ifndef QCVPROCESSOR_H_
#define QCVPROCESSOR_H_

#include <QObject>
#include <QDebug>
#include <QString>
#include <QRegExp>
#include <QMutex>
#include <QThread>
#include "CvProcessor.h"
Q_DECLARE_METATYPE(CvProcessor::ProcessTime)

/**
 * Qt flavored class to process a source image with OpenCV 2+
 */
class QcvProcessor : public QObject, public virtual CvProcessor
{
    Q_OBJECT

    protected:

        /**
         * Default timeout to show messages
         */
        static int defaultTimeOut;

        /**
         * Number format used to format numbers into QStrings
         */
        static QString numberFormat;

        /**
         * The regular expression used to validate new number formats
         * @see #setNumberFormat
         */
        static QRegExp numberRegExp;

        /**
         * format used to format Mean/Std time values : <mean> Â± <std>
         */
        static QString meanStdFormat;

        /**
         * format used to format Min/Max time values : <min> / <max>
         */
        static QString minMaxFormat;

        /**
         * The Source image mutex in order to avoid concurrent access to
         * the source image (typically the source image may be currently
         * modified by the capture for instance)
         */
        QMutex * sourceLock;

        /**
         * the thread in which this processor should run
         */
        QThread * updateThread;

        /**
         * Message to send when something changes
         */
        QString message;

        /**
         * String used to store formatted process time value
         */
        QString processTimeString;

        /**
         * String used to store formatted min/max time values
         */
        QString processMinMaxTimeString;

    public:

        /**
         * QcvProcessor constructor
         * @param image the source image
         * @param imageLock the mutex for concurrent access to the source image.
         * In order to avoid concurrent access to the same image
         * @param updateThread the thread in which this processor should run
         * @param parent parent QObject
```



```
         */
        QcvProcessor(Mat * image,
                     QMutex * imageLock = NULL,
                     QThread * updateThread = NULL,
                     QObject * parent = NULL);

        /**
         * QcvProcessor destructor
         */
        virtual ~QcvProcessor();

        /**
         * Sets new number format
         * @param format the new number format
         * @pre format string should look like "%8.1f" or at least not be longer
         * than 10 chars since format is a 10 chars array.
         * @post id format string is valid and shorter than 10 chars
         * it has been applied as the new format string.
         */
        static void setNumberFormat(const char * format);

        /**
         * Get the format c-string for numbers
         * @return the format string for numbers (e.g.: "%5.2f")
         */
        static const char * getNumberFormat();

        /**
         * Get the format c-string for std dev of numbers
         * @return the format string for numbers (e.g.: " Â± %4.2f")
         */
        static const char * getStdFormat();

        /**
         * Get the format c-string for min / max of numbers
         * @return the format string for numbers (e.g.: "%5.2f / %5.2f")
         */
        static const char * getMinMaxFormat();

        /**
         * Send to debug stream (for showing processor attributes values)
         * @param dbg the debug stream to send to
         * @return a reference to the output stream
         */
        virtual QDebug & toDBStream(QDebug & dbg) const;

        /**
         * Friend QDebug output operator
         * @param dbg the debug stream
         * @param proc the QcvProcessor to send to debug stream
         * @return the debug stream
         */
        friend QDebug & operator <<(QDebug & dbg, const QcvProcessor & proc);

    public slots:
        /**
         * Update computed images slot and sends updated signal
         */
        virtual void update();

        /**
         * Changes source image slot.
         * Attributes needs to be cleaned up then set up again
         * @param image the new source Image
         * @throw CvProcessorException#NULL_IMAGE when new source image is NULL
         * @post Various signals are emitted:
         *  - imageChanged(sourceImage)
         *  - imageCchanged()
         *  - if image size changed then imageSizeChanged() is emitted
         *  - if image color space changed then imageColorsChanged() is emitted
         */
        virtual void setSourceImage(Mat * image) throw (CvProcessorException);

        /**
         * Sets Time per feature processing time unit (reimplemented as a slot).
         * @param value the time per feature value (true or false)
         */
        virtual void setTimePerFeature(const bool value);

        /**
         * Reset mean and std process time in order to re-start computing
         * (reimplemented as a slot)
         * new mean and std process time values.
         */
        virtual void resetMeanProcessTime();

    signals:
        /**
         * Signal emitted when update is complete
         */
```

```
181         void updated();
182
183         /**
184          * Signal emitted when processor has finished.
185          * Used to tell helper threads to quit
186          */
187         void finished();
188
189         /**
190          * Signal emitted when source image is reallocated
191          */
192         void imageChanged();
193
194         /**
195          * Signal emitted when source image is reallocated
196          * @param image the new source image pointer or none if just
197          * image changed notification is required
198          */
199         void imageChanged(Mat * image);
200
201         /**
202          * Signal emitted when source image colors changes from color to gray
203          * or from gray to color
204          */
205         void imageColorsChanged();
206
207         /**
208          * Signal emitted when source image size changes
209          */
210         void imageSizeChanged();
211
212         /**
213          * Signal emitted when processing time has changed
214          * @param formattedValue the new value of the processing time
215          */
216         void processTimeUpdated(const QString & formattedValue);
217
218         /**
219          * Signal emitted when min/max processing time has changed
220          * @param formattedValue the new value of the processing time
221          */
222         void processTimeMinMaxUpdated(const QString & formattedValue);
223
224         /**
225          * Signal emitted when processing time has changed
226          * @param time the new processing time
227          */
228         void processTimeUpdated(const CvProcessor::ProcessTime * time);
229
230         /**
231          * Signal to set text somewhere
232          * @param message the message
233          */
234         void sendText(const QString & message);
235
236         /**
237          * Signal to send update message when something changes
238          * @param message the message
239          * @param timeout number of ms the message should be displayed
240          */
241         void sendMessage(const QString & message, int timeout = defaultTimeOut);
242 };
243
244 #endif /* QCVPROCESSOR_H_ */
```



```
1  /*
2   * QCvProcessor.cpp
3   *
4   *  Created on: 19 fÃ©vr. 2012
5   *      Author: davidroussel
6   */
7
8  #include <QRegExpValidator>
9  #include <QMetaType>
10 #include <QDebug>
11 #include "QcvProcessor.h"
12
13 /*
14  * Proto instantiation of CvProcessor template method
15  * Stream & CvProcessor::toStream_Impl<Stream>(Stream &) const with concrete
16  * type Qdebug
17  */
18 template QDebug & CvProcessor::toStream_Impl<QDebug>(QDebug &) const;
19
20 /*
21  * Default timeout to show messages
22  */
23 int QcvProcessor::defaultTimeOut = 5000;
24
25 /*
26  * Number format used to format numbers into QStrings
27  */
28 QString QcvProcessor::numberFormat = QString::fromUtf8("%7.0f");
29
30 /*
31  * The regular expression used to validate new number formats
32  * @see #setNumberFormat
33  */
34 QRegExp QcvProcessor::numberRegExp("%[+- 0#]*[0-9]*([.][0-9]+)?[efEF]");
35
36 /*
37  * format used to format Mean/Std time values : <mean> Â± <std>
38  */
39 QString QcvProcessor::meanStdFormat = numberFormat + QString::fromUtf8(" Â± %5.0f");
40
41 /*
42  * format used to format Min/Max time values : <min> / <max>
43  */
44 QString QcvProcessor::minMaxFormat = numberFormat +  QString::fromUtf8("/") +
45                                      numberFormat;
46
47 /*
48  * QcvProcessor constructor
49  * @param image the source image
50  * @param imageLock the mutex for concurrent access to the source image
51  * In order to avoid concurrent access to the same image
52  * @param updateThread the thread in which this processor should run
53  * @param parent parent QObject
54  */
55 QcvProcessor::QcvProcessor(Mat * image,
56                            QMutex * imageLock,
57                            QThread * updateThread,
58                            QObject * parent) :
59     CvProcessor(image), // <-- virtual base class constructor first
60     QObject(parent),
61     sourceLock(imageLock),
62     updateThread(updateThread),
63     message(),
64     processTimeString()
65 {
66     if (updateThread ≠ NULL)
67     {
68         this→moveToThread(updateThread);
69
70         connect(this, SIGNAL(finished()), updateThread, SLOT(quit()),
71                 Qt::DirectConnection);
72
73         updateThread→start();
74     }
75 }
76
77 /*
78  * QcvProcessor destructor
79  */
80 QcvProcessor::~QcvProcessor()
81 {
82     // Lock might be already destroyed in source object so don't try to unlock
83
84     message.clear();
85     processTimeString.clear();
86
87     emit finished();
88
89     if (updateThread ≠ NULL)
90     {
```

```
91          // Wait until update thread has received the "finished" signal through
92          // "quit" slot
93          updateThread→wait();
94      }
95  }
96
97  /*
98   * Sets new number format
99   * @param format the new number format
100  */
101 void QcvProcessor::setNumberFormat(const char * format)
102 {
103     /*
104      * The format string should validate the following regex
105      * %[+- 0#]*[0-9]*([.][0-9]+)?[efEF]
106      */
107     QRegExpValidator validator(numberRegExp, NULL);
108
109     QString qFormat(format);
110     int pos = 0;
111     if (validator.validate(qFormat,pos) ≡ QValidator::Acceptable)
112     {
113         numberFormat = format;
114         meanStdFormat = format + QString::fromUtf8(" Â± ") + format;
115         minMaxFormat = format + QString::fromUtf8("/") + format;
116     }
117     else
118     {
119         qWarning("QcvProcessor::setNumberFormat(%s) : invalid format", format);
120     }
121 }
122
123 /*
124  * Send to stream (for showing processor attributes values)
125  * @param dbg the debug stream to send to
126  * @return a reference to the output stream
127  */
128 QDebug & QcvProcessor::toDBStream(QDebug & dbg) const
129 {
130     return toStream_Impl<QDebug>(dbg);
131 }
132
133 /*
134  * Friend QDebug output operator
135  * @param dbg the debug stream
136  * @param proc the Qcvprocessor to send to debug stream
137  * @return the debug stream
138  */
139 QDebug & operator << (QDebug & dbg, const QcvProcessor & proc)
140 {
141     proc.toDBStream(dbg.nospace());
142     return dbg.space();
143 }
144
145 /*
146  * Update computed images slot and sends updated signal
147  * required
148  */
149 void QcvProcessor::update()
150 {
151     /*
152      * Important note : CvProcessor::update() should NOT be called here
153      * since it should be called in QcvXXXprocessor subclasses such that
154      * QcvXXXProcessor::update method should contain :
155      *  - call to CvXXXProcessor::update() (not QCvXXXProcessor)
156      *  - emit signals from QcvXXXProcessor
157      *  - call to QcvProcessor::update() (this method) to
158      *      - emit updated signal
159      *      - emit standard process time strings signals
160      *  - or
161      *      - emit updated signal in QcvXXXProcessor
162      *      - customize your processtimes and emit time strings signals
163      */
164     emit updated();
165     processTimeString.sprintf(meanStdFormat.toStdString().c_str(),
166                               getMeanProcessTime(0), getStdProcessTime(0));
167 //  processMinMaxTimeString.sprintf(minMaxFormat.toStdString().c_str(),
168 //                                  getMinProcessTime(0), getMaxProcessTime(0));
169     emit processTimeUpdated(processTimeString);
170 //  emit processTimeMinMaxUpdated(processMinMaxTimeString);
171     emit processTimeUpdated(&meanProcessTime);
172 }
173
174 /*
175  * Changes source image slot.
176  * Attributes needs to be cleaned up then set up again
177  * @param image the new source Image
178  * @post Various signals are emitted:
179  *  - imageChanged(sourceImage)
180  *  - imageCchanged()
```



```
181  *  - if image size changed then imageSizeChanged() is emitted
182  *  - if image color space changed then imageColorsChanged() is emitted
183  */
184 void QcvProcessor::setSourceImage(Mat *image)
185     throw (CvProcessorException)
186 {
187     Size previousSize(sourceImage→size());
188     int previousNbChannels(nbChannels);
189
190     if (sourceLock ≠ NULL)
191     {
192         sourceLock→lock();
193         // qDebug() << "QcvProcessor::setSourceImage: lock";
194     }
195
196     CvProcessor::setSourceImage(image);
197
198     if (sourceLock ≠ NULL)
199     {
200         // qDebug() << "QcvProcessor::setSourceImage: unlock";
201         sourceLock→unlock();
202     }
203
204     emit imageChanged(sourceImage);
205
206     emit imageChanged();
207
208     if ((previousSize.width ≠ image→cols) ∨
209         (previousSize.height ≠ image→rows))
210     {
211         emit imageSizeChanged();
212     }
213
214     if (previousNbChannels ≠ nbChannels)
215     {
216         emit imageColorsChanged();
217     }
218
219     // Force update
220     update();
221 }
222
223 /*
224  * Sets Time per feature processing time unit (reimplemented as a slot).
225  * @param value the time per feature value (true or false)
226  */
227 void QcvProcessor::setTimePerFeature(const bool value)
228 {
229     CvProcessor::setTimePerFeature(value);
230 }
231
232 /*
233  * Reset mean and std process time in order to re-start computing
234  * (reimplemented as a slot)
235  * new mean and std process time values.
236  */
237 void QcvProcessor::resetMeanProcessTime()
238 {
239     CvProcessor::resetMeanProcessTime();
240 }
241
242
243 /*
244  * Get the format c-string for numbers
245  * @return the format string for numbers (e.g.: "%5.2f")
246  */
247 const char * QcvProcessor::getNumberFormat()
248 {
249     return numberFormat.toStdString().c_str();
250 }
251
252 /*
253  * Get the format c-string for std dev of numbers
254  * @return the format string for numbers (e.g.: " Â± %4.2f")
255  */
256 const char * QcvProcessor::getStdFormat()
257 {
258     return meanStdFormat.toLocal8Bit().data();
259 }
260
261 /*
262  * Get the format c-string for min / max of numbers
263  * @return the format string for numbers (e.g.: "%5.2f / %5.2f")
264  */
265 const char * QcvProcessor::getMinMaxFormat()
266 {
267     return minMaxFormat.toLocal8Bit().data();
268 }
```

```
/*
 * CvGFilter.h
 *
 *  Created on: 26 fÃ©vr. 2012
 *      Author: davidroussel
 */

#ifndef CVGFILTER_H_
#define CVGFILTER_H_

#include "CvProcessor.h"

/**
 * Class to process source image with gaussian filters
 */
class CvGFilter: virtual public CvProcessor
{
    public:

        /**
         * Image Display type
         */
        typedef enum
        {
            INPUT_IM = 0, //!< Input image
            GRAY_IM, //!< Input gray image
            BLURRED_IM, //!< Gaussian Blurred gray image
            GRADIENT_X_IM, //!< Horizontal gradient component
            GRADIENT_Y_IM, //!< Vertical gradient component
            GRADIENT_MAG_IM, //!< Gradient Magnitude
            GRADIENT_ANGLE_IM, //!< Gradient angle
            EDGE_MAP_IM, //!< Edge Map from mag thresholding
            LAPLACIAN_IM, //!< Laplacian
            CORNERNESS_IM, //!< Cornerness measure
            HARRISCORNER_IM, //!< OpenCV Harris Cornerness measure
            NBDISPLAY_IM //!< Number of elements in this enum
        } ImageDisplay;

        typedef enum
        {
            THRESHOLD = 0, //!< Show only edge map from gradient magnitude threshold
            CANNY, //!< Show only Canny's edge map
            MERGED, //!< Show merged edge maps
            NBEDGEDISPLAY //!< Number of elements in this enum
        } EdgeDisplay;

    protected:
        /**
         * Size of all processed images: sourceImage->size()
         */
        Size dim;

        /**
         * image processing result type: CV_32FC1 or CV_64FC1
         */
        int procType;

        /**
         * image processing result type for display images: CV_8UC1
         */
        int displayType;

        /**
         * Source image converted to gray
         */
        Mat inFrameGray;
        // ------------------------------------------------------------------
        // Gaussian kernels parameters
        // ------------------------------------------------------------------
        /**
         * Gaussian kernel size (3, 5, 7..., 15): 7,
         * size is \f$(2n+1)\f$ with \f$n \in [1..7]\f$
         */
        int kernelSize;

        /**
         * Minimum kernel size: 3
         */
        static const int minKernelSize;

        /**
         * Maximum kernel size: 15
         */
        static const int maxKernelSize;

        /**
         * Gaussian Variance to apply on gaussian kernel: kernelSize/5.0
         */
        double sigma;

        /**
         * Minimum gaussian variance: kernelSize / 20.0
         */
        double minSigma;

        /**
         * Maximum gaussian variance : kernelSize / 2.0
         */
        double maxSigma;

        /**
         * gaussian variance steps
         */
        static const double sigmaStep;

        /**
         * Indicates sigma has changed so gaussian kernels should be
         * recomputed: true
         */
        bool sigmaChanged;

        // ------------------------------------------------------------------
        // Threshold for edge map
        // ------------------------------------------------------------------
        /**
         * Threshold value for edge map: 128
         */
        int thresholdLevel;

        /**
         * Minimum threshold value for edge map: 0
         */
        static const int minThreshold;

        /**
         * Maximum threshold value for edge map: 255
         */
        static const int maxThreshold;

        // ------------------------------------------------------------------
        // Harris K constant for cornerness measure
        // ------------------------------------------------------------------
        /**
         * Harris K constant.
         * initial value is 0.04
         */
        double harrisKappa;

        /**
         * Harris K constant mininal value: 0.04
         */
        static const double harrisKappaMin;

        /**
         * Harris K constant maximal value: 0.15
         */
        static const double harrisKappaMax;

        /**
         * Harris K constant steps: 0.01
         */
        static const double harrisKappaStep;

        // ------------------------------------------------------------------
        // Gaussian filters kernels (all kernels are procType)
        // ------------------------------------------------------------------
        /**
         * Gaussian 1D horizontal filter
         */
        Mat gX;

        /**
         * Gaussian 1D vertical filter
         */
        Mat gY;

        /**
         * Gaussian horizontal 1st derivative filter: dx
         */
        Mat gDx;

        /**
         * Gaussian vertical 1st derivative filter: dy
         */
        Mat gDy;

        /**
         * Gaussian horizontal 2nd derivative filter: d2x
         */
        Mat gD2x;
```

```
181
182         /**
183          * Gaussian vertical 2nd derivative filter: d2y
184          */
185         Mat gD2y;
186
187         /**
188          * Gaussian horizontal and vertical 1st derivative filter: dxy
189          */
190         Mat gDxy;
191
192         /**
193          * 2D Gaussian kernel for Ixx,Iyy and Ixy smoothing in cornerness
194          */
195         Mat g2D;
196
197         // ------------------------------------------------------------------
198         // Processing images results (all images are procType)
199         // ------------------------------------------------------------------
200         /**
201          * Image display mode.
202          */
203         ImageDisplay displayMode;
204
205         /**
206          * Edge display mode.
207          */
208         EdgeDisplay edgeMode;
209
210         /**
211          * Blurred image processed with gaussian vertical and horizontal
212          * kernels
213          */
214         Mat blurred;
215
216         /**
217          * Image processed with 1st derivative horizontal gaussian kernel
218          * and vertical gaussian kernel.
219          * Horizontal gradient: \f$ I_{x} \f$
220          */
221         Mat dX;
222
223         /**
224          * Image processed with 1st derivative vertical gaussian kernel and
225          * horizontal gaussian kernel.
226          * Vertical gradient: \f$ I_{y} \f$
227          */
228         Mat dY;
229
230         /**
231          * Gradient magnitude: \f$ \sqrt{I_{x}^{2} + I_{y}^{2}} \f$
232          */
233         Mat gradientMag;
234
235         /**
236          * Gradient angle: \f$ atan(\frac{I_{y}}{I_{x}}) \f$
237          */
238         Mat gradientAngle;
239
240         /**
241          * Image processed with horizontal 2nd derivative gaussian kernel and
242          * vertical gaussian kernel. Or eventually Horizontal gradient
243          * processed with 1st derivative horizontal gaussian kernel.
244          * Horizontal laplacian: \f$ I_{x^{2}} \f$
245          */
246         Mat d2X;
247
248         /**
249          * Image processed with vertical 2nd derivative gaussian kernel and
250          * horizontal gaussian kernel. Or eventually Vertical gradient
251          * processed with 1st derivative vertical gaussian kernel.
252          * Vertical laplacian: \f$ I_{y^{2}} \f$
253          */
254         Mat d2Y;
255
256         /**
257          * Laplacian image: sum of horizontal and vertical laplacian components.
258          * laplacian: \f$ I_{x^{2}} + I_{y^{2}} \f$
259          */
260         Mat laplacian;
261
262         /**
263          * dXY component needed by Hessian matrix (in cornerness computing):
264          * \f$ I_{xy} \f$
265          */
266         Mat dXY;
267
268         /**
269          * Cornerness image computed according to Harris Measure.
270          * Such as the Hessian matrix : \f[
```



```
271          * H =
272          *  \left(
273          *      \begin{array}{cc}
274          *          I_{x^{2}}   & I_{xy}\\
275          *          I_{yx}  & I_{y^{2}}\\
276          *      \end{array}
277          *  \right)
278          * \f]
279          * Harris measure is \f[
280          * det(H) - k trace(H) = I_{x}^{2} I_{y}^{2} - I_{xy}^{2} - k
281          * \left( I_{x}^{2} + I_{y}^{2} \right)^{2}
282          * \f]
283          */
284         Mat cornerness;
285
286         /**
287          * Cornerness image computed directly with OpenCV harris function
288          */
289         Mat harris;
290
291         // ------------------------------------------------------------------
292         // Processing images results (all images are procType)
293         // ------------------------------------------------------------------
294         /**
295          * Blurred image converted for display
296          */
297         Mat blurredDisplay;
298
299         /**
300          * Horizontal gradient image converted for display
301          */
302         Mat dXDisplay;
303
304         /**
305          * Vertical gradient image converted for display
306          */
307         Mat dYDisplay;
308
309         /**
310          * Gradient magnitude image converted for display
311          */
312         Mat gradientMagDisplay;
313
314         /**
315          * Gradient angle image converted for display
316          */
317         Mat gradientAngleDisplay;
318
319         /**
320          * Gradient magnitude display thresholded to show edges
321          */
322         Mat edgeMap;
323
324         /**
325          * Laplacian image converted for display
326          */
327         Mat laplacianDisplay;
328
329         /**
330          * Cornerness image converted for display
331          */
332         Mat cornernessDisplay;
333
334         /**
335          * Canny edges image
336          */
337         Mat cannyEdgeMap;
338
339         /**
340          * Edge maps components.
341          * allow to mix inFrameGray, edgeMap and cannyEdge to produce a mmixed
342          * edge image
343          */
344         vector<Mat> edgeMapComponents;
345
346         /**
347          * Mixed edges image
348          */
349         Mat mixEdge;
350
351         /**
352          * Harris cornerness image converted for display
353          */
354         Mat harrisDisplay;
355
356         /**
357          * Setup attributes when source image is changed
358          * @param image source Image
359          * @param completeSetup is true when used to change source image,
360          * and false when used in constructor
```

```
361         */
362         virtual void setup(Mat *image, bool completeSetup);
363
364         /**
365          * Cleanup attributes before changing source image or cleaning class
366          * before destruction
367          */
368         virtual void cleanup();
369
370     public:
371         /**
372          * Gaussian filtering class constructor
373          * @param sourceImage
374          */
375         CvGFilter(Mat *sourceImage);
376
377         /**
378          * Gaussian filtering class destructor
379          */
380         virtual ~CvGFilter();
381
382         /**
383          * Gaussian filtering update
384          *  - convert source image to gray
385          *  - if sigma changed recompute gaussian kernels
386          *  - compute blurred image and convert it for display
387          *  - compute horizontal and vertical gradients and convert them
388          *  for display
389          *  - compute gradient magnitude and angle and convert them for display
390          *  - threshold gradient magintude to edgeMap
391          *  - compute horizontal and vertical laplacian components and
392          *  laplacian image
393          *  - compute dXY to prepare cornerness measure
394          */
395         virtual void update();
396
397         /**
398          * Get current kernel size
399          * @return the current kernel size
400          */
401         int getKernelSize() const;
402
403         /**
404          * Sets the a new kernel size
405          * @param kernelSize the new kernel size
406          * @post if new size is in range [3..15]
407          * with a step of 2;
408          *  - the new kernel size is set up, and remains unchanged otherwise.
409          *  - gaussian kernels are eventually recomputed
410          */
411         virtual void setKernelSize(int kernelSize);
412
413         /**
414          * Gets maximum kernel size
415          * @return the maximum kernel size
416          */
417         static int getMaxKernelSize();
418
419         /**
420          * Gets minimum kernel size
421          * @return the minimum kernel size
422          */
423         static int getMinKernelSize();
424
425         /**
426          * Gets the current value of gaussian variance
427          * @return the current value of gaussian variance
428          */
429         double getSigma() const;
430
431         /**
432          * Sets a new value for gaussian variance
433          * @param sigma the new value of gaussian variance
434          */
435         virtual void setSigma(double sigma);
436
437         /**
438          * Gets the minimum possible value of gaussian variance
439          * according to kernel size
440          * @return the minimum gaussian variance
441          */
442         double getMinSigma() const;
443
444         /**
445          * Gets the maximum possible value of gaussian variance
446          * according to kernel size
447          * @return the maximum gaussian variance
448          */
449         double getMaxSigma() const;
450
```



```
451         /**
452          * Gets the steps for sigma increments
453          * @return steps for sigma increments
454          */
455         static double getSigmaStep();
456
457         /**
458          * Gets the sigma changed status, in order to recompute
459          * gaussian kernels (if needed)
460          * @return true if sigma is different from last update,
461          * false otherwise
462          */
463         bool isSigmaChanged() const;
464
465         /**
466          * Gets the current threshold level of edgemap
467          * @return the current threshold level
468          */
469         int getThresholdLevel() const;
470
471         /**
472          * Sets new threshold level for edge map
473          * @param thresholdLevel the new threshold level
474          */
475         virtual void setThresholdLevel(int thresholdLevel);
476
477         /**
478          * Gets minimum threshold value for edgemap
479          * @return the minimum threshold value for edgemap
480          */
481         static int getMinThreshold();
482
483         /**
484          * Gets maximum threshold value for edgemap
485          * @return the maximum threshold value for edgemap
486          */
487         static int getMaxThreshold();
488
489         /**
490          * Gets the current Harris parameter Kappa
491          * @return the current value of Kappa
492          */
493         double getHarrisKappa() const;
494
495         /**
496          * Sets new Harris parameter Kappa
497          * @param harrisKappa the new parameter to set
498          */
499         virtual void setHarrisKappa(double harrisKappa);
500
501         /**
502          * Gets maximum Harris parameter Kappa
503          * @return the maximum Harris parameter Kappa
504          */
505         static double getHarrisKappaMax();
506
507         /**
508          * Gets minimum Harris parameter Kappa
509          * @return the minimum Harris parameter Kappa
510          */
511         static double getHarrisKappaMin();
512
513         /**
514          * Gets Harris parameter Kappa increment
515          * @return the Harris parameter Kappa increment
516          */
517         static double getHarrisKappaStep();
518
519         /**
520          * Get current display mode
521          * @return the current display mode
522          */
523         ImageDisplay getDisplayMode() const;
524
525         /**
526          * Sets a new display mode
527          * @param displayMode the new display mode to set
528          */
529         virtual void setDisplayMode(const ImageDisplay displayMode);
530
531         /**
532          * Gets the current edge display mode
533          * @return the current edge display mode
534          */
535         EdgeDisplay getEdgeMode() const;
536
537         /**
538          * Set a new edge didsplay mode
539          * @param edgeMode the new edge mode
540          */
```

```
541         void setEdgeMode(const EdgeDisplay edgeMode);
542
543         /**
544          * Gets Image reference corresponding to the current displayMode and
545          * edgeMode
546          * @return Image reference corresponding to the current displayMode and
547          * edgeMode
548          */
549         const Mat & getDisplayModeImage();
550
551         /**
552          * Gets Image pointer corresponding to the current displayMode and
553          * edgeMode
554          * @return Image reference corresponding to the current displayMode and
555          * edgeMode
556          */
557         Mat * getDisplayModeImagePtr();
558
559     protected:
560         // ------------------------------------------------------------------
561         // Utility methods
562         // ------------------------------------------------------------------
563         /**
564          * Compute 1D or 2D normalized gaussian into kernel with sigma variance and
565          * amp amplitude.
566          * \f[
567          * g(x,y) = A \cdot e^{-\left(\frac{\left(x - x_0\right)^2+\left(y - y_0\right)^2}
568          * {2\sigma^2}\right)}
569          * \f]
570          * where \f$ A = \frac{1}{\sum {y}\sum {x}g(x,y)}\f$ and
571          * \f$x_0 = floor(\frac{kernel.cols}{2}) + 1\f$ and
572          * \f$y_0 = floor(\frac{kernel.rows}{2}) + 1\f$.
573          * @param kernel matrix to store gaussian kernel:
574          *  - if kernel size is \f$ 1 \times N\f$ or \f$ N \times 1\f$ produces a 1D
575          *  line or column filter.
576          *  - if kernel size is \f$ M \times N\f$ produces a 2D gaussian filter.
577          * @param sigma standard deviation \f$\sigma\f$ of the gauss curve (or surface)
578          * expressed in pixels
579          * @param derivOrderX horizontal derivative order. Should be 0, 1 or 2:
580          *  - derivOrderX = 1: \f$\frac{\delta g(x,y)}{\delta x}=
581          *                        \frac{-(x - x_0)}{\sigma^2}\cdot g(x,y)\f$
582          *  - derivOrderX = 2: \f$\frac{\delta^2 g(x,y)}{\delta x^2}=
583          *                        \frac{(x - x_0)^2 - \sigma^2}{\sigma^4}\cdot g(x,y)\f$
584          * @param derivOrderY vertical derivative order. Should be 0, 1 or 2
585          *  - derivOrderY = 1: \f$\frac{\delta g(x,y)}{\delta y}=
586          *                        \frac{-(y - y_0)}{\sigma^2}\cdot g(x,y)\f$
587          *  - derivOrderY = 2: \f$\frac{\delta^2 g(x,y)}{\delta y^2}=
588          *                        \frac{(y - y_0)^2 - \sigma^2}{\sigma^4}\cdot g(x,y)\f$
589          * @warning kernel size should be odd (2n+1) in order to have an application
590          * point at the center odf the matrix
591          * The type parameter T applied to gaussian2D depends on the type of
592          * the kernel matrix:
593          *  - if kernel type is CV_64FC1, then you should use
594          *  gaussian<double>(kernel...)
595          *  - if kernel type is CV_32FC1, then you should use
596          *  gaussian<float>(kernel...)
597          *  - Other kernels Mat types haven't been tested
598          */
599         template<typename T>
600         void gaussian(Mat & kernel,
601                       const double sigma,
602                       const unsigned int derivOrderX = 0,
603                       const unsigned int derivOrderY = 0);
604
605         /**
606          * Compute Cornerness measure based on Harris criteria.
607          * This criteria is based on the Hessian matrix build upon
608          * 2nd order horizontal and vertical \f$I_{xx} \& I_{xx}\f$ derivatives and
609          * also 1st order cross derivatives \f$I_{xy}\f$.
610          * \f[
611          * H =
612          * \left(
613          *  \begin{array}{cc}
614          *      I_{xx} & I_{xy} \\
615          *      I_{xy} & I_{yy}
616          *  \end{array}
617          * \right)
618          * \f]
619          * each component of the hessian matrix is fisrt weighted (filtered) by
620          * a weighting kernel (such as gaussian kernel)
621          * \f[
622          * A = \sum_{x} \sum_{y} w(x,y)
623          * \left(
624          *  \begin{array}{cc}
625          *      I_{xx} & I_{xy} \\
626          *      I_{xy} & I_{yy}
627          *  \end{array}
628          * \right)
629          * =
630          * \left(
```



```
631          *  \begin{array}{cc}
632          *      \langle I_{xx}\rangle & \langle I_{xy}\rangle \\
633          *      \langle I_{xy}\rangle & \langle I_{yy}\rangle
634          *  \end{array}
635          * \right)
636          * \f]
637          * The Harris cornerness measure is then computed by :
638          * \f$det(A) - \kappa \cdot trace(A)^2\f$ with \f$\kappa \in
639          * \left[0.04 \cdots 0.15\right]\f$
640          * which is equivalent to \f$I_{xx}I_{yy} - I_{xy}^{2} + \kappa
641          * \left(I_{xx} + I_{xy}\right)^2\f$
642          *
643          *
644          * An approximate version can be obtained with
645          * \f$det(H) - \kappa \cdot trace(H)^2\f$
646          * @param Ixx Horizontal laplacian
647          * @param Iyy Vertical laplacian
648          * @param Ixy Cross derivatives
649          * @param wKernel weighting kernel (typically a gaussian)
650          * @param Kappa trace Factor
651          * @param dst destination matrix
652          * @note if matrices are CV_64FC1 we should use computeCornerness<double>
653          * @note if matrices are CV_32FC1 we should use computeCornerness<float>
654          */
655         template<typename T>
656         void computeCornerness(const Mat & Ixx,
657                                const Mat & Iyy,
658                                const Mat & Ixy,
659                                const Mat & wKernel,
660                                const double Kappa,
661                                Mat & dst);
662
663         /**
664          * Prints info about min and max value of this matrix
665          * @param m the matrix to investigate
666          */
667         void minMaxInfo(const Mat & m);
668 };
669
670 #endif /* CVGFILTER_H_ */
```

```cpp
/*
 * CvGFilter.cpp
 *
 *  Created on: 26 fÃ©vr. 2012
 *      Author: davidroussel
 */

#include <assert.h>
#include <opencv2/imgproc/imgproc.hpp>

#include "CvGFilter.h"

/*
 * Minimum kernel size: 3
 */
const int CvGFilter::minKernelSize = 3;

/*
 * Maximum kernel size: 15
 */
const int CvGFilter::maxKernelSize = 15;

/*
 * gaussian variance steps
 */
const double CvGFilter::sigmaStep = 0.1;

/*
 * Minimum threshold value for edge map: 0
 */
const int CvGFilter::minThreshold = 0;

/*
 * Maximum threshold value for edge map: 255
 */
const int CvGFilter::maxThreshold = 255;

/*
 * Harris K constant mininal value: 0.04
 */
const double CvGFilter::harrisKappaMin = 0.04;

/*
 * Harris K constant maximal value: 0.15
 */
const double CvGFilter::harrisKappaMax = 0.15;

/*
 * Harris K constant steps: 0.01
 */
const double CvGFilter::harrisKappaStep = 0.01;

/*
 * Gaussian filtering class constructor
 * @param sourceImage
 */
CvGFilter::CvGFilter(Mat * sourceImage) :
    CvProcessor(sourceImage),
    dim(sourceImage->size()),
    procType(CV_64FC1),
    displayType(CV_8UC1),
    inFrameGray(dim, displayType, Scalar(0)),
    kernelSize(7),
    sigma((double) kernelSize/5.0),
    minSigma((double) kernelSize / 20.0),
    maxSigma((double) kernelSize / 2.0),
    sigmaChanged(true),
    thresholdLevel(128),
    harrisKappa(harrisKappaMin),
    gX(1, kernelSize, procType, Scalar(0)),
    gY(kernelSize, 1, procType, Scalar(0)),
    gDx(1, kernelSize, procType, Scalar(0)),
    gDy(kernelSize, 1, procType, Scalar(0)),
    gD2x(1, kernelSize, procType, Scalar(0)),
    gD2y(kernelSize, 1, procType, Scalar(0)),
    gDxy(kernelSize, 1, procType, Scalar(0)),
    g2D(kernelSize, kernelSize, procType, Scalar(0)),
    displayMode(INPUT_IM),
    edgeMode(THRESHOLD),
    blurred(dim, procType, Scalar(0)),
    dX(dim, procType, Scalar(0)),
    dY(dim, procType, Scalar(0)),
    gradientMag(dim, procType, Scalar(0)),
    gradientAngle(dim, procType, Scalar(0)),
    d2X(dim, procType, Scalar(0)),
    d2Y(dim, procType, Scalar(0)),
    laplacian(dim, procType, Scalar(0)),
    dXY(dim, procType, Scalar(0)),
    cornerness(dim, procType, Scalar(0)),
    harris(dim, procType, Scalar(0)),
    blurredDisplay(dim, displayType, Scalar(0)),
    dXDisplay(dim, displayType, Scalar(0)),
    dYDisplay(dim, displayType, Scalar(0)),
    gradientMagDisplay(dim, displayType, Scalar(0)),
    gradientAngleDisplay(dim, displayType, Scalar(0)),
    edgeMap(dim, displayType, Scalar(0)),
    laplacianDisplay(dim, displayType, Scalar(0)),
    cornernessDisplay(dim, displayType, Scalar(0)),
    cannyEdgeMap(dim, displayType, Scalar(0)),
    mixEdge(dim, CV_8UC3, Scalar(0, 0, 0)),
    harrisDisplay(dim, displayType, Scalar(0))

{
    setup(sourceImage, false);

    // Adds named image to additional images map
    addImage("gray", &inFrameGray);
    addImage("blurred", &blurredDisplay);
    addImage("dx", &dXDisplay);
    addImage("dy", &dYDisplay);
    addImage("gradientmag", &gradientMagDisplay);
    addImage("gradientangle", &gradientAngleDisplay);
    addImage("edgemap", &edgeMap);
    addImage("laplacian", &laplacianDisplay);
    addImage("cornerness", &cornernessDisplay);
    addImage("canny", &cannyEdgeMap);
    addImage("mixedges", &mixEdge);
    addImage("harris", &harrisDisplay);
}

/*
 * Gaussian filtering class destructor
 */
CvGFilter::~CvGFilter()
{
    cleanup();
}

/*
 * Setup attributes when source image is changed
 * @param image source Image
 * @param completeSetup is true when used to change source image,
 * and false when used in constructor
 */
void CvGFilter::setup(Mat *image, bool completeSetup)
{
    assert (image != NULL);

    CvProcessor::setup(image, completeSetup);

    if (completeSetup) // complete setup
    {
        dim = sourceImage->size();
        inFrameGray = Mat(dim, displayType, Scalar(0));
        CvGFilter::setKernelSize(7);
        thresholdLevel = 128;
        harrisKappa = harrisKappaMin;
        displayMode = INPUT_IM;
        edgeMode = THRESHOLD;
        blurred = Mat(dim, procType, Scalar(0));
        dX = Mat(dim, procType, Scalar(0));
        dY = Mat(dim, procType, Scalar(0));
        gradientMag = Mat(dim, procType, Scalar(0));
        gradientAngle = Mat(dim, procType, Scalar(0));
        d2X = Mat(dim, procType, Scalar(0));
        d2Y = Mat(dim, procType, Scalar(0));
        laplacian = Mat(dim, procType, Scalar(0));
        dXY = Mat(dim, procType, Scalar(0));
        cornerness = Mat(dim, procType, Scalar(0));
        harris = Mat(dim, procType, Scalar(0));
        blurredDisplay = Mat(dim, displayType, Scalar(0));
        dXDisplay = Mat(dim, displayType, Scalar(0));
        dYDisplay = Mat(dim, displayType, Scalar(0));
        gradientMagDisplay = Mat(dim, displayType, Scalar(0));
        gradientAngleDisplay = Mat(dim, displayType, Scalar(0));
        edgeMap = Mat(dim, displayType, Scalar(0));
        laplacianDisplay = Mat(dim, displayType, Scalar(0));
        cornernessDisplay = Mat(dim, displayType, Scalar(0));
        cannyEdgeMap = Mat(dim, displayType, Scalar(0));
        mixEdge = Mat(dim, CV_8UC3, Scalar(0, 0, 0));
        harrisDisplay = Mat(dim, displayType, Scalar(0));
    }
    else // during constructor only
    {

    }

    // in any cases
    edgeMapComponents.push_back(inFrameGray);
    edgeMapComponents.push_back(edgeMap);
```

```
181     edgeMapComponents.push_back(cannyEdgeMap);
182
183 }
184
185 /*
186  * Cleanup attributes before changing source image or cleaning class
187  * before destruction
188  */
189 void CvGFilter::cleanup()
190 {
191     vector<Mat>::iterator it = edgeMapComponents.begin();
192     for (; it ≠ edgeMapComponents.end(); ++it)
193     {
194         (*it).release();
195     }
196     edgeMapComponents.clear();
197
198     harrisDisplay.release();
199     mixEdge.release();
200     cannyEdgeMap.release();
201     cornernessDisplay.release();
202     laplacianDisplay.release();
203     edgeMap.release();
204     gradientAngleDisplay.release();
205     gradientMagDisplay.release();
206     dYDisplay.release();
207     dXDisplay.release();
208     blurredDisplay.release();
209     harris.release();
210     cornerness.release();
211     dXY.release();
212     laplacian.release();
213     d2Y.release();
214     d2X.release();
215     gradientAngle.release();
216     gradientMag.release();
217     dY.release();
218     dX.release();
219     blurred.release();
220     inFrameGray.release();
221 }
222
223 /*
224  * Get current kernel size
225  * @return the current kernel size
226  */
227 int CvGFilter::getKernelSize() const
228 {
229     return kernelSize;
230 }
231
232 /*
233  * Sets the a new kernel size
234  * @param kernelSize the new kernel size
235  * @post if new size is in range [3..15]
236  * with a step of 2:
237  *  - the new kernel size is set up, and remains unchanged otherwise.
238  *  - gaussian kernels are eventually recomputed
239  */
240 void CvGFilter::setKernelSize(int kernelSize)
241 {
242     if (this→kernelSize ≠ kernelSize)
243     {
244         g2D.release();
245         gDxy.release();
246         gD2y.release();
247         gD2x.release();
248         gDy.release();
249         gDx.release();
250         gY.release();
251         gX.release();
252
253         // Kernel size should be odd
254         if (((kernelSize - 1)%2) ≠ 0)
255         {
256             kernelSize++;
257         }
258
259         if (kernelSize > maxKernelSize)
260         {
261             this→kernelSize = maxKernelSize;
262         }
263         else if (kernelSize < minKernelSize)
264         {
265             this→kernelSize = minKernelSize;
266         }
267         else
268         {
269             this→kernelSize = kernelSize;
270         }
```



```
271
272         sigma = (double) kernelSize/5.0;
273         minSigma = (double) kernelSize / 20.0;
274         maxSigma = (double) kernelSize / 2.0;
275         sigmaChanged = true;
276
277         gX = Mat(1, this→kernelSize, procType, Scalar(0));
278         gY = Mat(this→kernelSize, 1, procType, Scalar(0));
279         gDx = Mat(1, this→kernelSize, procType, Scalar(0));
280         gDy = Mat(this→kernelSize, 1, procType, Scalar(0));
281         gD2x = Mat(1, this→kernelSize, procType, Scalar(0));
282         gD2y = Mat(this→kernelSize, 1, procType, Scalar(0));
283         gDxy = Mat(this→kernelSize, 1, procType, Scalar(0));
284         g2D = Mat(this→kernelSize, this→kernelSize, procType, Scalar(0));
285
286         sigmaChanged = true;
287     }
288 }
289
290 /*
291  * Gets maximum kernel size
292  * @return the maximum kernel size
293  */
294 int CvGFilter::getMaxKernelSize()
295 {
296     return maxKernelSize;
297 }
298
299 /*
300  * Gets minimum kernel size
301  * @return the minimum kernel size
302  */
303 int CvGFilter::getMinKernelSize()
304 {
305     return minKernelSize;
306 }
307
308 /*
309  * Gets the current value of gaussian variance
310  * @return the current value of gaussian variance
311  */
312 double CvGFilter::getSigma() const
313 {
314     return sigma;
315 }
316
317 /*
318  * Sets a new value for gaussian variance
319  * @param sigma the new value of gaussian variance
320  */
321 void CvGFilter::setSigma(double sigma)
322 {
323     if (sigma < minSigma)
324     {
325         this→sigma = minSigma;
326     }
327     else if (sigma > maxSigma)
328     {
329         this→sigma = maxSigma;
330     }
331     else
332     {
333         this→sigma = sigma;
334     }
335     sigmaChanged = true;
336 }
337
338 /*
339  * Gets the minimum possible value of gaussian variance
340  * according to kernel size
341  * @return the minimum gaussian variance
342  */
343 double CvGFilter::getMinSigma() const
344 {
345     return minSigma;
346 }
347
348 /*
349  * Gets the maximum possible value of gaussian variance
350  * according to kernel size
351  * @return the maximum gaussian variance
352  */
353 double CvGFilter::getMaxSigma() const
354 {
355     return maxSigma;
356 }
357
358 /*
359  * Gets the steps for sigma increments
360  * @return steps for sigma increments
```

```
 */
double CvGFilter::getSigmaStep()
{
    return sigmaStep;
}

/*
 * Gets the sigma changed status, in order to recompute
 * gaussian kernels (if needed)
 * @return true if sigma is different from last update,
 * false otherwise
 */
bool CvGFilter::isSigmaChanged() const
{
    return sigmaChanged;
}

/*
 * Gets the current threshold level of edgemap
 * @return the current threshold level
 */
int CvGFilter::getThresholdLevel() const
{
    return thresholdLevel;
}

/*
 * Sets new threshold level for edge map
 * @param thresholdLevel the new threshold level
 */
void CvGFilter::setThresholdLevel(int thresholdLevel)
{
    if (thresholdLevel < minThreshold)
    {
        this→thresholdLevel = minThreshold;
    }
    else if (thresholdLevel > maxThreshold)
    {
        this→thresholdLevel = maxThreshold;
    }
    else
    {
        this→thresholdLevel = thresholdLevel;
    }
}

/*
 * Gets minimum threshold value for edgemap
 * @return the minimum threshold value for edgemap
 */
int CvGFilter::getMinThreshold()
{
    return minThreshold;
}

/*
 * Gets maximum threshold value for edgemap
 * @return the maximum threshold value for edgemap
 */
int CvGFilter::getMaxThreshold()
{
    return maxThreshold;
}

/*
 * Gets the current Harris parameter Kappa
 * @return the current value of Kappa
 */
double CvGFilter::getHarrisKappa() const
{
    return harrisKappa;
}

/*
 * Sets new Harris parameter Kappa
 * @param harrisKappa the new parameter to set
 */
void CvGFilter::setHarrisKappa(double harrisKappa)
{
    if (harrisKappa > harrisKappaMax)
    {
        this→harrisKappa = harrisKappaMax;
    }
    else if (harrisKappa < harrisKappaMin)
    {
        this→harrisKappa = harrisKappaMin;
    }
    else
    {
        this→harrisKappa = harrisKappa;
```



```
    }
}

/*
 * Gets maximum Harris parameter Kappa
 * @return the maximum Harris parameter Kappa
 */
double CvGFilter::getHarrisKappaMax()
{
    return harrisKappaMax;
}

/*
 * Gets minimum Harris parameter Kappa
 * @return the minimum Harris parameter Kappa
 */
double CvGFilter::getHarrisKappaMin()
{
    return harrisKappaMin;
}

/*
 * Gets Harris parameter Kappa increment
 * @return the Harris parameter Kappa increment
 */
double CvGFilter::getHarrisKappaStep()
{
    return harrisKappaStep;
}

/*
 * Get current display mode
 * @return the current display mode
 */
CvGFilter::ImageDisplay CvGFilter::getDisplayMode() const
{
    return displayMode;
}

void CvGFilter::setDisplayMode(const ImageDisplay displayMode)
{
    if ((displayMode ≥ INPUT_IM) ∧ (displayMode < NBDISPLAY_IM))
    {
        this→displayMode = displayMode;
    }
    else
    {
        cerr << "display mode out of range: " << displayMode << endl;
    }
}

/*
 * Gets the current edge display mode
 * @return the current edge display mode
 */
CvGFilter::EdgeDisplay CvGFilter::getEdgeMode() const
{
    return edgeMode;
}

/*
 * Set a new edge didsplay mode
 * @param edgeMode the new edge mode
 */
void CvGFilter::setEdgeMode(const EdgeDisplay edgeMode)
{
    if ( (edgeMode ≥ THRESHOLD) ∧ (edgeMode < NBEDGEDISPLAY))
    {
        this→edgeMode = edgeMode;
    }
    else
    {
        cerr << "edge mode out of range: " << edgeMode << endl;
    }
}

/*
 * Gets Image reference corresponding to the current displayMode and
 * edgeMode
 * @return Image reference corresponding to the current displayMode and
 * edgeMode
 */
const Mat & CvGFilter::getDisplayModeImage()
{
    switch (this→displayMode)
    {
        case INPUT_IM:
            return getImage("source");
            break;
        case GRAY_IM:
```

```
541                 return getImage("gray");
542                 break;
543             case BLURRED_IM:
544                 return getImage("blurred");
545                 break;
546             case GRADIENT_X_IM:
547                 return getImage("dx");
548                 break;
549             case GRADIENT_Y_IM:
550                 return getImage("dy");
551                 break;
552             case GRADIENT_MAG_IM:
553                 return getImage("gradientmag");
554                 break;
555             case GRADIENT_ANGLE_IM:
556                 return getImage("gradientangle");
557                 break;
558             case EDGE_MAP_IM:
559                 switch (edgeMode)
560                 {
561                     case THRESHOLD:
562                         return getImage("edgemap");
563                         break;
564                     case CANNY:
565                         return getImage("canny");
566                         break;
567                     case MERGED:
568                     case NBEDGEDISPLAY:
569                     default:
570                         return getImage("mixedges");
571                         break;
572                 }
573                 break;
574             case LAPLACIAN_IM:
575                 return getImage("laplacian");
576                 break;
577             case CORNERNESS_IM:
578                 return getImage("cornerness");
579                 break;
580             case HARRISCORNER_IM:
581                 return getImage("harris");
582                 break;
583             case NBDISPLAY_IM:
584             default:
585                 break;
586         }
587
588         // by default return source
589         return getImage("source");
590     }
591
592     /*
593      * Gets Image pointer corresponding to the current displayMode and
594      * edgeMode
595      * @return Image reference corresponding to the current displayMode and
596      * edgeMode
597      */
598     Mat * CvGFilter::getDisplayModeImagePtr()
599     {
600         switch (this→displayMode)
601         {
602             case INPUT_IM:
603                 return getImagePtr("source");
604                 break;
605             case GRAY_IM:
606                 return getImagePtr("gray");
607                 break;
608             case BLURRED_IM:
609                 return getImagePtr("blurred");
610                 break;
611             case GRADIENT_X_IM:
612                 return getImagePtr("dx");
613                 break;
614             case GRADIENT_Y_IM:
615                 return getImagePtr("dy");
616                 break;
617             case GRADIENT_MAG_IM:
618                 return getImagePtr("gradientmag");
619                 break;
620             case GRADIENT_ANGLE_IM:
621                 return getImagePtr("gradientangle");
622                 break;
623             case EDGE_MAP_IM:
624                 switch (edgeMode)
625                 {
626                     case THRESHOLD:
627                         return getImagePtr("edgemap");
628                         break;
629                     case CANNY:
630                         return getImagePtr("canny");
```



```
631                         break;
632                 case MERGED:
633                 case NBEDGEDISPLAY:
634                 default:
635                     return getImagePtr("mixedges");
636                     break;
637             }
638             break;
639         case LAPLACIAN_IM:
640             return getImagePtr("laplacian");
641             break;
642         case CORNERNESS_IM:
643             return getImagePtr("cornerness");
644             break;
645         case HARRISCORNER_IM:
646             return getImagePtr("harris");
647             break;
648         case NBDISPLAY_IM:
649         default:
650             break;
651     }
652     // by default return source
653     return getImagePtr("source");
654
655 }
656
657 /*
658  * Gaussian filtering update
659  *  - convert source image to gray
660  *  - if sigma changed recompute gaussian kernels
661  *  - compute blurred image and convert it for display
662  *  - compute horizontal and vertical gradients and convert them
663  *  for display
664  *  - compute gradient magnitude and angle and convert them for display
665  *  - threshold gradient magintude to edgeMap
666  *  - compute horizontal and vertical laplacian components and
667  *  laplacian image
668  *  - compute dXY to prepare cornerness measure
669  */
670 void CvGFilter::update()
671 {
672     // ----------------------------------------------------------------------
673     // convert image to gray
674     // ----------------------------------------------------------------------
675     cvtColor(*sourceImage, inFrameGray, CV_BGR2GRAY);
676
677     // ----------------------------------------------------------------------
678     // recompute filters kernels if sigma has changed
679     // ----------------------------------------------------------------------
680     if (sigmaChanged)
681     {
682         // Gaussian 1D horizontal filter
683         // TODO gaussian<double> (gX, ...);
684
685         // Gaussian 1D vertical filter
686         // TODO gaussian<double> (gY, ...);
687
688         // Gaussian horizontal 1st derivative filter: dx
689         // TODO gaussian<double> (gDx, ...);
690
691         // Gaussian vertical 1st derivative filter: dy
692         // TODO gaussian<double> (gDy, sigma, 0, 1);
693
694         // Gaussian horizontal and vertical 1st derivative filter: dxy
695         // TODO gaussian<double> (gDxy, ...);
696
697         // 2D Gaussian kernel for Ixx,Iyy and Ixy smoothing in cornerness
698         // with 2*sigma+1 sigma
699         // TODO gaussian<double> (g2D, ...);
700
701         sigmaChanged = false;
702     }
703
704     // ----------------------------------------------------------------------
705     // Compute blurred image
706     // ----------------------------------------------------------------------
707     if (displayMode ≡ BLURRED_IM ∨
708         displayMode ≡ EDGE_MAP_IM ∨
709         displayMode ≡ HARRISCORNER_IM)
710     {
711         // Compute gaussian blurred image inFrameGray --> blurred
712         // TODO sepFilter2D(...);
713
714         // Convert blurred image for display or other purpose
715         convertScaleAbs(blurred, blurredDisplay);
716     }
717
718     // ----------------------------------------------------------------------
719     // Compute gradients
720     // ----------------------------------------------------------------------
```

```
721     if (displayMode ≥ GRADIENT_X_IM ∧ displayMode ≤LAPLACIAN_IM)
722     {
723         // Compute horizontal gradient: inFrameGray --> dX
724         // TODO sepFilter2D(...);
725 
726         // Compute vertical gradient: inFrameGray --> dY
727         // TODO sepFilter2D(...);
728     }
729 
730     // -----------------------------------------------------------------------
731     // Converts horizontal gradient for display
732     // -----------------------------------------------------------------------
733     if (displayMode ≡ GRADIENT_X_IM)
734     {
735         // convert horizontal gradient for display
736         dX.convertTo(dXDisplay,dXDisplay.type(),0.5,128);
737     }
738 
739     // -----------------------------------------------------------------------
740     // Converts vertical gradient for display
741     // -----------------------------------------------------------------------
742     if (displayMode ≡ GRADIENT_Y_IM)
743     {
744         // convert vertical gradient for display
745         dY.convertTo(dYDisplay,dYDisplay.type(),0.5,128);
746     }
747 
748     // -----------------------------------------------------------------------
749     // Compute gradient magnitude and angle
750     // -----------------------------------------------------------------------
751     if (displayMode ≡ GRADIENT_MAG_IM ∨
752         displayMode ≡ GRADIENT_ANGLE_IM ∨
753         (displayMode ≡ EDGE_MAP_IM ∧ edgeMode ≠ CANNY))
754     {
755         // Compute gradient magnitude and angle with cartToPolar
756         // dX, dY --> gradientMag, gradientAngle (angle in degrees)
757         // TODO cartToPolar(...);
758     }
759 
760     // -----------------------------------------------------------------------
761     // Converts gradient magnitude for display
762     // -----------------------------------------------------------------------
763     if (displayMode ≡ GRADIENT_MAG_IM ∨
764         (displayMode ≡ EDGE_MAP_IM ∧ edgeMode ≠ CANNY))
765     {
766         // convert magnitude for display
767         convertScaleAbs(gradientMag, gradientMagDisplay);
768     }
769 
770     // -----------------------------------------------------------------------
771     // Converts gradient angle for display
772     // -----------------------------------------------------------------------
773     if (displayMode ≡ GRADIENT_ANGLE_IM)
774     {
775         // convert angle for display
776         convertScaleAbs(gradientAngle, gradientAngleDisplay);
777     }
778 
779     // -----------------------------------------------------------------------
780     // Threshold display gradient magnitude to get edge map
781     // -----------------------------------------------------------------------
782     if (displayMode ≡ EDGE_MAP_IM ∧ edgeMode ≠ CANNY)
783     {
784         // threshold gradient magnitude for edge map at thresholdLevel
785         // gradientMagDisplay --> edgeMap
786         // TODO threshold(...);
787     }
788 
789     // -----------------------------------------------------------------------
790     // Compute canny edges from burred image
791     // -----------------------------------------------------------------------
792     if ( displayMode ≡ EDGE_MAP_IM ∧
793         ((edgeMode ≡ CANNY) ∨ (edgeMode ≡ MERGED)) )
794     {
795         // Compute Canny Edges from blurred image
796         // blurredDisplay --> cannyEdgeMap with 1st threshold= thresholdLevel
797         // and second threshold = thresholdLevel / 2 or 3
798         // Caution : sobel aperture should not be bigger than 7 so use
799         // MIN(7, kernelSize) for sobel aperture
800         // Use L2 norm rather than L1
801         // TODO Canny(...,
802         //      ...,
803         //      ....        // first threshold for histeresis
804         //      ....    // second threshold for histeresis
805         //      ...,    // sobel aperture (1, 3, 5, 7)
806         //      ...);               // slower L2 norm
807     }
808 
809     // -----------------------------------------------------------------------
810     // Compute :
```



```
811     //  - Horizontal and vertical laplacian components
812     //  - laplacian
813     // -----------------------------------------------------------------------
814     if (displayMode ≡  LAPLACIAN_IM ∨
815         displayMode ≡ CORNERNESS_IM)
816     {
817         // Compute Laplacian X component by computing x gradient on dX (already a gradient)
818         // dX --> d2X
819         // TODO Ã  complÃ©ter ...
820 
821         // Compute Laplacian Y component by computing y gradient on dY (already a gradient)
822         // dY -->d2Y
823         // TODO Ã  complÃ©ter ...
824 
825         // Compute Laplacian
826         // d2X + d2Y --> laplacian;
827         // TODO Ã  complÃ©ter ...
828     }
829 
830     // -----------------------------------------------------------------------
831     // Converts laplacian for display
832     // -----------------------------------------------------------------------
833     if (displayMode ≡ LAPLACIAN_IM)
834     {
835         // Convert laplacian for display
836         laplacian.convertTo(laplacianDisplay, laplacianDisplay.type(), 0.5,
837                 128);
838     }
839 
840     // -----------------------------------------------------------------------
841     // Compute cornerness image and converts it for display
842     // -----------------------------------------------------------------------
843     if (displayMode ≡ CORNERNESS_IM)
844     {
845         /*
846          * Compute Cornerness measure from Hessian matrix
847          *  H = | d2X dXY |
848          *      | dXY d2Y |
849          *  det(H) - k Trace(H) with k in [0.04 ... 0.15]
850          *    = d2X * d2Y - dXY^2 - k (d2X + d2Y)^2
851          */
852         // Compute dXY: inFrameGray -> dXY cross derivative image
853         // TODO sepFilter2D(...);
854 
855         // compute cornerness measure
856         // TODO ComplÃ©ter la mÃ©thode computeCornerness
857         computeCornerness<double>(d2X,d2Y,dXY,g2D,harrisKappa,cornerness);
858 
859         // Cornerness values are unknown yet so take a look:
860         // minMaxInfo(cornerness);
861 
862         // convert cornerness for display
863         normalize(cornerness, cornernessDisplay, 0, 255, NORM_MINMAX,
864                 cornernessDisplay.type());
865     }
866 
867     // -----------------------------------------------------------------------
868     // merge edges map components into a color image
869     // -----------------------------------------------------------------------
870     if (displayMode ≡ EDGE_MAP_IM ∧ edgeMode ≡ MERGED)
871     {
872         // merge edgeMap, cannyEdgeMap and gray component into mixEdge color image
873         merge(edgeMapComponents, mixEdge);
874     }
875 
876 
877     // -----------------------------------------------------------------------
878     // Compute Harris cornerness image with harris function and convert it
879     // for display
880     // -----------------------------------------------------------------------
881     if (displayMode ≡ HARRISCORNER_IM)
882     {
883         // Compute Harris corners from blurred image
884         // blurredDisplay --> harris
885         // with kernelSize neighborhood
886         // use MIN(7,kernelSize) for sobel aperture
887         // harrisKappa for kappa
888         //TODO cornerHarris(...);
889 
890         // Harris corner measures are unknown so take a look
891         // minMaxInfo(harris);
892 
893         // Convert harris measure for display
894         normalize(harris, harrisDisplay, 0, 255, NORM_MINMAX, harrisDisplay.type());
895     }
896 }
897 
898 /*
899  * Compute 1D or 2D normalized gaussian into kernel with sigma variance and
900  * @param kernel matrix to store gaussian kernel:
```

```
901  *  - if kernel size is \f$ 1 \times N\f$ or \f$ N \times 1\f$ produces a 1D
902  *  line or column filter.
903  *  - if kernel size is \f$ M \times N\f$ produces a 2D gaussian filter.
904  * @param sigma standard deviation \f$\sigma\f$ of the gauss curve (or surface)
905  * expressed in pixels
906  * @param derivOrderX horizontal derivative order. Should be 0, 1 or 2:
907  * @param derivOrderY vertical derivative order. Should be 0, 1 or 2
908  */
909 template <typename T>
910 void CvGFilter::gaussian(Mat & kernel,
911                          const double sigma,
912                          const unsigned int derivOrderX,
913                          const unsigned int derivOrderY)
914 {
915     // clog << "kernel to compute : [" << kernel.rows << "x" << kernel.cols
916     //      << "] = " << kernel << endl;
917
918     if ((kernel.rows > 0) ∧ (kernel.cols > 0))
919     {
920         // x center point
921         double x0 = floor(kernel.cols/2.0)+1.0;
922         // y center point
923         double y0 = floor(kernel.rows/2.0)+1.0;
924
925         // clog << "kernel center is [" << y0 << ", " << x0 << "]" << endl;
926
927         double sum = 0.0;
928
929         if (sigma > 0.0)
930         {
931             double sigmaFactor = 2.0 * sigma * sigma;
932
933             // clog << "Sigma factor = " << sigmaFactor << endl;
934
935             // Compute gaussian values ----------------------------------------
936             for (int i = 0; i < kernel.rows; i++)
937             {
938                 // yterms = (y - y0)^2 / sigmaFactor
939                 double yterms = double(i+1) - y0;
940                 yterms *= yterms;
941                 yterms /= sigmaFactor;
942                 T yExp = (T) exp(-yterms);
943                 for (int j = 0; j < kernel.cols; j++)
944                 {
945                     // xterms = (x - x0)^2 / sigmaFactor
946                     double xterms = double(j+1) - x0;
947                     xterms *= xterms;
948                     xterms /= sigmaFactor;
949                     T xExp = (T) exp(-xterms);
950
951                     // FIXME kernel.at<T> (i, j) = (T)(exp(-(xterms + yterms)));
952                     kernel.at<T> (i, j) = xExp * yExp;
953
954                     // clog << "g(" << i << ", " << j << ") = "
955                     //      << kernel.at<T> (i, j) << " = " << " exp(-(" << xterms
956                     //      << " + " << yterms << "))" << endl;
957
958                 }
959             }
960
961             // clog << "gaussian[" << kernel.rows << "x" << kernel.cols << "] sum = "
962             //      << sum << endl;
963
964             // Compute derivatives if any ------------------------------------
965             double sigma2 = sigma * sigma;
966             double sigma4 = sigma2 * sigma2;
967
968             if (derivOrderX > 0)
969             {
970                 switch (derivOrderX)
971                 {
972                     case 1:
973                         for (int j = 0;  j < kernel.cols; j++)
974                         {
975                             /*
976                              * dg(x,y)/dx = (-(x-x0) / sigma^2) * g(x,y)
977                              */
978                             double dFactor = -((double)(j+1) - x0) / sigma2;
979
980                             for (int i=0; i < kernel.rows; i++)
981                             {
982                                 kernel.at<T> (i, j) *= dFactor;
983                                 // clog << "dg(" << i << ", " << j << ")/dj = "
984                                 //      << kernel.at<T> (i, j) << endl;
985                             }
986                         }
987                         break;
988                     case 2:
989                         for (int j = 0;  j < kernel.cols; j++)
990                         {
```



```
991                             /*
992                              * d^2g(x,y)/dx^2 = ((x-x0)^2 - sigma^2 / sigma^4)
993                              *                  * g(x,y)
994                              */
995                             double dFactor = (pow(((double) (j + 1) - x0), 2.0)
996                                     - sigma2) / sigma4;
997
998                             for (int i=0; i < kernel.rows; i++)
999                             {
1000                                kernel.at<T> (i, j) *= dFactor;
1001                                // clog << "d^2g(" << i << ", " << j << ")/dj^2 = "
1002                                //      << kernel.at<T> (i, j) << endl;
1003                            }
1004                        }
1005                        break;
1006                    default:
1007                        cerr << "gaussian deriv order X should be 0, 1 or 2 : "
1008                             << derivOrderX << endl;
1009                        break;
1010                }
1011            }
1012
1013            if (derivOrderY > 0)
1014            {
1015                switch (derivOrderY)
1016                {
1017                    case 1:
1018                        for (int i = 0;  i < kernel.rows; i++)
1019                        {
1020                            /*
1021                             * dg(x,y)/dy = (-(y-y0) / sigma^2) * g(x,y)
1022                             */
1023                            double dFactor = -((double)(i+1) - y0) / sigma2;
1024
1025                            for (int j=0; j < kernel.cols; j++)
1026                            {
1027                                kernel.at<T> (i, j) *= dFactor;
1028                                // clog << "dg(" << i << ", " << j << ")/di = "
1029                                //      << kernel.at<T> (i, j) << endl;
1030                            }
1031                        }
1032                        break;
1033                    case 2:
1034                        for (int i = 0;  i < kernel.rows; i++)
1035                        {
1036                            /*
1037                             * d^2g(x,y)/dv^2 = ((v-y0)^2 - sigma^2 / sigma^4)
1038                             *                  * g(x,y)
1039                             */
1040                            double dFactor = (pow(((double) (i + 1) - y0), 2.0)
1041                                    - sigma2) / sigma4;
1042
1043                            for (int j=0; j < kernel.cols; j++)
1044                            {
1045                                kernel.at<T> (i, j) *= dFactor;
1046                                // clog << "d^2g(" << i << ", " << j << ")/di^2 = "
1047                                //      << kernel.at<T> (i, j) << endl;
1048                            }
1049                        }
1050                        break;
1051                    default:
1052                        cerr << "gaussian deriv order Y should be 0, 1 or 2 : "
1053                             << derivOrderY << endl;
1054                        break;
1055                }
1056            }
1057
1058            // Normalize values ----------------------------------------------
1059            // compute filter gain on first half (or quarter)
1060            sum = 0.0;
1061            double quarterSum = 0.0;
1062            double borderXSum = 0.0;
1063            double borderYSum = 0.0;
1064            int centerI = (int)y0;
1065            int centerJ = (int)x0;
1066            // quarter sum
1067            for (int i = 0; i < centerI-1; i++)
1068            {
1069                for (int j = 0; j < centerJ-1; j++)
1070                {
1071                    quarterSum +=kernel.at<T> (i, j);
1072                }
1073            }
1074            // border X sum
1075            for (int j=0; j < centerJ-1; j++)
1076            {
1077                borderXSum+=kernel.at<T>(centerI-1,j);
1078            }
1079            // border Y sum
1080            for (int i = 0; i < centerI-1; ++i)
```

```
1081                {
1082                    borderYSum+=kernel.at<T>(i,centerJ-1);
1083                }
1084                // Adds center Value
1085                sum += kernel.at<T>(centerI-1,centerJ-1);
1086
1087                // The compute the whole sum =
1088                //  - quarterSum * 4 + borderXSum*2 + boderYSum*2
1089                //    + center (already added in sum)
1090                sum += borderXSum*2.0 + borderYSum*2.0 + quarterSum*4.0;
1091
1092                // Normalize values
1093                for (int i = 0; i < kernel.rows; i++)
1094                {
1095                    for (int j = 0; j < kernel.cols; j++)
1096                    {
1097                        kernel.at<T> (i, j) /= sum;
1098                    }
1099                }
1100            }
1101            else
1102            {
1103                cerr << "gaussian sigma is <= 0 --> no output" << endl;
1104            }
1105        }
1106        else
1107        {
1108            cerr << "gaussian kernel rows or cols is 0 --> no output" << endl;
1109        }
1110    }
1111
1112    /*
1113     * Harris corenerness measure
1114     * det(H) - kappa trace(H)^2
1115     * = IxxIyy - Ixy^2 - kappa (Ixx + Iyy)^2
1116     */
1117    template <typename T>
1118    void CvGFilter::computeCornerness(const Mat & Ixx,
1119                                      const Mat & Iyy,
1120                                      const Mat & Ixy,
1121                                      const Mat & wKernel,
1122                                      const double Kappa,
1123                                      Mat & dst)
1124    {
1125        // Local smoothed Ixx, Iyy and Iyy
1126        Mat IxxW(Ixx.size(), Ixx.type(), Scalar(0));
1127        Mat IyyW(Iyy.size(), Iyy.type(), Scalar(0));
1128        Mat IxyW(Ixy.size(), Ixy.type(), Scalar(0));
1129
1130        // Smooth Ixx, Iyy and Ixy with wKernel using filter2D
1131        // Ixx --> IxxW with wKernel
1132        // TODO filter2D(...);
1133        // Iyy --> IyyW with wKernel
1134        // TODO filter2D(...);
1135        // Ixy --> IxyW with wKernel
1136        // TODO filter2D(...);
1137
1138        // initiate iterators on IxxW, IyyW, IxyW and dst
1139        MatConstIterator_<T> xxIt = IxxW.begin<T>();
1140        MatConstIterator_<T> xxItEnd = IxxW.end<T>();
1141        MatConstIterator_<T> yyIt = IyyW.begin<T>();
1142        MatConstIterator_<T> xyIt = IxyW.begin<T>();
1143        MatIterator_<T> destIt = dst.begin<T>();
1144
1145        // intialize variables of the equation
1146        T IxxIyy = 0.0;
1147        T Ixy2 = 0.0;
1148        T IxxSumIyy = 0.0;
1149
1150        // Compute cornerness value on each pixel
1151        for(; xxIt ≠ xxItEnd; ++xxIt, ++yyIt, ++xyIt, ++destIt)
1152        {
1153            /*
1154             *      / *xxIt  *xyIt \
1155             * H = |               |
1156             *      \ *xyIt  *yyIt /
1157             *
1158             * *destIt = det(H) - Kappa x trace(H)^2
1159             */
1160
1161            // TODO (*destIt) = ...
1162        }
1163
1164        IxxW.release();
1165        IyyW.release();
1166        IxyW.release();
1167    }
1168
1169    /*
1170     * Prints info about min and max value of this matrix
```



```
1171     * @param m the matrix to investigate
1172     */
1173    void CvGFilter::minMaxInfo(const Mat & m)
1174    {
1175        double minVal, maxVal;
1176        Point minLoc;
1177        Point maxLoc;
1178        minMaxLoc(m,&minVal, &maxVal, &minLoc, &maxLoc);
1179        clog << "values : min = " << minVal << " at (" << minLoc.x
1180             << ", " << minLoc.y << ") max = " << maxVal << " at ("
1181             << maxLoc.x << ", " << maxLoc.y << ")" << endl;
1182    }
```

```
/*
 * QcvGFilter.h
 *
 *  Created on: 27 fÃ©vr. 2012
 *      Author: davidroussel
 */

#ifndef QCVGFILTER_H_
#define QCVGFILTER_H_

#include "QcvProcessor.h"
#include "CvGFilter.h"

/**
 * QT flavored class to process source image with gaussian filters
 */
class QcvGFilter: public QcvProcessor, public CvGFilter
{
    Q_OBJECT

    protected:
        /**
         * Self lock for operations from multiple threads
         * @note may be NULL ift there is no update thread
         */
        QMutex * selfLock;

    public:

        /**
         * QcvGFilter constructor
         * @param image the source image
         * @param imageLock the mutex on source image
         * @param updateThread the thread in which this processor runs
         * @param parent parent QObject
         */
        QcvGFilter(Mat * image,
                   QMutex * imageLock = NULL,
                   QThread * updateThread = NULL,
                   QObject * parent = NULL);

        /**
         * QcvGFilter destructor
         */
        virtual ~QcvGFilter();

        // ------------------------------------------------------------------
        // Options settings with message notification
        // ------------------------------------------------------------------

    public slots:
        /**
         * Update computed images slot and sends updated signal
         * required
         */
        void update();

        /**
         * Changes source image slot.
         * Attributes needs to be cleaned up then set up again
         * @param image the new source Image
         */
        void setSourceImage(Mat * image)
            throw (CvProcessorException);

        /**
         * Sets the a new kernel size
         * @param kernelSize the new kernel size
         * @post if new size is in range [3..15]
         * with a step of 2:
         *  - the new kernel size is set up, and remains unchanged otherwise.
         *  - gaussian kernels are eventually recomputed
         */
        void setKernelSize(int kernelSize);

        /**
         * Sets a new value for gaussian variance
         * @param sigma the new value of gaussian variance
         */
        void setSigma(double sigma);

        /**
         * Sets new threshold level for edge map
         * @param thresholdLevel the new threshold level
         */
        void setThresholdLevel(int thresholdLevel);

        /**
         * Sets new Harris parameter Kappa
         * @param harrisKappa the new parameter to set
```



```
         */
        void setHarrisKappa(double harrisKappa);

        /**
         * Sets a new display mode and emit corresponding imageChanged(Mat*)
         * signal
         * @param displayMode the new display mode to set
         */
        void setDisplayMode(const ImageDisplay displayMode);

        /**
         * Set a new edge didsplay mode and emit corresponding
         * imageChanged(Mat*) signal if needed
         * @param edgeMode the new edge mode
         * @param standalone if setEdgeMode is used by itself then true, but
         * false when used inside setDisplayMode
         */
        void setEdgeMode(const EdgeDisplay edgeMode, const bool standalone = true);

    signals:
        /**
         * Signal emitted when kernelSize changed because sigma values (min,
         * max, step & value) are changed by new kernel size values
         * @param kernelSize the new kernelSize
         */
        void kernelSizeChanged(int kernelSize);

};

#endif /* QCVGFILTER_H_ */
```

```cpp
/*
 * QcvGFilter.cpp
 *
 *  Created on: 27 fÃ©vr. 2012
 *      Author: davidroussel
 */

#include "QcvGFilter.h"


/*
 * QcvGFilter constructor
 * @param image the source image
 * @param imageLock the mutex on source image
 * @param updateThread the thread in which this processor runs
 * @param parent parent QObject
 */
QcvGFilter::QcvGFilter(Mat *image,
                       QMutex * imageLock,
                       QThread * updateThread,
                       QObject *parent) :
    CvProcessor(image), // <-- virtual base class constructor first
    QcvProcessor(image, imageLock, updateThread, parent),
    CvGFilter(image),
    selfLock(updateThread ≠ NULL ? new QMutex() :
                                   (imageLock ≠ NULL ? imageLock : NULL))
{
}

/*
 * QcvGFilter destructor
 */
QcvGFilter::~QcvGFilter()
{
    message.clear();
    if (selfLock ≠ NULL)
    {
        selfLock→lock();
        selfLock→unlock();
        delete selfLock;
    }
}

/*
 * Changes source image slot.
 * Attributes needs to be cleaned up then set up again
 * @param image the new source Image
 */
void QcvGFilter::setSourceImage(Mat *image)
    throw (CvProcessorException)
{
    Size previousSize(sourceImage→size());
    int previousNbChannels(nbChannels);
    bool hasLock = selfLock ≠ NULL;
    if (hasLock)
    {
        selfLock→lock();
    }

    CvProcessor::setSourceImage(image);

    if (hasLock)
    {
        selfLock→unlock();
    }

    emit imageChanged(sourceImage);

    emit imageChanged();

    if ((previousSize.width ≠ image→cols) ∨
        (previousSize.height ≠ image→rows))
    {
        emit imageSizeChanged();
    }

    if (previousNbChannels ≠ nbChannels)
    {
        emit imageColorsChanged();
    }

    // Force update
    // update();
}

/*
 * Sets the a new kernel size
 * @param kernelSize the new kernel size
 * @post if new size is in range [3..15]
 * with a step of 2;
```



```cpp
 *  - the new kernel size is set up, and remains unchanged otherwise.
 *  - gaussian kernels are eventually recomputed
 */
void QcvGFilter::setKernelSize(int kernelSize)
{
    bool hasLock = selfLock ≠ NULL;
    if (hasLock)
    {
        selfLock→lock();
    }

    CvGFilter::setKernelSize(kernelSize);

    if (hasLock)
    {
        selfLock→unlock();
    }
}

/*
 * Sets a new value for gaussian variance
 * @param sigma the new value of gaussian variance
 */
void QcvGFilter::setSigma(double sigma)
{
    bool hasLock = selfLock ≠ NULL;
    if (hasLock)
    {
        selfLock→lock();
    }

    CvGFilter::setSigma(sigma);

    if (hasLock)
    {
        selfLock→unlock();
    }
}

/*
 * Sets new threshold level for edge map
 * @param thresholdLevel the new threshold level
 */
void QcvGFilter::setThresholdLevel(int thresholdLevel)
{
    bool hasLock = selfLock ≠ NULL;
    if (hasLock)
    {
        selfLock→lock();
    }

    CvGFilter::setThresholdLevel(thresholdLevel);

    if (hasLock)
    {
        selfLock→unlock();
    }
}

/*
 * Sets new Harris parameter Kappa
 * @param harrisKappa the new parameter to set
 */
void QcvGFilter::setHarrisKappa(double harrisKappa)
{
    bool hasLock = selfLock ≠ NULL;
    if (hasLock)
    {
        selfLock→lock();
    }

    CvGFilter::setHarrisKappa(harrisKappa);

    if (hasLock)
    {
        selfLock→unlock();
    }
}


/*
 * Sets a new display mode and emit corresponding imageChanged(Mat*)
 * signal
 * @param displayMode the new display mode to set
 */
void QcvGFilter::setDisplayMode(ImageDisplay displayMode)
{
    bool hasLock = selfLock ≠ NULL;
    if (hasLock)
    {
```

```
181         selfLock→lock();
182     }
183 
184     CvGFilter::setDisplayMode(displayMode);
185 
186     if (hasLock)
187     {
188         selfLock→unlock();
189     }
190 
191     message.clear();
192 
193     switch (this→displayMode)
194     {
195         case INPUT_IM:
196             message.append("Source image");
197             emit imageChanged(getImagePtr("source"));
198             break;
199         case GRAY_IM:
200             message.append("Gray converted image");
201             emit imageChanged(getImagePtr("gray"));
202             break;
203         case BLURRED_IM:
204             message.append("Gray converted image, blurred with gaussian kernel");
205             emit imageChanged(getImagePtr("blurred"));
206             break;
207         case GRADIENT_X_IM:
208             message.append("Horizontal gradient");
209             emit imageChanged(getImagePtr("dx"));
210             break;
211         case GRADIENT_Y_IM:
212             message.append("Vertical gradient");
213             emit imageChanged(getImagePtr("dy"));
214             break;
215         case GRADIENT_MAG_IM:
216             message.append("Gradient magnitude");
217             emit imageChanged(getImagePtr("gradientmag"));
218             break;
219         case GRADIENT_ANGLE_IM:
220             message.append("Gradient angle");
221             emit imageChanged(getImagePtr("gradientangle"));
222             break;
223         case EDGE_MAP_IM:
224             // Embeded setEdgeMode
225             setEdgeMode(edgeMode, false);
226             break;
227         case LAPLACIAN_IM:
228             message.append("Laplacian image");
229             emit imageChanged(getImagePtr("laplacian"));
230             break;
231         case CORNERNESS_IM:
232             message.append("Cornerness");
233             emit imageChanged(getImagePtr("cornerness"));
234             break;
235         case HARRISCORNER_IM:
236             message.append("Harris cornerness");
237             emit imageChanged(getImagePtr("harris"));
238             break;
239         case NBDISPLAY_IM:
240         default:
241             break;
242     }
243 
244     emit sendMessage(message, defaultTimeOut);
245 }
246 
247 /*
248  * Set a new edge didsplay mode and emit corresponding
249  * imageChanged(Mat*) signal if needed
250  * @param edgeMode the new edge mode
251  */
252 void QcvGFilter::setEdgeMode(EdgeDisplay edgeMode, const bool standalone)
253 {
254     if (standalone)
255     {
256         bool hasLock = selfLock ≠ NULL;
257         if (hasLock)
258         {
259             selfLock→lock();
260         }
261 
262         CvGFilter::setEdgeMode(edgeMode);
263 
264         if (hasLock)
265         {
266             selfLock→unlock();
267         }
268     }
269 
270     if (displayMode ≡ EDGE_MAP_IM)
```



```
271     {
272         if (standalone)
273         {
274             message.clear();
275         }
276 
277         switch (edgeMode)
278         {
279             case THRESHOLD:
280                 message.append("Egde map by thresholding gradient magnitude");
281                 emit imageChanged(getImagePtr("edgemap"));
282                 break;
283             case CANNY:
284                 message.append("Canny edges");
285                 emit imageChanged(getImagePtr("canny"));
286                 break;
287             case MERGED:
288                 message.append("Mixed edge image");
289                 emit imageChanged(getImagePtr("mixedges"));
290                 break;
291             case NBEDGEDISPLAY:
292             default:
293                 break;
294         }
295 
296         if (standalone)
297         {
298             emit sendMessage(message, defaultTimeOut);
299         }
300     }
301 }
302 
303 
304 /*
305  * Update computed images slot and sends updated signal
306  * required
307  */
308 void QcvGFilter::update()
309 {
310     bool hasSourceLock = (sourceLock ≠ NULL) ∧ (sourceLock ≠ selfLock);
311 
312     if (hasSourceLock)
313     {
314         sourceLock→lock();
315         // qDebug() << "QcvColorSpaces::update : lock";
316     }
317 
318     bool hasLock = selfLock ≠ NULL;
319     if (hasLock)
320     {
321         selfLock→lock();
322     }
323 
324     /*
325      * Update filtered images
326      */
327     CvGFilter::update();
328 
329     if (hasSourceLock)
330     {
331         // qDebug() << "QcvColorSpaces::update : unlock";
332         sourceLock→unlock();
333     }
334 
335     if (hasLock)
336     {
337         selfLock→unlock();
338     }
339 
340     /*
341      * emit updated signal
342      */
343     QcvProcessor::update();
344 }
```

```cpp
/*
 * QcvMatWidget.cpp
 *
 *  Created on: 28 fÃ©vr. 2011
 *      Author: davidroussel
 */
#include <QtDebug>

#include <opencv2/imgproc.hpp>

#include "QcvMatWidget.h"

/*
 * Default size when no image has been set
 */
QSize QcvMatWidget::defaultSize(640, 480);

/*
 * Default aspect ratio when image is not set yet
 */
double QcvMatWidget::defaultAspectRatio = 4.0/3.0;

/*
 * Drawing color
 */
const Scalar QcvMatWidget::drawingColor(0xFF,0xCC,0x00,0x88);

/*
 * Drawing width
 */
const int QcvMatWidget::drawingWidth(3);

/*
 * OpenCV QT Widget default constructor
 * @param parent parent widget
 * @param mouseSense mouse sensivity
 */
QcvMatWidget::QcvMatWidget(QWidget *parent,
                           MouseSense mouseSense) :
    QWidget(parent),
    sourceImage(NULL),
    aspectRatio(defaultAspectRatio),
    mousePressed(false),
    mouseSense(mouseSense),
//  count(0)
    pixelScale(devicePixelRatioF())
{
    setup();
}

/*
 * OpenCV QT Widget constructor
 * @param the source image
 * @param parent parent widget
 * @param mouseSense mouse sensivity
 */
QcvMatWidget::QcvMatWidget(Mat * sourceImage,
                           QWidget *parent,
                           MouseSense mouseSense) :
    QWidget(parent),
    sourceImage(sourceImage),
    aspectRatio((double)sourceImage->cols / (double)sourceImage->rows),
    mousePressed(false),
    mouseSense(mouseSense),
//  count(0)
    pixelScale(devicePixelRatioF())
{
    setup();
}

/*
 * OpenCV Widget destructor.
 * Releases displayImage.
 */
QcvMatWidget::~QcvMatWidget()
{
    displayImage.release();
}

/*
 * paint event reimplemented to draw content (in this case only
 * draw in display image since final rendering method is not yet available)
 * @param event the paint event
 */
void QcvMatWidget::paintEvent(QPaintEvent * event)
{
    Q_UNUSED(event);

    if (displayImage.data != NULL)
    {
```



```cpp
        // evt draw in image
        if (mousePressed)
        {
            // if MOUSE_CLICK only draws a cross
            if (mouseSense > MOUSE_NONE)
            {
                if (¬(mouseSense & MOUSE_DRAG))
                {
                    if (mouseMoved)
                    {
                        drawCross(draggedPoint);
                    }
                    else
                    {
                        drawCross(pressedPoint);
                    }
                }
                else  // else if MOUSE_DRAG starts drawing a rectangle
                {
                    drawRectangle(selectionRect);
                }
            }
        }
    }
    else
    {
        qWarning("QcvMatWidget::paintEvent : image.data is NULL");
    }
}

/*
 * Widget setup
 */
void QcvMatWidget::setup()
{
    layout = new QHBoxLayout();
    layout->setContentsMargins(0,0,0,0);
    setLayout(layout);
}

/*
 * Sets new source image
 * @param sourceImage the new source image
 */
void QcvMatWidget::setSourceImage(Mat * sourceImage)
{
    // qDebug("QcvMatWidget::setSourceImage");

    this->sourceImage = sourceImage;

    // re-setup geometry since height x width may have changed
    aspectRatio = (double)sourceImage->cols / (double)sourceImage->rows;
    // qDebug ("aspect ratio changed to %4.2f", aspectRatio);

}

/*
 * Converts BGR or Gray source image to RGB display image
 * @see #sourceImage
 * @see #displayImage
 */
void QcvMatWidget::convertImage()
{
//  qDebug("Convert image");

    int depth = sourceImage->depth();
    int channels = sourceImage->channels();

    // Converts any image type to RGB format
    switch (depth)
    {
        case CV_8U:
            switch (channels)
            {
                case 1: // gray level image
                    cvtColor(*sourceImage, displayImage,CV_GRAY2RGB);
                    break;
                case 3: // Color image (OpenCV produces BGR images)
                    cvtColor(*sourceImage, displayImage, CV_BGR2RGB);
                    break;
                default:
                    qFatal("This number of channels (%d) is not supported",
                           channels);
                    break;
            }
            break;
        default:
            qFatal("This image depth (%d) is not implemented in QcvMatWidget",
                   depth);
            break;
```

```
    }
}

/*
 * Callback called when mouse button pressed event occurs.
 * reimplemented to send pressPoint signal when left mouse button is
 * pressed
 * @param event mouse event
 */
void QcvMatWidget::mousePressEvent(QMouseEvent *event)
{
    if (mouseSense > MOUSE_NONE)
    {
//      qDebug("mousePressEvent(%d, %d) with button %d",
//             event->pos().x(), event->pos().y(), event->button());
        mousePressed = true;
        pressedPoint = event->pos();
        pressedButton = event->button();

        if((event->button() == Qt::LeftButton) & (mouseSense & MOUSE_DRAG))
        {
            // initialise selection rect
            selectionRect.setTopLeft(pressedPoint);
            selectionRect.setBottomRight(pressedPoint);
        }

        emit pressPoint(pressedPoint, pressedButton);
    }
}

/*
 * Callback called when mouse move event occurs.
 * reimplemented to send dragPoint signal when mouse is dragged
 * (after left mouse button has been pressed)
 * @param event mouse event
 */
void QcvMatWidget::mouseMoveEvent(QMouseEvent *event)
{
    mouseMoved = true;
    draggedPoint = event->pos();

    if ((mouseSense & MOUSE_DRAG) & mousePressed)
    {
//      qDebug("mouseMoveEvent(%d, %d) with button %d",
//             event->pos().x(), event->pos().y(), event->button());

        selectionRectFromPoints(pressedPoint, draggedPoint);

        emit dragPoint(draggedPoint);
    }
}

/*
 * Callback called when mouse button released event occurs.
 * reimplemented to send releasePoint signal when left mouse button is
 * released
 * @param event mouse event
 */
void QcvMatWidget::mouseReleaseEvent(QMouseEvent *event)
{
    if ((mouseSense > MOUSE_NONE) & mousePressed)
    {
//      qDebug("mouseReleaseEvent(%d, %d) with button %d",
//             event->pos().x(), event->pos().y(), event->button());
        mousePressed = false;
        mouseMoved = false;
        releasedPoint = event->pos();
        emit releasePoint(releasedPoint, pressedButton);

        if ((event->button() == Qt::LeftButton) & (mouseSense & MOUSE_DRAG))
        {
            selectionRectFromPoints(pressedPoint, releasedPoint);
            emit releaseSelection(selectionRect, event->button());
        }
    }
}

/*
 * Draw Cross
 * @param p the cross center
 */
void QcvMatWidget::drawCross(const QPoint & p)
{
    int x0 = p.x();
    int y0 = p.y();
    int x1, x2, x3, x4;
    int y1, y2, y3, y4;
    int offset = 10;

    x1 = x0 - 2*offset;
```



```
    x2 = x0 - offset;
    x3 = x0 + offset;
    x4 = x0 + 2*offset;
    y1 = y0 - 2*offset;
    y2 = y0 - offset;
    y3 = y0 + offset;
    y4 = y0 + 2*offset;

    Point p1a(x1, y0);
    Point p1b(x2, y0);
    Point p2a(x3, y0);
    Point p2b(x4, y0);
    Point p3a(x0, y1);
    Point p3b(x0, y2);
    Point p4a(x0, y3);
    Point p4b(x0, y4);

    line(displayImage, p1a, p1b, drawingColor, drawingWidth, CV_AA);
    line(displayImage, p2a, p2b, drawingColor, drawingWidth, CV_AA);
    line(displayImage, p3a, p3b, drawingColor, drawingWidth, CV_AA);
    line(displayImage, p4a, p4b, drawingColor, drawingWidth, CV_AA);
}

/*
 * Draw rectangle
 * @param r the rectangle to draw
 */
void QcvMatWidget::drawRectangle(const QRect & r)
{
    int x1 = r.left();
    int x2 = r.right();
    int y1 = r.top();
    int y2 = r.bottom();

    Point p1(x1, y1);
    Point p2(x2, y2);

    rectangle(displayImage, p1, p2, drawingColor, drawingWidth, CV_AA);
}

/*
 * Modifiy selectionRect using two points
 * @param p1 first point
 * @param p2 second point
 */
void QcvMatWidget::selectionRectFromPoints(const QPoint & p1, const QPoint & p2)
{
    int left, right, top, bottom;
    if (p1.x() < p2.x())
    {
        left = p1.x();
        right = p2.x();
    }
    else
    {
        left = p2.x();
        right = p1.x();
    }

    if (p1.y() < p2.y())
    {
        top = p1.y();
        bottom = p2.y();
    }
    else
    {
        top = p2.y();
        bottom = p1.y();
    }

    selectionRect.setLeft(left);
    selectionRect.setRight(right);
    selectionRect.setTop(top);
    selectionRect.setBottom(bottom);
}



/*
 * Widget minimum size is set to the contained image size
 * @return le size of the image within
 */
//QSize QcvMatWidget::minimumSize() const
//{
//  return sizeHint();
//}


/*
 * Size hint (because size depends on sourceImage properties)
```

```
361  * @return size obtained from sourceImage
362  */
363 QSize QcvMatWidget::sizeHint() const
364 {
365     if (sourceImage ≠ NULL)
366     {
367         return QSize(sourceImage→cols, sourceImage→rows);
368     }
369     else
370     {
371         return defaultSize;
372     }
373 }
374
375 /*
376  * Gets Mat widget mouse clickable status
377  * @return true if widget is sensitive to mouse click
378  */
379 bool QcvMatWidget::isMouseClickable() const
380 {
381     return (mouseSense & MOUSE_CLICK);
382 }
383
384 /*
385  * Gets Mat widget mouse dragable status
386  * @return true if widget is sensitive to mouse drag
387  */
388 bool QcvMatWidget::isMouseDragable() const
389 {
390     return (mouseSense & MOUSE_DRAG);
391 }
392
393 /*
394  * Update slot customized to include convertImage before actually
395  * updating
396  */
397 void QcvMatWidget::update()
398 {
399 //  count++;
400 //  qDebug() << "QcvMatWidget::update " << count;
401 //  std::cerr << "{o";
402     convertImage();
403     QWidget::update();
404 //  std::cerr << "}";
405 }
406
407 /*
408  * Recompute pixel scale according to screen pixel scale.
409  * Used with Hi DPI devices (such as retina screens)
410  * @post pixel scale have been updated according to
411  * devicePixelRatioF provided by the QPaintDevice super class
412  */
413 void QcvMatWidget::screenChanged()
414 {
415     pixelScale = devicePixelRatioF();
416 //  qDebug() << "Pixel scale updated to" << pixelScale;
417 }
418
419 // ----------------------------------------------------------------------------
420 // convertImage old algorithm
421 // ----------------------------------------------------------------------------
422 //  int cvIndex, cvLineStart;
423 //  // switch between bit depths
424 //  switch (displayImage.depth())
425 //  {
426 //      case CV_8U:
427 //          switch (displayImage.channels())
428 //          {
429 //              case 1: // Grav level images
430 //                  if ( (displayImage.cols != image.width()) ||
431 //                       (displayImage.rows != image.height()) )
432 //                  {
433 //                      QImage temp(displayImage.cols, displayImage.rows,
434 //                              QImage::Format_RGB32);
435 //                      image = temp;
436 //                  }
437 //                  cvIndex = 0;
438 //                  cvLineStart = 0;
439 //                  for (int y = 0; y < displayImage.rows; y++)
440 //                  {
441 //                      unsigned char red, green, blue;
442 //                      cvIndex = cvLineStart;
443 //                      for (int x = 0; x < displayImage.cols; x++)
444 //                      {
445 //                          // DO it
446 //                          red   = displayImage.data[cvIndex];
447 //                          green = displayImage.data[cvIndex];
448 //                          blue  = displayImage.data[cvIndex];
449 //
450 //                          image.setPixel(x, y, qRgb(red, green, blue));
```



```
451 //                          cvIndex++;
452 //                      }
453 //                      cvLineStart += displayImage.step;
454 //                  }
455 //                  break;
456 //              case 3: // BGR images (Regular OpenCV Color Capture)
457 //                  if ( (displayImage.cols != image.width()) ||
458 //                       (displayImage.rows != image.height()) )
459 //                  {
460 //                      QImage temp(displayImage.cols, displayImage.rows,
461 //                              QImage::Format_RGB32);
462 //                      image = temp;
463 //                  }
464 //                  cvIndex = 0;
465 //                  cvLineStart = 0;
466 //                  for (int y = 0; y < displayImage.rows; y++)
467 //                  {
468 //                      unsigned char red, green, blue;
469 //                      cvIndex = cvLineStart;
470 //                      for (int x = 0; x < displayImage.cols; x++)
471 //                      {
472 //                          // DO it
473 //                          red   = displayImage.data[cvIndex + 2];
474 //                          green = displayImage.data[cvIndex + 1];
475 //                          blue  = displayImage.data[cvIndex + 0];
476 //
477 //                          image.setPixel(x, y, qRgb(red, green, blue));
478 //                          cvIndex += 3;
479 //                      }
480 //                      cvLineStart += displayImage.step;
481 //                  }
482 //                  break;
483 //              default:
484 //                  printf("This number of channels is not supported\n");
485 //                  break;
486 //          }
487 //          break;
488 //      default:
489 //          printf("This type of Image is not implemented in QcvMatWidget\n");
490 //          break;
491 //  }
492
```

```cpp
//#include <iostream>
#include <QtDebug>
#include "QcvMatWidgetLabel.h"

using namespace std;

/*
 * OpenCV QT Widget default constructor
 * @param parent parent widget
 */
QcvMatWidgetLabel::QcvMatWidgetLabel(QWidget *parent,
                                     MouseSense mouseSense) :
    QcvMatWidget(parent, mouseSense),
    imageLabel(new QLabel())
{
    setup();
}

/*
 * OpenCV QT Widget constructor
 * @param the source OpenCV qImage
 * @param parent parent widget
 */
QcvMatWidgetLabel::QcvMatWidgetLabel(Mat * sourceImage,
                                     QWidget *parent,
                                     MouseSense mouseSense) :
    QcvMatWidget(sourceImage, parent, mouseSense),
    imageLabel(new QLabel())
{
    setup();
}

/*
 * Widget setup
 * @pre imageLabel has been allocated
 */
void QcvMatWidgetLabel::setup()
{
    layout->addWidget(imageLabel,0,Qt::AlignCenter);
}

/*
 * OpenCV Widget destructor.
 */
QcvMatWidgetLabel::~QcvMatWidgetLabel(void)
{
    delete imageLabel;
}

/*
 * paint event reimplemented to draw content
 * @param event the paint event
 */
void QcvMatWidgetLabel::paintEvent(QPaintEvent * event)
{
//  qDebug("QcvMatWidgetLabel::paintEvent");
    QcvMatWidget::paintEvent(event);

    if (displayImage.data != NULL)
    {
        // Builds QImage from RGB image data
        // and sets image as Label pixmap
        imageLabel->setPixmap(QPixmap::fromImage(QImage((uchar *) displayImage.data,
                                                        displayImage.cols,
                                                        displayImage.rows,
                                                        displayImage.step,
                                                        QImage::Format_RGB888)));
    }
    else
    {
        qWarning("QcvMatWidgetLabel::paintEvent : image.data is NULL");
    }
}
```



```cpp
/*
 * QcvMatWidgetImage.cpp
 *
 *  Created on: 31 janv. 2012
 *      Author: davidroussel
 */

#include "QcvMatWidgetImage.h"
#include <QPaintEvent>
#include <QSizePolicy>
#include <QDebug>

/*
 * Default Constructor
 * @param parent parent widget
 */
QcvMatWidgetImage::QcvMatWidgetImage(QWidget *parent,
                                     MouseSense mouseSense) :
    QcvMatWidget(parent, mouseSense),
    qImage(NULL)
{
    setup();
}

/*
 * Constructor
 * @param sourceImage source image
 * @param parent parent widget
 */
QcvMatWidgetImage::QcvMatWidgetImage(Mat * sourceImage,
                                     QWidget *parent,
                                     MouseSense mouseSense) :
    QcvMatWidget(sourceImage, parent, mouseSense),
    qImage(NULL)
{
    setSourceImage(sourceImage);

    setup();
}

/*
 * Setup widget (defines size policy)
 */
void QcvMatWidgetImage::setup()
{
//  qDebug("QcvMatWidgetImage::Setup");

    /*
     * Customize size policy
     */
    QSizePolicy qsp(QSizePolicy::Fixed, QSizePolicy::Fixed);
    // sets height depends on width (also need to reimplement heightForWidth())
    qsp.setHeightForWidth(true);
    setSizePolicy(qsp);

    /*
     * Customize layout
     */

    // size policy has changed to call updateGeometry
    updateGeometry();
}

/*
 * Destructor.
 */
QcvMatWidgetImage::~QcvMatWidgetImage()
{
    if (qImage != NULL)
    {
        delete qImage;
    }
}

/*
 * Sets new source image
 * @param sourceImage the new source image
 */
void QcvMatWidgetImage::setSourceImage(Mat * sourceImage)
{
    if (qImage != NULL)
    {
        delete qImage;
    }
    // setup and convert image
    QcvMatWidget::setSourceImage(sourceImage);
    convertImage();
    qImage = new QImage((uchar *) displayImage.data, displayImage.cols,
        displayImage.rows, displayImage.step,
        QImage::Format_RGB888);
```

```
91
92      // re-setup geometry since height x width may have changed
93      updateGeometry();
94  }
95
96  /*
97   * Size policy to keep aspect ratio right
98   * @return
99   */
100 //QSizePolicy QcvMatWidgetImage::sizePolicy () const
101 //{
102 //  return policy;
103 //}
104
105 /*
106  * aspect ratio method
107  * @param w width
108  * @return the required height fo r this width
109  */
110 int QcvMatWidgetImage::heightForWidth(int w) const
111 {
112 //  qDebug ("height  = %d for width  = %d called", (int)((double)w/aspectRatio), w);
113     return (int)((double)w/aspectRatio);
114 }
115
116 /*
117  * Minimum size hint according to aspect ratio and min height of 100
118  * @return minimum size hint
119  */
120 //QSize QcvMatWidgetImage::minimumSizeHint () const
121 //{
122 //  // qDebug("min size called");
123 //  // return QSize((int)(100.0*aspectRatio), 100);
124 //  return sizeHint();
125 //}
126
127
128 /*
129  * paint event reimplemented to draw content
130  * @param event the paint event
131  */
132 void QcvMatWidgetImage::paintEvent(QPaintEvent *event)
133 {
134 //  qDebug("QcvMatWidgetImage::paintEvent");
135
136     // evt draws in image directly
137     QcvMatWidget::paintEvent(event);
138
139     if (displayImage.data ≠ NULL)
140     {
141         // then draw image
142         QPainter painter(this);
143         painter.setRenderHint(QPainter::SmoothPixmapTransform, true);
144         if (event ≡ NULL)
145         {
146             painter.drawImage(0, 0, *qImage);
147         }
148         else // partial repaint
149         {
150             painter.drawImage(event→rect(), *qImage);
151         }
152     }
153     else
154     {
155         qWarning("QcvMatWidgetImage::paintEvent : image.data is NULL");
156     }
157 }
```



```
1   /*
2    * QcvMatWidgetGL.cpp
3    *
4    *  Created on: 28 fÃ©vr. 2011
5    *      Author: davidroussel
6    */
7   #include <QDebug>
8
9   #include "QcvMatWidgetGL.h"
10
11  /*
12   * OpenCV QT Widget default constructor
13   * @param parent parent widget
14   */
15  QcvMatWidgetGL::QcvMatWidgetGL(QWidget *parent,
16                                 MouseSense mouseSense) :
17      QcvMatWidget(parent, mouseSense),
18      gl(NULL)
19  {
20  }
21
22  /*
23   * OpenCV QT Widget constructor
24   * @param parent parent widget
25   */
26  QcvMatWidgetGL::QcvMatWidgetGL(Mat * sourceImage,
27                                 QWidget *parent,
28                                 MouseSense mouseSense) :
29      QcvMatWidget(sourceImage, parent, mouseSense),
30      gl(NULL)
31  {
32      setSourceImage(sourceImage);
33  }
34
35  /*
36   * OpenCV Widget destructor.
37   */
38  QcvMatWidgetGL::~QcvMatWidgetGL()
39  {
40      if (gl ≠ NULL)
41      {
42          layout→removeWidget(gl);
43          delete gl;
44      }
45  }
46
47  /*
48   * Sets new source image
49   * @param sourceImage the new source image
50   */
51  void QcvMatWidgetGL::setSourceImage(Mat *sourceImage)
52  {
53      QcvMatWidget::setSourceImage(sourceImage);
54
55      if (gl ≠ NULL)
56      {
57          layout→removeWidget(gl);
58          delete gl;
59      }
60
61      convertImage();
62
63      gl = new QGLImageRender(displayImage, GL_RGB, &pixelScale, this);
64
65      layout→addWidget(gl, 0, Qt::AlignCenter);
66  }
67
68  /*
69   * paint event reimplemented to draw content
70   * @param event the paint event
71   */
72  void QcvMatWidgetGL::paintEvent(QPaintEvent * event)
73  {
74      QcvMatWidget::paintEvent(event);
75      gl→update();
76  }
```

```cpp
/*
 * QGLImageRender.cpp
 *
 *  Created on: 28 fÃ©vr. 2011
 *      Author: davidroussel
 */
#include <QDebug>
#ifdef __APPLE__
    #include <gl.h>
    #include <glu.h>
#else
    #include <GL/gl.h>
    #include <GL/glu.h>
#endif
#include "QGLImageRender.h"

/*
 * QGLImageRender Constructor
 * @param image the RGB image to draw in the pixel buffer
 * @param format pixel format
 * @param pixelScale pixel scale pointer from container
 * @param parent the parent widget
 */
QGLImageRender::QGLImageRender(const Mat & image,
                               const GLenum format,
                               float * pixelScale,
                               QWidget *parent) :
    QGLWidget(parent),
    image(image),
    pixelFormat(format),
    pixelScale(pixelScale)
{
    if (¬doubleBuffer())
    {
        qWarning("QGLImageRender::QGLImageRender caution : no double buffer");
    }
    if (this→image.data ≡ NULL)
    {
        qWarning("QGLImageRender::QGLImageRender caution : image data is null");
    }
    if (this→pixelScale ≡ NULL)
    {
        qCritical("QGLImageRender::QGLImageRender caution : pixel scale is null");
    }
}

QGLImageRender::~QGLImageRender()
{
    image.release();
}

void QGLImageRender::initializeGL()
{
//  qDebug("GL init ...");
    glClearColor(0.0, 0.0, 0.0, 0.0);
//  glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
}

void QGLImageRender::resizeGL(int width, int height)
{
//  qDebug("GL resizeGL ...");

    glViewport(0, 0, (GLsizei) width, (GLsizei) height);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (image.data ≠ NULL)
    {
        glOrtho(0, (GLdouble) image.cols, 0, (GLdouble) image.rows, 1.0, -1.0);
    }

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void QGLImageRender::paintGL()
{
//  qDebug("GL drawing pixels ...");

    glClear(GL_COLOR_BUFFER_BIT);

    if (image.data ≠ NULL)
    {
        /* apply the right translate so the image drawing starts top left */
        glRasterPos4f(0.0f,(GLfloat)(image.rows), 0.0f, 1.0f);
        /*
         * for hi dpi displays
         *  typically pixelScale =
         *  - 1.0 for normal displays
         *  - 2.0 for hidpi displays
```



```cpp
         */
        glPixelZoom(*pixelScale, -(*pixelScale));

        glDrawPixels(image.cols, image.rows, pixelFormat,
                     GL_UNSIGNED_BYTE, image.data);
        // In any circumstance you should NOT use glFlush or swapBuffers() here
    }
    else
    {
        qWarning("Nothing to draw");
    }
}

QSize QGLImageRender::sizeHint () const
{
    return minimumSizeHint();
}

QSize QGLImageRender::minimumSizeHint() const
{
    if (image.data ≠ NULL)
    {
        return QSize(image.cols, image.rows);
    }
    else
    {
        qWarning("QGLImageRender::minimumSizeHint : probably invalid sizeHint");
        return QSize(320,240);
    }
}

QSizePolicy QGLImageRender::sizePolicy () const
{
    return QSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
}
```

```
/*
 * QcvVideoCapture.cpp
 *
 *  Created on: 29 janv. 2012
 *      Author: davidroussel
 */

#include <QElapsedTimer>
#include <QDebug>

#include "QcvVideoCapture.h"

#include <opencv2/imgproc/imgproc.hpp>

/*
 * default time interval between refresh
 */
int QcvVideoCapture::defaultFrameDelay = 33;

/*
 * Number of frames to test frame rate
 */
size_t QcvVideoCapture::defaultFrameNumberTest = 5;

/*
 * Default message showing time (at least 2000 ms)
 */
int QcvVideoCapture::messageDelay = 5000;

/*
 * QcvVideoCapture constructor.
 * Opens the default camera (0)
 * @param flipVideo mirror image status
 * @param gray convert image to gray status
 * @param skip indicates capture can skip an image. When the capture
 * result has not been processed yet, or when false that capture should
 * wait for the result to be processed before grabbing a new image.
 * This only applies when #updateThread is not NULL.
 * @param width desired width or 0 to keep capture width
 * @param height desired height or 0 to keep capture height
 * otherwise capture is updated in the current thread.
 * @param updateThread the thread used to run this capture
 * @param parent the parent QObject
 */
QcvVideoCapture::QcvVideoCapture(const bool flipVideo,
                                 const bool gray,
                                 const bool skip,
                                 const unsigned int width,
                                 const unsigned int height,
                                 QThread * updateThread,
                                 QObject * parent) :
    QcvVideoCapture(0, flipVideo, gray, skip, width, height, updateThread,
                    parent)
{
}

/*
 * QcvVideoCapture constructor with device Id
 * @param deviceId the id of the camera to open
 * @param flipVideo mirror image
 * @param gray convert image to gray
 * @param skip indicates capture can skip an image. When the capture
 * result has not been processed yet, or when false that capture should
 * wait for the result to be processed before grabbing a new image.
 * This only applies when #updateThread is not NULL.
 * @param width desired width or 0 to keep capture width
 * @param height desired height or 0 to keep capture height
 * @param updateThread the thread used to run this capture
 * @param parent the parent QObject
 */
QcvVideoCapture::QcvVideoCapture(const int deviceId,
                                 const bool flipVideo,
                                 const bool gray,
                                 const bool skip,
                                 const unsigned int width,
                                 const unsigned int height,
                                 QThread * updateThread,
                                 QObject * parent) :
    QObject(parent),
    filename(),
    capture(deviceId),
    timer(new QTimer(updateThread != NULL ? this : NULL)),
    updateThread(updateThread),
    mutex(QMutex::NonRecursive),
    lockLevel(0),
    liveVideo(true),
    flipVideo(flipVideo),
    resize(false),
    directResize(false),
    gray(gray),
```



```
    skip(skip),
    size(0, 0),
    originalSize(0, 0),
    frameRate(0.0),
    statusMessage()
{
    if (updateThread != NULL)
    {
        moveToThread(this->updateThread);
        connect(this, SIGNAL(finished()), updateThread, SLOT(quit()),
                Qt::DirectConnection);
    }

    timer->setSingleShot(false);
    connect(timer, SIGNAL(timeout()), SLOT(update()));

    if (grabTest())
    {
        setSize(width, height);
        QString message("Camera ");
        message.append(QString::number(deviceId));
        message.append(" ");
        int delay = grabInterval(message);
        if (updateThread != NULL)
        {
            updateThread->start();
        }
        timer->start(delay);
        qDebug("timer started with %d ms delay", delay);
        emit timerChanged(delay);
    }
    else
    {
        qDebug() << "QcvVideoCapture::QcvVideoCapture(" << deviceId
                 << "): grab test failed";
    }
}

/*
 * QcvVideoCapture constructor from file name
 * @param fileName video file to open
 * @param flipVideo mirror image
 * @param gray convert image to gray
 * @param skip indicates capture can skip an image. When the capture
 * result has not been processed yet, or when false that capture should
 * wait for the result to be processed before grabbing a new image.
 * This only applies when #updateThread is not NULL.
 * @param width desired width or 0 to keep capture width
 * @param height desired height or 0 to keep capture height
 * @param updateThread the thread used to run this capture
 * @param parent the parent QObject
 */
QcvVideoCapture::QcvVideoCapture(const QString & fileName,
                                 const bool flipVideo,
                                 const bool gray,
                                 const bool skip,
                                 const unsigned int width,
                                 const unsigned int height,
                                 QThread * updateThread,
                                 QObject * parent) :
    QObject(parent),
    filename(fileName),
    capture(fileName.toStdString()),
    timer(new QTimer(updateThread != NULL ? this : NULL)),
    updateThread(updateThread),
    mutex(QMutex::NonRecursive),
    lockLevel(0),
    liveVideo(false),
    flipVideo(flipVideo),
    resize(false),
    directResize(false),
    gray(gray),
    skip(skip),
    size(0, 0),
    originalSize(0, 0),
    frameRate(0.0),
    statusMessage()
{
    if (updateThread != NULL)
    {
        moveToThread(this->updateThread);
        connect(this, SIGNAL(finished()), updateThread, SLOT(quit()),
                Qt::DirectConnection);
    }

    timer->setSingleShot(false);
    connect(timer, SIGNAL(timeout()), SLOT(update()));

    if (grabTest())
    {
```

```
181         setSize(width, height);
182         QString message("File ");
183         message.append(fileName);
184         message.append(" ");
185 
186         int delay = grabInterval(message);
187         if (updateThread ≠ NULL)
188         {
189             updateThread→start();
190         }
191         timer→start(delay);
192         qDebug("timer started with %d ms delay", delay);
193         emit timerChanged(delay);
194     }
195 }
196 
197 /*
198  * QcvVideoCapture destructor.
199  * releases video capture and image
200  */
201 QcvVideoCapture::~QcvVideoCapture()
202 {
203     // wait for the end of an update
204     if (updateThread ≠ NULL)
205     {
206         if (lockLevel ≡ 0)
207         {
208             // qDebug() << "QcvVideoCapture::~QcvVideoCapture: lock in thread"
209             //          << QThread::currentThread();
210             mutex.lock();
211         }
212         lockLevel++;
213 
214         emit finished();
215     }
216 
217     if (timer ≠ NULL)
218     {
219         if (timer→isActive())
220         {
221             timer→stop();
222             qDebug("timer stopped");
223         }
224 
225         timer→disconnect(SIGNAL(timeout()), this, SLOT(update()));
226     }
227 
228     if (updateThread ≠ NULL)
229     {
230         lockLevel--;
231         if (lockLevel ≡ 0)
232         {
233             mutex.unlock();
234         }
235 
236         // Wait until the updateThread receives the "finished" signal through
237         // "quit" slot
238         updateThread→wait();
239 
240         delete timer; // delete unparented timer
241     }
242 
243     // releasae OpenCV ressources
244     filename.clear();
245     capture.release();
246     imageDisplay.release();
247     imageFlipped.release();
248     imageResized.release();
249     image.release();
250 
251 //  qDebug() << "QcvVideoCapture destroyed";
252 }
253 
254 /*
255  * Open new device Id
256  * @param deviceId device number to open
257  * @param width desired width or 0 to keep capture width
258  * @param height desired height or 0 to keep capture height
259  * @return true if device has been opened and checked and timer launched
260  */
261 bool QcvVideoCapture::open(const int deviceId,
262                            const unsigned int width,
263                            const unsigned int height)
264 {
265     if (updateThread ≠ NULL)
266     {
267         if (lockLevel ≡ 0)
268         {
269             mutex.lock();
270         }
```



```
271         lockLevel++;
272     }
273 
274     filename.clear();
275     if (timer→isActive())
276     {
277         timer→stop();
278         qDebug("timer stopped");
279     }
280 
281     if (capture.isOpened())
282     {
283         capture.release();
284     }
285 
286     if (¬image.empty())
287     {
288         image.release();
289     }
290 
291     capture.open(deviceId);
292 
293     bool grabbed = grabTest();
294 
295     if (grabbed)
296     {
297         setSize(width, height);
298 
299         statusMessage.clear();
300         statusMessage.append("Camera ");
301         statusMessage.append(QString::number(deviceId));
302         statusMessage.append(" ");
303         int delay = grabInterval(statusMessage);
304         timer→start(delay);
305         liveVideo = true;
306         qDebug("timer started with %d ms delay", delay);
307         emit timerChanged(delay);
308         emit imageChanged(&imageDisplay);
309     }
310     if (updateThread ≠ NULL)
311     {
312         lockLevel--;
313         if (lockLevel ≡ 0)
314         {
315             mutex.unlock();
316         }
317     }
318 
319     return grabbed;
320 }
321 
322 /*
323  * Open new video file
324  * @param fileName video file to open
325  * @param width desired width or 0 to keep capture width
326  * @param height desired height or 0 to keep capture height
327  * @return true if video has been opened and timer launched
328  */
329 bool QcvVideoCapture::open(const QString & fileName,
330                            const unsigned int width,
331                            const unsigned int height)
332 {
333     filename = fileName;
334 
335     if (timer→isActive())
336     {
337         timer→stop();
338         qDebug("timer stopped");
339     }
340 
341     if (updateThread ≠ NULL)
342     {
343         if (lockLevel ≡ 0)
344         {
345             mutex.lock();
346         }
347         lockLevel++;
348     }
349 
350     if (capture.isOpened())
351     {
352         capture.release();
353     }
354 
355     if (¬image.empty())
356     {
357         image.release();
358     }
359 
360     capture.open(fileName.toStdString());
```

```
361
362     bool grabbed = grabTest();
363
364     if (grabbed)
365     {
366         setSize(width, height);
367         // qDebug() << "open setSize done";
368         statusMessage.clear();
369         statusMessage.append("file ");
370         statusMessage.append(fileName);
371         statusMessage.append(" opened");
372
373         int delay = grabInterval(statusMessage);
374         timer→start(delay);
375         liveVideo = false;
376         qDebug("timer started with %d ms delay", delay);
377         emit timerChanged(delay);
378         emit imageChanged(&imageDisplay);
379     }
380
381     if (updateThread ≠ NULL)
382     {
383         lockLevel--;
384         if(lockLevel ≡ 0)
385         {
386             mutex.unlock();
387         }
388     }
389
390     return grabbed;
391 }
392
393 /*
394  * Size accessor
395  * @return the image size
396  */
397 const QSize & QcvVideoCapture::getSize() const
398 {
399     return size;
400 }
401
402 /*
403  * Sets #imageDisplay size according to preferred width and height
404  * @param width desired width
405  * @param height desired height
406  * @pre a first image have been grabbed
407  */
408 void QcvVideoCapture::setSize(const unsigned int width,
409                               const unsigned int height)
410 {
411     if ((updateThread ≠ NULL))
412     {
413         if (lockLevel ≡ 0)
414         {
415             mutex.lock();
416         }
417         lockLevel++;
418     }
419
420     unsigned int preferredWidth;
421     unsigned int preferredHeight;
422
423     // if not empty then release it
424     if (¬imageResized.empty())
425     {
426         imageResized.release();
427     }
428
429     if ((width ≡ 0) ∧ (height ≡ 0)) // reset to original size
430     {
431         if (directResize) // direct set size to original size
432         {
433             setDirectSize((unsigned int)originalSize.width(),
434                           (unsigned int)originalSize.height());
435             // image is updated into setDirectSize
436         }
437         preferredWidth = image.cols;
438         preferredHeight = image.rows;
439
440         resize = false;
441         imageResized = image;
442     }
443     else // width != 0 or height != 0
444     {
445         if ((width ≡ (unsigned int)image.cols) ∧
446             (height ≡ (unsigned int)image.rows)) // unchanged
447         {
448             preferredWidth = image.cols;
449             preferredHeight = image.rows;
450             imageResized = image;
```



```
451
452             if (((int)preferredWidth ≡ originalSize.width()) ∧
453                 ((int)preferredHeight ≡ originalSize.height()))
454             {
455                 resize = false;
456             }
457             else
458             {
459                 resize = true;
460             }
461         }
462         else // width or height have changed
463         {
464             /*
465              * Resize needed
466              */
467             preferredWidth = width;
468             preferredHeight = height;
469
470             resize = true;
471
472             if (directResize)
473             {
474                 setDirectSize(preferredWidth, preferredHeight);
475                 imageResized = image;
476             }
477             else
478             {
479                 imageResized = Mat(preferredHeight, preferredWidth, image.type());
480             }
481         }
482     }
483
484     if (updateThread ≠ NULL)
485     {
486         lockLevel--;
487         if (lockLevel ≡ 0)
488         {
489             mutex.unlock();
490         }
491     }
492
493     qDebug("QcvVideoCapture resize is %s [%s]",
494            (resize ? "ON" : "OFF"),
495            (directResize ? "direct" : "soft"));
496
497     size.setWidth(preferredWidth);
498     size.setHeight(preferredHeight);
499     statusMessage.clear();
500     statusMessage.sprintf("Size set to %dx%d", preferredWidth, preferredHeight);
501     emit messageChanged(statusMessage, messageDelay);
502
503
504     /*
505      * imageChanged signal is delayed until setGray is called into
506      * setFlipVideo
507      */
508     // Refresh image chain
509     setFlipVideo(flipVideo);
510 }
511
512 /*
513  * Sets #imageDisplay size according to preferred width and height
514  * @param size new desired size to set
515  * @pre a first image have been grabbed
516  */
517 void QcvVideoCapture::setSize(const QSize & size)
518 {
519     setSize(size.width(), size.height());
520 }
521
522 /*
523  * Sets video flipping
524  * @param flipVideo flipped video or not
525  */
526 void QcvVideoCapture::setFlipVideo(const bool flipVideo)
527 {
528     bool previousFlip = this→flipVideo;
529     this→flipVideo = flipVideo;
530
531     if (updateThread ≠ NULL)
532     {
533         if (lockLevel ≡ 0)
534         {
535             mutex.lock();
536         }
537         lockLevel++;
538     }
539
540     if (¬imageFlipped.empty())
```

```
541     {
542         imageFlipped.release();
543     }
544
545     if (flipVideo)
546     {
547         imageFlipped = Mat(imageResized.size(), imageResized.type());
548     }
549     else
550     {
551         imageFlipped = imageResized;
552     }
553
554     if (updateThread ≠ NULL)
555     {
556         lockLevel--;
557         if (lockLevel ≡ 0)
558         {
559             mutex.unlock();
560         }
561     }
562
563     if (previousFlip ≠ flipVideo)
564     {
565         statusMessage.clear();
566         statusMessage.sprintf("flip video is %s", (flipVideo ? "on" : "off"));
567         emit messageChanged(statusMessage, messageDelay);
568         emit imageChanged(&imageDisplay);
569     }
570
571     /*
572      * imageChanged signal is delayed until setGray is called
573      */
574     // refresh image chain
575     setGray(gray);
576 }
577
578 /*
579  * Sets video conversion to gray
580  * @param grayConversion the gray conversion status
581  */
582 void QcvVideoCapture::setGray(const bool grayConversion)
583 {
584     bool previousGray = gray;
585
586     gray = grayConversion;
587
588     if (updateThread ≠ NULL)
589     {
590         if (lockLevel ≡ 0)
591         {
592             mutex.lock();
593         }
594         lockLevel++;
595     }
596
597     if (¬imageDisplay.empty())
598     {
599         imageDisplay.release();
600     }
601
602     if (gray)
603     {
604         imageDisplay = Mat(imageFlipped.size(), CV_8UC1);
605     }
606     else
607     {
608         imageDisplay = imageFlipped;
609     }
610
611     if (updateThread ≠ NULL)
612     {
613         lockLevel--;
614         if (lockLevel ≡ 0)
615         {
616             mutex.unlock();
617         }
618     }
619
620     if (previousGray ≠ grayConversion)
621     {
622         statusMessage.clear();
623         statusMessage.sprintf("gray video is %s", (gray ? "on" : "off"));
624         emit messageChanged(statusMessage, messageDelay);
625     }
626
627     /*
628      * In any cases emit image changed since
629      *  - setSize may have been called
630      *  - setFlipVideo may have been called
```



```
631      */
632     emit imageChanged(&imageDisplay);
633 }
634
635 /*
636  * Gets resize state.
637  * @return true if imageDisplay have been resized to preferred width and
638  * height, false otherwise
639  */
640 bool QcvVideoCapture::isResized() const
641 {
642     return resize;
643 }
644
645 /*
646  * Gets direct resize state.
647  * @return true if image can be resized directly into capture.
648  * @note direct resize capabilities are tested into #grabTest which is
649  * called in all constructors. So #isDirectResizeable should not be
650  * called before #grabTest
651  */
652 bool QcvVideoCapture::isDirectResizeable() const
653 {
654     return directResize;
655 }
656
657 /*
658  * Gets video flipping status
659  * @return flipped video status
660  */
661 bool QcvVideoCapture::isFlipVideo() const
662 {
663     return flipVideo;
664 }
665
666 /*
667  * Gets video gray converted status
668  * @return the converted to gray status
669  */
670 bool QcvVideoCapture::isGray() const
671 {
672     return gray;
673 }
674
675 /*
676  * Gets the image skipping policy
677  * @return true if new image can be skipped when previous one has not
678  * been processed yet, false otherwise.
679  */
680 bool QcvVideoCapture::isSkippable() const
681 {
682     return skip;
683 }
684
685 /*
686  * Gets the current frame rate
687  * @return the current frame rate
688  */
689 double QcvVideoCapture::getFrameRate() const
690 {
691     return frameRate;
692 }
693
694 /*
695  * Image accessor
696  * @return the image
697  */
698 Mat * QcvVideoCapture::getImage()
699 {
700     return &imageDisplay;
701 }
702
703 /*
704  * The source image mutex
705  * @return  the mutex used on image access
706  */
707 QMutex * QcvVideoCapture::getMutex()
708 {
709     return &mutex;
710 }
711
712 /*
713  * Performs a grab test to fill #image
714  * @return true if capture is opened and successfully grabs a first
715  * frame into #image, false otherwise
716  */
717 bool QcvVideoCapture::grabTest()
718 {
719 //  qDebug("Grab test");
720     bool result = false;
```

```
721
722     if (capture.isOpened())
723     {
724 #ifndef Q_OS_LINUX // V4L does not support these queries
725         int capWidth = capture.get(CV_CAP_PROP_FRAME_WIDTH);
726         int capHeight = capture.get(CV_CAP_PROP_FRAME_HEIGHT);
727
728         qDebug("Capture grab test with %d x %d image", capWidth, capHeight);
729 #endif
730         // grabs first frame
731         if (capture.grab())
732         {
733             bool retrieved = capture.retrieve(image);
734             if (retrieved)
735             {
736                 size.setWidth(image.cols);
737                 size.setHeight(image.rows);
738                 originalSize.setWidth(image.cols);
739                 originalSize.setHeight(image.rows);
740
741                 /*
742                  * Tries to determine if direct resizing in capture is possible
743                  * by setting original size through properties
744                  * Typically :
745                  *  - camera capture might be resizable
746                  *  - video file capture may not be resizable
747                  */
748                 directResize = setDirectSize(image.cols, image.rows);
749
750                 qDebug("Capture direct resizing is %s",
751                        (directResize ? "on" : "off"));
752
753                 result = true;
754             }
755             else
756             {
757                 qFatal("Video Capture unable to retreive image");
758             }
759         }
760         else
761         {
762             qFatal("Video Capture can not grab");
763         }
764     }
765     else
766     {
767         qFatal("Video Capture is not opened");
768     }
769
770     return result;
771 }
772
773 /*
774  * Get or compute interval between two frames
775  * @return interval between two frames
776  * @pre capture is already instanciated
777  */
778 int QcvVideoCapture::grabInterval(const QString & message)
779 {
780     int frameDelay = defaultFrameDelay;
781
782     // Tries to get framerate from capture
783     // ------------------------------------------------------------------
784     // Caution : on some systems getting video parameters is forbidden !
785     // For instance it does not work with linuxes equipped with V4L
786     // ------------------------------------------------------------------
787 #ifndef Q_OS_LINUX
788     frameRate = capture.get(CV_CAP_PROP_FPS);
789 #else
790     frameRate = -1.0;
791 #endif
792
793     /*
794      * if capture obtained frameRate is inconsistent, then we'll try to find out
795      * by ourselves
796      */
797     if (frameRate ≤ 0.0)
798     {
799         /*
800          * If live Video : grab a few images and measure elapsed time
801          */
802         if (liveVideo)
803         {
804             QElapsedTimer localTimer;
805             localTimer.start();
806
807             for (size_t i=0; i < defaultFrameNumberTest; i++)
808             {
809                 capture >> image;
810             }
```



```
811
812             frameDelay = (int)(localTimer.elapsed() / defaultFrameNumberTest);
813             frameRate = 1.0/((double)frameDelay/1000.0);
814             qDebug("Measured capture frame rate is  %4.2f images/s", frameRate);
815         }
816         /*
817          * FIXME else ???
818          * video files read through capture should provide framerate with
819          * capture.get(CV_CAP_PROP_FPS) but what happens if they don't ???
820          */
821     }
822     else
823     {
824         qDebug("%s Capture frame rate = %4.2f", message.toStdString().c_str(),
825                                                 frameRate);
826         frameDelay = 1000/frameRate;
827     }
828
829     statusMessage.sprintf("%s frame rate = %4.2f images/s",
830                           message.toStdString().c_str(), frameRate);
831     emit messageChanged(statusMessage, messageDelay);
832
833     return frameDelay;
834 }
835
836 /*
837  * Tries to set capture size directly on capture by using properties.
838  *  - CV_CAP_PROP_FRAME_WIDTH to set frame width
839  *  - CV_CAP_PROP_FRAME_HEIGHT to set frame height
840  * @param width the width property to set on capture
841  * @param height the height property to set on capture
842  * @return true if capture is opened and if width and height have been
843  * set successfully through @code capture.set(...) @endcode. Returns
844  * false otherwise.
845  * @post if at least width or height have been set successfully, capture
846  * image is released then updated again so it will have the right
847  * dimensions.
848  */
849 bool QcvVideoCapture::setDirectSize(const unsigned int width,
850                                     const unsigned int height)
851 {
852 #ifdef Q_OS_LINUX
853     Q_UNUSED(width);
854     Q_UNUSED(height);
855 #endif
856     bool done = false;
857
858     /*
859      * We absolutely need this lock in order to safely set width and
860      * height directly into the capture, so if mutex is already locked
861      * we should wait for it to be unlocked before continuing. Moreover,
862      * if mutex is NON-recursive and already locked, the call to lock() could
863      * lead to a DEADlock, so mutex HAS to be recursive !
864      */
865
866 #ifndef Q_OS_LINUX
867     if (capture.isOpened())
868     {
869         bool setWidth = capture.set(CV_CAP_PROP_FRAME_WIDTH, (double)width);
870         bool setHeight = capture.set(CV_CAP_PROP_FRAME_HEIGHT, (double)height);
871         if (setWidth ∨ setHeight)
872         {
873             // release old capture image
874             image.release();
875
876             // force image update to get the right size
877             capture >> image;
878
879             done = true;
880         }
881     }
882 #endif
883
884     return done;
885 }
886
887 /*
888  * update slot triggered by timer : Grabs a new image and sends updated()
889  * signal iff new image has been grabbed, otherwise there is no more
890  * images to grab so kills timer
891  */
892 void QcvVideoCapture::update()
893 {
894     bool locked = true;
895     bool image_updated = false;
896
897     if (updateThread ≠ NULL)
898     {
899         if (skip)
900         {
```

```
901             locked = mutex.tryLock();
902             if (locked)
903             {
904                 lockLevel++;
905             }
906         }
907         else
908         {
909             if (lockLevel ≡ 0)
910             {
911                 mutex.lock();
912             }
913             lockLevel++;
914         }
915     }
916
917     if (capture.isOpened() ∧ locked)
918     {
919         capture >> image;
920
921         if (¬image.data) // captured image has no data
922         {
923             statusMessage.clear();
924
925             if (liveVideo)
926             {
927                 if (timer→isActive())
928                 {
929                     timer→stop();
930                     qDebug("timer stopped");
931                 }
932
933                 capture.release();
934
935                 statusMessage.sprintf("No more frames to capture ...");
936                 emit messageChanged(statusMessage, 0);
937                 qDebug("%s", statusMessage.toStdString().c_str());
938             }
939             else // not live video ==> video file
940             {
941                 // We'll try to rewind the file back to frame 0
942                 bool restart = capture.set(CV_CAP_PROP_POS_FRAMES, 0.0);
943
944                 if (restart)
945                 {
946                     statusMessage.sprintf("Capture restarted");
947                     emit messageChanged(statusMessage,
948                                         QcvVideoCapture::messageDelay);
949                     emit restarted();
950                     qDebug("%s", statusMessage.toStdString().c_str());
951
952                     // Refresh image chain resized -> flipped -> gray
953                     setSize(size);
954                 }
955                 else
956                 {
957                     capture.release();
958
959                     statusMessage.sprintf("Failed to restart capture ...");
960                     emit messageChanged(statusMessage, 0);
961                     emit finished();
962                     qDebug("%s", statusMessage.toStdString().c_str());
963                 }
964             }
965         }
966         else // capture image has data
967         {
968             /*
969              * CAUTION
970              * image->imageResized->imageFlipped->imageDisplay
971              * constitute an image chain, so when size is changed with
972              * setSize it should call setFlipVideo which should call
973              * setGray
974              */
975
976             // resize image
977             if (resize ∧ ¬directResize)
978             {
979                 cv::resize(image, imageResized, imageResized.size(), 0, 0,
980                     INTER_AREA);
981             }
982             /*
983              * else imageResized.data is already == image.data
984              */
985
986             // flip image horizontally if required
987             if (flipVideo)
988             {
989                 flip(imageResized, imageFlipped, 1);
990             }
```



```
991             /*
992              * else imageFlipped.data is already == imageResized.data
993              */
994
995             // convert image to gray if required
996             if (gray)
997             {
998                 cvtColor(imageFlipped, imageDisplay, CV_BGR2GRAY);
999             }
1000            /*
1001             * else imageDisplay.data is already == imageFlipped.data
1002             */
1003            image_updated = true;
1004        }
1005
1006        if (updateThread ≠ NULL)
1007        {
1008            lockLevel--;
1009            if (lockLevel ≡ 0)
1010            {
1011                mutex.unlock();
1012            }
1013        }
1014
1015        if (image_updated)
1016        {
1017            emit updated();
1018        }
1019    }
1020    else
1021    {
1022        // mutex hasn't been locked, so we skipped one capture
1023        // qDebug() << "Capture skipped an image (level " << lockLevel << ")";
1024    }
1025 }
```

```
/*
 * CaptureFactory.cpp
 *
 *  Created on: 11 fÃ©vr. 2012
 *      Author: davidroussel
 */

#include <cstdlib>  // for NULL
#include <QDebug>
#include <QFile>
#include <QtGlobal>
#include <QStringListIterator>
#include "CaptureFactory.h"

/*
 * Capture Factory constructor.
 * Arguments can be
 *  - [-d | --device] <device number> : camera number
 *  - [-f | --file] <filename> : video file name
 *  - [-m | --mirror] : flip image horizontally
 *  - [-g | --gray] : convert to gray level
 *  - [-s | --size] <width>x<height>: preferred width and height
 * @param argList program the argument list provided as a list of
 * strings
 */
CaptureFactory::CaptureFactory(const QStringList & argList) :
    capture(NULL),
    deviceNumber(0),
    liveVideo(true),
    flippedVideo(false),
    grayVideo(false),
    skipImages(false),
    preferredWidth(0),
    preferredHeight(0),
    videoPath()
{
    // C++ Like iterator
    // for (QStringList::const_iterator it = argList.begin(); it != argList.end(); ++it)
    // Java like iterator (because we use hasNext multiple times)
    for (QListIterator<QString> it(argList); it.hasNext(); )
    {
        QString currentArg(it.next());

        if (currentArg == "-d" || currentArg =="--device")
        {
            // Next argument should be device number integer
            if (it.hasNext())
            {
                QString deviceString(it.next());
                bool convertOk;
                deviceNumber = deviceString.toInt(&convertOk,10);
                if (!convertOk || deviceNumber < 0)
                {
                    qWarning("Warning: Invalid device number %d",deviceNumber);
                    deviceNumber = 0;
                }
                liveVideo = true;
            }
            else
            {
                qWarning("Warning: device tag found with no following device number");
            }
        }
        else if (currentArg == "-v" || currentArg == "--video")
        {
            // Next argument should be a path name to video file or URL
            if (it.hasNext())
            {
                videoPath = it.next();
                liveVideo = false;
            }
            else
            {
                qWarning("file tag found with no following filename");
            }
        }
        else if (currentArg == "-m" || currentArg == "--mirror")
        {
            flippedVideo = true;
        }
        else if (currentArg == "-g" || currentArg == "--gray")
        {
            grayVideo = true;
        }
        else if (currentArg == "-k" || currentArg == "--skip")
        {
            skipImages = true;
        }
        else if (currentArg == "-s" || currentArg == "--size")
        {
            if (it.hasNext())
            {
                // search for <width>x<height>
                QString sizeString = it.next();
                int xIndex = sizeString.indexOf(QChar('x'), 0,
                    Qt::CaseInsensitive);
                if (xIndex != -1)
                {
                    QString widthString = sizeString.left(xIndex);
                    preferredWidth = widthString.toUInt();
                    qDebug("preferred width is %d", preferredWidth);

                    QString heightString = sizeString.remove(0, xIndex+1);
                    preferredHeight = heightString.toUInt();
                    qDebug("preferred height is %d", preferredHeight);
                }
                else
                {
                    qWarning("invalid <width>x<height>");
                }
            }
            else
            {
                qWarning("size not found after --size");
            }
        }
    }
}

/*
 * Capture factory destructor
 */
CaptureFactory::~CaptureFactory()
{
}

/*
 * Set the capture to live (webcam) or file source
 * @param live the video source
 */
void CaptureFactory::setLiveVideo(const bool live)
{
    liveVideo = live;
}

/*
 * Set device number to use when instanciating the capture with
 * live video.
 * @param deviceNumber the device number to use
 */
void CaptureFactory::setDeviceNumber(const int deviceNumber)
{
    if (deviceNumber >= 0)
    {
        this->deviceNumber = deviceNumber;
    }
    else
    {
        qWarning("CaptureFactory::setDeviceNumber: invalid number %d", deviceNumber);
    }
}

/*
 * Set path to video file when #liveVideo is false
 * @param path the path to the video file source
 */
void CaptureFactory::setFile(const QString & path)
{
    if (QFile::exists(path))
    {
        videoPath = path;
    }
    else
    {
        qWarning() << QObject::tr("CaptureFactory::setFile: path") << path
                   << QObject::tr(" does not exist");
    }
}

/*
 * Set video horizontal flip state (useful for selfies)
 * @param flipped the horizontal flip state
 */
void CaptureFactory::setFlipped(const bool flipped)
{
    flippedVideo = flipped;
}

/*
 * Set gray conversion
```

```
 * @param gray the gray conversion state
 */
void CaptureFactory::setGray(const bool gray)
{
    grayVideo = gray;
}

/*
 * Set video grabbing skippable. When true, grabbing is skipped when
 * previously grabbed image has not been processed yet. Otherwise,
 * grabbing new image wait for the previous image to be processed.
 * This only applies if capture is run in a separate thread.
 * @param skip the video grabbing skippable state
 */
void CaptureFactory::setSkippable(const bool skip)
{
    skipImages = skip;
}

/*
 * Set video size (independently of video source actual size)
 * @param width the desired image width
 * @param height the desired image height
 */
void CaptureFactory::setSize(const size_t width, const size_t height)
{
    preferredWidth = (int)width;
    preferredHeight = (int)height;
}

/*
 * Set video size (independently of video source actual size)
 * @param size the desired video size
 */
void CaptureFactory::setSize(const QSize & size)
{
    preferredWidth = size.width();
    preferredHeight = size.height();
}

/*
 * Provide capture instanciated according to values
 * extracted from argument lists
 * @param updateThread the thread to run this capture or NULL if this
 * capture run in the current thread
 * @return the new capture instance
 */
QcvVideoCapture * CaptureFactory::getCaptureInstance(QThread * updateThread)
{
    // ------------------------------------------------------------------------
    // Opening Video Capture
    // ------------------------------------------------------------------------
    if (liveVideo)
    {
        qDebug() << "opening device # " << deviceNumber;
    }
    else
    {
        qDebug() << "opening video file " << videoPath;
    }

    qDebug() << "Opening ";
    if (liveVideo)
    {
        // Live video feed
        qDebug() << "Live Video ... from camera # " << deviceNumber;
        capture = new QcvVideoCapture(deviceNumber,
                                      flippedVideo,
                                      grayVideo,
                                      skipImages,
                                      preferredWidth,
                                      preferredHeight,
                                      updateThread);
    }
    else
    {
        // Video file or stream
        qDebug() << videoPath << " ... ";
        capture = new QcvVideoCapture(videoPath,
                                      flippedVideo,
                                      grayVideo,
                                      skipImages,
                                      preferredWidth,
                                      preferredHeight,
                                      updateThread);
    }

    return capture;
}
```



```
#include <cmath>
#include <opencv2/core/core.hpp>    // for MeanValue<cv::Mat, cv::Mat> specialization

#include "MeanValue.h"

/*
 * Constructor.
 * Initialize sum & sum2 to T(0) and count to 0
 * @param initialValue [optional] a T specimen can be provided in order
 * to initialise sum and sum2 by copying the specimen
 * @param initialMinimum [optional] initial value of minimum and minimum
 * reset value
 */
template <typename T, typename R>
MeanValue<T, R>::MeanValue(const T & initialValue,
                           const T & initialMinimum) :
    sum(initialValue),
    sum2(initialValue),
    count(0),
    minValue(initialMinimum),
    maxValue(initialValue),
    resetMinValue(initialMinimum),
    resetMaxValue(initialValue)
{
}

/*
 * Copy constructor
 * @param mv the other mean value to copy
 */
template <typename T, typename R>
MeanValue<T, R>::MeanValue(const MeanValue<T, R> & mv) :
    sum(mv.sum),
    sum2(mv.sum2),
    count(mv.count),
    minValue(mv.minValue),
    maxValue(mv.maxValue),
    resetMinValue(mv.resetMinValue),
    resetMaxValue(mv.resetMaxValue)
{
}

/*
 * Move constructor
 * @param mv the other mean value to copy
 */
template <typename T, typename R>
MeanValue<T, R>::MeanValue(MeanValue<T, R> ∧ mv) :
    sum(mv.sum),
    sum2(mv.sum2),
    count(mv.count),
    minValue(mv.minValue),
    maxValue(mv.maxValue),
    resetMinValue(mv.resetMinValue),
    resetMaxValue(mv.resetMaxValue)
{
}

/*
 * Destructor
 */
template <typename T, typename R>
MeanValue<T, R>::~MeanValue()
{
}

/*
 * Function call operator
 * @param value value to add to the values sum and values square sum
 * @post elements count has been increased
 */
template <typename T, typename R>
void MeanValue<T, R>::operator ()(const T & value)
{
    sum += value;
    sum2 += value * value;
    count++;
    if (value > maxValue)
    {
        maxValue = value;
    }
    if (value < minValue)
    {
        minValue = value;
    }
}

/*
 * Self increment operator
 * @param value value to add to the values sum and values square sum
```

```
 * @post elements count has been increased
 * @note does the same thing as Function call operator
 */
template <typename T, typename R>
void MeanValue<T, R>::operator +=(const T & value)
{
    operator()(value);
}

/*
 * Copy operator from another mean value
 * @param mv the mean value to copy
 * @return a reference to the current mean value
 */
template <typename T, typename R>
MeanValue<T, R> & MeanValue<T, R>::operator =(const MeanValue<T, R> & mv)
{
    sum = mv.sum;
    sum2 = mv.sum2;
    count = mv.count;
    minValue = mv.minValue;
    maxValue = mv.maxValue;
    // can't copy resetMinValue & resetMaxValue 'cause they're constants

    return *this;
}

/*
 * Move operator from another mean value
 * @param mv the mean value to move
 * @return a reference to the current mean value
 */
template <typename T, typename R>
MeanValue<T, R> & MeanValue<T, R>::operator =(MeanValue<T, R> ^ mv)
{
    sum = mv.sum;
    sum2 = mv.sum2;
    count = mv.count;
    minValue = mv.minValue;
    maxValue = mv.maxValue;
    // can't copy resetMinValue & resetMaxValue 'cause they're constants

    return *this;
}

/*
 * Cast operator to result type
 * @return the mean value
 */
template <typename T, typename R>
MeanValue<T, R>::operator R() const
{
    return mean();
}

/*
 * Compute mean value : E(X) = sum/nbElements
 * @return the mean value of all added elements.
 */
template <typename T, typename R>
R MeanValue<T, R>::mean() const
{
    if (count ≠ 0)
    {
        return R(sum / (R) count);
    }
    else
    {
        return R(0);
    }
}

/*
 * Compute standard deviation of values : sqrt(E(X^2) - E(X)^2)
 * @return the standard deviation of all added elements.
 */
template <typename T, typename R>
R MeanValue<T, R>::std() const
{
    if (count ≠ 0)
    {
        R ex = mean();
        double ex2 = sum2 / (double) count;
        return R(sqrt(ex2 - double(ex * ex)));
    }
    else
    {
        return R(0);
    }
}
```



```

/*
 * Minimum recorded value accessor
 * @return the minimum recorded value (until reset)
 */
template <typename T, typename R>
T MeanValue<T, R>::min() const
{
    if (count ≠ 0)
    {
        return minValue;
    }
    else
    {
        return T(0);
    }
}

/*
 * Maximum recorded value accessor
 * @return the maximum recorded value (until reset)
 */
template <typename T, typename R>
T MeanValue<T, R>::max() const
{
    if (count ≠ 0)
    {
        return maxValue;
    }
    else
    {
        return T(0);
    }
}

/*
 * Reset added values, square values and count to 0
 */
template <typename T, typename R>
void MeanValue<T, R>::reset()
{
    sum = T(0);
    sum2 = T(0);
    count = 0;
    minValue = resetMinValue;
    maxValue = resetMaxValue;
}

/*
 * Output operator for MeanValue
 * @param out the output stream
 * @param mv the MeanValue to print on the output stream
 * @return a reference to the current output stream
 * @post put mean value Â± std value on the stream
 */
template <typename T, typename R>
ostream & operator <<(ostream & out, const MeanValue<T, R> & mv)
{
    out << mv.mean() << "Â±" << mv.std() << "[" << mv.min() << "…"
        << mv.max() << "]";

    return out;
}

// ----------------------------------------------------------------------------
// Specializations for MeanValue<cv::Mat, cv::Mat>
// ----------------------------------------------------------------------------

/**
 * Function call operator (specialization for MeanValue<cv::Mat, cv::Mat>)
 * @param value value to add to the values sum and values square sum
 * @post elements count has been increased
 */
template <>
void MeanValue<cv::Mat>::operator ()(const cv::Mat & value)
{
    sum += value;
    sum2 += value * value.t();
    count++;
    int rows = value.rows;
    int cols = value.cols;
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            /*
             * FIXME Caution accessing pixels values in double only works
             * with matrices of double
             */
            double & currentMin = minValue.at<double>(i, j);
```

```
271             double & currentMax = maxValue.at<double>(i, j);
272             double currentValue = value.at<double>(i, j);
273             if (currentValue < currentMin)
274             {
275                 currentMin = currentValue;
276             }
277             if (currentValue > currentMax)
278             {
279                 currentMax = currentValue;
280             }
281         }
282     }
283 }
284
285 /**
286  * Compute mean value (specialization for MenValue<cv::Mat, cv::Mat>):
287  * E(X) = sum/nbElements
288  * @return the mean value of all added elements.
289  */
290 template <>
291 cv::Mat MeanValue<cv::Mat>::mean() const
292 {
293     if (count ≠ 0)
294     {
295         return cv::Mat(sum * double(1.0/(double)count));
296     }
297     else
298     {
299         return cv::Mat(sum * double(0));
300     }
301 }
302
303 /**
304  * Compute standard deviation of values (specialization for
305  * MeanValue<cv::Mat; cv::Mat>): sqrt(E(X^2) - E(X)^2)
306  * @return the standard deviation of all added elements.
307  */
308 template <>
309 cv::Mat MeanValue<cv::Mat>::std() const
310 {
311     if (count ≠ 0)
312     {
313         cv::Mat ex = mean();
314         cv::Mat ex2 = sum2 * double(1.0 / (double) count);
315         int rows = sum.rows;
316         int cols = sum.cols;
317         cv::Mat result(rows, cols, CV_64FC1);
318
319         for (int i = 0; i < rows; i++)
320         {
321             for (int j = 0; j < cols; j++)
322             {
323                 double exij = ex.at<double>(i,j);
324                 result.at<double>(i,j) = sqrt( ex2.at<double>(i,j) - (exij * exij) );
325             }
326         }
327
328         return result;
329     }
330     else
331     {
332         return cv::Mat(sum2 * double(0.0));
333     }
334 }
335
336 /**
337  * Minimum recorded value accessor (specialization for
338  * MeanValue<cv::Mat; cv::Mat>)
339  * @return the minimum recorded value (until reset)
340  */
341 template <>
342 cv::Mat  MeanValue<cv::Mat>::min() const
343 {
344     if (count ≠ 0)
345     {
346         return minValue;
347     }
348     else
349     {
350         return cv::Mat();
351     }
352 }
353
354 /**
355  * Maximum recorded value accessor (specialization for
356  * MeanValue<cv::Mat; cv::Mat>)
357  * @return the maximum recorded value (until reset)
358  */
359 template <>
360 cv::Mat MeanValue<cv::Mat>::max() const
```



```
361 {
362     if (count ≠ 0)
363     {
364         return maxValue;
365     }
366     else
367     {
368         return cv::Mat();
369     }
370 }
371
372 /**
373  * Reset added values (specialization for MeanValue<cv::Mat, cv::Mat>),
374  * square values and count to 0
375  */
376 template <>
377 void MeanValue<cv::Mat>::reset()
378 {
379     sum *= double(0);
380     sum2 *= double(0);
381     count = 0;
382     minValue = resetMinValue;
383     maxValue = resetMaxValue;
384 }
385
386 // ----------------------------------------------------------------------------
387 // Template protoinstanciations for
388 //  - int
389 //  - clock_t (unsigned long)
390 //  - float
391 //  - double
392 //  - cv::Mat
393 //  - Pose
394 // ----------------------------------------------------------------------------
395
396 // Proto instanciations
397 template class MeanValue<int, double>;
398 template class MeanValue<clock_t, double>;
399 template class MeanValue<float, double>;
400 template class MeanValue<double>;
401 template class MeanValue<int, float>;
402 template class MeanValue<clock_t, float>;
403 template class MeanValue<float>;
404 template class MeanValue<double, float>;
405 template class MeanValue<cv::Mat>;
406
407 // Output operators proto-instanciations
408 template ostream & operator << (ostream &, const MeanValue<int, double> &);
409 template ostream & operator << (ostream &, const MeanValue<clock_t, double> &);
410 template ostream & operator << (ostream &, const MeanValue<float, double> &);
411 template ostream & operator << (ostream &, const MeanValue<double> &);
412 template ostream & operator << (ostream &, const MeanValue<int, float> &);
413 template ostream & operator << (ostream &, const MeanValue<clock_t, float> &);
414 template ostream & operator << (ostream &, const MeanValue<float> &);
415 template ostream & operator << (ostream &, const MeanValue<double, float> &);
416 template ostream & operator << (ostream &, const MeanValue<cv::Mat> &);
```

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "QcvVideoCapture.h"
#include "QcvGFilter.h"

/**
 * Namespace for generated UI
 */
namespace Ui {
    class MainWindow;
}

/**
 * Channels index 2 Widget index conversion
 */
static const CvGFilter::Channels RGB[3] = {CvGFilter::RED,
                                           CvGFilter::GREEN,
                                           CvGFilter::BLUE};

/**
 * OpenCV/Qt Application Main window
 */
class MainWindow : public QMainWindow
{
    Q_OBJECT

    public:
        /**
         * Rendering mode for main image
         */
        typedef enum
        {
            RENDER_IMAGE = 0,//!< QImage rendering mode
            RENDER_PIXMAP,   //!< QPixmap in a QLabel rendering mode
            RENDER_GL        //!< OpenGL in a QGLWidget rendering mode
        } RenderMode;

        /**
         * MainWindow constructor.
         * @param capture the capture QObject to capture frames from devices
         * or video files
         * @param processor the openCV image processor
         * @param parent parent widget
         */
        explicit MainWindow(QcvVideoCapture * capture,
                            QcvGFilter * processor,
                            QWidget *parent = NULL);

        /**
         * MainWindow destructor
         */
        virtual ~MainWindow();

    signals:
        /**
         * Signal to send update message when something changes
         * @param message the message
         * @param timeout number of ms the message should be displayed
         */
        void sendMessage(const QString & message, int timeout = 0);

        /**
         * Signal to send when video size change is requested
         * @param size the new video size
         */
        void sizeChanged(const QSize & size);

        /**
         * Signal to send for opening a device (camera) with the capture
         * @param deviceId device number to open
         * @param width desired width or 0 to keep capture width
         * @param height desired height or 0 to keep capture height
         * @return true if device has been opened and checked and timer launched
         */
        void deviceChanged(const int deviceId,
                           const unsigned int width,
                           const unsigned int height);

        /**
         * Signal to send for opening a video file in the capture
         * @param fileName video file to open
         * @param width desired width or 0 to keep capture width
         * @param height desired height or 0 to keep capture height
         * @return true if video has been opened and timer launched
         */
        void fileChanged(const QString & fileName,
                         const unsigned int width,
                         const unsigned int height);
```



```

        /**
         * Signal to send when requesting video flip
         * @param flip the video flip status
         */
        void flipChanged(const bool flip);

        /**
         * Signal to send when requesting gray changed
         * @param gray the gray status
         */
        void grayChanged(const bool gray);

    private:
        /**
         * The UI built in QtDesigner or QtCreator
         */
        Ui::MainWindow *ui;

        /**
         * The Capture object grabs frame using OpenCV HiGui
         */
        QcvVideoCapture * capture;

        /**
         * The filter processor
         */
        QcvGFilter * processor;

        /**
         * Image preferred width
         */
        int preferredWidth;

        /**
         * Image preferred height
         */
        int preferredHeight;

        /**
         * Message to send to statusBar
         */
        QString message;

        /**
         * Changes widgetImage nature according to desired rendering mode.
         * Possible values for mode are:
         *  - IMAGE: widgetImage is assigned to a QcvMatWidgetImage instance
         *  - PIXMAP: widgetImage is assigned to a QcvMatWidgetLabel instance
         *  - GL: widgetImage is assigned to a QcvMatWidgetGL instance
         * @param mode
         */
        void setRenderingMode(const RenderMode mode);

        /**
         * Setup UI values from capture settings
         */
        void setupUIfromCatpure();

        /**
         * Setup UI values from processor settings
         */
        void setupUIfromProcessor();

        /**
         * Setup Sigma slider and double spinbox according to processor
         * values
         */
        void setupSigma();

        /**
         * Convert value of double spin box to integer slider
         * @param min minimum value of the double spin box
         * @param step step value of the double spin box
         * @param value current value of the double spin box or max value to
         * obtain integer slider max value
         * @return the integer value to set on the integer slider
         */
        static int double2intValue(const double min,
                                   const double step,
                                   const double value);

        /**
         * Convert integer value from slider to double value of double spin box
         * @param dmin the minimum double value of the spin box
         * @param dstep the step value of the double spin box
         * @param ivalue the integer value of the integer slider
         * @return the value of the double spin box
         */
        static double int2doubleValue(const double dmin,
```

```
                                const double dstep,
                                const int ivalue);

    private slots:

        /**
         * Re setup processor from UI settings when source image changes
         */
        void setupProcessorFromUI();

        /**
         * Menu action when Sources->camera 0 is selected
         * Sets capture to open device 0. If device is not available
         * menu item is set to inactive.
         */
        void on_actionCamera_0_triggered();

        /**
         * Menu action when Sources->camera 1 is selected
         * Sets capture to open device 0. If device is not available
         * menu item is set to inactive
         */
        void on_actionCamera_1_triggered();

        /**
         * Menu action when Sources->file is selected.
         * Opens file dialog and tries to open selected file (is not empty),
         * then sets capture to open the selected file
         */
        void on_actionFile_triggered();

        /**
         * Menu action to quit application.
         */
        void on_actionQuit_triggered();

        /**
         * Menu action when flip image is selected.
         * Sets capture to change flip status which leads to reverse
         * image horizontally
         */
        void on_actionFlip_triggered();

        /**
         * Menu action when original image size is selected.
         * Sets capture not to resize image
         */
        void on_actionOriginalSize_triggered();

        /**
         * Menu action when constrained image size is selected.
         * Sets capture resize to preferred width and height
         */
        void on_actionConstrainedSize_triggered();

        /**
         * Menu action to replace current image rendering widget by a
         * QcvMatWidgetImage instance.
         */
        void on_actionRenderImage_triggered();

        /**
         * Menu action to replace current image rendering widget by a
         * QcvMatWidgetLabel with pixmap instance.
         */
        void on_actionRenderPixmap_triggered();

        /**
         * Menu action to replace current image rendering widget by a
         * QcvMatWidgetGL instance.
         */
        void on_actionRenderOpenGL_triggered();

        /**
         * Original size radioButton action.
         * Sets capture resize to off
         */
        void on_radioButtonOrigSize_clicked();

        /**
         * Custom size radioButton action.
         * Sets capture resize to preferred width and height
         */
        void on_radioButtonCustomSize_clicked();

        /**
         * Width spinbox value change.
         * Changes the preferred width and if custom size is selected apply
         * this custom width
         * @param value the desired width
```



```
         */
        void on_spinBoxWidth_valueChanged(int value);

        /**
         * Height spinbox value change.
         * Changes the preferred height and if custom size is selected apply
         * this custom height
         * @param value the desired height
         */
        void on_spinBoxHeight_valueChanged(int value);

        /**
         * Flip capture image horizontally.
         * changes capture flip status
         */
        void on_checkBoxFlip_clicked();

        /**
         * Kernel size changed in spin box
         * @param value the new kernel size
         */
        void on_spinBoxKernel_valueChanged(int value);

        /**
         * Sigma gaussian variance changed in double spin box
         * @param dvalue the new sigma value
         * @post the corresponding integer value is computed and applied
         * to the integer sigma slider
         */
        void on_doubleSpinBoxSigma_valueChanged(double dvalue);

        /**
         * Threshold value changed in Threshold spinbox
         * @param value the new threshold value
         */
        void on_spinBoxThreshold_valueChanged(int value);

        /**
         * Kappa harris parameter changed in double spinbox
         * @param dvalue the new kappa value
         * @post the corresponding integer value is computed and applied
         * to the integer kappa slider
         */
        void on_doubleSpinBoxKappa_valueChanged(double dvalue);

        /**
         * Sigma slider value changed
         * @param ivalue the new Sigma integer value
         * @post the corresponding double value is computed and applied
         * to the Sigma double spinbox
         */
        void on_horizontalSliderSigma_valueChanged(int ivalue);

        /**
         * Sigma slider value changed
         * @param ivalue the new Sigma integer value
         * @post the corresponding double value is computed and applied
         * to the Sigma double spinbox
         */
        void on_horizontalSliderKappa_valueChanged(int ivalue);

        /**
         * Display image selection
         * @param index the new selected image index
         */
        void on_comboBoxImages_currentIndexChanged(int index);

        /**
         * Edge mode selection
         * @param index the new edge mode index
         */
        void on_comboBoxEdges_currentIndexChanged(int index);
};

#endif // MAINWINDOW_H
```

```cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include <QObject>
#include <QFileDialog>
#include <QWindow>
#include <QDebug>

#include <assert.h>

#include "QcvMatWidgetImage.h"
#include "QcvMatWidgetLabel.h"
#include "QcvMatWidgetGL.h"

/*
 * MainWindow constructor.
 * @param capture the capture QObject to capture frames from devices
 * or video files
 * @param processor the openCV image processor
 * @param parent parent widget
 */
MainWindow::MainWindow(QcvVideoCapture * capture,
                       QcvGFilter * processor,
                       QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow),
    capture(capture),
    processor(processor),
    preferredWidth(320),
    preferredHeight(240)
{
    ui->setupUi(this);
    ui->scrollAreaSource->setBackgroundRole(QPalette::Mid);

    // ------------------------------------------------------------------------
    // Assertions
    // ------------------------------------------------------------------------
    assert(capture != NULL);

    assert(processor != NULL);

    // ------------------------------------------------------------------------
    // Special widgets initialisation
    // ------------------------------------------------------------------------
    // Replace QcvMatWidget instances with QcvMatWidgetImage instances
    // sets image widget sources for the first time
    // connects processor->update to image Widgets->updated
    // connects processor->image changed to image widgets->setSourceImage
    setRenderingMode(RENDER_IMAGE);

    // ------------------------------------------------------------------------
    // rest of Signal/Slot connections
    // ------------------------------------------------------------------------

    // Capture, processor and this messages to status bar
    connect(capture, SIGNAL(messageChanged(QString,int)),
            ui->statusBar, SLOT(showMessage(QString,int)));

    connect(processor, SIGNAL(sendMessage(QString,int)),
            ui->statusBar, SLOT(showMessage(QString,int)));

    connect(this, SIGNAL(sendMessage(QString,int)),
            ui->statusBar, SLOT(showMessage(QString,int)));

    // When Processor source image changes, some attributes are reinitialised
    // So we have to set them up again according to current UI values
    connect(processor, SIGNAL(imageChanged()),
            this, SLOT(setupProcessorFromUI()));

    // Connects UI requests to capture
    connect(this, SIGNAL(sizeChanged(const QSize &)),
            capture, SLOT(setSize(const QSize &)), Qt::DirectConnection);
    connect(this, SIGNAL(deviceChanged(int,uint,uint)),
            capture, SLOT(open(int,uint,uint)), Qt::DirectConnection);
    connect(this, SIGNAL(fileChanged(QString,uint,uint)),
            capture, SLOT(open(QString,uint,uint)), Qt::DirectConnection);
    connect(this, SIGNAL(flipChanged(bool)),
            capture, SLOT(setFlipVideo(bool)), Qt::DirectConnection);
    connect(this, SIGNAL(grayChanged(bool)),
            capture, SLOT(setGray(bool)), Qt::DirectConnection);

    // ------------------------------------------------------------------------
    // UI setup according to capture and processor options
    // ------------------------------------------------------------------------
    setupUIfromCatpure();
    setupUIfromProcessor();
}

/*
 * MainWindow destructor
```



```cpp
 */
MainWindow::~MainWindow()
{
    delete ui;
}

/*
 * Changes widgetImage nature according to desired rendering mode.
 * Possible values for mode are:
 *  - IMAGE: widgetImage is assigned to a QcvMatWidgetImage instance
 *  - PIXMAP: widgetImage is assigned to a QcvMatWidgetLabel instance
 *  - GL: widgetImage is assigned to a QcvMatWidgetGL instance
 * @param mode
 */
void MainWindow::setRenderingMode(const RenderMode mode)
{
    // Disconnect signals from slots first
    disconnect(processor, SIGNAL(updated()),
               ui->sourceImage, SLOT(update()));

    disconnect(processor, SIGNAL(imageChanged(Mat*)),
               ui->sourceImage, SLOT(setSourceImage(Mat*)));

    QWindow * currentWindow = windowHandle();
    if (mode == RENDER_GL)
    {
        disconnect(currentWindow,
                   SIGNAL(screenChanged(QScreen *)),
                   ui->sourceImage,
                   SLOT(screenChanged()));
    }

    // remove widgets in scroll areas
    QWidget * wSource = ui->scrollAreaSource->takeWidget();

    if (wSource == ui->sourceImage)
    {
        // delete removed widgets
        delete ui->sourceImage;

        // create new widget
        Mat * sourceMat = processor->getImagePtr("source");

        switch (mode)
        {
            case RENDER_PIXMAP:
                ui->sourceImage = new QcvMatWidgetLabel(sourceMat);
                break;
            case RENDER_GL:
                ui->sourceImage = new QcvMatWidgetGL(sourceMat);
                break;
            case RENDER_IMAGE:
            default:
                ui->sourceImage = new QcvMatWidgetImage(sourceMat);
                break;
        }

        if (ui->sourceImage != NULL)
        {
            // Name the new images widgets with same name as in UI files
            ui->sourceImage->setObjectName(QString::fromUtf8("sourceImage"));

            // add to scroll areas
            ui->scrollAreaSource->setWidget(ui->sourceImage);

            // Reconnect signals to slots
            connect(processor, SIGNAL(updated()),
                    ui->sourceImage, SLOT(update()));

            connect(processor, SIGNAL(imageChanged(Mat*)),
                    ui->sourceImage, SLOT(setSourceImage(Mat*)));

            if (mode == RENDER_GL)
            {
                connect(currentWindow,
                        SIGNAL(screenChanged(QScreen *)),
                        ui->sourceImage,
                        SLOT(screenChanged()));
            }

            // Sends message to status bar and sets menu checks
            message.clear();
            message.append(tr("Render more set to "));
            switch (mode)
            {
                case RENDER_IMAGE:
                    ui->actionRenderPixmap->setChecked(false);
                    ui->actionRenderOpenGL->setChecked(false);
                    message.append(tr("QImage"));
                    break;
```

```
                    case RENDER_PIXMAP:
                        ui->actionRenderImage->setChecked(false);
                        ui->actionRenderOpenGL->setChecked(false);
                        message.append(tr("QPixmap in QLabel"));
                        break;
                    case RENDER_GL:
                        ui->actionRenderImage->setChecked(false);
                        ui->actionRenderPixmap->setChecked(false);
                        message.append(tr("QGLWidget"));
                        break;
                    default:
                    break;
                }
                emit sendMessage(message, 5000);
            }
            else
            {
                qDebug("MainWindow::on_actionRenderXXX some new widget is null");
            }
        }
        else
        {
            qDebug("MainWindow::on_actionRenderXXX removed widget is not in ui->");
        }
}

/*
 * Setup UI values from capture settings
 */
void MainWindow::setupUIfromCatpure()
{
    // ----------------------------------------------------------------------
    // UI setup according to capture options
    // ----------------------------------------------------------------------
    // Sets size radioButton states
    if (capture->isResized())
    {
        /*
         * Initial Size radio buttons configuration
         */
        ui->radioButtonOrigSize->setChecked(false);
        ui->radioButtonCustomSize->setChecked(true);
        /*
         * Initial Size menu items configuration
         */
        ui->actionOriginalSize->setChecked(false);
        ui->actionConstrainedSize->setChecked(true);

        QSize size = capture->getSize();
        qDebug("Capture->size is %dx%d", size.width(), size.height());
        preferredWidth = size.width();
        preferredHeight = size.height();
    }
    else
    {
        /*
         * Initial Size radio buttons configuration
         */
        ui->radioButtonCustomSize->setChecked(false);
        ui->radioButtonOrigSize->setChecked(true);

        /*
         * Initial Size menu items configuration
         */
        ui->actionConstrainedSize->setChecked(false);
        ui->actionOriginalSize->setChecked(true);
    }

    // Sets spinboxes preferred size
    ui->spinBoxWidth->setValue(preferredWidth);
    ui->spinBoxHeight->setValue(preferredHeight);

    // Sets flipCheckbox and menu item states
    bool flipped = capture->isFlipVideo();
    ui->actionFlip->setChecked(flipped);
    ui->checkBoxFlip->setChecked(flipped);
}

/*
 * Setup UI values from processor attributes
 */
void MainWindow::setupUIfromProcessor()
{
    ui->comboBoxImages->setCurrentIndex((int)processor->getDisplayMode());
    ui->comboBoxEdges->setCurrentIndex((int)processor->getEdgeMode());

    // Kernel
    ui->labelKernelMin->setText(QString::number(processor->getMinKernelSize()));
    ui->labelKernelMax->setText(QString::number(processor->getMaxKernelSize()));
    ui->spinBoxKernel->setValue(processor->getKernelSize());
```



```

    // Sigma
    setupSigma();

    // Threshold
    ui->horizontalSliderThreshold->setMinimum(processor->getMinThreshold());
    ui->horizontalSliderThreshold->setMaximum(processor->getMaxThreshold());
    ui->horizontalSliderThreshold->setSingleStep(1);
    ui->horizontalSliderThreshold->setValue(processor->getThresholdLevel());

    // Kappa
    double kappaMin = QcvGFilter::getHarrisKappaMin();
    double kappaMax = QcvGFilter::getHarrisKappaMax();
    double kappaStep = QcvGFilter::getHarrisKappaStep();
    double kappa = processor->getHarrisKappa();
//  qDebug("Kappa : %f [%f:%f:%f]",kappa, kappaMin, kappaStep, kappaMax);

    ui->labelKappaMin->setText(QString::number(kappaMin,'f',2));
    ui->labelKappaMax->setText(QString::number(kappaMax,'f',2));

    ui->doubleSpinBoxKappa->setMinimum(kappaMin);
    ui->doubleSpinBoxKappa->setMaximum(kappaMax);
    ui->doubleSpinBoxKappa->setSingleStep(kappaStep);
    ui->doubleSpinBoxKappa->setValue(kappa);

    ui->horizontalSliderKappa->setMinimum(0);
    ui->horizontalSliderKappa->setMaximum(
        double2intValue(kappaMin,
                        kappaStep,
                        kappaMax));
    ui->horizontalSliderKappa->setSingleStep(1);
    ui->horizontalSliderKappa->setValue(
        double2intValue(kappaMin,
                        kappaStep,
                        kappa));

}

/*
 * Setup Sigma slider and double spinbox according to processor
 * values
 */
void MainWindow::setupSigma()
{
    double sigmaMin = processor->getMinSigma();
    double sigmaMax = processor->getMaxSigma();
    double sigmaStep = QcvGFilter::getSigmaStep();
    double sigma = processor->getSigma();

//  qDebug("Sigma : %f [%f:%f:%f]",sigma, sigmaMin, sigmaStep, sigmaMax);

    ui->labelSigmaMin->setText(QString::number(sigmaMin,'f',2));
    ui->labelSigmaMax->setText(QString::number(sigmaMax,'f',2));

    ui->doubleSpinBoxSigma->setMinimum(sigmaMin);
    ui->doubleSpinBoxSigma->setMaximum(sigmaMax);
    ui->doubleSpinBoxSigma->setSingleStep(sigmaStep);
    ui->doubleSpinBoxSigma->setValue(sigma);

    ui->horizontalSliderSigma->setMinimum(0);
    ui->horizontalSliderSigma->setMaximum(
        double2intValue(sigmaMin,
                        sigmaStep,
                        sigmaMax));
    ui->horizontalSliderSigma->setSingleStep(1);
    ui->horizontalSliderSigma->setValue(
        double2intValue(sigmaMin,
                        sigmaStep,
                        sigma));
}

/*
 * Re setup processor from UI settings when source changes
 */
void MainWindow::setupProcessorFromUI()
{
    processor->setDisplayMode((CvGFilter::ImageDisplay)ui->comboBoxImages->currentIndex());
    processor->setEdgeMode((CvGFilter::EdgeDisplay)ui->comboBoxEdges->currentIndex());
    processor->setKernelSize(ui->spinBoxKernel->value());
    setupSigma();
    processor->setThresholdLevel(ui->spinBoxThreshold->value());
    processor->setHarrisKappa(ui->doubleSpinBoxKappa->value());
}

/*
 * Convert value of double spin box to integer slider
 * @param min minimum value of the double spin box
 * @param step step value of the double spin box
 * @param value current value of the double spin box or max value to obtain
 * integer slider max value
```

```
 * @return the integer value to set on the integer slider
 */
int MainWindow::double2intValue(const double min,
                                const double step,
                                const double value)
{
    return (int)((value - min)/step)+1;
}

/*
 * Convert integer value from slider to double value of double spin box
 * @param dmin the minimum double value of the spin box
 * @param dstep the step value of the double spin box
 * @param ivalue the integer value of the integer slider
 * @return the value of the double spin box
 */
double MainWindow::int2doubleValue(const double dmin,
                                   const double dstep,
                                   const int ivalue)
{
    return (dmin + ((double)ivalue * dstep));
}

/*
 * Menu action when Sources->camera 0 is selected
 * Sets capture to open device 0. If device is not available
 * menu item is set to inactive.
 */
void MainWindow::on_actionCamera_0_triggered()
{
    int width = 0;
    int height = 0;

    if (ui->radioButtonCustomSize->isChecked())
    {
        width = preferredWidth;
        height = preferredHeight;
    }

    qDebug("Opening device 0 ...");
//  if (!capture->open(0, width, height))
//  {
//      qWarning("Unable to open device 0");
//      // disable menu item if camera 0 does not exist
//      ui->actionCamera_0->setDisabled(true);
//  }

    emit deviceChanged(0, width, height);
}

/*
 * Menu action when Sources->camera 1 is selected
 * Sets capture to open device 0. If device is not available
 * menu item is set to inactive
 */
void MainWindow::on_actionCamera_1_triggered()
{
    int width = 0;
    int height = 0;

    if (ui->radioButtonCustomSize->isChecked())
    {
        width = preferredWidth;
        height = preferredHeight;
    }

    qDebug("Opening device 1 ...");
//  if (!capture->open(1, width, height))
//  {
//      qWarning("Unable to open device 1");
//      // disable menu item if camera 1 does not exist
//      ui->actionCamera_1->setDisabled(true);
//  }

    emit deviceChanged(1, width, height);
}

/*
 * Menu action when Sources->file is selected.
 * Opens file dialog and tries to open selected file (is not empty),
 * then sets capture to open the selected file
 */
void MainWindow::on_actionFile_triggered()
{
    int width = 0;
    int height = 0;

    if (ui->radioButtonCustomSize->isChecked())
    {
        width = preferredWidth;
```



```
        height = preferredHeight;
    }

    QString fileName =
            QFileDialog::getOpenFileName(this,
                                         tr("Open Video"),
                                         "./",
                                         tr("Video Files (*.avi *.mkv *.mp4 *.m4v)"),
                                         NULL,
                                         QFileDialog::ReadOnly);

    qDebug("Opening file %s ...", fileName.toStdString().c_str());

    if (fileName.length() > 0)
    {
//      if (!capture->open(fileName))
//      {
//          qWarning("Unable to open device file : %s",
//                   fileName.toStdString().c_str());
//      }
//      // setupProcessorFromUI(); // already done from connection
        emit fileChanged(fileName, width, height);
    }
    else
    {
        qWarning("empty file name");
    }
}

/*
 * Menu action to qui application
 */
void MainWindow::on_actionQuit_triggered()
{
    this->close();
}

/*
 * Menu action when flip image is selected.
 * Sets capture to change flip status which leads to reverse
 * image horizontally
 */
void MainWindow::on_actionFlip_triggered()
{
    emit flipChanged(¬capture->isFlipVideo());
    /*
     * There is no need to update ui->checkBoxFlip since it is connected
     * to ui->actionFlip through signals/slots
     */
}

/*
 * Menu action when original image size is selected.
 * Sets capture not to resize image
 */
void MainWindow::on_actionOriginalSize_triggered()
{

    ui->actionConstrainedSize->setChecked(false);

    emit sizeChanged(QSize(0, 0));
}

/*
 * Menu action when constrained image size is selected.
 * Sets capture resize to preferred width and height
 */
void MainWindow::on_actionConstrainedSize_triggered()
{
    ui->actionOriginalSize->setChecked(false);

    emit sizeChanged(QSize(preferredWidth, preferredHeight));
}

/*
 * Menu action to replace current image rendering widget by a
 * QcvMatWidgetImage instance.
 */
void MainWindow::on_actionRenderImage_triggered()
{
    qDebug("Setting image rendering to: images");
    setRenderingMode(RENDER_IMAGE);
}

/*
 * Menu action to replace current image rendering widget by a
 * QcvMatWidgetLabel with pixmap instance.
 */
void MainWindow::on_actionRenderPixmap_triggered()
{
```

```
541     qDebug("Setting image rendering to: pixmaps");
542     setRenderingMode(RENDER_PIXMAP);
543 }
544
545 /*
546  * Menu action to replace current image rendering widget by a
547  * QcvMatWidgetGL instance.
548  */
549 void MainWindow::on_actionRenderOpenGL_triggered()
550 {
551     qDebug("Setting image rendering to: opengl");
552     setRenderingMode(RENDER_GL);
553 }
554
555 /*
556  * Original size radioButton action.
557  * Sets capture resize to off
558  */
559 void MainWindow::on_radioButtonOrigSize_clicked()
560 {
561     ui->actionConstrainedSize->setChecked(false);
562     emit sizeChanged(QSize(0, 0));
563 }
564
565 /*
566  * Custom size radioButton action.
567  * Sets capture resize to preferred width and height
568  */
569 void MainWindow::on_radioButtonCustomSize_clicked()
570 {
571     ui->actionOriginalSize->setChecked(false);
572     emit sizeChanged(QSize(preferredWidth, preferredHeight));
573 }
574
575 /*
576  * Width spinbox value change.
577  * Changes the preferred width and if custom size is selected apply
578  * this custom width
579  * @param value the desired width
580  */
581 void MainWindow::on_spinBoxWidth_valueChanged(int value)
582 {
583     preferredWidth = value;
584     if (ui->radioButtonCustomSize->isChecked())
585     {
586         emit sizeChanged(QSize(preferredWidth, preferredHeight));
587     }
588 }
589
590 /*
591  * Height spinbox value change.
592  * Changes the preferred height and if custom size is selected apply
593  * this custom height
594  * @param value the desired height
595  */
596 void MainWindow::on_spinBoxHeight_valueChanged(int value)
597 {
598     preferredHeight = value;
599     if (ui->radioButtonCustomSize->isChecked())
600     {
601         emit sizeChanged(QSize(preferredWidth, preferredHeight));
602     }
603 }
604
605 /*
606  * Flip capture image horizontally.
607  * changes capture flip status
608  */
609 void MainWindow::on_checkBoxFlip_clicked()
610 {
611     /*
612      * There is no need to update ui->actionFlip since it is connected
613      * to ui->checkBoxFlip through signals/slots
614      */
615     emit flipChanged(ui->checkBoxFlip->isChecked());
616 }
617
618 /*
619  * Kernel size changed in spin box
620  * @param value the new kernel size
621  */
622 void MainWindow::on_spinBoxKernel_valueChanged(int value)
623 {
624     processor->setKernelSize(value);
625
626     setupSigma();
627 }
628
629 /*
630  * Sigma gaussian variance changed in double spin box
```



```
631  * @param dvalue the new sigma value
632  * @post the corresponding integer value is computed and applied
633  * to the integer sigma slider
634  */
635 void MainWindow::on_doubleSpinBoxSigma_valueChanged(double dvalue)
636 {
637     processor->setSigma(dvalue);
638
639     int ivalue = double2intValue(ui->doubleSpinBoxSigma->minimum(),
640                                  ui->doubleSpinBoxSigma->singleStep(),
641                                  dvalue);
642
643 //  qDebug("Sigma Spinbox->Slider : convert %f [%f:%f:%f] to integer value %d [%d:%d:%d]",
644 //         dvalue,
645 //         ui->doubleSpinBoxSigma->minimum(),
646 //         ui->doubleSpinBoxSigma->singleStep(),
647 //         ui->doubleSpinBoxSigma->maximum(),
648 //         ivalue,
649 //         ui->horizontalSliderSigma->minimum(),
650 //         ui->horizontalSliderSigma->singleStep(),
651 //         ui->horizontalSliderSigma->maximum());
652
653     ui->horizontalSliderSigma->blockSignals(true);
654     ui->horizontalSliderSigma->setValue(ivalue);
655     ui->horizontalSliderSigma->blockSignals(false);
656 }
657
658 void MainWindow::on_spinBoxThreshold_valueChanged(int value)
659 {
660     processor->setThresholdLevel(value);
661 }
662
663 /*
664  * Kappa harris parameter changed in double spinbox
665  * @param dvalue the new kappa value
666  * @post the corresponding integer value is computed and applied
667  * to the integer kappa slider
668  */
669 void MainWindow::on_doubleSpinBoxKappa_valueChanged(double dvalue)
670 {
671     processor->setHarrisKappa(dvalue);
672
673     int ivalue = double2intValue(ui->doubleSpinBoxKappa->minimum(),
674                                  ui->doubleSpinBoxKappa->singleStep(),
675                                  dvalue);
676
677 //  qDebug("Kappa Spinbox->Slider : convert %f [%f:%f:%f] to integer value %d [%d:%d:%d]",
678 //         dvalue,
679 //         ui->doubleSpinBoxKappa->minimum(),
680 //         ui->doubleSpinBoxKappa->singleStep(),
681 //         ui->doubleSpinBoxKappa->maximum(),
682 //         ivalue,
683 //         ui->horizontalSliderKappa->minimum(),
684 //         ui->horizontalSliderKappa->singleStep(),
685 //         ui->horizontalSliderKappa->maximum());
686
687     ui->horizontalSliderKappa->blockSignals(true);
688     ui->horizontalSliderKappa->setValue(ivalue);
689     ui->horizontalSliderKappa->blockSignals(false);
690 }
691
692 void MainWindow::on_horizontalSliderSigma_valueChanged(int ivalue)
693 {
694     double dvalue = int2doubleValue(ui->doubleSpinBoxSigma->minimum(),
695                                     ui->doubleSpinBoxSigma->singleStep(),
696                                     ivalue);
697
698 //  qDebug("Sigma Slider->spinbox : convert %d [%d:%d:%d] to double value = %f [%f:%f:%f]",
699 //         ivalue,
700 //         ui->horizontalSliderSigma->minimum(),
701 //         ui->horizontalSliderSigma->singleStep(),
702 //         ui->horizontalSliderSigma->maximum(),
703 //         dvalue,
704 //         ui->doubleSpinBoxSigma->minimum(),
705 //         ui->doubleSpinBoxSigma->singleStep(),
706 //         ui->doubleSpinBoxSigma->maximum());
707
708     ui->doubleSpinBoxSigma->setValue(dvalue);
709 }
710
711 void MainWindow::on_horizontalSliderKappa_valueChanged(int ivalue)
712 {
713     double dvalue = int2doubleValue(ui->doubleSpinBoxKappa->minimum(),
714                                     ui->doubleSpinBoxKappa->singleStep(),
715                                     ivalue);
716
717 //  qDebug("Kappa Slider->spinbox : convert %d [%d:%d:%d] to double value = %f [%f:%f:%f]",
718 //         ivalue,
719 //         ui->horizontalSliderKappa->minimum(),
720 //         ui->horizontalSliderKappa->singleStep(),
```

```
721 //          ui->horizontalSliderKappa->maximum(),
722 //          dvalue,
723 //          ui->doubleSpinBoxKappa->minimum(),
724 //          ui->doubleSpinBoxKappa->singleStep(),
725 //          ui->doubleSpinBoxKappa->maximum());
726
727     ui->doubleSpinBoxKappa->setValue(dvalue);
728 }
729
730 /*
731  * Display image selection
732  * @param index the new selected image index
733  */
734 void MainWindow::on_comboBoxImages_currentIndexChanged(int index)
735 {
736     processor->setDisplayMode((CvGFilter::ImageDisplay)index);
737 }
738
739 /*
740  * Edge mode selection
741  * @param index the new edge mode index
742  */
743 void MainWindow::on_comboBoxEdges_currentIndexChanged(int index)
744 {
745     processor->setEdgeMode((CvGFilter::EdgeDisplay)index);
746 }
```



```
1  #include <QApplication>
2  #include <libgen.h>     // for basename
3  #include <iostream>     // for cout
4
5  using namespace std;
6
7  #include "QcvVideoCapture.h"
8  #include "CaptureFactory.h"
9  #include "QcvGFilter.h"
10 #include "mainwindow.h"
11
12 /**
13  * Usage function shown just before launching QApp
14  * @param name the name of the program (argv[0])
15  */
16 void usage(char * name);
17
18 /**
19  * Test program OpenCV2 + QT5
20  * @param argc argument count
21  * @param argv argument values
22  * @return QTApp return value
23  * @par usage : <Progname> [--device | -d] <#> | [--file | -f ] <filename>
24  * [--mirror | -m] [--size | -s] <width>x<height>
25  *  - device : [--device | -d] <device #> (0, 1, ...) Opens capture device #
26  *  - filename : [--file | -f ] <filename> Opens a video file or URL (including rtsp)
27  *  - mirror : mirrors image horizontally before display
28  *  - size : [--size | -s] <width>x<height> resize capture to fit desired <width>
29  *  and <height>
30  */
31 int main(int argc, char *argv[])
32 {
33     CvProcessor::VerboseLevel verboseLevel = CvProcessor::VERBOSE_WARNINGS; // verbose up to notif
34     // CvProcessor::VerboseLevel verboseLevel = CvProcessor::VERBOSE_ACTIVITY; // verbose all
35
36     // -----------------------------------------------------------------------
37     // Instanciate QApplication to receive special QT args
38     // -----------------------------------------------------------------------
39     QApplication app(argc, argv);
40
41     // Gets arguments after QT specials removed
42     QStringList argList = QCoreApplication::arguments();
43
44     int threadNumber = 3;
45     // parse arguments for --threads tag
46     for (QListIterator<QString> it(argList); it.hasNext(); )
47     {
48         QString currentArg(it.next());
49
50         if (currentArg == "-t" || currentArg =="--threads")
51         {
52             // Next argument should be thread number integer
53             if (it.hasNext())
54             {
55                 QString threadString(it.next());
56                 bool convertOk;
57                 threadNumber = threadString.toInt(&convertOk,10);
58                 if (!convertOk || threadNumber < 1 || threadNumber > 3)
59                 {
60                     qWarning("Warning: Invalid thread number %d",threadNumber);
61                     threadNumber = 3;
62                 }
63             }
64             else
65             {
66                 qWarning("Warning: thread tag found with no following thread number");
67             }
68         }
69         else if (currentArg == "-v" || currentArg == "--verbose")
70         {
71             // next argument should be a verbose level (from 0 to 4)
72             if (it.hasNext())
73             {
74                 QString verboseLevelString(it.next());
75                 bool convertOk;
76                 int newVerboseLevel = verboseLevelString.toUInt(&convertOk, 10);
77                 if (!convertOk ||
78                     newVerboseLevel < 0 ||
79                     newVerboseLevel > (int)CvProcessor::NBVERBOSELEVEL)
80                 {
81                     qWarning("Invalid verbose level %d", newVerboseLevel);
82                 }
83                 else
84                 {
85                     verboseLevel = (CvProcessor::VerboseLevel)newVerboseLevel;
86                 }
87             }
88             else
89             {
90                 // by default set it to max verbose
```

```
91                  verboseLevel = CvProcessor::VERBOSE_ACTIVITY;
92              }
93          }
94      }
95
96      // ------------------------------------------------------------------------
97      // Create Capture factory using program arguments and
98      // open Video Capture
99      // ------------------------------------------------------------------------
100     CaptureFactory factory(argList);
101     factory.setSkippable(true);
102
103     // Helper thread for capture
104     QThread * capThread = NULL;
105     if (threadNumber > 1)
106     {
107         capThread = new QThread();
108     }
109
110     // Capture
111     QcvVideoCapture * capture = factory.getCaptureInstance(capThread);
112
113     // ------------------------------------------------------------------------
114     // Create Filtering processor
115     // ------------------------------------------------------------------------
116     // Helper thread for processor
117     QThread * procThread = NULL;
118     if (threadNumber > 2)
119     {
120         procThread = new QThread();
121     }
122     else
123     {
124         if (threadNumber > 1)
125         {
126             procThread = capThread;
127         }
128     }
129
130     // Processsor
131     QcvGFilter * processor = NULL;
132     if (procThread ≡ NULL)
133     {
134         processor = new QcvGFilter(capture→getImage());
135     }
136     else
137     {
138         if (procThread ≠ capThread)
139         {
140             processor = new QcvGFilter(capture→getImage(),
141                                        capture→getMutex(),
142                                        procThread);
143         }
144         else // procThread == capThread
145         {
146             processor = new QcvGFilter(capture→getImage(),
147                                        NULL,
148                                        procThread);
149         }
150     }
151
152     processor→setVerboseLevel(verboseLevel);
153
154     // ------------------------------------------------------------------------
155     // Connects capture to processor
156     // ------------------------------------------------------------------------
157     // Connects capture update to processor update
158     QObject::connect(capture, SIGNAL(updated()),
159                      processor, SLOT(update()),
160                      ((threadNumber < 3) ? Qt::DirectConnection :
161                                            Qt::QueuedConnection));
162
163     // connect capture changed image to processor set input
164     QObject::connect(capture, SIGNAL(imageChanged(Mat*)),
165                      processor, SLOT(setSourceImage(Mat*)),
166                      ((threadNumber < 3) ? Qt::DirectConnection :
167                                            Qt::QueuedConnection));
168
169     // ------------------------------------------------------------------------
170     // Now that Capture & processor are on then
171     // add our MainWindow as toplevel
172     // and launches app
173     // ------------------------------------------------------------------------
174     MainWindow w(capture, processor);
175     w.show();
176
177     usage(argv[0]);
178
179     int retVal = app.exec();
180
```



```
181     // ------------------------------------------------------------------------
182     // Cleanup & return
183     // ------------------------------------------------------------------------
184     delete processor;
185     delete capture;
186
187     bool sameThread = capThread ≡ procThread;
188
189     if (capThread ≠ NULL)
190     {
191         delete capThread;
192     }
193
194     if (procThread ≠ NULL ∧ ¬sameThread)
195     {
196         delete procThread;
197     }
198
199     return retVal;
200 }
201
202 /*
203  * Usage function shown just before launching QApp
204  * @param name the name of the program (argv[0])
205  */
206 void usage(char * name)
207 {
208     cout << "usage : " << basename(name) << " "
209          << "[-d | --device] <device number> "
210          << "[-v | --video] <video file> "
211          << "[-s | --size] <width>x<height> "
212          << "[-m | --mirror] "
213          << "[-t | --threads] <number of threads>"
214          << endl;
215 }
```