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

#ifndef CVPROCESSOR_H_
#define CVPROCESSOR_H_

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

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

#include "CvProcessorException.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;

    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;

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

        /**
         * 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 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 than 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;

        /**
         * 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);

    protected:
        // -------------------------------------------------------------------
        // Setup and cleanup attributes
        // -------------------------------------------------------------------
        /**
         * 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
         */
        virtual void setup(Mat * sourceImage, const bool fullSetup = true);

        /**
         * Clean up internal attributes before changing source image or
         * cleaning up class before destruction
         * @note this method should be reimplemented in sub classes
         */
        virtual void cleanup();
};

#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),
    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;
    addImage("source", this→sourceImage);
}

/*
 * Clean up internal atrtibutes 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)
{
    // 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)
```

```
165         {
166             cerr << "CvProcessor::addImage(\"" << name
167                  << "\",...) : already added" << endl;
168         }
169 
170         retValue = false;
171     }
172     else
173     {
174         retValue = true;
175     }
176 
177     return retValue;
178 }
179 /*
180  * Update named image in additionnal images.
181  * @param name the name of the image
182  * @param image the image reference
183  * @post the image located at key name is updated.
184  */
185 //void CvProcessor::updateImage(const char * name, Mat * image)
186 //{
187 //  // Search for this name in the map
188 //  map<string, Mat*>::iterator it;
189 //  for (it = images.begin(); it != images.end(); ++it)
190 //  {
191 //      if (it->first == name)
192 //      {
193 //          (it->second->release();
194 //          images.erase(it);
195 //      }
196 //  }
197 //
198 //  string sname(name);
199 //
200 //  updateImage(sname, image);
201 //}
202 
203 /*
204  * Update named image in additionnal images.
205  * @param name the name of the image
206  * @param image the image reference
207  * @post the image located at key name is updated.
208  */
209 //void CvProcessor::updateImage(const string & name, const Mat & image)
210 //{
211 //  clog << "update image " << name << " with " << (long) &image << endl;
212 //  images.erase(name);
213 //
214 //  addImage(name, image);
215 //}
216 
217 /*
218  * Get image by name
219  * @param name the name of the image we're looking for
220  * @return the image registered by this name in the additionnal images
221  * map
222  * @throw CvProcessorException#INVALID_NAME is used name is not already
223  * registerd in the images
224  */
225 const Mat & CvProcessor::getImage(const char *name) const
226     throw (CvProcessorException)
227 {
228     string sname(name);
229 
230     return getImage(sname);
231 }
232 
233 /*
234  * Get image pointer by name
235  * @param name the name of the image we're looking for
236  * @return the image pointer registered by this name in the additionnal
237  * images map
238  * @throw CvProcessorException#INVALID_NAME is used name is not already
239  * registerd in the images
240  */
241 const Mat & CvProcessor::getImage(const string & name) const
242     throw (CvProcessorException)
243 {
244     // Search for this name
245     map<string, Mat*>::const_iterator cit;
246     for (cit = images.begin(); cit ≠ images.end(); ++cit)
```



```
247     {
248         if (cit→first ≡ name)
249         {
250             if (cit→second→data ≡ NULL)
251             {
252                 // image contains no data
253                 throw CvProcessorException(CvProcessorException::NULL_DATA,
254                                            name.c_str());
255             }
256             return *(cit→second);
257         }
258     }
259 
260     // not found : throw exception
261     throw CvProcessorException(CvProcessorException::INVALID_NAME,
262                                name.c_str());
263 }
264 
265 /*
266  * Get image pointer by name
267  * @param name the name of the image we're looking for
268  * @return the image pointer registered by this name in the additionnal
269  * images map
270  * @throw CvProcessorException#INVALID_NAME is used name is not already
271  * registerd in the images
272  */
273 Mat * CvProcessor::getImagePtr(const char *name)
274     throw (CvProcessorException)
275 {
276     string sname(name);
277 
278     return getImagePtr(sname);
279 }
280 
281 /*
282  * Get image pointer by name
283  * @param name the name of the image we're looking for
284  * @return the image registered by this name in the additionnal images
285  * map
286  * @throw CvProcessorException#INVALID_NAME is used name is not already
287  * registerd in the images
288  */
289 Mat * CvProcessor::getImagePtr(const string & name)
290     throw (CvProcessorException)
291 {
292     // Search for this name
293     map<string, Mat*>::const_iterator cit;
294     for (cit = images.begin(); cit ≠ images.end(); ++cit)
295     {
296         if (cit→first ≡ name)
297         {
298             if (verboseLevel ≥ VERBOSE_ACTIVITY)
299             {
300                 clog << "getImagePtr(" << name << "): returning : "
301                      << (long) (cit→second) << endl;
302             }
303             return cit→second;
304         }
305     }
306 
307     // not found : throw exception
308     throw CvProcessorException(CvProcessorException::INVALID_NAME, name.c_str());
309 }
310 
311 /*
312  * Number of channels in source image
313  * @return the number of channels of source image
314  */
315 int CvProcessor::getNbChannels() const
316 {
317     return nbChannels;
318 }
319 
320 /*
321  * Type of the source image
322  * @return the openCV type of the source image
323  */
324 int CvProcessor::getType() const
325 {
326     return type;
327 }
328 
```

```
329 /*
330  * Get the current verbose level
331  * @return the current verbose level
332  */
333 CvProcessor::VerboseLevel CvProcessor::getVerboseLevel() const
334 {
335     return verboseLevel;
336 }
337
338 /*
339  * Set new verbose level
340  * @param level the new verobse level
341  */
342 void CvProcessor::setVerboseLevel(const VerboseLevel level)
343 {
344     if ((level ≥ VERBOSE_NONE) ∧ (level < NBVERBOSELEVEL))
345     {
346         verboseLevel = level;
347     }
348
349     cout << "Verbose level set to: ";
350     switch (verboseLevel)
351     {
352         case VERBOSE_NONE:
353             cout << "no messages";
354             break;
355         case VERBOSE_ERRORS:
356             cout << "unrecoverable errors only";
357             break;
358         case VERBOSE_WARNINGS:
359             cout << "errors and warnings";
360             break;
361         case VERBOSE_NOTIFICATIONS:
362             cout << "errors, warnings and notifications";
363             break;
364         case VERBOSE_ACTIVITY:
365             cout << "All messages";
366             break;
367         case NBVERBOSELEVEL:
368         default:
369             cout << "Unknown verobse mode (unchanged)";
370             break;
371     }
372     cout << endl;
373 }
374
375 /*
376  * Return processor processing time of step index [default implementation
377  * returning only processTime, should be reimplemented in subclasses]
378  * @param index index of the step which processing time is required,
379  * 0 indicates all steps, and values above 0 indicates step #. If
380  * required index is bigger than number of steps than all steps value
381  * should be returned.
382  * @return the processing time of step index.
383  * @note should be reimplemented in subclasses in order to define
384  * time/feature behaviour
385  */
386 double CvProcessor::getProcessTime(const size_t) const
387 {
388     return processTime;
389 }
390
391
392 /*
393  * Indicates if processing time is per feature processed in the current
394  * image or absolute
395  * @return
396  */
397 bool CvProcessor::isTimePerFeature() const
398 {
399     return timePerFeature;
400 }
401
402 /*
403  * Sets Time per feature processing time unit
404  * @param value the time per feature value (true or false)
405  */
406 void CvProcessor::setTimePerFeature(const bool value)
407 {
408     timePerFeature = value;
409 }
410
```



```
411
```

```
#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;
         * }
         */

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

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

#endif /*CVPROCESSOREXCEPTION_H_*/
```

```
#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;
        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 <QString>
#include <QRegExp>
#include <QMutex>
#include <QThread>
#include "CvProcessor.h"

/**
 * 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 char numberFormat[10];

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

        /**
         * The Source image mutex in order to avoid concurrent access to
         * the source image (typically the source image may be modified
         */
        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;

    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);

    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 slot.
         * @param value the time per feature value (true or false)
         */
        virtual void setTimePerFeature(const bool value);

    signals:
        /**
         * Signal emitted when update is complete
         */
        void updated();

        /**
         * Signal emitted when processor has finished.
         * Used to tell helper threads to quit
         */
        void finished();

        /**
         * Signal emitted when source image is reallocated
         */
        void imageChanged();

        /**
         * Signal emitted when source image is reallocated
         * @param image the new source image pointer or none if just
         * image changed notification is required
         */
        void imageChanged(Mat * image);

        /**
         * Signal emitted when source image colors changes from color to gray
         * or from gray to color
         */
        void imageColorsChanged();

        /**
         * Signal emitted when source image size changes
         */
        void imageSizeChanged();

        /**
         * Signal emited when processing time has channged
         * @param value the new value of the processing time
         */
        void processTimeUpdated(const QString & formattedValue);

        /**
         * Signal to set text somewhere
         * @param message the message
         */
        void sendText(const QString & message);
```

```
165
166         /**
167          * Signal to send update message when something changes
168          * @param message the message
169          * @param timeout number of ms the message should be displayed
170          */
171         void sendMessage(const QString & message, int timeout = defaultTimeOut);
172
173 };
174
175 #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 <QDebug>
10  #include <cstring>      // for strcpy
11  #include "QcvProcessor.h"
12
13  /*
14   * Default timeout to show messages
15   */
16  int QcvProcessor::defaultTimeOut = 5000;
17
18  /*
19   * Number format used to format numbers into QStrings
20   */
21  char QcvProcessor::numberFormat[10] = {"%8.1f ms"};
22
23  /*
24   * The regular expression used to validate new number formats
25   * @see #setNumberFormat
26   */
27  QRegExp QcvProcessor::numberRegExp("%[+- 0#]*[0-9]*([.][0-9]+)?[efEF]");
28
29  /*
30   * QcvProcessor constructor
31   * @param image the source image
32   * @param imageLock the mutex for concurrent access to the source image
33   * In order to avoid concurrent access to the same image
34   * @param updateThread the thread in which this processor should run
35   * @param parent parent QObject
36   */
37  QcvProcessor::QcvProcessor(Mat * image,
38                             QMutex * imageLock,
39                             QThread * updateThread,
40                             QObject * parent) :
41      CvProcessor(image), // <-- virtual base class constructor first
42      QObject(parent),
43      sourceLock(imageLock),
44      updateThread(updateThread),
45      message(),
46      processTimeString()
47  {
48      if (updateThread != NULL)
49      {
50          this->moveToThread(updateThread);
51
52          connect(this, SIGNAL(finished()), updateThread, SLOT(quit()),
53                  Qt::DirectConnection);
54
55          updateThread->start();
56      }
57  }
58
59  /*
60   * QcvProcessor destructor
61   */
62  QcvProcessor::~QcvProcessor()
63  {
64      // Lock might be already destroyed in source object so don't try to unlock
65
66      message.clear();
67      processTimeString.clear();
68
69      emit finished();
70
71      if (updateThread != NULL)
72      {
73          // Wait until update thread has received the "finished" signal through
74          // "quit" slot
75          updateThread->wait();
76      }
77  }
78
79  /*
80   * Sets new number format
81   * @param format the new number format
82   */
```

```
83  void QcvProcessor::setNumberFormat(const char * format)
84  {
85      /*
86       * The format string should validate the following regex
87       * %[+- 0#]*[0-9]*([.][0-9]+)?[efEF]
88       */
89      QRegExpValidator validator(numberRegExp, NULL);
90
91      QString qFormat(format);
92      int pos = 0;
93      if ((validator.validate(qFormat,pos) ≡ QValidator::Acceptable) ∧
94          (strlen(format) ≤ 10))
95      {
96          strcpy(numberFormat, format);
97      }
98      else
99      {
100         qWarning("QcvProcessor::setNumberFormat(%s) : invalid format", format);
101     }
102 }
103
104
105 /*
106  * Update computed images slot and sends updated signal
107  * required
108  */
109 void QcvProcessor::update()
110 {
111     /*
112      * Important note : CvProcessor::update() should NOT be called here
113      * since it should be called in QcvXXXprocessor subclasses such that
114      * QcvXXXProcessor::update method should contain :
115      *  - call to CvXXXProcessor::update() (not QCvXXXProcessor)
116      *  - emit signals from QcvXXXProcessor
117      *  - call to QcvProcessor::update() (this method)
118      */
119     emit updated();
120     processTimeString.sprintf(numberFormat, getProcessTime(0) / 1000.0);
121     emit processTimeUpdated(processTimeString);
122 }
123
124 /*
125  * Changes source image slot.
126  * Attributes needs to be cleaned up then set up again
127  * @param image the new source Image
128  * @post Various signals are emitted:
129  *  - imageChanged(sourceImage)
130  *  - imageCchanged()
131  *  - if image size changed then imageSizeChanged() is emitted
132  *  - if image color space changed then imageColorsChanged() is emitted
133  */
134 void QcvProcessor::setSourceImage(Mat *image)
135     throw (CvProcessorException)
136 {
137     if (verboseLevel ≥ VERBOSE_NOTIFICATIONS)
138     {
139         clog << "QcvProcessor::setSourceImage(" << (ulong) image << ")" << endl;
140     }
141
142     Size previousSize(sourceImage→size());
143     int previousNbChannels(nbChannels);
144
145     if (sourceLock ≠ NULL)
146     {
147         sourceLock→lock();
148         // qDebug() << "QcvProcessor::setSourceImage: lock";
149     }
150
151     CvProcessor::setSourceImage(image);
152
153     if (sourceLock ≠ NULL)
154     {
155         // qDebug() << "QcvProcessor::setSourceImage: unlock";
156         sourceLock→unlock();
157     }
158
159     emit imageChanged(sourceImage);
160
161     emit imageChanged();
162
163     if ((previousSize.width ≠ image→cols) ∨
164         (previousSize.height ≠ image→rows))
```



```
165     {
166         emit imageSizeChanged();
167     }
168
169     if (previousNbChannels ≠ nbChannels)
170     {
171         emit imageColorsChanged();
172     }
173
174     // Force update
175     update();
176 }
177
178 /*
179  * Sets Time per feature processing time unit slot
180  * @param value the time per feature value (true or false)
181  */
182 void QcvProcessor::setTimePerFeature(const bool value)
183 {
184     CvProcessor::setTimePerFeature(value);
185 }
```

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

#ifndef CVDFT_H_
#define CVDFT_H_

#include <vector>
using namespace std;

#include <cv.h>
using namespace cv;

#include "CvProcessor.h"

/**
 * Class to compute DFT on input image
 */
class CvSimpleDFT : virtual public CvProcessor
{
    public:
        /**
         * Minimum log scale factor.
         * Default value is 5.
         */
        static const double minLogScaleFactor;

        /**
         * Maximum log scale factor.
         * Default value is 20 or 30.
         */
        static const double maxLogScaleFactor;

    protected:
        /**
         * Minimum of source image rows & cols for cropping source
         */
        int minSize;

        /**
         * Maximum of source image rows & cols for cropping source
         */
        int maxSize;

        /**
         * Border size to crop on source image
         */
        int borderSize;

        /**
         * DFT optimal size
         */
        int optimalDFTSize;

        /**
         * Optimal Fourier size
         */
        Size dftSize;

        /**
         * Input frame cropped to square size for FFT: CV_8UC<nbChannels>
         */
        Mat inFrameSquare;

        /**
         * Input frame cropped color channels: CV_8UC1 x <nbChannels>
         */
        vector<Mat> channels;

        /**
         * Input frame square channels converted to doubles: CV_64FC1 x <nbChannels>
         */
        vector<Mat> channelsDouble;

        /**
         * Input frame square channels complex channels:
         * CV_64FC1 x 2 x <nbChannels>
         */
        vector<vector<Mat> > channelsDoubleComplexComponents;
```



```

        /**
         * Input frame square complex image: CV_64FC2 x <nbChannels>
         */
        vector<Mat> channelsComplexImages;

        /**
         * Complex spectrum images: CV_64FC2 x <nbChannels>
         */
        vector<Mat> channelsComplexSpectrums;

        /**
         * Complex spectrum channels: CV_64FC1 x 2 x <nbChannels>
         */
        vector<vector<Mat> > channelsComplexSpectrumComponents;

        /**
         * Spectrum magnitude: CV_64FC1 x <nbChannels>
         */
        vector<Mat> channelsSpectrumMagnitude;

        /**
         * LogScale factor to apply on log magnitude to show spectrum.
         */
        double logScaleFactor;

        /**
         * log spectrum magnitude: CV_64FC1 x <nbChannels>
         */
        vector<Mat> channelsSpectrumLogMagnitude;

        /**
         * [Log] spectrum magnitude channels converted for display:
         * CV_8UC1 x <nbChannels>
         */
        vector<Mat> channelsSpectrumLogMagnitudeDisplay;

        /**
         * [Log] spectrum magnitude image converted for display:
         * CV_8UC<nbChannels>
         */
        Mat spectrumMagnitudeImage;

    public:
        /**
         * DFT processor constructor
         * @param sourceImage the source image
         * @pre source image is not NULL
         */
        CvSimpleDFT(Mat * sourceImage);

        /**
         * DFT Processor destructor
         */
        virtual ~CvSimpleDFT();

        /**
         * DFT Update.
         * Steps in update
         *  - crop source image to a square according to optima FFT size
         *  - split in frame square into color channels
         *  - converts these color channels to double
         *  - apply frequency shift on double channels to
         *      - produce the shifted real component of source channels
         *      - produce later a spectrum with low frequencies at image center
         *  - merge real/image channels into complex image per channel
         *  - compute dft on each channel
         *  - split channels complex spectrum in to real/imag components
         *  - compute channels spectrum magnitude from real/imag components
         *  - log scale channels spectrum magnitude
         *  - converts channels log magnitude for display
         */
        virtual void update();

        // ------------------------------------------------------------------
        // Options settings and gettings
        // ------------------------------------------------------------------

        /**
         * Optimal dft size for current source image
         * @return the current optimal dft size
         */
```

```
165         int getOptimalDftSize() const;
166
167         /**
168          * Get current log scale factor
169          * @return the current log scale factor
170          */
171         double getLogScaleFactor() const;
172
173         /**
174          * Setting the log scale factor
175          * @param logScaleFactor the new log scale factor
176          */
177         virtual void setLogScaleFactor(double logScaleFactor);
178
179     protected:
180
181         // --------------------------------------------------------------------
182         // Setup and cleanup attributes
183         // --------------------------------------------------------------------
184
185         /**
186          * Setup internal attributes according to source image
187          * @param sourceImage a new source image
188          * @param fullSetup full setup is needed when source image is changed
189          */
190         void setup(Mat * sourceImage, bool fullSetup = true);
191
192         /**
193          * Clean up internal atrtibutes before changing source image or
194          * cleaning up class before destruction
195          */
196         void cleanup();
197
198         // --------------------------------------------------------------------
199         // Utility methods
200         // --------------------------------------------------------------------
201         /**
202          * Modify image to obtain reverse frequencies on the Fourier transform
203          * (low frequencies at the center of the image and high frequencies on
204          * the border), or modify image obtained from reverse Fourier transform
205          * with reversed frequencies.
206          * @param imgIn source image
207          * @param imgOut destination image
208          * @par Algorithm:
209          * This is based on the following property of the Z transform :
210          * \f[
211          *  TZ\left\{a^{k} x_{k}\right\} = X\left(\frac{z}{a}\right)
212          * \f]
213          * if \f$y_{k} = (-1)^{k} x_{k}\f$ then \f$Y(z) = X(-z)\f$
214          * which can be explained in Fourier space by replacing
215          * \f$z\f$ by \f$e^{j 2 \pi F}\f$:
216          * \f[
217          * Y\left[e^{j 2 \pi F}\right] = X\left[-e^{j2\pi F}\right] =
218          * X\left[e^{j\pi}e^{j2\pi F}\right] =
219          * X\left[e^{j2\pi\left(F + \frac{1}{2}\right)}\right]
220          * \f]
221          * hence
222          * \f[
223          * Y(F) = X\left(F + \frac{1}{2}\right)
224          * \f]
225          * or
226          * \f[
227          * Y(f) = X\left(f + \frac{f_{e}}{2}\right)
228          * \f]
229          * where \f$f_{e}\f$ is the sampling frequency, which means the
230          * resulting Fourier transform will present an \f$\frac{f_{e}}{2}\f$
231          * frequency offset. And since the sampling frequency lies in the middle
232          * of the spectrum in the DFT. Low frequencies will appear centered
233          * around the middle of the spectrum.
234          *
235          * In 2D the algorithm is the following:
236          * \f[
237          * imgOut(i,j) = (-1)^{i+j} \cdot imgIn(i,j)
238          * \f]
239          * \f$f_{e}\f$ is at the center of the spectrum image in 2D, which
240          * means, low frequencies will be located at the center of the image.
241          */
242         template <typename T>
243         void frequencyShift(Mat & imgIn, Mat & imgOut);
244
245         /**
246          * Log scale T valued image
```



```
247          * @param imgIn input image
248          * @param imgOut output image
249          * @param scaleFactor such as
250          * \f$ imgOut = scaleFactor \times \log(1 + imgIn)\f$
251          */
252         template <typename T>
253         void logScaleImg(const Mat & imgIn, Mat & imgOut, const T scaleFactor);
254 };
255
256 #endif /* CVDFT_H_ */
```

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

#include <limits>
#include <cmath>

//#include <iostream>
//using namespace std;

#include "CvSimpleDFT.h"

/*
 * Minimum log scale factor.
 * Default value is 5.
 */
const double CvSimpleDFT::minLogScaleFactor = 5.0;

/*
 * Maximum log scale factor.
 * Default value is 20.
 */
const double CvSimpleDFT::maxLogScaleFactor = 30.0;

/*
 * DFT processor constructor
 * @param sourceImage the source image
 */
CvSimpleDFT::CvSimpleDFT(Mat * sourceImage) :
    CvProcessor(sourceImage),
    minSize(MIN(sourceImage->rows, sourceImage->cols)),
    maxSize(MAX(sourceImage->rows, sourceImage->cols)),
    borderSize((maxSize-minSize)/2),
    optimalDFTSize(getOptimalDFTSize(minSize)),
    dftSize(optimalDFTSize, optimalDFTSize),
    inFrameSquare(dftSize, type),
    logScaleFactor(10.0),
    spectrumMagnitudeImage(dftSize, type)
{
    setup(sourceImage, false);

    addImage("square", &inFrameSquare);
    addImage("spectrum", &spectrumMagnitudeImage);
}

/*
 * DFT Processor destructor
 */
CvSimpleDFT::~CvSimpleDFT()
{
    cleanup();
}

/*
 * Setup internal attributes according to source image
 * @param sourceImage a new source image
 * @param fullSetup full setup is needed when source image is changed
 */
void CvSimpleDFT::setup(Mat *sourceImage, bool fullSetup)
{
    // Full setup starting point (already performed in constructor)
    if (fullSetup)
    {
        CvProcessor::setup(sourceImage, fullSetup);
        minSize = MIN(sourceImage->rows, sourceImage->cols);
        maxSize = MAX(sourceImage->rows, sourceImage->cols);
        borderSize = (maxSize-minSize)/2;
        optimalDFTSize = getOptimalDFTSize(minSize);
        dftSize.height = optimalDFTSize;
        dftSize.width = optimalDFTSize;
        inFrameSquare = Mat(dftSize, type);
//      logScaleFactor = 10.0;
        spectrumMagnitudeImage = Mat(dftSize, type);
    }

    // Partial setup starting point
    for (int i=0; i < nbChannels; i++)
    {
        channels.push_back(Mat(dftSize, CV_8UC1));
```



```
        channelsDouble.push_back(Mat(dftSize, CV_64FC1));
        channelsDoubleComplexComponents.push_back(vector<Mat>());
        channelsComplexImages.push_back(Mat(dftSize, CV_64FC2));
        channelsComplexSpectrums.push_back(Mat(dftSize, CV_64FC2));
        channelsComplexSpectrumComponents.push_back(vector<Mat>());
        channelsSpectrumMagnitude.push_back(Mat(dftSize, CV_64FC1));
        channelsSpectrumLogMagnitude.push_back(Mat(dftSize, CV_64FC1));
        channelsSpectrumLogMagnitudeDisplay.push_back(Mat(dftSize, CV_8UC1));

        // complex channels
        for (int j=0; j < 2; j++)
        {
            channelsDoubleComplexComponents[i].push_back(Mat(dftSize, CV_64FC1));
            channelsComplexSpectrumComponents[i].push_back(Mat(dftSize, CV_64FC1));
        }

        // fill complex channels of channelsDoubleComplexComponents with 0
        channelsDoubleComplexComponents[i][1] = Scalar(0.0);
    }
}

void CvSimpleDFT::cleanup()
{
    for (int i=0; i < nbChannels; i++)
    {
        // complex channels
        for (int j=0; j < 2; j++)
        {
            channelsComplexSpectrumComponents[i][j].release();
            channelsDoubleComplexComponents[i][j].release();
        }

        channelsSpectrumLogMagnitudeDisplay[i].release();
        channelsSpectrumLogMagnitude[i].release();
        channelsSpectrumMagnitude[i].release();
        channelsComplexSpectrumComponents[i].clear();
        channelsComplexSpectrums[i].release();
        channelsComplexImages[i].release();
        channelsDoubleComplexComponents[i].clear();
        channelsDouble[i].release();
        channels[i].release();
    }

    channelsSpectrumLogMagnitudeDisplay.clear();
    channelsSpectrumLogMagnitude.clear();
    channelsSpectrumMagnitude.clear();
    channelsComplexSpectrumComponents.clear();
    channelsComplexSpectrums.clear();
    channelsComplexImages.clear();
    channelsDoubleComplexComponents.clear();
    channelsDouble.clear();
    channels.clear();

    spectrumMagnitudeImage.release();
    inFrameSquare.release();

    // super cleanup
    CvProcessor::cleanup();
}

/*
 * Update
 */
void CvSimpleDFT::update()
{
//  clog << "CvSimpleDFT::update()" << endl;

    /*
     * Crop source image to center square and resize it to nearest
     * DFT optimal size
     * *sourceImage -> inFrameSquare
     */
    if (sourceImage->cols > sourceImage->rows)
    {
        // wider than high : resize a colRange(borderSize, borderSize + minSize)
        // of sourceImage to dftSize in inFrameSquare
        resize(sourceImage->colRange(borderSize, borderSize + minSize),
               inFrameSquare,
               dftSize,
               0,
               0,
               INTER_AREA);
```

```
    }
    else
    {
        // higher than wide : resize a rowRange(borderSize, borderSize + minSize)
        // of sourceImage to dftSize in inFrameSquare
        resize(sourceImage→rowRange(borderSize, borderSize + minSize),
               inFrameSquare,
               dftSize,
               0,
               0,
               INTER_AREA);
    }

    /*
     * Split input frame square to individual channels
     * inFrameSquare -> channels
     */
    // TODO Ã  complÃ©ter ...

    // Process each component (1 for gray images, 3 for color images)
    for (int i=0; i < nbChannels; i++)
    {
        /*
         * Fourier transform processing
         *  - Convert uchar center square image to CV_64F real component
         *  - perform frequency shift on real image to obtain low frequencies
         *      in the middle of the DFT image rather than in the corners
         *  - merge real & imag component to complexImage before DFT
         *      imag component could be filled with 0
         *  - compute DFT
         *  - split DFT channels
         *  - compute DFT magnitude from DFT channels
         *  - logScale magnitude with factor (5 to 20)
         *  - convertScaleAbs logMagnitude to CV_8UC1 to display image
         *
         */

        // convert component to double
        // channels[] -> channelsDouble
        // TODO Ã  complÃ©ter ...

        // Frequency shift channelsDouble to real complex component with
        // frequencyShift<double>(...)
        // Frequency shift allow to prepare spatial image components to
        // produce frequency image later with low frequencies in the center
        // of frequency image
        // channelsDouble[] -> channelsDoubleComplexComponents[][0]
        // TODO Ã  complÃ©ter ...
        // channelsDoubleComplexComponents[i][1] is already filled with 0 in
        // setup method so frequency shift is not necessary on imaginary part

        // Merge Real and Imaginary into a complex component image
        // channelsDoubleComplexComponents[] -> channelsComplexImages[]
        // TODO Ã  complÃ©ter ...

        // Perform Fourier transform (dft) on Complex component image
        // channelsComplexImages[] -> channelsComplexSpectrums[] with
        // DFT_COMPLEX_OUTPUT
        // TODO Ã  complÃ©ter ...

        // Split component Complex spectrum to real/imag channels
        // channelsComplexSpectrums[] -> channelsComplexSpectrumComponents[]
        // TODO Ã  complÃ©ter ...

        // Compute component spectrum magnitude
        // channelsComplexSpectrumComponents[][0 & 1] -> channelsSpectrumMagnitude[]
        // TODO Ã  complÃ©ter ...

        // Log scale magnitude with logScaleImg<double>(...) and logScaleFactor
        // channelsSpectrumMagnitude[] -> channelsSpectrumLogMagnitude[]
        // TODO Ã  complÃ©ter ...

        // Convert Log scale channels Spectrum to display channels
        // channelsSpectrumLogMagnitude[] -> channelsSpectrumLogMagnitudeDisplay[]
        // TODO Ã  complÃ©ter ...
    }

    // Merge channels spectrum Log magnitude to color spectrum image
    // channelsSpectrumLogMagnitudeDisplay -> spectrumMagnitudeImage
    // TODO Ã  complÃ©ter ...

}
```



```

/*
 * Optimal dft size for current source image
 * @return the current optimal dft size
 */
int CvSimpleDFT::getOptimalDftSize() const
{
    return optimalDFTSize;
}

/*
 * Get current log scale factor
 * @return the current log scale factor
 */
double CvSimpleDFT::getLogScaleFactor() const
{
    return logScaleFactor;
}

/*
 * Setting the log scale factor
 * @param logScaleFactor the new log scale factor
 */
void CvSimpleDFT::setLogScaleFactor(double logScaleFactor)
{
    if (logScaleFactor > maxLogScaleFactor)
    {
        this→logScaleFactor = maxLogScaleFactor;
    }
    else if (logScaleFactor < minLogScaleFactor)
    {
        this→logScaleFactor = minLogScaleFactor;
    }
    else
    {
        this→logScaleFactor = logScaleFactor;
    }
}

// ----------------------------------------------------------------------
// Utility methods
// ----------------------------------------------------------------------
/*
 * Modify image to obtain reverse frequencies on the Fourier transform
 * (low frequencies at the center of the image and high frequencies on
 * the border), or modify image obtained from reverse Fourier transform
 * with reversed frequencies.
 * @param imgIn source image
 * @param imgOut destination image
 */
template <typename T>
void CvSimpleDFT::frequencyShift(Mat & imgIn, Mat & imgOut)
{
    int i, j;

    for (i = 0; i < imgIn.rows; i++)
    {
        for (j = 0; j < imgIn.cols; j++)
        {
            imgOut.at<T> (i, j) = imgIn.at<T> (i, j) * (T)pow(-1.0, i + j);
        }
    }
}

/*
 * Log scale T valued image
 * @param imgIn input image
 * @param imgOut output image
 * @param scaleFactor such as
 * \f$ imgOut = scaleFactor \times \log(1 + imgIn)\f$
 */
template <typename T>
void CvSimpleDFT::logScaleImg(const Mat & imgIn, Mat & imgOut,
    const T scaleFactor)
{
    MatConstIterator_<T> inIt = imgIn.begin<T>();
    MatConstIterator_<T> inItEnd = imgIn.end<T>();
    MatIterator_<T> outIt = imgOut.begin<T>();
    MatIterator_<T> outItEnd = imgOut.end<T>();
    for (; inIt ≠ inItEnd ∧ outIt ≠ outItEnd; ++inIt, ++outIt)
    {
        (*outIt) = scaleFactor * (T)log(1.0 + (*inIt));
```

```
329         }
330 }
```



```
1   /*
2    * QcvSimpleDFT.h
3    *
4    *  Created on: 22 fÃ©vr. 2012
5    *      Author: davidroussel
6    */
7
8   #ifndef QCVDFT_H_
9   #define QCVDFT_H_
10
11  #include "QcvProcessor.h"
12  #include "CvSimpleDFT.h"
13
14  /**
15   * Qt flavored Simple Fourier transform
16   */
17  class QcvSimpleDFT: public QcvProcessor, public CvSimpleDFT
18  {
19      Q_OBJECT
20
21      public:
22
23          /**
24           * QcvSimpleDFT constructor
25           * @param image the source image
26           * @param imageLock the mutex on source image
27           * @param updateThread the thread in which this processor runs
28           * @param parent parent QObject
29           */
30          QcvSimpleDFT(Mat * image,
31                       QMutex * imageLock = NULL,
32                       QThread * updateThread = NULL,
33                       QObject * parent = NULL);
34
35          /**
36           * QcvSimpleDFT destructor
37           */
38          virtual ~QcvSimpleDFT();
39
40          // ------------------------------------------------------------------
41          // Options settings with message notification
42          // ------------------------------------------------------------------
43
44      public slots:
45          /**
46           * Update computed images slot and sends updated signal
47           * required
48           */
49          void update();
50
51          /**
52           * Changes source image slot.
53           * Attributes needs to be cleaned up then set up again
54           * @param image the new source Image
55           */
56          void setSourceImage(Mat * image)
57              throw (CvProcessorException);
58
59      signals:
60
61  //      /**
62  //       * Signal sent when source image changes to adjust max filter sizes
63  //       */
64  //      void dftSizeChanged();
65
66          /**
67           * Signal sent when input dftSize square image has been reallocated
68           * @param image the new in square image
69           */
70          void squareImageChanged(Mat * image);
71
72          /**
73           * Signal sent when spectrum image has been reallocated
74           * @param image the new spectrum image
75           */
76          void spectrumImageChanged(Mat * image);
77
78          /**
79           * Signal sent when inverse image has been reallocated
80           * @param image the new inverse image
81           */
82          void inverseImageChanged(Mat * image);
```

```
};

#endif /* QCVDFT_H_ */
```



```
/*
 * QcvSimpleDFT.cpp
 *
 *  Created on: 22 fÃ©vr. 2012
 *      Author: davidroussel
 */

#include "QcvSimpleDFT.h"


/*
 * QcvSimpleDFT 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
 */
QcvSimpleDFT::QcvSimpleDFT(Mat * image,
                           QMutex * imageLock,
                           QThread * updateThread,
                           QObject * parent) :
    CvProcessor(image), // <-- virtual base class constructor first
    QcvProcessor(image, imageLock, updateThread, parent),
    CvSimpleDFT(image)
{
}

/*
 * QcvSimpleDFT destructor
 */
QcvSimpleDFT::~QcvSimpleDFT()
{
    message.clear();
}

/*
 * Update computed images slot and sends updated signal
 * required
 */
void QcvSimpleDFT::update()
{
    if (sourceLock ≠ NULL)
    {
        sourceLock→lock();
        // qDebug() << "QcvSimpleDFT::update : lock";
    }

    /*
     * Update DFT images
     */
    CvSimpleDFT::update();

    if (sourceLock ≠ NULL)
    {
        // qDebug() << "QcvSimpleDFT::update : unlock";
        sourceLock→unlock();
    }

    /*
     * emit updated signal
     */
    QcvProcessor::update();
}

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

    QcvProcessor::setSourceImage(image);

    emit squareImageChanged(&inFrameSquare);

    emit spectrumImageChanged(&spectrumMagnitudeImage);

    if ((previousDftSize.width ≠ dftSize.width) ∨
        (previousDftSize.height ≠ dftSize.height))
```

```
83      {
84          emit imageSizeChanged();
85          emit sendText(QString::number(optimalDFTSize));
86      }
87
88      // Force update
89      update();
90  }
```



```
1   /*
2    * QcvMatWidget.h
3    *
4    *  Created on: 28 fÃ©vr. 2011
5    *^H      Author: davidroussel
6    */
7
8   #ifndef QCVMATWIDGET_H_
9   #define QCVMATWIDGET_H_
10
11  #include <QWidget>
12  #include <QHBoxLayout>
13  #include <QMouseEvent>
14  #include <QPoint>
15
16  #include <cv.h>
17  using namespace cv;
18
19  /**
20   * Abstract widget to show OpenCV Mat image into QT.
21   * Should be refined in
22   *  - QcvMatWidgetLabel
23   *  - QcvMatWidgetImage
24   *  - QcvMatWidgetGL
25   */
26  class QcvMatWidget : public QWidget
27  {
28      Q_OBJECT
29
30      public:
31          /**
32           * Mouse sensivity of the image widget
33           */
34          typedef enum
35          {
36              /**
37               * Sensitive to no mouse click or drag
38               */
39              MOUSE_NONE = 0,
40              /**
41               * Sensitive to mouse clicks
42               */
43              MOUSE_CLICK = 1,
44              /**
45               * Sensitive to mouse drag
46               */
47              MOUSE_DRAG = 2,
48              /**
49               * Sensitive to mouse click and drag
50               */
51              MOUSE_CLICK_AND_DRAG = 3
52          } MouseSense;
53
54      protected:
55          /**
56           * The widget layout
57           */
58          QHBoxLayout * layout;
59
60          /**
61           * The OpenCV BGR or gray image
62           */
63          Mat * sourceImage;
64
65          /**
66           * The OpenCV RGB image converted from gray or BGR OpenCV image
67           */
68          Mat displayImage;
69
70          /**
71           * Default size when no image has been set
72           */
73          static QSize defaultSize;
74
75          /**
76           * the aspect ratio ofthe image to draw
77           */
78          double aspectRatio;
79
80          /**
81           * Default aspect ratio when image is not set yet
82           */
```

```
83          static double defaultAspectRatio;
84
85          /**
86           * Indicate a mouse button is currently pressed within the widget
87           */
88          bool mousePressed;
89
90          /**
91           * Indicate a mouse is moved after a button has been pressed
92           */
93          bool mouseMoved;
94
95          /**
96           * Mouse sensivity
97           */
98          MouseSense mouseSense;
99
100         /**
101          * mouse pressed location
102          */
103         QPoint pressedPoint;
104
105         /**
106          * Mouse pressed button
107          */
108         Qt::MouseButton pressedButton;
109
110         /**
111          * mouse drag location
112          */
113         QPoint draggedPoint;
114
115         /**
116          * mouse release location
117          */
118         QPoint releasedPoint;
119
120         /**
121          * Selection rectangle
122          */
123         QRect selectionRect;
124
125         /**
126          * Drawing color
127          */
128         static const Scalar drawingColor;
129
130         /**
131          * Drawing width
132          */
133         static const int drawingWidth;
134
135 //      size_t count;
136
137     public:
138
139         /**
140          * OpenCV QT Widget default constructor
141          * @param parent parent widget
142          * @param mouseSense mouse sensivity
143          */
144         QcvMatWidget(QWidget *parent = NULL,
145                      MouseSense mouseSense = MOUSE_NONE);
146
147         /**
148          * OpenCV QT Widget constructor
149          * @param sourceImage the source image
150          * @param parent parent widget
151          * @param mouseSense mouse sensivity
152          * @pre sourceImage is not NULL
153          */
154         QcvMatWidget(Mat * sourceImage,
155                      QWidget *parent = NULL,
156                      MouseSense mouseSense = MOUSE_NONE);
157
158         /**
159          * OpenCV Widget destructor.
160          * Releases displayImage.
161          */
162         virtual ~QcvMatWidget(void);
163
164 //^H    ^H  /**
```



```
165 //^H    ^H   * Widget minimum size is set to the contained image size
166 //^H    ^H   * @return le size of the image within
167 //^H    ^H   */
168 //^H    ^H  QSize minimumSize() const;
169
170         /**
171          * Size hint (because size depends on sourceImage properties)
172          * @return size obtained from sourceImage or defaultSize if sourceImage
173          * is not set yet
174          */
175         QSize sizeHint() const;
176
177         /**
178          * Gets Mat widget mouse clickable status
179          * @return true if widget is sensitive to mouse click
180          */
181         bool isMouseClickable() const;
182
183         /**
184          * Gets Mat widget mouse dragable status
185          * @return true if widget is sensitive to mouse drag
186          */
187         bool isMouseDragable() const;
188
189     protected:
190
191         /**
192          * paint event reimplemented to draw content (in this case only
193          * draw in display image since final rendering method is not yet available)
194          * @param event the paint event
195          */
196         virtual void paintEvent(QPaintEvent * event);
197
198         /**
199          * Widget setup
200          * @post new Layout has been created and set for this widget
201          */
202         void setup();
203
204         /**
205          * Converts BGR or Gray source image to RGB display image
206          * @pre sourceImage is not NULL
207          * @post BGR or Gray source image has been converted to RGB displayimage
208          * @see #sourceImage
209          * @see #displayImage
210          */
211         void convertImage();
212
213         /**
214          * Callback called when mouse button pressed event occurs.
215          * reimplemented to send pressPoint signal when left mouse button is
216          * pressed
217          * @param event mouse event
218          */
219         void mousePressEvent(QMouseEvent *event);
220
221         /**
222          * Callback called when mouse move event occurs.
223          * reimplemented to send dragPoint signal when mouse is dragged
224          * (after left mouse button has been pressed)
225          * @param event mouse event
226          */
227         void mouseMoveEvent(QMouseEvent *event);
228
229         /**
230          * Callback called when mouse button released event occurs.
231          * reimplemented to send releasePoint signal when left mouse button is
232          * released
233          * @param event mouse event
234          */
235         void mouseReleaseEvent(QMouseEvent *event);
236
237         /**
238          * Draw Cross
239          * @param p the cross center
240          */
241         virtual void drawCross(const QPoint & p);
242
243         /**
244          * Draw rectangle
245          * @param r the rectangle to draw
246          */
```

```
247         virtual void drawRectangle(const QRect & r);
248
249 //      /*
250 //       * paint event reimplemented to draw content
251 //       * @param event the paint event
252 //       */
253 //      virtual void paintEvent(QPaintEvent * event) = 0;
254
255         /**
256          * Modifiy selectionRect using two points
257          * @param p1 first point
258          * @param p2 second point
259          */
260         void selectionRectFromPoints(const QPoint & p1, const QPoint & p2);
261
262     public slots:
263         /**
264          * Sets new source image
265          * @param sourceImage the new source image
266          * @pre sourceimage is not NULL
267          * @post new sourceImage has been set and aspectRatio has been updated
268          */
269         virtual void setSourceImage(Mat * sourceImage);
270
271         /**
272          * Update slot customized to include convertImage before actually
273          * updating
274          * @post sourceImage have been converted to RGB and widget updated
275          */
276         virtual void update();
277
278     signals:
279
280         /**
281          * Signal sent to transmit the point in the widget where a mouse
282          * button has been pressed
283          * @param p the point where any mouse button has been pressed
284          * @param button the button pressed
285          */
286         void pressPoint(const QPoint & p, const Qt::MouseButton & button);
287
288         /**
289          * Signal sent to transmit the point in the widget where mouse cursor is
290          * currently dragged to (which suppose a mouse button has been
291          * previously pressed)
292          * @param p the point where the mouse cursor is dragged to
293          */
294         void dragPoint(const QPoint & p);
295
296         /**
297          * Signal sent to transmit the point in the widget where a mouse
298          * button has been released
299          * @param p the point where left mouse button has been released
300          * @param button the button pressed
301          */
302         void releasePoint(const QPoint & p, const Qt::MouseButton & button);
303
304         /**
305          * Signal sent to transmit the rectangle selection when mouse button
306          * has been clicked, dragged and released
307          * @param r the rectangle selection
308          * @param button the button pressed during dragging
309          */
310         void releaseSelection(const QRect & r, const Qt::MouseButton & button);
311 };
312
313 #endif /* QCVMATWIDGET_H_ */
```



```
1  /*
2   * QcvMatWidget.cpp
3   *
4   *  Created on: 28 fÃ©vr. 2011
5   *      Author: davidroussel
6   */
7  #include <QtDebug>
8  #include "QcvMatWidget.h"
9
10 /*
11  * Default size when no image has been set
12  */
13 QSize QcvMatWidget::defaultSize(640, 480);
14
15 /*
16  * Default aspect ratio when image is not set yet
17  */
18 double QcvMatWidget::defaultAspectRatio = 4.0/3.0;
19
20 /*
21  * Drawing color
22  */
23 const Scalar QcvMatWidget::drawingColor(0xFF,0xCC,0x00,0x88);
24
25 /*
26  * Drawing width
27  */
28 const int QcvMatWidget::drawingWidth(3);
29
30 /*
31  * OpenCV QT Widget default constructor
32  * @param parent parent widget
33  * @param mouseSense mouse sensivity
34  */
35 QcvMatWidget::QcvMatWidget(QWidget *parent,
36                            MouseSense mouseSense) :
37     QWidget(parent),
38     sourceImage(NULL),
39     aspectRatio(defaultAspectRatio),
40     mousePressed(false),
41     mouseSense(mouseSense)
42 //  count(0)
43 {
44     setup();
45 }
46
47 /*
48  * OpenCV QT Widget constructor
49  * @param the source image
50  * @param parent parent widget
51  * @param mouseSense mouse sensivity
52  */
53 QcvMatWidget::QcvMatWidget(Mat * sourceImage,
54                            QWidget *parent,
55                            MouseSense mouseSense) :
56     QWidget(parent),
57     sourceImage(sourceImage),
58     aspectRatio((double)sourceImage→cols / (double)sourceImage→rows),
59     mousePressed(false),
60     mouseSense(mouseSense)
61 //  count(0)
62 {
63     setup();
64 }
65
66 /*
67  * OpenCV Widget destructor.
68  * Releases displayImage.
69  */
70 QcvMatWidget::~QcvMatWidget()
71 {
72     displayImage.release();
73 }
74
75 /*
76  * paint event reimplemented to draw content (in this case only
77  * draw in display image since final rendering method is not yet available)
78  * @param event the paint event
79  */
80 void QcvMatWidget::paintEvent(QPaintEvent * event)
81 {
82     Q_UNUSED(event);
```

```
83
84      if (displayImage.data ≠ NULL)
85      {
86          // evt draw in image
87          if (mousePressed)
88          {
89              // if MOUSE_CLICK only draws a cross
90              if (mouseSense > MOUSE_NONE)
91              {
92                  if (¬(mouseSense & MOUSE_DRAG))
93                  {
94                      if (mouseMoved)
95                      {
96                          drawCross(draggedPoint);
97                      }
98                      else
99                      {
100                         drawCross(pressedPoint);
101                     }
102                 }
103                 else  // else if MOUSE_DRAG starts drawing a rectangle
104                 {
105                     drawRectangle(selectionRect);
106                 }
107             }
108         }
109     }
110     else
111     {
112         qWarning("QcvMatWidget::paintEvent : image.data is NULL");
113     }
114 }
115
116
117 /*
118  * Widget setup
119  */
120 void QcvMatWidget::setup()
121 {
122     layout = new QHBoxLayout();
123     layout→setContentsMargins(0,0,0,0);
124     setLayout(layout);
125 }
126
127 /*
128  * Sets new source image
129  * @param sourceImage the new source image
130  */
131 void QcvMatWidget::setSourceImage(Mat * sourceImage)
132 {
133     // qDebug("QcvMatWidget::setSourceImage");
134
135     this→sourceImage = sourceImage;
136
137     // re-setup geometry since height x width may have changed
138     aspectRatio = (double)sourceImage→cols / (double)sourceImage→rows;
139     // qDebug ("aspect ratio changed to %4.2f", aspectRatio);
140
141 }
142
143 /*
144  * Converts BGR or Gray source image to RGB display image
145  * @see #sourceImage
146  * @see #displayImage
147  */
148 void QcvMatWidget::convertImage()
149 {
150 //  qDebug("Convert image");
151
152     int depth = sourceImage→depth();
153     int channels = sourceImage→channels();
154
155     // Converts any image type to RGB format
156     switch (depth)
157     {
158         case CV_8U:
159             switch (channels)
160             {
161                 case 1: // gray level image
162                     cvtColor(*sourceImage, displayImage,CV_GRAY2RGB);
163                     break;
164                 case 3: // Color image (OpenCV produces BGR images)
```



```
165                     cvtColor(*sourceImage, displayImage, CV_BGR2RGB);
166                     break;
167                 default:
168                     qFatal("This number of channels (%d) is not supported",
169                            channels);
170                     break;
171             }
172             break;
173         default:
174             qFatal("This image depth (%d) is not implemented in QOpenCVWidget",
175                    depth);
176             break;
177     }
178 }
179
180 /*
181  * Callback called when mouse button pressed event occurs.
182  * reimplemented to send pressPoint signal when left mouse button is
183  * pressed
184  * @param event mouse event
185  */
186 void QcvMatWidget::mousePressEvent(QMouseEvent *event)
187 {
188     if (mouseSense > MOUSE_NONE)
189     {
190 //      qDebug("mousePressEvent(%d, %d) with button %d",
191 //             event->pos().x(), event->pos().y(), event->button());
192         mousePressed = true;
193         pressedPoint = event→pos();
194         pressedButton = event→button();
195
196         if((event→button() ≡ Qt::LeftButton) ∧ (mouseSense & MOUSE_DRAG))
197         {
198             // initialise selection rect
199             selectionRect.setTopLeft(pressedPoint);
200             selectionRect.setBottomRight(pressedPoint);
201         }
202
203         emit pressPoint(pressedPoint, pressedButton);
204     }
205 }
206
207 /*
208  * Callback called when mouse move event occurs.
209  * reimplemented to send dragPoint signal when mouse is dragged
210  * (after left mouse button has been pressed)
211  * @param event mouse event
212  */
213 void QcvMatWidget::mouseMoveEvent(QMouseEvent *event)
214 {
215     mouseMoved = true;
216     draggedPoint = event→pos();
217
218     if ((mouseSense & MOUSE_DRAG) ∧ mousePressed)
219     {
220 //      qDebug("mouseMoveEvent(%d, %d) with button %d",
221 //             event->pos().x(), event->pos().y(), event->button());
222
223         selectionRectFromPoints(pressedPoint, draggedPoint);
224
225         emit dragPoint(draggedPoint);
226     }
227 }
228
229 /*
230  * Callback called when mouse button released event occurs.
231  * reimplemented to send releasePoint signal when left mouse button is
232  * released
233  * @param event mouse event
234  */
235 void QcvMatWidget::mouseReleaseEvent(QMouseEvent *event)
236 {
237     if ((mouseSense > MOUSE_NONE) ∧ mousePressed)
238     {
239 //      qDebug("mouseReleaseEvent(%d, %d) with button %d",
240 //             event->pos().x(), event->pos().y(), event->button());
241         mousePressed = false;
242         mouseMoved = false;
243         releasedPoint = event→pos();
244         emit releasePoint(releasedPoint, pressedButton);
245
246         if ((event→button() ≡ Qt::LeftButton) ∧ (mouseSense & MOUSE_DRAG))
```

```
247         {
248             selectionRectFromPoints(pressedPoint, releasedPoint);
249             emit releaseSelection(selectionRect, event→button());
250         }
251     }
252 }
253
254 /*
255  * Draw Cross
256  * @param p the cross center
257  */
258 void QcvMatWidget::drawCross(const QPoint & p)
259 {
260     int x0 = p.x();
261     int y0 = p.y();
262     int x1, x2, x3, x4;
263     int y1, y2, y3, y4;
264     int offset = 10;
265
266     x1 = x0 − 2*offset;
267     x2 = x0 − offset;
268     x3 = x0 + offset;
269     x4 = x0 + 2*offset;
270     y1 = y0 − 2*offset;
271     y2 = y0 − offset;
272     y3 = y0 + offset;
273     y4 = y0 + 2*offset;
274
275     Point p1a(x1, y0);
276     Point p1b(x2, y0);
277     Point p2a(x3, y0);
278     Point p2b(x4, y0);
279     Point p3a(x0, y1);
280     Point p3b(x0, y2);
281     Point p4a(x0, y3);
282     Point p4b(x0, y4);
283
284     line(displayImage, p1a, p1b, drawingColor, drawingWidth, CV_AA);
285     line(displayImage, p2a, p2b, drawingColor, drawingWidth, CV_AA);
286     line(displayImage, p3a, p3b, drawingColor, drawingWidth, CV_AA);
287     line(displayImage, p4a, p4b, drawingColor, drawingWidth, CV_AA);
288 }
289
290 /*
291  * Draw rectangle
292  * @param r the rectangle to draw
293  */
294 void QcvMatWidget::drawRectangle(const QRect & r)
295 {
296     int x1 = r.left();
297     int x2 = r.right();
298     int y1 = r.top();
299     int y2 = r.bottom();
300
301     Point p1(x1, y1);
302     Point p2(x2, y2);
303
304     rectangle(displayImage, p1, p2, drawingColor, drawingWidth, CV_AA);
305 }
306
307 /**
308  * Modifiy selectionRect using two points
309  * @param p1 first point
310  * @param p2 second point
311  */
312 void QcvMatWidget::selectionRectFromPoints(const QPoint & p1, const QPoint & p2)
313 {
314     int left, right, top, bottom;
315     if (p1.x() < p2.x())
316     {
317         left = p1.x();
318         right = p2.x();
319     }
320     else
321     {
322         left = p2.x();
323         right = p1.x();
324     }
325
326     if (p1.y() < p2.y())
327     {
328         top = p1.y();
```



```
329         bottom = p2.y();
330     }
331     else
332     {
333         top = p2.y();
334         bottom = p1.y();
335     }
336
337     selectionRect.setLeft(left);
338     selectionRect.setRight(right);
339     selectionRect.setTop(top);
340     selectionRect.setBottom(bottom);
341 }
342
343
344
345 /*
346  * Widget minimum size is set to the contained image size
347  * @return le size of the image within
348  */
349 //QSize QcvMatWidget::minimumSize() const
350 //{
351 //  return sizeHint();
352 //}
353
354
355 /*
356  * Size hint (because size depends on sourceImage properties)
357  * @return size obtained from sourceImage
358  */
359 QSize QcvMatWidget::sizeHint() const
360 {
361     if (sourceImage ≠ NULL)
362     {
363         return QSize(sourceImage→cols, sourceImage→rows);
364     }
365     else
366     {
367         return defaultSize;
368     }
369 }
370
371 /*
372  * Gets Mat widget mouse clickable status
373  * @return true if widget is sensitive to mouse click
374  */
375 bool QcvMatWidget::isMouseClickable() const
376 {
377     return (mouseSense & MOUSE_CLICK);
378 }
379
380 /*
381  * Gets Mat widget mouse dragable status
382  * @return true if widget is sensitive to mouse drag
383  */
384 bool QcvMatWidget::isMouseDragable() const
385 {
386     return (mouseSense & MOUSE_DRAG);
387 }
388
389 /*
390  * Update slot customized to include convertImage before actually
391  * updating
392  */
393 void QcvMatWidget::update()
394 {
395 //  count++;
396 //  qDebug() << "QcvMatWidget::update " << count;
397 //  std::cerr << "{o";
398     convertImage();
399     QWidget::update();
400 //  std::cerr << "}";
401 }
402
403 // ------------------------------------------------------------------------
404 // convertImage old algorithm
405 // ------------------------------------------------------------------------
406 //  int cvIndex, cvLineStart;
407 //  // switch between bit depths
408 //  switch (displayImage.depth())
409 //  {
410 //      case CV_8U:
```

```
411 //          switch (displayImage.channels())
412 //          {
413 //              case 1: // Gray level images
414 //                  if ( (displayImage.cols != image.width()) ||
415 //                       (displayImage.rows != image.height()) )
416 //                  {
417 //                      QImage temp(displayImage.cols, displayImage.rows,
418 //                                  QImage::Format_RGB32);
419 //                      image = temp;
420 //                  }
421 //                  cvIndex = 0;
422 //                  cvLineStart = 0;
423 //                  for (int y = 0; y < displayImage.rows; y++)
424 //                  {
425 //                      unsigned char red, green, blue;
426 //                      cvIndex = cvLineStart;
427 //                      for (int x = 0; x < displayImage.cols; x++)
428 //                      {
429 //                          // DO it
430 //                          red   = displayImage.data[cvIndex];
431 //                          green = displayImage.data[cvIndex];
432 //                          blue  = displayImage.data[cvIndex];
433 //
434 //                          image.setPixel(x, y, qRgb(red, green, blue));
435 //                          cvIndex++;
436 //                      }
437 //                      cvLineStart += displayImage.step;
438 //                  }
439 //                  break;
440 //              case 3: // BGR images (Regular OpenCV Color Capture)
441 //                  if ( (displayImage.cols != image.width()) ||
442 //                       (displayImage.rows != image.height()) )
443 //                  {
444 //                      QImage temp(displayImage.cols, displayImage.rows,
445 //                                  QImage::Format_RGB32);
446 //                      image = temp;
447 //                  }
448 //                  cvIndex = 0;
449 //                  cvLineStart = 0;
450 //                  for (int y = 0; y < displayImage.rows; y++)
451 //                  {
452 //                      unsigned char red, green, blue;
453 //                      cvIndex = cvLineStart;
454 //                      for (int x = 0; x < displayImage.cols; x++)
455 //                      {
456 //                          // DO it
457 //                          red   = displayImage.data[cvIndex + 2];
458 //                          green = displayImage.data[cvIndex + 1];
459 //                          blue  = displayImage.data[cvIndex + 0];
460 //
461 //                          image.setPixel(x, y, qRgb(red, green, blue));
462 //                          cvIndex += 3;
463 //                      }
464 //                      cvLineStart += displayImage.step;
465 //                  }
466 //                  break;
467 //              default:
468 //                  printf("This number of channels is not supported\n");
469 //                  break;
470 //          }
471 //          break;
472 //      default:
473 //          printf("This type of Image is not implemented in QcvMatWidget\n");
474 //          break;
475 //  }
476
```



```
1
2   #ifndef QCVMATWIDGETLABEL_H
3   #define QCVMATWIDGETLABEL_H
4
5   #include <cv.h>
6   #include <QLabel>
7
8   using namespace cv;
9
10  #include "QcvMatWidget.h"
11
12  /**
13   * OpenCV Widget for QT with QImage display
14   */
15  class QcvMatWidgetLabel : public QcvMatWidget
16  {
17      private:
18          /**
19           * The Image Label
20           */
21          QLabel * imageLabel;
22
23      public:
24          /**
25           * OpenCV QT Widget default constructor
26           * @param parent parent widget
27           * @param mouseSense mouse sensivity
28           */
29          QcvMatWidgetLabel(QWidget *parent = NULL,
30                            MouseSense mouseSense = MOUSE_NONE);
31
32          /**
33           * OpenCV QT Widget constructor
34           * @param sourceImage the source OpenCV qImage
35           * @param parent parent widget
36           * @param mouseSense mouse sensivity
37           */
38          QcvMatWidgetLabel(Mat * sourceImage,
39                            QWidget *parent = NULL,
40                            MouseSense mouseSense = MOUSE_NONE);
41
42          /**
43           * OpenCV Widget destructor.
44           */
45          virtual ~QcvMatWidgetLabel(void);
46
47      protected:
48          /**
49           * Widget setup
50           * @pre imageLabel has been allocated
51           * @post imageLabel has been added to the layout
52           */
53          void setup();
54
55          /**
56           * paint event reimplemented to draw content
57           * @param event the paint event
58           * @pre imageLabel has been allocated
59           * @post displayImage has been set as pixmap of the imageLabel
60           */
61          void paintEvent(QPaintEvent * event);
62
63  };
64
65  #endif //QCVMATWIDGETLABEL_H
```

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

#ifndef QCVMATWIDGETIMAGE_H_
#define QCVMATWIDGETIMAGE_H_

#include <QImage>
#include <QPainter>

#include "QcvMatWidget.h"

/**
 * OpenCV Widget for QT with a QPainter to draw image
 */
class QcvMatWidgetImage: public QcvMatWidget
{
    protected:
        /**
         * the Qimage to display in the widget with a QPainter
         */
        QImage * qImage;

//      /**
//       * Size Policy returned by
//       */
//      QSizePolicy policy;

    public:
        /**
         * Default Constructor
         * @param parent parent widget
         * @param mouseSense mouse sensivity
         */
        QcvMatWidgetImage(QWidget *parent = NULL,
                          MouseSense mouseSense = MOUSE_NONE);

        /**
         * Constructor
         * @param sourceImage source image
         * @param parent parent widget
         * @param mouseSense mouse sensivity
         */
        QcvMatWidgetImage(Mat * sourceImage,
                          QWidget *parent = NULL,
                          MouseSense mouseSense = MOUSE_NONE);

        /**
         * Destructor.
         */
        virtual ~QcvMatWidgetImage();

//      /**
//       * Minimum size hint according to aspect ratio and min height of 100
//       * @return minimum size hint
//       */
//      QSize minimumSizeHint() const;

        /**
         * aspect ratio method
         * @param w width
         * @return the required height fo r this width
         */
        int heightForWidth ( int w ) const;

//      /**
//       * Size policy to keep aspect ratio right
//       * @return
//       */
//      QSizePolicy sizePolicy () const;

        /**
         * Sets new source image
         * @param sourceImage the new source image
         */
        virtual void setSourceImage(Mat * sourceImage);

    protected:
        /**
```

```
83          * Setup widget (defines size policy)
84          */
85         void setup();
86
87         /**
88          * paint event reimplemented to draw content
89          * @param event the paint event
90          */
91         void paintEvent(QPaintEvent * event);
92
93 };
94
95 #endif /* QCVMATWIDGETIMAGE_H_ */
```



```
1  /*
2   * QcvMatWidgetImage.cpp
3   *
4   *  Created on: 31 janv. 2012
5   *    Author: davidroussel
6   */
7
8  #include "QcvMatWidgetImage.h"
9  #include <QPaintEvent>
10 #include <QSizePolicy>
11 #include <QDebug>
12
13 /*
14  * Default Constructor
15  * @param parent parent widget
16  */
17 QcvMatWidgetImage::QcvMatWidgetImage(QWidget *parent,
18                                      MouseSense mouseSense) :
19     QcvMatWidget(parent, mouseSense),
20     qImage(NULL)
21 {
22     setup();
23 }
24
25 /*
26  * Constructor
27  * @param sourceImage source image
28  * @param parent parent widget
29  */
30 QcvMatWidgetImage::QcvMatWidgetImage(Mat * sourceImage,
31                                      QWidget *parent,
32                                      MouseSense mouseSense) :
33     QcvMatWidget(sourceImage, parent, mouseSense),
34     qImage(NULL)
35 {
36     setSourceImage(sourceImage);
37
38     setup();
39 }
40
41 /*
42  * Setup widget (defines size policy)
43  */
44 void QcvMatWidgetImage::setup()
45 {
46 //  qDebug("QcvMatWidgetImage::Setup");
47
48     /*
49      * Customize size policy
50      */
51     QSizePolicy qsp(QSizePolicy::Fixed, QSizePolicy::Fixed);
52     // sets height depends on width (also need to reimplement heightForWidth())
53     qsp.setHeightForWidth(true);
54     setSizePolicy(qsp);
55
56     /*
57      * Customize layout
58      */
59
60     // size policy has changed to call updateGeometry
61     updateGeometry();
62 }
63
64 /*
65  * Destructor.
66  */
67 QcvMatWidgetImage::~QcvMatWidgetImage()
68 {
69     if (qImage ≠ NULL)
70     {
71         delete qImage;
72     }
73 }
74
75 /*
76  * Sets new source image
77  * @param sourceImage the new source image
78  */
79 void QcvMatWidgetImage::setSourceImage(Mat * sourceImage)
80 {
81     if (qImage ≠ NULL)
82     {
```

```
83          delete qImage;
84      }
85      // setup and convert image
86      QcvMatWidget::setSourceImage(sourceImage);
87      convertImage();
88      qImage = new QImage((uchar *) displayImage.data, displayImage.cols,
89          displayImage.rows, displayImage.step,
90          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.h
3    *
4    *  Created on: 28 fÃ©vr. 2011
5    *    Author: davidroussel
6    */
7
8   #ifndef QOPENCVWIDGETQGL_H_
9   #define QOPENCVWIDGETQGL_H_
10
11  #include <QGLWidget>
12
13  #include "QcvMatWidget.h"
14  #include "QGLImageRender.h"
15
16  /**
17   * OpenCV Widget for QT with QGLWidget display
18   */
19  class QcvMatWidgetGL: public QcvMatWidget
20  {
21      private:
22          /**
23           * QGLWidget to draw in
24           */
25          QGLImageRender * gl;
26
27  //      size_t glCount;
28
29      public:
30
31          /**
32           * OpenCV QT Widget default constructor
33           * @param parent parent widget
34           * @param mouseSense mouse sensivity
35           */
36          QcvMatWidgetGL(QWidget *parent = NULL,
37                         MouseSense mouseSense = MOUSE_NONE);
38
39          /**
40           * OpenCV QT Widget constructor
41           * @param sourceImage the source image
42           * @param parent parent widget
43           * @param mouseSense mouse sensivity
44           */
45          QcvMatWidgetGL(Mat * sourceImage,
46                         QWidget *parent = NULL,
47                         MouseSense mouseSense = MOUSE_NONE);
48
49          /**
50           * Sets new source image
51           * @param sourceImage the new source image
52           */
53          void setSourceImage(Mat * sourceImage);
54
55          /**
56           * OpenCV Widget destructor.
57           */
58          virtual ~QcvMatWidgetGL();
59
60      protected:
61          /*
62           * paint event reimplemented to draw content
63           * @param event the paint event
64           */
65          void paintEvent(QPaintEvent * event);
66  };
67
68  #endif /* QOPENCVWIDGETQGL_H_ */
```

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

#include "QcvMatWidgetGL.h"

/*
 * OpenCV QT Widget default constructor
 * @param parent parent widget
 */
QcvMatWidgetGL::QcvMatWidgetGL(QWidget *parent,
                               MouseSense mouseSense) :
    QcvMatWidget(parent, mouseSense),
    gl(NULL)
//  glCount(0)
{
}

/*
 * OpenCV QT Widget constructor
 * @param parent parent widget
 */
QcvMatWidgetGL::QcvMatWidgetGL(Mat * sourceImage,
                               QWidget *parent,
                               MouseSense mouseSense) :
    QcvMatWidget(sourceImage, parent, mouseSense),
    gl(NULL)
//  glCount(0)
{
    setSourceImage(sourceImage);
}

/*
 * OpenCV Widget destructor.
 */
QcvMatWidgetGL::~QcvMatWidgetGL()
{
    if (gl ≠ NULL)
    {
        layout→removeWidget(gl);
        delete gl;
    }
}

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

    if (gl ≠ NULL)
    {
        layout→removeWidget(gl);
        delete gl;
    }

    convertImage();

    gl = new QGLImageRender(displayImage, this);

    layout→addWidget(gl, 0, Qt::AlignCenter);
}

/*
 * paint event reimplemented to draw content
 * @param event the paint event
 */
void QcvMatWidgetGL::paintEvent(QPaintEvent * event)
{
    QcvMatWidget::paintEvent(event);
//  qDebug() << "Paint event # " << glCount++;
    gl→update();
}
```



```cpp
/*
 * QGLImageRender.h
 *
 *  Created on: 28 fÃ©vr. 2011
 *      Author: davidroussel
 */

#ifndef QGLIMAGERENDER_H_
#define QGLIMAGERENDER_H_

#include <QGLWidget>
#include <QSize>
#include <QSizePolicy>
#include <cv.h>

using namespace cv;

/**
 * A Class allowing to draw OpenCV Mat images using OpenGL
 */
class QGLImageRender: public QGLWidget
{
    private:
        /**
         * The RGB image to draw
         */
        Mat image;

//      size_t fCount;

    public:
        /**
         * QGLImageRender Constructor
         * @param image the RGB image to draw in the pixel buffer
         * @param parent the parent widget
         */
        QGLImageRender(const Mat & image, QWidget *parent = NULL);

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

        /**
         * Size hint
         * @return Qsize containing size hint
         */
        QSize sizeHint () const;

        /**
         * Minimum Size hint
         * @return QSize containing the minimum size hint
         */
        QSize minimumSizeHint() const;

        /**
         * Size Policy for this widget
         * @return A No resize at all policy
         */
        QSizePolicy sizePolicy () const;

    protected :
        /**
         * Initialise GL drawing (called once on each QGLContext)
         */
        void initializeGL();
        /**
         * Paint GL : called whenever the widget needs to be painted
         */
        void paintGL();
        /**
         * Resize GL : called whenever the widget has been resized
         */
        void resizeGL(int width, int height);
};

#endif /* QGLIMAGERENDER_H_ */
```

```
/*
 * 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::QGLImageRender(const Mat & image, QWidget *parent) :
    QGLWidget(parent),
    image(image)
//  fCount(0)
{
    if (¬doubleBuffer())
    {
        qWarning("QGLImageRender::QGLImageRender caution : no double buffer");
    }
    if (this→image.data ≡ NULL)
    {
        qWarning("QGLImageRender::QGLImageRender caution : image data 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::paintGL()
{
//  qDebug("GL drawing pixels ...");

    glClear(GL_COLOR_BUFFER_BIT);

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

}

void QGLImageRender::resizeGL(int width, int height)
{
//  qDebug("GL resizeGL ...");
//  glViewport(0, 0, width, height);
//  glMatrixMode(GL_PROJECTION);
//  glLoadIdentity();
//  gluOrtho2D(0.0, 0.0, (GLdouble)width, (GLdouble)height);

    qDebug("GL Resize (%d, %d)",width, height);

//  GLfloat zoom, xZoom, yZoom;
//
//  xZoom = (GLfloat)width/(GLfloat)image.cols;
//  yZoom = (GLfloat)height/(GLfloat)image.rows;
//
//  if (xZoom < yZoom)
//  {
//      zoom = xZoom;
//  }
//  else
```



```
//  {
//      zoom = yZoom;
//  }

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

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (image.data ≠ NULL)
    {
//      gluOrtho2D(0, (GLdouble) image.cols, 0, (GLdouble) image.rows);
        glOrtho(0, (GLdouble) image.cols, 0, (GLdouble) image.rows, 1.0, -1.0);
    }

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    /* apply the right translate so the image drawing starts top left */
    if (image.data ≠ NULL)
    {
        /*
         * For some reason we should not start drawing exactly at the limit
         * of the drawing plane so we start drawing at image.rows - something
         * which could be very tiny
         */
        glRasterPos2i(0,image.rows);
    }
    else
    {
        qWarning("QGLImageRender::resizeGL(...) : image.data is NULL");
    }

    /* apply the right zoom factor so image are displayed top 2 bottom */
    glPixelZoom(1.0, -1.0);
}


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.h
 *
 *  Created on: 29 janv. 2012
 *    Author: davidroussel
 */

#ifndef QCVVIDEOCAPTURE_H_
#define QCVVIDEOCAPTURE_H_

#include <QObject>
#include <QSize>
#include <QTimer>
#include <QThread>
#include <QMutex>

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

/**
 * Qt Class for capturing videos from cameras of files with OpenCV.
 * QcvVideoCapture opens streams and refresh itself automatically.
 * When frame has been refreshed a signal is emitted.
 */
class QcvVideoCapture: public QObject
{
    Q_OBJECT

    protected:

        /**
         * file name used to open video file.
         * Used to reopen video file when video is finished.
         */
        QString filename;

        /**
         * Video capture instance
         * @warning capture is regularly updated by a timer, but can also be
         * manipulated by other methods (such as #setDirectSize). So capture
         * access for new images should be protected by a mutex to ensure
         * atomic access to capture object at a time.
         */
        VideoCapture capture;

        /**
         * refresh timer
         */
        QTimer * timer;

        /**
         * Independant thread to update capture.
         * If independant thread is required, then update method is called
         * from within this thread. Otherwise, update method is called from
         * main thread.
         */
        QThread * updateThread;

        /**
         * Mutex lock to ensure atomic access capture grabbing new image.
         * @warning if QcvVideoCapture object is not updated in the
         * #updateThread, then trying to lock mutex multiple times with
         * mutex.lock() will lead to a deadlock, so if this object has no
         * #updateThread (if #updateThread == NULL) we should use
         * mutex.tryLock() instead and give up when lock can't be obtained with
         * tryLock(). For instance when tryLock into #update method fails, this
         * means that capture object is locked in some other method, so we don't
         * grab any new image this time and hope, we'll be able to do it next
         * time #update will be called.
         */
        QMutex mutex;

        /**
         * Mutex lock state memory to avoid locking the mutex multiple times
         * across multiple methods. When a mutex.lock() is performed locked
         * should be set to true until mutex.unlock(). Hence, if a method
         * requiring lock is performed, a second lock is avoided by checking
         * this attribute.
         */
        size_t lockLevel;

        /**
```



```
         * Image Matrix to obtain from capture
         */
        Mat image;

        /**
         * image resized (if required)
         */
        Mat imageResized;

        /**
         * [resized] image flipped (if required)
         */
        Mat imageFlipped;

        /**
         * Image converted for display:
         *  - scaled
         *  - flipped horizontally
         *  - converted to gray
         */
        Mat imageDisplay;

        /**
         * Live video indication (from cam)
         */
        bool liveVideo;

        /**
         * flipVideo to mirror image
         */
        bool flipVideo;

        /**
         * scale image to preferred width and height
         */
        bool resize;

        /**
         * scaling is performed into capture rather than through cv::resize
         * function
         */
        bool directResize;

        /**
         * image converted to gray
         */
        bool gray;

        /**
         * Allow capture to skip an image capture when lock can't be acquired
         * before grabbing a new image. Otherwise we'll wait until the lock
         * is acquired before grabbing an new image. The lock might be acquired
         * by another lenghty thread/processor during image processing.
         */
        bool skip;

        /**
         * Current Image size (might be different from natural capture image
         * size)
         */
        QSize size;

        /**
         * Capture natural image size (without resizing)
         */
        QSize originalSize;

        /**
         * Capture frame rate obtained either by getting the CV_CAP_PROP_FPS
         * VideoCapture property or by computing capture time on several images
         * @see #grabInterval
         */
        double frameRate;

        /**
         * default time interval between refresh
         */
        static int defaultFrameDelay;

        /**
         * Number of frames to test frame rate
         */
```

```
        static size_t defaultFrameNumberTest;

        /**
         * Status message to send when something changes
         */
        QString statusMessage;

        /**
         * Default message showing time (at least 2000 ms)
         */
        static int messageDelay;

    public:
        /**
         * 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(const bool flipVideo = false,
                        const bool gray = false,
                        const bool skip = true,
                        const unsigned int  width = 0,
                        const unsigned int height = 0,
                        QThread * updateThread = NULL,
                        QObject * parent = NULL);

        /**
         * 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(const int deviceId,
                        const bool flipVideo = false,
                        const bool gray = false,
                        const bool skip = true,
                        const unsigned int  width = 0,
                        const unsigned int height = 0,
                        QThread * updateThread = NULL,
                        QObject * parent = NULL);

        /**
         * 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(const QString & fileName,
                        const bool flipVideo = false,
                        const bool gray = false,
                        const bool skip = true,
                        const unsigned int  width = 0,
                        const unsigned int height = 0,
                        QThread * updateThread = NULL,
                        QObject * parent = NULL);

```



```
        /**
         * QcvVideoCapture destructor.
         * releases video capture and image
         */
        virtual ~QcvVideoCapture();

        /**
         * Size accessor
         * @return the image size
         */
        const QSize & getSize() const;

        /**
         * Gets resize state.
         * @return true if imageDisplay have been resized to preferred width and
         * height, false otherwise
         */
        bool isResized() const;

        /**
         * Gets direct resize state.
         * @return true if image can be resized directly into capture.
         * @note direct resize capabilities are tested into #grabTest which is
         * called in all constructors. So #isDirectResizeable should not be
         * called before #grabTest
         */
        bool isDirectResizeable() const;

        /**
         * Gets video flipping status
         * @return flipped video status
         */
        bool isFlipVideo() const;

        /**
         * Gets video gray converted status
         * @return the converted to gray status
         */
        bool isGray() const;

        /**
         * Gets the image skipping policy
         * @return true if new image can be skipped when previous one has not
         * been processed yet, false otherwise.
         */
        bool isSkippable() const;

        /**
         * Gets the current frame rate
         * @return the current frame rate
         */
        double getFrameRate() const;

        /**
         * Image accessor
         * @return the image to display
         */
        Mat * getImage();

        /**
         * The source image mutex
         * @return  the mutex used on image access
         */
        QMutex * getMutex();

    public slots:
        /**
         * Open new device Id
         * @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
         */
        bool open(const int deviceId,
                  const unsigned int width = 0,
                  const unsigned int height = 0);

        /**
         * Open new video file
         * @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
```

```
329          * @return true if video has been opened and timer launched
330          */
331         bool open(const QString & fileName,
332                   const unsigned int width = 0,
333                   const unsigned int height = 0);
334         /**
335          * Sets video flipping
336          * @param flipVideo flipped video or not
337          */
338         void setFlipVideo(const bool flipVideo);
339 
340         /**
341          * Sets video conversion to gray
342          * @param grayConversion the gray conversion status
343          */
344         void setGray(const bool grayConversion);
345 
346         /**
347          * Sets #imageDisplay size according to preferred width and height
348          * @param size new desired size to set
349          * @param alreadyLocked mutex lock has already been aquired so setSize does not have
350          * to acquire the lock
351          * @pre a first image have been grabbed
352          */
353         void setSize(const QSize & size);
354 
355     protected:
356         /**
357          * Performs a grab test to fill #image.
358          * if capture is opened then tries to grab and if grab succeeds then
359          * tries to retrieve image from grab and sets image size.
360          * @return true if capture is opened and successfully grabbed a first
361          * frame into #image, false otherwise
362          * @post Moreover this method determines if direct resizing is allowed
363          * on this capture instance by trying to set
364          * CV_CAP_PROP_FRAME_WIDTH and CV_CAP_PROP_FRAME_HEIGHT.
365          */
366         bool grabTest();
367 
368         /**
369          * Get or compute interval between two frames in ms and sets the
370          * frameRate attribute.
371          * Tries to get CV_CAP_PROP_FPS from capture and if not available
372          * computes times between frames by grabbing defaultNumberTest images
373          * @return interval between two frames
374          * @param message message passed to grabInterval and display ahead of
375          * the framerate computed during grabInterval
376          * @pre capture is already instanciated
377          * @post message indicating frame rate has been emitted and interval
378          * between two frames has been returned
379          */
380         int grabInterval(const QString & message);
381 
382         /**
383          * Sets #imageDisplay size according to preferred width and height
384          * @param width desired width
385          * @param height desired height
386          * @pre a first image have been grabbed
387          */
388         void setSize(const unsigned int width,
389                      const unsigned int height);
390 
391         /**
392          * Tries to set capture size directly on capture by setting properties.
393          *  - CV_CAP_PROP_FRAME_WIDTH to set frame width
394          *  - CV_CAP_PROP_FRAME_HEIGHT to set frame height
395          * @param width the width property to set on capture
396          * @param height the height property to set on capture
397          * @return true if capture is opened and if width and height have been
398          * set successfully through @code capture.set(...) @endcode. Returns
399          * false otherwise.
400          * @post if at least width or height have been set successfully, capture
401          * image is released then updated again so it will have the right
402          * dimensions.
403          * @warning if mutex lock can't be obtained to ensure atomic access to
404          * capture object, then we start recursing until we obtain that lock,
405          * which is gross and should be fixed !!!
406          */
407         bool setDirectSize(const unsigned int width, const unsigned int height);
408 
409     protected slots:
410         /**
```



```
411          * update slot trigerred by timer : Grabs a new image and sends updated()
412          * signal iff new image has been grabbed, otherwise there is no more
413          * images to grab so kills timer.
414          * @note If lock on OpenCV capture object can not be obtained then
415          * capture is skipped. This is not critical since update is called
416          * regularly by the #timer, so we'll try updating image next time.
417          */
418         void update();
419 
420     signals:
421         /**
422          * Signal emitted when a new image has been grabbed
423          */
424         void updated();
425 
426         /**
427          * Signal emitted when capture is released
428          */
429         void finished();
430 
431         /**
432          * Signal to send update message when something changes
433          * @param message the message
434          * @param timeout number of ms the message should be displayed
435          */
436         void messageChanged(const QString & message, int timeout = 0);
437 
438         /**
439          * Signal to send when image has changed after opening new device or
440          * setting new display size
441          * @param image the new image to send
442          */
443         void imageChanged(Mat * image);
444 };
445 
446 #endif /* QCVVIDEOCAPTURE_H_ */
447 
```

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

#include <QElapsedTimer>
#include <QMutexLocker>
#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);
    }
    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())
    {
        setSize(width, height);
        QString message("File ");
        message.append(fileName);
        message.append(" ");

        int delay = grabInterval(message);
        if (updateThread ≠ NULL)
        {
            updateThread→start();
        }
        timer→start(delay);
        qDebug("timer started with %d ms delay", delay);
    }
}

/*
 * QcvVideoCapture destructor.
 * releases video capture and image
 */
QcvVideoCapture::~QcvVideoCapture()
{
    // wait for the end of an update
    if (updateThread ≠ NULL)
    {
        if (lockLevel ≡ 0)
        {
            mutex.lock();
            // qDebug() << "QcvVideoCapture::~QcvVideoCapture: lock";
        }
        lockLevel++;
    }

    if (timer ≠ NULL)
    {
        if (timer→isActive())
        {
            timer→stop();
            qDebug("timer stopped");
        }

        timer→disconnect(SIGNAL(timeout()), this, SLOT(update()));
    }

    if (updateThread ≠ NULL)
    {
        lockLevel−−;
        if (lockLevel ≡ 0)
        {
            // qDebug() << "QcvVideoCapture::~QcvVideoCapture: unlock";
            mutex.unlock();
        }

        emit finished();

        // Wait until the updateThread receives the "finished" signal through
        // "quit" slot
        updateThread→wait();

        delete timer; // delete unparented timer
    }

    // relesase OpenCV ressources
    filename.clear();
    capture.release();
    imageDisplay.release();
    imageFlipped.release();
```



```
    imageResized.release();
    image.release();
}

/*
 * Open new device Id
 * @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
 */
bool QcvVideoCapture::open(const int deviceId,
                           const unsigned int width,
                           const unsigned int height)
{
    if (updateThread ≠ NULL)
    {
        if (lockLevel ≡ 0)
        {
            mutex.lock();
            // qDebug() << "QcvVideoCapture::open(" << deviceId << "...): lock";
        }
        lockLevel++;
    }

    filename.clear();
    if (timer→isActive())
    {
        timer→stop();
        qDebug("timer stopped");
    }

    if (capture.isOpened())
    {
        capture.release();
    }

    if (¬image.empty())
    {
        image.release();
    }

    capture.open(deviceId);

    bool grabbed = grabTest();

    if (grabbed)
    {
        setSize(width, height);

        statusMessage.clear();
        statusMessage.append("Camera ");
        statusMessage.append(QString::number(deviceId));
        statusMessage.append(" ");
        int delay = grabInterval(statusMessage);
        timer→start(delay);
        liveVideo = true;
        qDebug("timer started with %d ms delay", delay);

        // emit
        // message changed already emitted by grabInterval()
        emit imageChanged(&imageDisplay);

    }
    if (updateThread ≠ NULL)
    {
        lockLevel−−;
        if (lockLevel ≡ 0)
        {
            // qDebug() << "QcvVideoCapture::open(" << deviceId << "...): unlock";
            mutex.unlock();
        }
    }

    return grabbed;
}

/*
 * Open new video file
 * @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
 */
bool QcvVideoCapture::open(const QString & fileName,
                           const unsigned int width,
                           const unsigned int height)
{
    filename = fileName;

    if (timer→isActive())
    {
        timer→stop();
        qDebug("timer stopped");
    }

    if (updateThread ≠ NULL)
    {
        if (lockLevel ≡ 0)
        {
            mutex.lock();
            // qDebug() << "QcvVideoCapture::open(" << fileName << "...): lock";
        }
        lockLevel++;
    }

    if (capture.isOpened())
    {
        capture.release();
    }

    if (¬image.empty())
    {
        image.release();
    }

    capture.open(fileName.toStdString());

    bool grabbed = grabTest();

    if (grabbed)
    {
        setSize(width, height);
        qDebug() << "open setSize done";
        statusMessage.clear();
        statusMessage.append("file ");
        statusMessage.append(fileName);
        statusMessage.append(" opened");

        int delay = grabInterval(statusMessage);
        timer→start(delay);
        liveVideo = false;
        qDebug("timer started with %d ms delay", delay);

        // emit changes
        // messageChanged already emitted by grabInterval
        emit imageChanged(&imageDisplay);

    }

    if (updateThread ≠ NULL)
    {
        lockLevel−−;
        if(lockLevel ≡ 0)
        {
            // qDebug() << "QcvVideoCapture::open(" << filename << "...): unlock";
            mutex.unlock();
        }
    }

    return grabbed;
}

/*
 * Size accessor
 * @return the image size
 */
const QSize & QcvVideoCapture::getSize() const
{
    return size;
}

/*
 * Sets #imageDisplay size according to preferred width and height
```



```
 * @param width desired width
 * @param height desired height
 * @pre a first image have been grabbed
 */
void QcvVideoCapture::setSize(const unsigned int width,
                              const unsigned int height)
{
    if ((updateThread ≠ NULL))
    {
        if (lockLevel ≡ 0)
        {
            mutex.lock();
            // qDebug("QcvVideoCapture::setSize(%d, %d) locked", width, height);
        }
        lockLevel++;
    }

    unsigned int preferredWidth;
    unsigned int preferredHeight;

    // qDebug("QcvVideoCapture::setSize(%d, %d)", width, height);

    // if not empty then release it
    if (¬imageResized.empty())
    {
        imageResized.release();
    }

    if ((width ≡ 0) ∧ (height ≡ 0)) // reset to original size
    {
        if (directResize) // direct set size to original size
        {
            setDirectSize((unsigned int)originalSize.width(),
                          (unsigned int)originalSize.height());
            // image is updated into setDirectSize
        }
        preferredWidth = image.cols;
        preferredHeight = image.rows;

        resize = false;
        imageResized = image;
    }
    else // width != 0 or height != 0
    {
        if ((width ≡ (unsigned int)image.cols) ∧
            (height ≡ (unsigned int)image.rows)) // unchanged
        {
            preferredWidth = image.cols;
            preferredHeight = image.rows;
            imageResized = image;

            if (((int)preferredWidth ≡ originalSize.width()) ∧
                ((int)preferredHeight ≡ originalSize.height()))
            {
                resize = false;
            }
            else
            {
                resize = true;
            }
        }
        else // width or height have changed
        {
            /*
             * Resize needed
             */
            preferredWidth = width;
            preferredHeight = height;

            resize = true;

            if (directResize)
            {
                setDirectSize(preferredWidth, preferredHeight);
                imageResized = image;
            }
            else
            {
                imageResized = Mat(preferredHeight, preferredWidth, image.type());
            }
        }
    }
```

```
493
494     if (updateThread ≠ NULL)
495     {
496         lockLevel−−;
497         if (lockLevel ≡ 0)
498         {
499             // qDebug("QcvVideoCapture::setSize unlocked");
500             mutex.unlock();
501         }
502     }
503
504     qDebug("QcvVideoCapture resize is %s [%s]",
505            (resize ? "ON" : "OFF"),
506            (directResize ? "direct" : "soft"));
507
508     size.setWidth(preferredWidth);
509     size.setHeight(preferredHeight);
510     statusMessage.clear();
511     statusMessage.sprintf("Size set to %dx%d", preferredWidth, preferredHeight);
512     emit messageChanged(statusMessage, messageDelay);
513
514
515     /*
516      * imageChanged signal is delayed until setGray is called into
517      * setFlipVideo
518      */
519     // Refresh image chain
520     setFlipVideo(flipVideo);
521 }
522
523 /*
524  * Sets #imageDisplay size according to preferred width and height
525  * @param size new desired size to set
526  * @pre a first image have been grabbed
527  */
528 void QcvVideoCapture::setSize(const QSize & size)
529 {
530     setSize(size.width(), size.height());
531 }
532
533 /*
534  * Sets video flipping
535  * @param flipVideo flipped video or not
536  */
537 void QcvVideoCapture::setFlipVideo(const bool flipVideo)
538 {
539     bool previousFlip = this→flipVideo;
540     this→flipVideo = flipVideo;
541
542     if (updateThread ≠ NULL)
543     {
544         if (lockLevel ≡ 0)
545         {
546             mutex.lock();
547             // qDebug() << "QcvVideoCapture::setFlipVideo(): lock";
548         }
549         lockLevel++;
550     }
551
552     if (¬imageFlipped.empty())
553     {
554         imageFlipped.release();
555     }
556
557     if (flipVideo)
558     {
559         imageFlipped = Mat(imageResized.size(), imageResized.type());
560     }
561     else
562     {
563         imageFlipped = imageResized;
564     }
565
566     if (updateThread ≠ NULL)
567     {
568         lockLevel−−;
569         if (lockLevel ≡ 0)
570         {
571             // qDebug() << "QcvVideoCapture::setFlipVideo(): unlock";
572             mutex.unlock();
573         }
574     }
```



```
575
576     if (previousFlip ≠ flipVideo)
577     {
578         statusMessage.clear();
579         statusMessage.sprintf("flip video is %s", (flipVideo ? "on" : "off"));
580         emit messageChanged(statusMessage, messageDelay);
581         emit imageChanged(&imageDisplay);
582     }
583
584     /*
585      * imageChanged signal is delayed until setGray is called
586      */
587     // refresh image chain
588     setGray(gray);
589 }
590
591 /*
592  * Sets video conversion to gray
593  * @param grayConversion the gray conversion status
594  */
595 void QcvVideoCapture::setGray(const bool grayConversion)
596 {
597     bool previousGray = gray;
598
599     gray = grayConversion;
600
601     if (updateThread ≠ NULL)
602     {
603         if (lockLevel ≡ 0)
604         {
605             mutex.lock();
606             // qDebug() << "QcvVideoCapture::setGray(): lock";
607         }
608         lockLevel++;
609     }
610
611     if (¬imageDisplay.empty())
612     {
613         imageDisplay.release();
614     }
615
616     if (gray)
617     {
618         imageDisplay = Mat(imageFlipped.size(), CV_8UC1);
619     }
620     else
621     {
622         imageDisplay = imageFlipped;
623     }
624
625     if (updateThread ≠ NULL)
626     {
627         lockLevel−−;
628         if (lockLevel ≡ 0)
629         {
630             mutex.unlock();
631             // qDebug() << "QcvVideoCapture::setGray(): unlock";
632         }
633     }
634
635     if (previousGray ≠ grayConversion)
636     {
637         statusMessage.clear();
638         statusMessage.sprintf("gray video is %s", (gray ? "on" : "off"));
639         emit messageChanged(statusMessage, messageDelay);
640     }
641
642     /*
643      * In any cases emit image changed since
644      *  − setSize may have been called
645      *  − setFlipVideo may have been called
646      */
647     emit imageChanged(&imageDisplay);
648 }
649
650 /*
651  * Gets resize state.
652  * @return true if imageDisplay have been resized to preferred width and
653  * height, false otherwise
654  */
655 bool QcvVideoCapture::isResized() const
656 {
```

```
657     return resize;
658 }
659
660 /*
661  * Gets direct resize state.
662  * @return true if image can be resized directly into capture.
663  * @note direct resize capabilities are tested into #grabTest which is
664  * called in all constructors. So #isDirectResizeable should not be
665  * called before #grabTest
666  */
667 bool QcvVideoCapture::isDirectResizeable() const
668 {
669     return directResize;
670 }
671
672 /*
673  * Gets video flipping status
674  * @return flipped video status
675  */
676 bool QcvVideoCapture::isFlipVideo() const
677 {
678     return flipVideo;
679 }
680
681 /*
682  * Gets video gray converted status
683  * @return the converted to gray status
684  */
685 bool QcvVideoCapture::isGray() const
686 {
687     return gray;
688 }
689
690 /*
691  * Gets the image skipping policy
692  * @return true if new image can be skipped when previous one has not
693  * been processed yet, false otherwise.
694  */
695 bool QcvVideoCapture::isSkippable() const
696 {
697     return skip;
698 }
699
700
701 /*
702  * Gets the current frame rate
703  * @return the current frame rate
704  */
705 double QcvVideoCapture::getFrameRate() const
706 {
707     return frameRate;
708 }
709
710
711 /*
712  * Image accessor
713  * @return the image
714  */
715 Mat * QcvVideoCapture::getImage()
716 {
717     return &imageDisplay;
718 }
719
720 /*
721  * The source image mutex
722  * @return  the mutex used on image access
723  */
724 QMutex * QcvVideoCapture::getMutex()
725 {
726     return &mutex;
727 }
728
729
730 /*
731  * Performs a grab test to fill #image
732  * @return true if capture is opened and successfully grabs a first
733  * frame into #image, false otherwise
734  */
735 bool QcvVideoCapture::grabTest()
736 {
737 //  qDebug("Grab test");
738     bool result = false;
```



```
739
740     if (capture.isOpened())
741     {
742 #ifndef Q_OS_LINUX // V4L does not support these queries
743         int capWidth = capture.get(CV_CAP_PROP_FRAME_WIDTH);
744         int capHeight = capture.get(CV_CAP_PROP_FRAME_HEIGHT);
745
746         qDebug("Capture grab test with %d x %d image", capWidth, capHeight);
747 #endif
748         // grabs first frame
749         if (capture.grab())
750         {
751             bool retrieved = capture.retrieve(image);
752             if (retrieved)
753             {
754                 size.setWidth(image.cols);
755                 size.setHeight(image.rows);
756                 originalSize.setWidth(image.cols);
757                 originalSize.setHeight(image.rows);
758
759                 /*
760                  * Tries to determine if direct resizing in capture is possible
761                  * by setting original size through properties
762                  * Typically :
763                  *  - camera capture might be resizable
764                  *  - video file capture may not be resizable
765                  */
766                 directResize = setDirectSize(image.cols, image.rows);
767
768                 qDebug("Capture direct resizing is %s",
769                        (directResize ? "on" : "off"));
770
771                 result = true;
772             }
773             else
774             {
775                 qFatal("Video Capture unable to retreive image");
776             }
777         }
778         else
779         {
780             qFatal("Video Capture can not grab");
781         }
782     }
783     else
784     {
785         qFatal("Video Capture is not opened");
786     }
787
788     return result;
789 }
790
791 /*
792  * Get or compute interval between two frames
793  * @return interval between two frames
794  * @pre capture is already instanciated
795  */
796 int QcvVideoCapture::grabInterval(const QString & message)
797 {
798     int frameDelay = defaultFrameDelay;
799
800     // Tries to get framerate from capture
801     // ------------------------------------------------------------------
802     // Caution : on some systems getting video parameters is forbidden !
803     // For instance it does not work wirh linuxes equipped with V4L
804     // ------------------------------------------------------------------
805 #ifndef Q_OS_LINUX
806     frameRate = capture.get(CV_CAP_PROP_FPS);
807 #else
808     frameRate = -1.0;
809 #endif
810
811 //  qDebug("framerate direct query = %f", frameRate);
812
813     /*
814      * if capture obtained frameRate is inconsistent, then we'll try to find out
815      * by ourselves
816      */
817     if (frameRate ≤ 0.0)
818     {
819         /*
820          * If live Video : grab a few images and measure elapsed time
```

```
821             */
822         if (liveVideo)
823         {
824             QElapsedTimer localTimer;
825             localTimer.start();
826 
827             for (size_t i=0; i < defaultFrameNumberTest; i++)
828             {
829                 capture >> image;
830             }
831 
832             frameDelay = (int)(localTimer.elapsed() / defaultFrameNumberTest);
833             frameRate = 1.0/((double)frameDelay/1000.0);
834             qDebug("Measured capture frame rate is  %4.2f images/s", frameRate);
835         }
836         /*
837          * FIXME else ???
838          * video files read through capture should provide framerate with
839          * capture.get(CV_CAP_PROP_FPS) but what happens if they don't ???
840          */
841     }
842     else
843     {
844         qDebug("%s Capture frame rate = %4.2f", message.toStdString().c_str(),
845                                                 frameRate);
846         frameDelay = 1000/frameRate;
847     }
848 
849     statusMessage.sprintf("%s frame rate = %4.2f images/s",
850                           message.toStdString().c_str(), frameRate);
851     emit messageChanged(statusMessage, messageDelay);
852 
853     return frameDelay;
854 }
855 
856 /*
857  * Tries to set capture size directly on capture by using properties.
858  *  - CV_CAP_PROP_FRAME_WIDTH to set frame width
859  *  - CV_CAP_PROP_FRAME_HEIGHT to set frame height
860  * @param width the width property to set on capture
861  * @param height the height property to set on capture
862  * @return true if capture is opened and if width and height have been
863  * set successfully through @code capture.set(...) @endcode. Returns
864  * false otherwise.
865  * @post if at least width or height have been set successfully, capture
866  * image is released then updated again so it will have the right
867  * dimensions.
868  */
869 bool QcvVideoCapture::setDirectSize(const unsigned int width,
870                                     const unsigned int height)
871 {
872 #ifdef Q_OS_LINUX
873     Q_UNUSED(width);
874     Q_UNUSED(height);
875 #endif
876     bool done = false;
877 
878     /*
879      * We absolutely need this lock in order to safely set width and
880      * height directly into the capture, so if mutex is already locked
881      * we should wait for it to be unlocked before continuing. Moreover,
882      * if mutex is NON-recursive and already locked, the call to lock() could
883      * lead to a DEADlock, so mutex HAS to be recursive !
884      */
885 
886 #ifndef Q_OS_LINUX
887     if (capture.isOpened())
888     {
889         bool setWidth = capture.set(CV_CAP_PROP_FRAME_WIDTH, (double)width);
890         bool setHeight = capture.set(CV_CAP_PROP_FRAME_HEIGHT, (double)height);
891         if (setWidth ∨ setHeight)
892         {
893             // release old capture image
894             image.release();
895 
896             // force image update to get the right size
897             capture >> image;
898 
899             done = true;
900         }
901     }
902 #endif
```



```
903 
904     return done;
905 }
906 
907 /*
908  * update slot trigerred by timer : Grabs a new image and sends updated()
909  * signal iff new image has been grabbed, otherwise there is no more
910  * images to grab so kills timer
911  */
912 void QcvVideoCapture::update()
913 {
914     bool locked = true;
915     bool image_updated = false;
916 
917     if (updateThread ≠ NULL)
918     {
919         if (skip)
920         {
921             locked = mutex.tryLock();
922             // qDebug() << "QcvVideoCapture::update trylock"
923             //          << (locked ? "granted" : "failed");
924             if (locked)
925             {
926                 lockLevel++;
927             }
928         }
929         else
930         {
931             if (lockLevel ≡ 0)
932             {
933                 mutex.lock();
934                 // qDebug() << "QcvVideoCapture::update lock";
935             }
936             lockLevel++;
937         }
938     }
939 
940     if (capture.isOpened() ∧ locked)
941     {
942         capture >> image;
943 
944         if (¬image.data) // captured image has no data
945         {
946             statusMessage.clear();
947 
948             if (liveVideo)
949             {
950                 if (timer→isActive())
951                 {
952                     timer→stop();
953                     qDebug("timer stopped");
954                 }
955 
956                 capture.release();
957 
958                 statusMessage.sprintf("No more frames to capture ...");
959                 emit messageChanged(statusMessage, 0);
960                 qDebug("%s", statusMessage.toStdString().c_str());
961             }
962             else // not live video ==> video file
963             {
964                 // We'll try to rewinds the file back to frame 0
965                 bool restart = capture.set(CV_CAP_PROP_POS_FRAMES, 0.0);
966 
967                 if (restart)
968                 {
969                     statusMessage.sprintf("Capture restarted");
970                     emit messageChanged(statusMessage,
971                                         QcvVideoCapture::messageDelay);
972                     qDebug("%s", statusMessage.toStdString().c_str());
973 
974                     // Refresh image chain resized -> flipped -> gray
975                     setSize(size);
976                 }
977                 else
978                 {
979                     capture.release();
980 
981                     statusMessage.sprintf("Failed to restart capture ...");
982                     emit messageChanged(statusMessage, 0);
983                     emit(finished());
984                     qDebug("%s", statusMessage.toStdString().c_str());
```

```
985                 }
986             }
987         }
988         else // capture image has data
989         {
990             /*
991              * CAUTION
992              * image->imageResized->imageFlipped->imageDisplay
993              * constitute an image chain, so when size is changed with
994              * setSize it should call setFlipVideo which should call
995              * setGray
996              */
997 
998             // resize image
999             if (resize ∧ ¬directResize)
1000            {
1001                cv::resize(image, imageResized, imageResized.size(), 0, 0,
1002                    INTER_AREA);
1003            }
1004            /*
1005             * else imageResized.data is already == image.data
1006             */
1007 
1008            // flip image horizontally if required
1009            if (flipVideo)
1010            {
1011                flip(imageResized, imageFlipped, 1);
1012            }
1013            /*
1014             * else imageFlipped.data is already == imageResized.data
1015             */
1016 
1017            // convert image to gray if required
1018            if (gray)
1019            {
1020                cvtColor(imageFlipped, imageDisplay, CV_BGR2GRAY);
1021            }
1022            /*
1023             * else imageDisplay.data is already == imageFlipped.data
1024             */
1025            image_updated = true;
1026        }
1027 
1028        if (updateThread ≠ NULL)
1029        {
1030            lockLevel--;
1031            if (lockLevel ≡ 0)
1032            {
1033                // qDebug() << "QcvVideoCapture::update unlock";
1034                mutex.unlock();
1035            }
1036        }
1037 
1038        if (image_updated)
1039        {
1040            emit updated();
1041        }
1042    }
1043    else
1044    {
1045        // mutex hasn't been locked, so we skipped one capture
1046        // qDebug() << "Capture skipped an image";
1047    }
1048 }
```



```
1  /*
2   * CaptureFactory.h
3   *
4   *  Created on: 11 fÃ©vr. 2012
5   *      Author: davidroussel
6   */
7 
8  #ifndef CAPTUREFACTORY_H_
9  #define CAPTUREFACTORY_H_
10 
11 #include <QString>
12 #include <QStringList>
13 #include <QThread>
14 #include "QcvVideoCapture.h"
15 
16 /**
17  * Capture Factory creates QcvVideoCapture from arguments list
18  */
19 class CaptureFactory
20 {
21     protected:
22         /**
23          * The capture instance to create
24          */
25         QcvVideoCapture *capture;
26 
27         /**
28          * Device number to open. Generally :
29          *  - 0 is internal or fisrt camera
30          *  - 1 is external or second camera
31          */
32         int deviceNumber;
33 
34         /**
35          * Indicates capture opens camera or file.
36          * Default value is true
37          */
38         bool liveVideo;
39 
40         /**
41          * Video should be flipped horizontally for mirror effect
42          * Default value is false
43          */
44         bool flippedVideo;
45 
46         /**
47          * Video should be converted to gray during capture.
48          * Default value is false
49          */
50         bool grayVideo;
51 
52         /**
53          * Capture can skip capturing new image when previous image has not
54          * been processed yet, or can wait for the previous image to be
55          * processed before grabbing a new image.
56          */
57         bool skipImages;
58 
59         /**
60          * Video preferred width (evt resize video)
61          * Default value is 0 which means no preferred width
62          */
63         int preferredWidth;
64 
65         /**
66          * Video preferred height (evt resize video)
67          * Default value is 0 which means no preferred height
68          */
69         int preferredHeight;
70 
71         /**
72          * Path to video file
73          */
74         QString videoPath;
75 
76     public:
77         /**
78          * Capture Factory constructor.
79          * Arguments can be
80          *  - [-d | --device] <device number> : camera number
81          *  - [-f | --file] <filename> : video file name
82          *  - [-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(const QStringList & argList);

        /**
         * Capture factory destructor
         */
        virtual ~CaptureFactory();

        /**
         * Set the capture to live (webcam) or file source
         * @param live the video source
         */
        void setLiveVideo(const bool live);

        /**
         * Set device number to use when instanciating the capture with
         * live video.
         * @param deviceNumber the device number to use
         */
        void setDeviceNumber(const int deviceNumber);

        /**
         * Set path to video file when #liveVideo is false
         * @param path the path to the video file source
         */
        void setFile(const QString & path);

        /**
         * Set video horizontal flip state (useful for selfies)
         * @param flipped the horizontal flip state
         */
        void setFlipped(const bool flipped);

        /**
         * Set gray conversion
         * @param gray the gray conversion state
         */
        void setGray(const bool 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 setSkippable(const bool skip);

        /**
         * Set video size (independently of video source actual size)
         * @param width the desired image width
         * @param height the desired image height
         */
        void setSize(const size_t width, const size_t height);

        /**
         * Set video size (independently of video source actual size)
         * @param size the desired video size
         */
        void setSize(const QSize & size);

        /**
         * 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 * getCaptureInstance(QThread * updatethread = NULL);
};

#endif /* CAPTUREFACTORY_H_ */
```



```
/*
 * 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;
```

```
247         capture = new QcvVideoCapture(deviceNumber,
248                                       flippedVideo,
249                                       grayVideo,
250                                       skipImages,
251                                       preferredWidth,
252                                       preferredHeight,
253                                       updateThread);
254     }
255     else
256     {
257         // Video file or stream
258         qDebug() << videoPath << "…";
259         capture = new QcvVideoCapture(videoPath,
260                                       flippedVideo,
261                                       grayVideo,
262                                       skipImages,
263                                       preferredWidth,
264                                       preferredHeight,
265                                       updateThread);
266     }
267
268     return capture;
269 }
270
```



```
1   #ifndef MAINWINDOW_H
2   #define MAINWINDOW_H
3
4   #include <QMainWindow>
5   #include "QcvVideoCapture.h"
6   #include "QcvSimpleDFT.h"
7
8   /**
9    * Namespace for generated UI
10   */
11  namespace Ui {
12      class MainWindow;
13  }
14
15  /**
16   * Rendering mode for main image
17   */
18  typedef enum
19  {
20      RENDER_IMAGE = 0,//!< QImage rendering mode
21      RENDER_PIXMAP,   //!< QPixmap in a QLabel rendering mode
22      RENDER_GL        //!< OpenGL in a QGLWidget rendering mode
23  } RenderMode;
24
25  /**
26   * Channels index 2 Widget index conversion
27   */
28  static const CvProcessor::Channels RGB[3] = {CvProcessor::RED,
29                                               CvProcessor::GREEN,
30                                               CvProcessor::BLUE};
31
32  /**
33   * OpenCV/Qt Histograms and LUT main window
34   */
35  class MainWindow : public QMainWindow
36  {
37      Q_OBJECT
38
39      public:
40          /**
41           * MainWindow constructor.
42           * @param capture the capture QObject to capture frames from devices
43           * or video files
44           * @param processor Fourier transform and filter processor
45           * @param parent parent widget
46           */
47          explicit MainWindow(QcvVideoCapture * capture,
48                              QcvSimpleDFT * processor,
49                              QWidget *parent = NULL);
50
51          /**
52           * MainWindow destructor
53           */
54          virtual ~MainWindow();
55
56      signals:
57          /**
58           * Signal to send update message when something changes
59           * @param message the message
60           * @param timeout number of ms the message should be displayed
61           */
62          void sendMessage(const QString & message, int timeout = 0);
63
64          /**
65           * Signal to send when video size change is requested
66           * @param size the new video size
67           */
68          void sizeChanged(const QSize & size);
69
70          /**
71           * Signal to send for opening a device (camera) with the capture
72           * @param deviceId device number to open
73           * @param width desired width or 0 to keep capture width
74           * @param height desired height or 0 to keep capture height
75           * @return true if device has been opened and checked and timer launched
76           */
77          void openDevice(const int deviceId,
78                          const unsigned int width,
79                          const unsigned int height);
80
81          /**
82           * Signal to send for opening a video file in the capture
```

```
83          * @param fileName video file to open
84          * @param width desired width or 0 to keep capture width
85          * @param height desired height or 0 to keep capture height
86          * @return true if video has been opened and timer launched
87          */
88         void openFile(const QString & fileName,
89                       const unsigned int width,
90                       const unsigned int height);
91
92         /**
93          * Signal to send when requesting video flip
94          * @param flip video flip
95          */
96         void flipVideo(const bool flip);
97
98         /**
99          * Signal to send when requesting gray image
100         * @param gray gray image status
101         */
102        void grayImage(const bool gray);
103
104    private:
105        /**
106         * The UI built in QtDesigner or QtCreator
107         */
108        Ui::MainWindow *ui;
109
110        /**
111         * The Capture object grabs frame using OpenCV HiGui
112         */
113        QcvVideoCapture * capture;
114
115        /**
116         * The Fourier Transform and filter processor
117         */
118        QcvSimpleDFT * processor;
119
120        /**
121         * Image preferred width
122         */
123        int preferredWidth;
124
125        /**
126         * Image preferred height
127         */
128        int preferredHeight;
129
130        /**
131         * Message to send to statusBar
132         */
133        QString message;
134
135        /**
136         * Changes widgetImage nature according to desired rendering mode.
137         * Possible values for mode are:
138         *  - IMAGE: widgetImage is assigned to a QcvMatWidgetImage instance
139         *  - PIXMAP: widgetImage is assigned to a QcvMatWidgetLabel instance
140         *  - GL: widgetImage is assigned to a QcvMatWidgetGL instance
141         * @param mode
142         */
143        void setRenderingMode(const RenderMode mode);
144
145        /**
146         * Set filters spinBoxes and sliders link state
147         * @param linked the link status
148         * @post When link is on all sliders/spinboxes of low pass and high pass
149         * filters are linked together, moving/changing one changes the others.
150         * When link os off sliders/spinboxes are not linked together
151         */
152        void setLinkedFilterSizes(bool linked);
153
154    private slots:
155
156        /**
157         * Re setup processor from UI settings when source image changes
158         */
159        void setupProcessorFromUI();
160
161        /**
162         * Menu action when Sources->camera 0 is selected
163         * Sets capture to open device 0. If device is not available
164         * menu item is set to inactive.
```



```
165         */
166        void on_actionCamera_0_triggered();
167
168        /**
169         * Menu action when Sources->camera 1 is selected
170         * Sets capture to open device 0. If device is not available
171         * menu item is set to inactive
172         */
173        void on_actionCamera_1_triggered();
174
175        /**
176         * Menu action when Sources->file is selected.
177         * Opens file dialog and tries to open selected file (is not empty),
178         * then sets capture to open the selected file
179         */
180        void on_actionFile_triggered();
181
182        /**
183         * Menu action to quit application.
184         */
185        void on_actionQuit_triggered();
186
187        /**
188         * Menu action when flip image is selected.
189         * Sets capture to change flip status which leads to reverse
190         * image horizontally
191         */
192        void on_actionFlip_triggered();
193
194        /**
195         * Menu action when gray image is selected.
196         * Sets capture to change gray status which leads convert captured image
197         * to gray or not
198         */
199        void on_actionGray_triggered();
200
201        /**
202         * Menu action when original image size is selected.
203         * Sets capture not to resize image
204         */
205        void on_actionOriginalSize_triggered();
206
207        /**
208         * Menu action when constrained image size is selected.
209         * Sets capture resize to preferred width and height
210         */
211        void on_actionConstrainedSize_triggered();
212
213        /**
214         * Menu action to replace current image rendering widget by a
215         * QcvMatWidgetImage instance.
216         */
217        void on_actionRenderImage_triggered();
218
219        /**
220         * Menu action to replace current image rendering widget by a
221         * QcvMatWidgetLabel with pixmap instance.
222         */
223        void on_actionRenderPixmap_triggered();
224
225        /**
226         * Menu action to replace current image rendering widget by a
227         * QcvMatWidgetGL instance.
228         */
229        void on_actionRenderOpenGL_triggered();
230
231        /**
232         * Original size radioButton action.
233         * Sets capture resize to off
234         */
235        void on_radioButtonOrigSize_clicked();
236
237        /**
238         * Custom size radioButton action.
239         * Sets capture resize to preferred width and height
240         */
241        void on_radioButtonCustomSize_clicked();
242
243        /**
244         * Width spinbox value change.
245         * Changes the preferred width and if custom size is selected apply
246         * this custom width
```

```
247          * @param value the desired width
248          */
249         void on_spinBoxWidth_valueChanged(int value);
250 
251         /**
252          * Height spinbox value change.
253          * Changes the preferred height and if custom size is selected apply
254          * this custom height
255          * @param value the desired height
256          */
257         void on_spinBoxHeight_valueChanged(int value);
258 
259         /**
260          * Flip capture image horizontally.
261          * changes capture flip status
262          */
263         void on_checkBoxFlip_clicked();
264 
265         /**
266          * convert capture image to gray level.
267          * changes cpature gray conversion status
268          */
269         void on_checkBoxGray_clicked();
270 
271         /**
272          * Changes logscale factor for spectrum
273          * @param value the new logscale factor
274          */
275         void on_spinBoxMag_valueChanged(int value);
276 
277 };
278 
279 #endif // MAINWINDOW_H
```



```
1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3  
4  #include <QObject>
5  #include <QFileDialog>
6  #include <QDebug>
7  #include <assert.h>
8  
9  #include "QcvMatWidgetImage.h"
10 #include "QcvMatWidgetLabel.h"
11 #include "QcvMatWidgetGL.h"
12 
13 /*
14  * MainWindow constructor.
15  * @param capture the capture QObject to capture frames from devices
16  * or video files
17  * @param processor Fourier transform and filter processor
18  * @param parent parent widget
19  */
20 MainWindow::MainWindow(QcvVideoCapture * capture,
21                        QcvSimpleDFT * processor,
22                        QWidget *parent) :
23     QMainWindow(parent),
24     ui(new Ui::MainWindow),
25     capture(capture),
26     processor(processor),
27     preferredWidth(341),
28     preferredHeight(256)
29 {
30     ui→setupUi(this);
31     ui→scrollAreaSource→setBackgroundRole(QPalette::Mid);
32     ui→scrollAreaSpectrum→setBackgroundRole(QPalette::Mid);
33 
34     // ----------------------------------------------------------------------
35     // Assertions
36     // ----------------------------------------------------------------------
37     assert(capture ≠ NULL);
38 
39     assert(processor ≠ NULL);
40 
41     // ----------------------------------------------------------------------
42     // Special widgets initialisation
43     // ----------------------------------------------------------------------
44     // Replace QcvMatWidget instances with QcvMatWidgetImage instances
45     // sets image widget sources for the first time
46     // connects processor->update to image Widgets->updated
47     // connects processor->image changed to image widgets->setSourceImage
48     setRenderingMode(RENDER_IMAGE);
49 
50     ui→labelFFTSizeValue→setText(QString::number(processor→getOptimalDftSize()));
51 
52     // ----------------------------------------------------------------------
53     // rest of Signal/Slot connections
54     // ----------------------------------------------------------------------
55     // processor->sendText --> labelFFTSizeValue->setText when source image
56     // changes, fft size might also change
57 
58     connect(processor, SIGNAL(sendText(QString)),
59             ui→labelFFTSizeValue, SLOT(setText(QString)));
60 
61     // Capture, processor and this messages to status bar
62     connect(capture, SIGNAL(messageChanged(QString,int)),
63             ui→statusBar, SLOT(showMessage(QString,int)));
64 
65     connect(processor, SIGNAL(sendMessage(QString,int)),
66             ui→statusBar, SLOT(showMessage(QString,int)));
67 
68     connect(this, SIGNAL(sendMessage(QString,int)),
69             ui→statusBar, SLOT(showMessage(QString,int)));
70 
71     // When Processor source image changes, some attributes are reinitialised
72     // So we have to set them up again according to current UI values
73     connect(processor, SIGNAL(imageChanged()),
74             this, SLOT(setupProcessorFromUI()));
75 
76     // Connects UI requests to capture
77     connect(this, SIGNAL(sizeChanged(const QSize &)),
78             capture, SLOT(setSize(const QSize &)));
79     connect(this, SIGNAL(openDevice(int,uint,uint)),
80             capture, SLOT(open(int,uint,uint)));
81     connect(this, SIGNAL(openFile(QString,uint,uint)),
82             capture, SLOT(open(QString,uint,uint)));
```

```
83      connect(this, SIGNAL(flipVideo(bool)), capture, SLOT(setFlipVideo(bool)));
84      connect(this, SIGNAL(grayImage(bool)), capture, SLOT(setGray(bool)));
85
86      // ----------------------------------------------------------------------
87
88      // ----------------------------------------------------------------------
89      // UI setup according to capture options
90      // ----------------------------------------------------------------------
91      // Sets size radioButton states
92      if (capture->isResized())
93      {
94          /*
95           * Initial Size radio buttons configuration
96           */
97          ui->radioButtonOrigSize->setChecked(false);
98          ui->radioButtonCustomSize->setChecked(true);
99          /*
100          * Initial Size menu items configuration
101          */
102         ui->actionOriginalSize->setChecked(false);
103         ui->actionConstrainedSize->setChecked(true);
104
105         QSize size = capture->getSize();
106         qDebug("Capture->size is %dx%d", size.width(), size.height());
107         preferredWidth = size.width();
108         preferredHeight = size.height();
109     }
110     else
111     {
112         /*
113          * Initial Size radio buttons configuration
114          */
115         ui->radioButtonCustomSize->setChecked(false);
116         ui->radioButtonOrigSize->setChecked(true);
117
118         /*
119          * Initial Size menu items configuration
120          */
121         ui->actionConstrainedSize->setChecked(false);
122         ui->actionOriginalSize->setChecked(true);
123     }
124
125     // Sets spinboxes preferred size
126     ui->spinBoxWidth->setValue(preferredWidth);
127     ui->spinBoxHeight->setValue(preferredHeight);
128
129     // Sets flipCheckbox and menu item states
130     bool flipped = capture->isFlipVideo();
131     ui->actionFlip->setChecked(flipped);
132     ui->checkBoxFlip->setChecked(flipped);
133
134     // Sets grayCheckbox and menu item states
135     bool gray = capture->isGray();
136     ui->actionGray->setChecked(gray);
137     ui->checkBoxGray->setChecked(gray);
138
139     // ----------------------------------------------------------------------
140     // UI setup according to DFTProcessor options
141     // ----------------------------------------------------------------------
142     // Setting up log scale spinbox value and boundaries
143     ui->spinBoxMag->setValue((int)processor->getLogScaleFactor());
144     ui->spinBoxMag->setMinimum((int)processor->minLogScaleFactor);
145     ui->spinBoxMag->setMaximum((int)processor->maxLogScaleFactor);
146 }
147
148 /*
149  * MainWindow destructor
150  */
151 MainWindow::~MainWindow()
152 {
153     delete ui;
154 }
155
156 /*
157  * Menu action when Sources->camera 0 is selected
158  * Sets capture to open device 0. If device is not available
159  * menu item is set to inactive.
160  */
161 void MainWindow::on_actionCamera_0_triggered()
162 {
163     int width = 0;
164     int height = 0;
```



```
165
166     if (ui->radioButtonCustomSize->isChecked())
167     {
168         width = preferredWidth;
169         height = preferredHeight;
170     }
171
172     qDebug("Opening device 0 ...");
173 //  if (!capture->open(0, width, height))
174 //  {
175 //      qWarning("Unable to open device 0");
176 //      // disable menu item if camera 0 does not exist
177 //      ui->actionCamera_0->setDisabled(true);
178 //  }
179
180     emit openDevice(0, width, height);
181 }
182
183 /*
184  * Menu action when Sources->camera 1 is selected
185  * Sets capture to open device 0. If device is not available
186  * menu item is set to inactive
187  */
188 void MainWindow::on_actionCamera_1_triggered()
189 {
190     int width = 0;
191     int height = 0;
192
193     if (ui->radioButtonCustomSize->isChecked())
194     {
195         width = preferredWidth;
196         height = preferredHeight;
197     }
198
199     qDebug("Opening device 1 ...");
200 //  if (!capture->open(1, width, height))
201 //  {
202 //      qWarning("Unable to open device 1");
203 //      // disable menu item if camera 1 does not exist
204 //      ui->actionCamera_1->setDisabled(true);
205 //  }
206
207     emit openDevice(1, width, height);
208 }
209
210 /*
211  * Menu action when Sources->file is selected.
212  * Opens file dialog and tries to open selected file (is not empty),
213  * then sets capture to open the selected file
214  */
215 void MainWindow::on_actionFile_triggered()
216 {
217     int width = 0;
218     int height = 0;
219
220     if (ui->radioButtonCustomSize->isChecked())
221     {
222         width = preferredWidth;
223         height = preferredHeight;
224     }
225
226     QString fileName =
227     QFileDialog::getOpenFileName(this,
228                                  tr("Open Video"),
229                                  "./",
230                                  tr("Video Files (*.avi *.mkv *.mp4 *.m4v)"),
231                                  NULL,
232                                  QFileDialog::ReadOnly);
233
234     qDebug("Opening file %s ...", fileName.toStdString().c_str());
235
236     if (fileName.length() > 0)
237     {
238 //      if (!capture->open(fileName, width, height))
239 //      {
240 //          qWarning("Unable to open device file : %s",
241 //                   fileName.toStdString().c_str());
242 //      }
243
244 //      setupProcessorFromUI(); // Should already be called by imageChanged signal
245
246         emit openFile(fileName, width, height);
```

```
247     }
248     else
249     {
250         qWarning("empty file name");
251     }
252 }
253 
254 /*
255  * Menu action to qui application
256  */
257 void MainWindow::on_actionQuit_triggered()
258 {
259     this→close();
260 }
261 
262 /*
263  * Menu action when flip image is selected.
264  * Sets capture to change flip status which leads to reverse
265  * image horizontally
266  */
267 void MainWindow::on_actionFlip_triggered()
268 {
269     // capture->setFlipVideo(!capture->isFlipVideo());
270     emit flipVideo(¬capture→isFlipVideo());
271     /*
272      * There is no need to update ui->checkBoxFlip since it is connected
273      * to ui->actionFlip through signals/slots
274      */
275 }
276 
277 /*
278  * Menu action when gray image is selected.
279  * Sets capture to change gray status which leads convert captured image
280  * to gray or not
281  */
282 void MainWindow::on_actionGray_triggered()
283 {
284     bool isGray = ¬capture→isGray();
285 
286     // capture->setGray(isGray);
287     emit grayImage(isGray);
288 }
289 
290 /*
291  * Menu action when original image size is selected.
292  * Sets capture not to resize image
293  */
294 void MainWindow::on_actionOriginalSize_triggered()
295 {
296 
297     ui→actionConstrainedSize→setChecked(false);
298 
299     // capture->setSize(0, 0);
300     emit sizeChanged(QSize(0, 0));
301 }
302 
303 /*
304  * Menu action when constrained image size is selected.
305  * Sets capture resize to preferred width and height
306  */
307 void MainWindow::on_actionConstrainedSize_triggered()
308 {
309     ui→actionOriginalSize→setChecked(false);
310 
311     // capture->setSize(preferredWidth, preferredHeight);
312     emit sizeChanged(QSize(preferredWidth, preferredHeight));
313 }
314 
315 /*
316  * Changes widgetImage nature according to desired rendering mode.
317  * Possible values for mode are:
318  *  - IMAGE: widgetImage is assigned to a QcvMatWidgetImage instance
319  *  - PIXMAP: widgetImage is assigned to a QcvMatWidgetLabel instance
320  *  - GL: widgetImage is assigned to a QcvMatWidgetGL instance
321  * @param mode
322  */
323 void MainWindow::setRenderingMode(const RenderMode mode)
324 {
325     // Disconnect signals from slots first
326     disconnect(processor, SIGNAL(updated()),
327                ui→sourceImage, SLOT(update()));
328     disconnect(processor, SIGNAL(updated()),
```



```
329                ui→spectrumImage, SLOT(update()));
330 
331     disconnect(processor, SIGNAL(squareImageChanged(Mat*)),
332                ui→sourceImage, SLOT(setSourceImage(Mat*)));
333     disconnect(processor, SIGNAL(spectrumImageChanged(Mat*)),
334                ui→spectrumImage, SLOT(setSourceImage(Mat*)));
335 
336     // remove widgets in scroll areas
337     QWidget * wSource = ui→scrollAreaSource→takeWidget();
338     QWidget * wSpectrum = ui→scrollAreaSpectrum→takeWidget();
339 
340     if ((wSource ≡ ui→sourceImage) ∧
341         (wSpectrum ≡ ui→spectrumImage))
342     {
343         // delete removed widgets
344         delete ui→sourceImage;
345         delete ui→spectrumImage;
346 
347         // create new widget
348         Mat * sourceMat = processor→getImagePtr("square");
349         Mat * spectrumMat = processor→getImagePtr("spectrum");
350 
351         switch (mode)
352         {
353             case RENDER_PIXMAP:
354                 ui→sourceImage = new QcvMatWidgetLabel(sourceMat);
355                 ui→spectrumImage = new QcvMatWidgetLabel(spectrumMat);
356                 break;
357             case RENDER_GL:
358                 ui→sourceImage = new QcvMatWidgetGL(sourceMat);
359                 ui→spectrumImage = new QcvMatWidgetGL(spectrumMat);
360                 break;
361             case RENDER_IMAGE:
362             default:
363                 ui→sourceImage = new QcvMatWidgetImage(sourceMat);
364                 ui→spectrumImage = new QcvMatWidgetImage(spectrumMat);
365                 break;
366         }
367 
368         if ((ui→sourceImage ≠ NULL) ∧
369             (ui→spectrumImage ≠ NULL))
370         {
371             // Name the new images widgets with same name as in UI files
372              ui→sourceImage→setObjectName(QString::fromUtf8("sourceImage"));
373              ui→spectrumImage→setObjectName(QString::fromUtf8("spectrumImage"));
374 
375             // add to scroll areas
376             ui→scrollAreaSource→setWidget(ui→sourceImage);
377             ui→scrollAreaSpectrum→setWidget(ui→spectrumImage);
378 
379             // Reconnect signals to slots
380             connect(processor, SIGNAL(updated()),
381                     ui→sourceImage, SLOT(update()));
382             connect(processor, SIGNAL(updated()),
383                     ui→spectrumImage, SLOT(update()));
384 
385             connect(processor, SIGNAL(squareImageChanged(Mat*)),
386                     ui→sourceImage, SLOT(setSourceImage(Mat*)));
387             connect(processor, SIGNAL(spectrumImageChanged(Mat*)),
388                     ui→spectrumImage, SLOT(setSourceImage(Mat*)));
389 
390             // Sends message to status bar and sets menu checks
391             message.clear();
392             message.append(tr("Render more set to "));
393             switch (mode)
394             {
395                 case RENDER_IMAGE:
396                     ui→actionRenderPixmap→setChecked(false);
397                     ui→actionRenderOpenGL→setChecked(false);
398                     message.append(tr("QImage"));
399                     break;
400                 case RENDER_PIXMAP:
401                     ui→actionRenderImage→setChecked(false);
402                     ui→actionRenderOpenGL→setChecked(false);
403                     message.append(tr("QPixmap in QLabel"));
404                     break;
405                 case RENDER_GL:
406                     ui→actionRenderImage→setChecked(false);
407                     ui→actionRenderPixmap→setChecked(false);
408                     message.append(tr("QGLWidget"));
409                     break;
410                 default:
```

```
411                 break;
412             }
413             emit sendMessage(message, 5000);
414         }
415         else
416         {
417             qDebug("MainWindow::on_actionRenderXXX some new widget is null");
418         }
419     }
420     else
421     {
422         qDebug("MainWindow::on_actionRenderXXX removed widget is not in ui->");
423     }
424 }
425
426 /*
427  * Re setup processor from UI settings when source changes
428  */
429 void MainWindow::setupProcessorFromUI()
430 {
431     processor->setLogScaleFactor((double)ui->spinBoxMag->value());
432 }
433
434 /*
435  * Menu action to replace current image rendering widget by a
436  * QcvMatWidgetImage instance.
437  */
438 void MainWindow::on_actionRenderImage_triggered()
439 {
440     qDebug("Setting image rendering to: images");
441     setRenderingMode(RENDER_IMAGE);
442 }
443
444 /*
445  * Menu action to replace current image rendering widget by a
446  * QcvMatWidgetLabel with pixmap instance.
447  */
448 void MainWindow::on_actionRenderPixmap_triggered()
449 {
450     qDebug("Setting image rendering to: pixmaps");
451     setRenderingMode(RENDER_PIXMAP);
452 }
453
454 /*
455  * Menu action to replace current image rendering widget by a
456  * QcvMatWidgetGL instance.
457  */
458 void MainWindow::on_actionRenderOpenGL_triggered()
459 {
460     qDebug("Setting image rendering to: opengl");
461     setRenderingMode(RENDER_GL);
462 }
463
464 /*
465  * Original size radioButton action.
466  * Sets capture resize to off
467  */
468 void MainWindow::on_radioButtonOrigSize_clicked()
469 {
470     ui->actionConstrainedSize->setChecked(false);
471     // capture->setSize(0, 0);
472     emit sizeChanged(QSize(0, 0));
473 }
474
475 /*
476  * Custom size radioButton action.
477  * Sets capture resize to preferred width and height
478  */
479 void MainWindow::on_radioButtonCustomSize_clicked()
480 {
481     ui->actionOriginalSize->setChecked(false);
482     // capture->setSize(preferredWidth, preferredHeight);
483     emit sizeChanged(QSize(preferredWidth, preferredHeight));
484 }
485
486 /*
487  * Width spinbox value change.
488  * Changes the preferred width and if custom size is selected apply
489  * this custom width
490  * @param value the desired width
491  */
492 void MainWindow::on_spinBoxWidth_valueChanged(int value)
```



```
493 {
494     preferredWidth = value;
495     if (ui->radioButtonCustomSize->isChecked())
496     {
497         // capture->setSize(preferredWidth, preferredHeight);
498         emit sizeChanged(QSize(preferredWidth, preferredHeight));
499     }
500 }
501
502 /*
503  * Height spinbox value change.
504  * Changes the preferred height and if custom size is selected apply
505  * this custom height
506  * @param value the desired height
507  */
508 void MainWindow::on_spinBoxHeight_valueChanged(int value)
509 {
510     preferredHeight = value;
511     if (ui->radioButtonCustomSize->isChecked())
512     {
513         // capture->setSize(preferredWidth, preferredHeight);
514         emit sizeChanged(QSize(preferredWidth, preferredHeight));
515     }
516 }
517
518 /*
519  * Flip capture image horizontally.
520  * changes capture flip status
521  */
522 void MainWindow::on_checkBoxFlip_clicked()
523 {
524     /*
525      * There is no need to update ui->actionFlip since it is connected
526      * to ui->checkBoxFlip through signals/slots
527      */
528     // capture->setFlipVideo(ui->checkBoxFlip->isChecked());
529     emit flipVideo(ui->checkBoxFlip->isChecked());
530 }
531
532 /*
533  * convert capture image to gray level.
534  * changes cpature gray conversion status
535  */
536 void MainWindow::on_checkBoxGray_clicked()
537 {
538     bool isGray = ui->checkBoxGray->isChecked();
539     // capture->setGray(isGray);
540     emit grayImage(isGray);
541 }
542
543 /*
544  * Changes logscale factor for spectrum
545  * @param value the new logscale factor
546  */
547 void MainWindow::on_spinBoxMag_valueChanged(int value)
548 {
549     processor->setLogScaleFactor((double)value);
550
551     double realScale = processor->getLogScaleFactor();
552
553     ui->spinBoxMag->setValue((int)realScale);
554 }
```

```
#include <QApplication>
#include <QThread>
#include <libgen.h>     // for basename
#include <iostream>     // for cout

#include "QcvVideoCapture.h"
#include "CaptureFactory.h"
#include "QcvSimpleDFT.h"
#include "mainwindow.h"

/**
 * Usage function shown just before launching QApp
 * @param name the name of the program (argv[0])
 */
void usage(char * name);

/**
 * Test program OpenCV2 + QT4
 * @param argc argument count
 * @param argv argument values
 * @return QTApp return value
 * @par usage : <Progname> [--device | -d] <#> | [--file | -f ] <filename>
 * [--mirror | -m] [--size | -s] <width>x<height>
 *  - device : [--device | -d] <device #> (0, 1, ...) Opens capture device #
 *  - filename : [--file | -f ] <filename> Opens a video file or URL (including rtsp)
 *  - mirror : mirrors image horizontally before display
 *  - size : [--size | -s] <width>x<height> resize capture to fit desired <width>
 *  and <height>
 */
int main(int argc, char *argv[])
{
    // -----------------------------------------------------------------------
    // Instanciate QApplication to receive special QT args
    // -----------------------------------------------------------------------
    QApplication app(argc, argv);

    // Gets arguments after QT specials removed
    QStringList argList = QCoreApplication::arguments();

    int threadNumber = 3;
    // parse arguments for --threads tag
    for (QListIterator<QString> it(argList); it.hasNext(); )
    {
        QString currentArg(it.next());

        if (currentArg ≡ "-t" ∨ currentArg ≡"--threads")
        {
            // Next argument should be thread number integer
            if (it.hasNext())
            {
                QString threadString(it.next());
                bool convertOk;
                threadNumber = threadString.toInt(&convertOk,10);
                if (¬convertOk ∨ threadNumber < 1 ∨ threadNumber > 3)
                {
                    qWarning("Warning: Invalid thread number %d",threadNumber);
                    threadNumber = 3;
                }
            }
            else
            {
                qWarning("Warning: thread tag found with no following thread number");
            }

        }
    }
    // -----------------------------------------------------------------------
    // Create Capture factory using program arguments and
    // open Video Capture
    // -----------------------------------------------------------------------
    CaptureFactory factory(argList);
    factory.setSkippable(true);

    // Helper thread for capture
    QThread * capThread = NULL;
    if (threadNumber > 1)
    {
        capThread = new QThread();
    }

    // Capture
    QcvVideoCapture * capture = factory.getCaptureInstance(capThread);
```



```

    // -----------------------------------------------------------------------
    // Create Fourier Processor
    // -----------------------------------------------------------------------
    // Helper thread for processor
    QThread * procThread = NULL;
    if (threadNumber > 2)
    {
        procThread = new QThread();
    }
    else
    {
        if (threadNumber > 1)
        {
            procThread = capThread;
        }
    }

    // Processsor
    QcvSimpleDFT * processor = NULL;
    if (procThread ≡ NULL)
    {
        processor = new QcvSimpleDFT(capture→getImage());
    }
    else
    {
        if (procThread ≠ capThread)
        {
            processor = new QcvSimpleDFT(capture→getImage(),
                                         capture→getMutex(),
                                         procThread);
        }
        else // procThread == capThread
        {
            processor = new QcvSimpleDFT(capture→getImage(),
                                         NULL,
                                         procThread);
        }
    }

    // -----------------------------------------------------------------------
    // Connects capture to processor
    // -----------------------------------------------------------------------
    // Connects capture update to QHistandLUT update
    QObject::connect(capture, SIGNAL(updated()),
                     processor, SLOT(update()));

    // connect capture changed image to QHistandLUT set input
    QObject::connect(capture, SIGNAL(imageChanged(Mat*)),
                     processor, SLOT(setSourceImage(Mat*)));
    // -----------------------------------------------------------------------
    // Now that Capture & Histogram are on then
    // add our MainWindow as toplevel
    // and launches app
    // -----------------------------------------------------------------------
    MainWindow w(capture, processor);
    w.show();

    usage(argv[0]);

    int retVal = app.exec();

    // -----------------------------------------------------------------------
    // Cleanup & return
    // -----------------------------------------------------------------------
    delete processor;
    delete capture;
    bool sameThread = capThread ≡ procThread;

    if (capThread ≠ NULL)
    {
        delete capThread;
    }

    if (procThread ≠ NULL ∧ ¬sameThread)
    {
        delete procThread;
    }

    return retVal;
}

```

```
165 /*
166  * Usage function shown just before launching QApp
167  * @param name the name of the program (argv[0])
168  */
169 void usage(char * name)
170 {
171     cout << "usage :" << basename(name) << " "
172          << "[-d | --device] <device number> "
173          << "[-v | --video] <video file> "
174          << "[-s | --size] <width>x<height> "
175          << "[-m | --mirror]"
176          << "[-g | --gray]"
177          << endl;
178 }
```