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

#ifndef CVPROCESSOR_H_
#define CVPROCESSOR_H_

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

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

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

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

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


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

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

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

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

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

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

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

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



```cpp

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

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

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

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

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

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

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

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

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

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

        /**
         * Get image by name
```

```
181             * @param name the name of the image we're looking for
182             * @return the image registered by this name in the additionnal images
183             * map
184             * @throw CvProcessorException#INVALID_NAME is used name is not already
185             * registerd in the images
186             */
187            const Mat & getImage(const char * name) const
188                throw (CvProcessorException);
189
190            /**
191             * Get image by name
192             * @param name the name of the image we're looking for
193             * @return the image registered by this name in the additionnal images
194             * map
195             * @throw CvProcessorException#INVALID_NAME is used name is not already
196             * registerd in the images
197             */
198            const Mat & getImage(const string & name) const
199                throw (CvProcessorException);
200
201            /**
202             * Get image pointer by name
203             * @param name the name of the image we're looking for
204             * @return the image pointer registered by this name in the additionnal
205             * images map
206             * @throw CvProcessorException#INVALID_NAME is used name is not already
207             * registerd in the images
208             */
209            Mat * getImagePtr(const char * name)
210                throw (CvProcessorException);
211
212            /**
213             * Get image pointer by name
214             * @param name the name of the image we're looking for
215             * @return the image registered by this name in the additionnal images
216             * map
217             * @throw CvProcessorException#INVALID_NAME is used name is not already
218             * registerd in the images
219             */
220            Mat * getImagePtr(const string & name)
221                throw (CvProcessorException);
222            // ------------------------------------------------------------------
223            // Options settings and gettings
224            // ------------------------------------------------------------------
225            /**
226             * Number of channels in source image
227             * @return the number of channels of source image
228             */
229            int getNbChannels() const;
230
231            /**
232             * Type of the source image
233             * @return the openCV type of the source image
234             */
235            int getType() const;
236
237            /**
238             * Get the current verbose level
239             * @return the current verbose level
240             */
241            VerboseLevel getVerboseLevel() const;
242
243            /**
244             * Set new verbose level
245             * @param level the new verobse level
246             */
247            virtual void setVerboseLevel(const VerboseLevel level);
248
249            /**
250             * Return processor processing time of step index [default implementation
251             * returning only processTime, should be reimplemented in subclasses]
252             * @param index index of the step which processing time is required,
253             * 0 indicates all steps, and values above 0 indicates step #. If
254             * required index is bigger than number of steps then all steps value
255             * should be returned.
256             * @return the processing time of step index.
257             * @note should be reimplemented in subclasses in order to define
258             * time/feature behaviour
259             */
260            virtual double getProcessTime(const size_t index = 0) const;
261
262            /**
263             * Return processor mean processing time of step index [default
264             * implementation returning only processTime, should be reimplemented
265             * in subclasses]
266             * @param index index of the step which processing time is required,
267             * 0 indicates all steps, and values above 0 indicates step #. If
268             * required index is bigger than number of steps then all steps value
269             * should be returned.
270             * @return the mean processing time of step index.
```



```
271             * @note should be reimplemented in subclasses in order to define
272             * time/feature behaviour
273             * @param index
274             */
275            virtual double getMeanProcessTime(const size_t index = 0) const;
276
277            /**
278             * Return processor processing time std of step index [default
279             * implementation returning only processTime, should be reimplemented
280             * in subclasses]
281             * @param index index of the step which processing time is required,
282             * 0 indicates all steps, and values above 0 indicates step #. If
283             * required index is bigger than number of steps than all steps value
284             * should be returned.
285             * @return the mean processing time of step index.
286             * @note should be reimplemented in subclasses in order to define
287             * time/feature behaviour
288             * @param index
289             */
290            virtual double getStdProcessTime(const size_t index = 0) const;
291
292            /**
293             * Return processor minimum processing time of step index [default
294             * implementation returning only processTime, should be reimplemented
295             * in subclasses]
296             * @param index index of the step which processing time is required,
297             * 0 indicates all steps, and values above 0 indicates step #. If
298             * required index is bigger than number of steps than all steps value
299             * should be returned.
300             * @return the mean processing time of step index.
301             * @note should be reimplemented in subclasses in order to define
302             * time/feature behaviour
303             * @param index
304             */
305            virtual clock_t getMinProcessTime(const size_t index = 0) const;
306
307            /**
308             * Return processor maximum processing time of step index [default
309             * implementation returning only processTime, should be reimplemented
310             * in subclasses]
311             * @param index index of the step which processing time is required,
312             * 0 indicates all steps, and values above 0 indicates step #. If
313             * required index is bigger than number of steps than all steps value
314             * should be returned.
315             * @return the mean processing time of step index.
316             * @note should be reimplemented in subclasses in order to define
317             * time/feature behaviour
318             * @param index
319             */
320            virtual clock_t getMaxProcessTime(const size_t index = 0) const;
321
322            /**
323             * Reset mean and std process time in order to re-start computing
324             * new mean and std process time values.
325             */
326            virtual void resetMeanProcessTime();
327
328            /**
329             * Indicates if processing time is per feature processed in the current
330             * image or absolute
331             * @return
332             */
333            bool isTimePerFeature() const;
334
335            /**
336             * Sets Time per feature processing time unit
337             * @param value the time per feature value (true or false)
338             */
339            virtual void setTimePerFeature(const bool value);
340
341            /**
342             * Send to stream (for showing processor attributes values)
343             * @param out the stream to send to
344             * @return a reference to the output stream
345             */
346            virtual ostream & toStream(ostream & out) const;
347
348            /**
349             * Send to any stream template
350             * @tparam Stream the stream type
351             * @param out the output stream
352             * @return a reference to the output stream
353             * @note this template method needs to be implemented in the header so
354             * it could be available in any source (.cpp) file that need a specific
355             * instantiation of this template method, for instance:
356             * @code
357             * template ostream & CvProcessor::toStream_Impl<ostream>(ostream &) const;
358             * @endcode
359             */
360            template <typename Stream>
```

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



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

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


#include "CvProcessor.h"

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

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

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

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

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

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

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



```
    }

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

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

    // clean up current attributes
    cleanup();

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

    // setup attributes again
    setup(sourceImage);
}

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

    return addImage(sname, image);
}

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

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

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

        retValue = false;
    }
    else
```

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



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

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



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

```
#ifndef CVPROCESSOREXCEPTION_H_
#define CVPROCESSOREXCEPTION_H_

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

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

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

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

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



```

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

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

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

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

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

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

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

#endif /*CVPROCESSOREXCEPTION_H_*/
```

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

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

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

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


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

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

    ostringstream output;

    output << initialWhat << ":";

    output << "CvProcessorException : ";

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

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



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

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


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

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

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

#ifndef QCVPROCESSOR_H_
#define QCVPROCESSOR_H_

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

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

    protected:

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

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

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

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

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

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

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

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

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

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

    public:

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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



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

```cpp
/*
 * CvDFT.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, then apply filters on the resulting
 * FFT and then process spectrum with inverse FFT to obtain filtered image.
 * input image can have single or multiple channels
 */
class CvDFT : virtual public CvProcessor
{
    public:

        /**
         * Frenquency filter type
         */
        typedef enum
        {
            /**
             * Frequency Box filter.
             * Inverse box filter is sinus cardinal
             */
            BOX_FILTER = 0,
            /**
             * Frequency gaussian filter.
             * Inverse gaussian filter is gaussian
             */
            GAUSS_FILTER,
            /**
             * frequency sinus cardinal filter.
             * Inverse sinus cardinal is box
             */
            SINC_FILTER,
            /**
             * Number of available filters
             */
            NB_FILTERS
        } FilterType;

        /**
         * 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;

        /**
         * Frequency filtering status
         */
        bool filtering;
```



```cpp

        /**
         * Type of frequency filter to apply
         */
        FilterType filterType;

        /**
         * 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.
         * Log scale factor
         */
        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;

        /**
         * Mask for lowpass filtering of each channel
         * (white door/gaussian on black):
         * CV_64FC1 x <nbChannels>
         */
        vector<Mat> channelsLowPassMask;

        /**
         * Mask for hipass filtering (black door/gaussian on white):
         * CV_64FC1 x <nbChannels>
         */
        vector<Mat> channelsHighPassMask;

        /**
         * Mask for reverse hipass when Hipass needs to be reversed:
```

```
181             * CV_64F1 x <nbChannels>
182             */
183            vector<Mat> channelsHighPassMaskReverse;
184
185            /**
186             * Complete channel spectrum mask = channelsLowPassMask x channelsHighPassMask:
187             * CV_64FC1 x <nbChannels>
188             */
189            vector<Mat> channelsSpectrumMask;
190
191            /**
192             * Channel spectrum mask converted for display :
193             * CV_8UC1 x <nbChannels>
194             */
195            vector<Mat> channelsSpectrumMaskDisplay;
196
197            /**
198             * Spectrum mask image : CV_8UC<nbChannels>
199             */
200            Mat spectrumMaskImage;
201
202            /**
203             * Complex spectrum masked: CV_64FC2 x <nbChannels>
204             */
205            vector<Mat> channelsComplexSpectrumsMasked;
206
207            /**
208             * Complex channels resulting from inverse Fourier transform
209             * of the masked complex spectrums: CV_64FC2 x <nbChannels>
210             */
211            vector<Mat> channelsComplexInverseImages;
212
213            /**
214             * Complex inverse image channels: CV_64FC1 x 2 x <nbChannels>
215             */
216            vector<vector<Mat> > channelsComplexInverseComponents;
217
218            /**
219             * Real part of FFT inverse channels (with fft shift):
220             * CV_64FC1 x <nbChannels>
221             */
222            vector<Mat> channelsRealInverse;
223
224            /**
225             * Real part of FFT inverse channels converted for display:
226             * CV_8UC1 x <nbChannels>
227             */
228            vector<Mat> channelsRealInverseDisplay;
229
230            /**
231             * Image composed with display real parts: CV_8UC<nbChannels>
232             */
233            Mat inverseImage;
234
235            /**
236             * Maximum size of the filters (dftSize / sqrt(2.0))
237             */
238            int filterMaxSize;
239
240            /**
241             * Minimum size of the filters
242             */
243            int filterMinSize;
244
245            /**
246             * Low pass filter size for each channel (initialised to filterMaxSize
247             * for no filtering)
248             */
249            vector<int> lowPassFilterSize;
250
251            /**
252             * High pass filter size for each channel (initialised to 0)
253             */
254            vector<int> highPassFilterSize;
255
256        public:
257            /**
258             * DFT processor constructor
259             * @param sourceImage the source image
260             * @pre source image is not NULL
261             */
262            CvDFT(Mat * sourceImage);
263
264            /**
265             * DFT Processor destructor
266             */
267            virtual ~CvDFT();
268
269            /**
270             * DFT Update.
```



```
271             * Steps in update
272             *  - crop source image to a square according to optima FFT size
273             *  - split in frame square into color channels
274             *  - converts these color channels to double
275             *  - apply frequency shift on double channels to
276             *      - produce the shifted real component of source channels
277             *      - produce later a spectrum with low frequencies at image center
278             *  - merge real/image channels into complex image per channel
279             *  - compute dft on each channel
280             *  - split channels complex spectrum in to real/imag components
281             *  - compute channels spectrum magnitude from real/imag components
282             *  - log scale channels spectrum magnitude
283             *  - if filtering
284             *      - fill lowpass channel masks with 0
285             *      - fill highpass channel masks with 1
286             *      - draw white low pass filter in lowpass channels
287             *      - draw white high pass filters in revrse highpass channels
288             *      - reverse highpass reverse channels to produce high pass
289             *      channels
290             *      - multiply lowpass & highpass channels into channel spectrum
291             *      masks
292             *      - multiply log magnitude spectrum channels by masks
293             *  - if not filtering then fill channels spectum masks with ones
294             *  - converts channels spectum masks for display
295             *  - converts channels log magnitude for display
296             *  - multiply channels complex spectrum components by masks
297             *  - perform inverse dft on masked spectrum channel complex to produce
298             *  inverse complex channels
299             *  - split inverse channels complex image into real/imag components
300             *  - perform frequency shift on real part o channels inverse component
301             *  - convert channels real inverse part for display
302             *  - merge channels spectrum log magnidude channels to displayable
303             *  image
304             *  - merge channels spectrum masks into a displayable image
305             *  - merge real channels of inverse dft into displayable image
306             */
307            virtual void update();
308
309            // ------------------------------------------------------------------------
310            // Options settings and gettings
311            // ------------------------------------------------------------------------
312
313            /**
314             * Filter type read access
315             * @return the current filter type
316             */
317            FilterType getFilterType() const;
318
319            /**
320             * Filter type setting
321             * @param filterType ne new filter type
322             */
323            virtual void setFilterType(const FilterType filterType);
324
325            /**
326             * Optimal dft size for current source image
327             * @return the current optimal dft size
328             */
329            int getOptimalDftSize() const;
330
331            /**
332             * Filtering status
333             * @return the filtering status. true if filtering is on, false
334             * otherwise
335             */
336            bool isFiltering() const;
337
338            /**
339             * Setting filtering status
340             * @param filtering ne new filtering status
341             */
342            virtual void setFiltering(bool filtering);
343
344            /**
345             * Get current log scale factor
346             * @return the current log scale factor
347             */
348            double getLogScaleFactor() const;
349
350            /**
351             * Setting the log scale factor
352             * @param logScaleFactor the new log scale factor
353             */
354            virtual void setLogScaleFactor(double logScaleFactor);
355
356            /**
357             * Returns min filter size for current image (generally 0)
358             * @return the minimum filter size
359             */
360            int getMinFilterSize() const;
```

```
361
362         /**
363          * Returns max filter size for current image
364          * @return the maximum filter size
365          */
366         int getMaxFilterSize() const;
367
368         /**
369          * Low pass filter size read access
370          * @param channel channel index
371          * @return the current low pass filter size or -1 if channel is invalid
372          */
373         int getLowPassFilterSize(const int channel = 0) const;
374
375         /**
376          * Low pass filter size setting
377          * @param channel channel index. If channel index == number of channels
378          * then set value for all channels
379          * @param filterSize the new value of low pass filter size.
380          * @note filterSize is limited to range
381          * [highPassFilterSize...filterMaxSize]
382          */
383         virtual void setLowPassFilterSize(const int channel,
384                                           const int filterSize);
385
386         /**
387          * High pass filter size read access
388          * @param channel channel index
389          * @return the current high pass filter size or -1 if channel is invalid
390          */
391         int getHighPassFilterSize(const int channel = 0) const;
392
393         /**
394          * High pass filter size setting
395          * @param channel channel index. If channel index == number of channels
396          * then set value for all channels
397          * @param filterSize the new value of high pass filter size.
398          * @note filterSize is limited to range
399          * [filterMinSize...lowPassFilterSize]
400          */
401         virtual void setHighPassFilterSize(const int channel,
402                                            const int filterSize);
403
404     protected:
405
406         // ------------------------------------------------------------------------
407         // Setup and cleanup attributes
408         // ------------------------------------------------------------------------
409
410         /**
411          * Setup internal attributes according to source image
412          * @param sourceImage a new source image
413          * @param fullSetup full setup is needed when source image is changed
414          */
415         void setup(Mat * sourceImage, bool fullSetup = true);
416
417         /**
418          * Clean up internal arrtibutes before changing source image or
419          * cleaning up class before destruction
420          */
421         void cleanup();
422
423         // ------------------------------------------------------------------------
424         // Utility methods
425         // ------------------------------------------------------------------------
426         /**
427          * Modify image to obtain reverse frequencies on the Fourier transform
428          * (low frequencies at the center of the image and high frequencies on
429          * the border), or modify image obtained from reverse Fourier transform
430          * with reversed frequencies.
431          * @param imgIn source image
432          * @param imgOut destination image
433          * @par Algorithm:
434          * This is based on the following property of the Z transform :
435          * \f[
436          *  TZ\left\{a^{k} x_{k}\right\} = X\left(\frac{z}{a}\right)
437          * \f]
438          * if \f$y_{k} = (-1)^{k} x_{k}\f$ then \f$Y(z) = X(-z)\f$
439          * which can be explained in Fourier space by replacing
440          * \f$z\f$ by \f$e^{j 2 \pi F}\f$:
441          * \f[
442          * Y\left[e^{i 2 \pi F}\right] = X\left[-e^{j2\pi F}\right] =
443          * X\left[e^{i\pile^{i2\pi F}\right] =
444          * X\left[e^{j2\pi\left(F + \frac{1}{2}\right)}\right]
445          * \f]
446          * hence
447          * \f[
448          * Y(F) = X\left(F + \frac{1}{2}\right)
449          * \f]
450          * or
```



```
451          * \f[
452          * Y(f) = X\left(f + \frac{f_{e}}{2}\right)
453          * \f]
454          * where \f$f_{e}\f$ is the sampling frequency, which means the
455          * resulting Fourier transform will present an \f$\frac{f_{e}}{2}\f$
456          * frequency offset. And since the sampling frequency lies in the middle
457          * of the spectrum in the DFT. Low frequencies will appear centered
458          * around the middle of the spectrum.
459          *
460          * In 2D the algorithm is the following:
461          * \f[
462          * imgOut(i,j) = (-1)^{i+j} \cdot imgIn(i,j)
463          * \f]
464          * \f$f_{e}\f$ is at the center of the spectrum image in 2D, which
465          * means, low frequencies will be located at the center of the image.
466          */
467         template <typename T>
468         void frequencyShift(Mat & imgIn, Mat & imgOut);
469
470         /**
471          * Computes a 2D gaussian on image
472          * @param image output (and/or input) image
473          * @param x0 x center
474          * @param y0 y center
475          * @param sigma gaussian width (at half height). If sigma <= 0 no
476          * output is performed
477          * @param amp amplitude
478          * @post Compute a 2D gaussian in image:
479          * \f[
480          * g(x,y) = amp \cdot exp^{-
481          * \left(
482          *  \frac{(x - x_{0})^{2} + (y - y_{0})^{2}}{2 \sigma^{2}}
483          *  \right)
484          *  }
485          * \f]
486          */
487         template <typename T>
488         void gaussian2D(Mat & image, double x0, double y0, double sigma,
489                         double amp);
490         /**
491          * Computes a 2D sinc on image
492          * @param image output (and/or input) image
493          * @param x0 x center
494          * @param y0 y center
495          * @param sigma width (at half height). If sigma <= 0 no
496          * output is performed
497          * @param amp amplitude
498          */
499         template <typename T>
500         void sinc2D(Mat & image, double x0, double y0, double sigma,
501                     double amp);
502
503         /**
504          * Log scale T valued image
505          * @param imgIn input image
506          * @param imgOut output image
507          * @param scaleFactor such as
508          * \f$ imgOut = scaleFactor \times \log(1 + imgIn)\f$
509          */
510         template <typename T>
511         void logScaleImg(const Mat & imgIn, Mat & imgOut, const T scaleFactor);
512
513         /**
514          * Negate image
515          * @param imgIn input image
516          * @param imgOut reversed output image
517          * @param value value to reverse from
518          * @code
519          * imageOut(i,j) = value - imageIn(i,j)
520          * @endcode
521          */
522         template <typename T>
523         void reverseValues(const Mat & imgIn, Mat & imgOut,
524                            const T value = numeric_limits<T>::max());
525 };
526
527 #endif /* CVDFT_H_ */
```

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

#include <limits>
#include <cmath>

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

#include <opencv2/imgproc/imgproc.hpp>

#include "CvDFT.h"

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

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

/*
 * DFT processor constructor
 * @param sourceImage the source image
 */
CvDFT::CvDFT(Mat * sourceImage) :
    CvProcessor(sourceImage),
    minSize(MIN(sourceImage->rows, sourceImage->cols)),
    maxSize(MAX(sourceImage->rows, sourceImage->cols)),
    borderSize((maxSize-minSize)/2),
    optimalDFTSize(getOptimalDFTSize(minSize)),
    filtering(false),
    filterType(BOX_FILTER),
    dftSize(optimalDFTSize, optimalDFTSize),
    inFrameSquare(dftSize, type),
    logScaleFactor(10.0),
    spectrumMagnitudeImage(dftSize, type),
    spectrumMaskImage(dftSize, type),
    inverseImage(dftSize, type),
    filterMaxSize((int)((double)optimalDFTSize / sqrt(2.0))),
    filterMinSize(0)
{
    setup(sourceImage, false);

    addImage("square", &inFrameSquare);
    addImage("mask", &spectrumMaskImage);
    addImage("spectrum", &spectrumMagnitudeImage);
    addImage("inverse", &inverseImage);
}

/*
 * DFT Processor destructor
 */
CvDFT::~CvDFT()
{
    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 CvDFT::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);
        spectrumMaskImage = Mat(dftSize, type);
        inverseImage = Mat(dftSize, type);
        filterMaxSize = (int)((double)optimalDFTSize / sqrt(2.0));
        filterMinSize = 0;
```



```cpp
    }

    // 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));
        channelsLowPassMask.push_back(Mat(dftSize, CV_64FC1));
        channelsHighPassMask.push_back(Mat(dftSize, CV_64FC1));
        channelsHighPassMaskReverse.push_back(Mat(dftSize, CV_64FC1));
        channelsSpectrumMask.push_back(Mat(dftSize, CV_64FC1));
        channelsSpectrumMaskDisplay.push_back(Mat(dftSize, CV_8UC1));
        channelsComplexSpectrumsMasked.push_back(Mat(dftSize, CV_64FC2));
        channelsComplexInverseImages.push_back(Mat(dftSize, CV_64FC2));
        channelsComplexInverseComponents.push_back(vector<Mat>());
        channelsRealInverse.push_back(Mat(dftSize, CV_64FC1));
        channelsRealInverseDisplay.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));
            channelsComplexInverseComponents[i].push_back(Mat(dftSize, CV_64FC1));
        }

        lowPassFilterSize.push_back(filterMaxSize);
        highPassFilterSize.push_back(filterMinSize);

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

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

        channelsRealInverseDisplay[i].release();
        channelsRealInverse[i].release();
        channelsComplexInverseComponents[i].clear();
        channelsComplexInverseImages[i].release();
        channelsComplexSpectrumsMasked[i].release();
        channelsSpectrumMaskDisplay[i].release();
        channelsSpectrumMask[i].release();
        channelsHighPassMask[i].release();
        channelsHighPassMaskReverse[i].release();
        channelsLowPassMask[i].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();
    }

    highPassFilterSize.clear();
    lowPassFilterSize.clear();
    channelsRealInverseDisplay.clear();
    channelsRealInverse.clear();
    channelsComplexInverseComponents.clear();
    channelsComplexInverseImages.clear();
    channelsComplexSpectrumsMasked.clear();
    channelsSpectrumMaskDisplay.clear();
    channelsSpectrumMask.clear();
    channelsHighPassMask.clear();
    channelsHighPassMaskReverse.clear();
    channelsLowPassMask.clear();
    channelsSpectrumLogMagnitudeDisplay.clear();
    channelsSpectrumLogMagnitude.clear();
    channelsSpectrumMagnitude.clear();
    channelsComplexSpectrumComponents.clear();
```

```
181     channelsComplexSpectrums.clear();
182     channelsComplexImages.clear();
183     channelsDoubleComplexComponents.clear();
184     channelsDouble.clear();
185     channels.clear();
186
187     inverseImage.release();
188     spectrumMaskImage.release();
189     spectrumMagnitudeImage.release();
190     inFrameSquare.release();
191
192     // super cleanup
193     CvProcessor::cleanup();
194 }
195
196 /*
197  * Update
198  */
199 void CvDFT::update()
200 {
201 //  clog << "CvDFT::update()" << endl;
202
203     Scalar one(1.0);
204     Scalar zero(0.0);
205     /*
206      * Crop source image to center square and resize it to nearest
207      * DFT optimal size
208      */
209     if (sourceImage->cols > sourceImage->rows)
210     {
211         resize(sourceImage->colRange(borderSize, borderSize + minSize),
212                inFrameSquare,
213                dftSize,
214                0,
215                0,
216                INTER_AREA);
217     }
218     else
219     {
220         resize(sourceImage->rowRange(borderSize, borderSize + minSize),
221                inFrameSquare,
222                dftSize,
223                0,
224                0,
225                INTER_AREA);
226     }
227
228     /*
229      * Split input frame square to individual channels
230      */
231     split(inFrameSquare, channels);
232
233     // Process each component
234     for (int i=0; i < nbChannels; i++)
235     {
236         /*
237          * Fourier transform processing
238          *  - Convert uchar center square image to CV_64F real component
239          *  - perform frequency shift on real image to obtain low frequencies
240          *      in the middle of the DFT image rather than in the corners
241          *  - merge real & imag component to complexImage before DFT
242          *      imag component could be filled with 0
243          *  - compute DFT
244          *  - split DFT channels
245          *  - compute DFT magnitude from DFT channels
246          *  - logScale magnitude with factor (5 to 20)
247          *  - convertScaleAbs logMagnitude to CV_8UC1 to display image
248          *
249          */
250
251         // convert component to float
252         channels[i].convertTo(channelsDouble[i], CV_64FC1);
253
254         // Frequency shift channelsDouble to real complex component
255         // Frequency shift allow to prepare spatial image components to
256         // produce frequency image later with low frequencies in the center
257         // of frequency image
258         frequencyShift<double>(channelsDouble[i],
259                                channelsDoubleComplexComponents[i][0]);
260         // channelsDoubleComplexComponents[i][1] is already filled with 0
261
262         // Merge Real and Imaginary into a complex component image
263         merge(channelsDoubleComplexComponents[i],
264               channelsComplexImages[i]);
265
266         // Perform Fourier transform on Complex component image
267         dft(channelsComplexImages[i],
268             channelsComplexSpectrums[i],
269             DFT_COMPLEX_OUTPUT);
270
```



```
271         // Split component Complex spectrum to real/imag channels
272         split(channelsComplexSpectrums[i],
273               channelsComplexSpectrumComponents[i]);
274
275         // Compute component spectrum magnitude
276         magnitude(channelsComplexSpectrumComponents[i][0],
277                   channelsComplexSpectrumComponents[i][1],
278                   channelsSpectrumMagnitude[i]);
279
280         // Log scale magnitude
281         logScaleImg<double>(channelsSpectrumMagnitude[i],
282                             channelsSpectrumLogMagnitude[i],
283                             logScaleFactor);
284
285         if (filtering)
286         {
287             // Clear lowpass mask with black
288             channelsLowPassMask[i] = zero;
289             // Clear highpass mask with white
290             channelsHighPassMask[i] = one;
291
292             // Compute lowpass and highpass masks
293             // Mask center
294             double gsize = (double) (optimalDFTSize-1) / 2.0;
295             // filter amplitude
296             double amplitude = 1.0;
297             switch (filterType)
298             {
299                 case BOX_FILTER:
300                     // Draw white filled circle on mask
301                     if (lowPassFilterSize[i] < filterMaxSize+1)
302                     {
303                         // Draws a white circle in channelsLowPassMask[i]
304                         // - point : (optimalDFTSize/2, optimalDFTSize/2)
305                         // - color : one (see above)
306                         // - filled circle
307                         // - with size from lowPassFilterSize[]
308                         // - use CV AA as lineType
309                         // - no shift
310                         // TODO ComplÃ©ter ...
311                         // circle(...);
312                     }
313
314                     // Draw black circle inside white filled circle
315                     if (highPassFilterSize[i] > 0)
316                     {
317                         // Draws a black circle in channelsHighPassMask[i]
318                         // - point : (optimalDFTSize/2, optimalDFTSize/2)
319                         // - color : zero (see above)
320                         // - filled circle
321                         // - with size from highPassFilterSize[]
322                         // - use CV AA as lineType
323                         // - no shift
324                         // TODO ComplÃ©ter ...
325                         // circle(...);
326                     }
327                     break;
328                 case GAUSS_FILTER:
329                     // TODO ComplÃ©ter la mÃ©thode gaussian2D<T>(...) en fin de fichier
330
331                     // positive gaussian for low pass freq filter
332                     if (lowPassFilterSize[i] < filterMaxSize+1)
333                     {
334                         gaussian2D<double> (channelsLowPassMask[i],
335                                             gsize,
336                                             gsize,
337                                             (double) lowPassFilterSize[i],
338                                             amplitude);
339                     }
340
341                     // negative gaussian for high pass freq filter
342                     if (highPassFilterSize[i] > filterMinSize)
343                     {
344                         gaussian2D<double> (channelsHighPassMaskReverse[i],
345                                             gsize,
346                                             gsize,
347                                             (double) highPassFilterSize[i],
348                                             amplitude);
349
350                         reverseValues<double>(channelsHighPassMaskReverse[i],
351                                               channelsHighPassMask[i],
352                                               1.0);
353                     }
354                     break;
355                 case SINC_FILTER:
356
357                     // TODO ComplÃ©ter la mÃ©thode sinc2D<T>(...) en fin de fichier
358
359                     // positive sinc
360                     if (lowPassFilterSize[i] < filterMaxSize+1)
```

```
                    {
                        sinc2D<double> (channelsLowPassMask[i],
                                        gsize,
                                        gsize,
                                        (double) lowPassFilterSize[i],
                                        amplitude);
                    }

                    // negative sinc
                    if (highPassFilterSize[i] > filterMinSize)
                    {
                        sinc2D<double> (channelsHighPassMaskReverse[i],
                                        gsize,
                                        gsize,
                                        (double) highPassFilterSize[i],
                                        amplitude);

                        reverseValues<double>(channelsHighPassMaskReverse[i],
                                              channelsHighPassMask[i],
                                              1.0);
                    }
                    break;
                default:
                    break;
            } // end switch (filterType)

            // multiply lowpass and highpass
            multiply(channelsLowPassMask[i],
                     channelsHighPassMask[i],
                     channelsSpectrumMask[i]);

            // multiply spectrum LogMagnitude by Spectrum mask when filtering
            multiply(channelsSpectrumLogMagnitude[i],
                     channelsSpectrumMask[i],
                     channelsSpectrumLogMagnitude[i]);

        } // end if (filtering)
        else // No filtering: spectrum mask is completely white
        {
            channelsSpectrumMask[i] = one;
        }

        // Converts Spectrum mask for display
        convertScaleAbs(channelsSpectrumMask[i],
                        channelsSpectrumMaskDisplay[i]);

        // Convert Log scale channels Spectrum to display channels
        convertScaleAbs(channelsSpectrumLogMagnitude[i],
                        channelsSpectrumLogMagnitudeDisplay[i]);

        /*
         * Inverse Fourier transform processing
         * Principle : Multiply spectrum by a mask and then compute inverse
         * transform which is equivalent convolve input image with the
         * inverse transform of the mask
         * - Creates a uchar mask same size as the DFT
         *  - Draw white filled circle on the mask to represent frequencies to filter
         * - Creates a complex floating point mask with the original mask
         * - Multiply complex spectrum with complex mask to procude the filtered spectrum
         * - Inverse filtered spectrum
         *  - split inverse Fourier complex image to channels
         *  - convert real channel to CV_8UC1 to display filtered image
         */

        // multiply spectrum real and imaginary channels with mask
        for (int j = 0; j < 2; j++)
        {
            multiply(channelsComplexSpectrumComponents[i][j],
                     channelsSpectrumMask[i],
                     channelsComplexSpectrumComponents[i][j]);
        }

        // remerge filtered component Spectrum Components to complex image
        merge(channelsComplexSpectrumComponents[i],
              channelsComplexSpectrumsMasked[i]);

        // perform inverse Fourier Transform
        // with specific flags : DFT_REAL_OUTPUT + DFT_SCALE
        idft(channelsComplexSpectrumsMasked[i],
             channelsComplexInverseImages[i],
             DFT_REAL_OUTPUT + DFT_SCALE);

        // split inverse FFT to real/imag channels
        split(channelsComplexInverseImages[i],
              channelsComplexInverseComponents[i]);

        // Reperform frequency shift on resulting real component
        frequencyShift<double>(channelsComplexInverseComponents[i][0],
                               channelsRealInverse[i]);
```



```
        // Convert real channel to display component
        convertScaleAbs(channelsRealInverse[i],
                        channelsRealInverseDisplay[i]);
    } // end for (int i=0; i < nbChannels; i++)

    // Merge channels spectrum Log magnitude to color spectrum image
    merge(channelsSpectrumLogMagnitudeDisplay,
          spectrumMagnitudeImage);

    // Merge channels spectrum masks to color mask image
    merge(channelsSpectrumMaskDisplay,
          spectrumMaskImage);

    // Merge channels inverse real parts into inverse image
    merge(channelsRealInverseDisplay, inverseImage);

}

/*
 * Filter type read access
 * @return the current filter type
 */
CvDFT::FilterType CvDFT::getFilterType() const
{
    return filterType;
}

/*
 * Filter type setting
 * @param filterType ne new filter type
 */
void CvDFT::setFilterType(const FilterType filterType)
{
    if (filterType < NB_FILTERS)
    {
        this->filterType = filterType;
    }
    else
    {
        cerr << "unknown filter type " << filterType;
    }
}

/*
 * Filtering status
 * @return the filtering status. true if filtering is on, false
 * otherwise
 */
bool CvDFT::isFiltering() const
{
    return filtering;
}

/*
 * Setting filtering status
 * @param filtering ne new filtering status
 */
void CvDFT::setFiltering(bool filtering)
{
    this->filtering = filtering;
}

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

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

/*
 * Setting the log scale factor
 * @param logScaleFactor the new log scale factor
 */
void CvDFT::setLogScaleFactor(double logScaleFactor)
{
    if (logScaleFactor > maxLogScaleFactor)
    {
        this->logScaleFactor = maxLogScaleFactor;
    }
```

```
541     else if (logScaleFactor < minLogScaleFactor)
542     {
543         this→logScaleFactor = minLogScaleFactor;
544     }
545     else
546     {
547         this→logScaleFactor = logScaleFactor;
548     }
549 }
550
551 /*
552  * Returns min filter size for current image (generally 0)
553  * @return the minimum filter size
554  */
555 int CvDFT::getMinFilterSize() const
556 {
557     return filterMinSize;
558 }
559
560 /*
561  * Returns max filter size for current image
562  * @return the maximum filter size
563  */
564 int CvDFT::getMaxFilterSize() const
565 {
566     return filterMaxSize;
567 }
568
569 /*
570  * Low pass filter size read access
571  * @param channel channel index
572  * @return the current low pass filter size or -1 if channel is invalid
573  */
574 int CvDFT::getLowPassFilterSize(const int channel) const
575 {
576     if ((channel ≥ 0) ∧ (channel < nbChannels))
577     {
578         return lowPassFilterSize[channel];
579     }
580     else
581     {
582         return -1;
583     }
584 }
585
586 /*
587  * High pass filter size read access
588  * @param channel channel index
589  * @return the current high pass filter size or -1 if channel is invalid
590  */
591 int CvDFT::getHighPassFilterSize(const int channel) const
592 {
593     if ((channel ≥ 0) ∧ (channel < nbChannels))
594     {
595         return highPassFilterSize[channel];
596     }
597     else
598     {
599         return -1;
600     }
601 }
602
603 /*
604  * Low pass filter size setting
605  * @param channel channel index. If channel index == number of channels
606  * then set value for all channels
607  * @param filterSize the new value of low pass filter size.
608  * @note filterSize is limited to range
609  * [highPassFilterSize...filterMaxSize]
610  */
611 void CvDFT::setLowPassFilterSize(const int channel, const int filterSize)
612 {
613     if ((channel ≥ 0) ∧ (channel < nbChannels))
614     {
615         if (filterSize < highPassFilterSize[channel])
616         {
617             lowPassFilterSize[channel] = highPassFilterSize[channel];
618         }
619         else if (filterSize > filterMaxSize)
620         {
621             lowPassFilterSize[channel] = filterMaxSize;
622         }
623         else
624         {
625             lowPassFilterSize[channel] = filterSize;
626         }
627     }
628 }
629
630 /*
```



```
631  * High pass filter size setting
632  * @param channel channel index. If channel index == number of channels
633  * then set value for all channels
634  * @param filterSize the new value of high pass filter size.
635  * @note filterSize is limited to range
636  * [filterMinSize...lowPassFilterSize]
637  */
638 void CvDFT::setHighPassFilterSize(const int channel, const int filterSize)
639 {
640     if ((channel ≥ 0) ∧ (channel < nbChannels))
641     {
642         if (filterSize > lowPassFilterSize[channel])
643         {
644             highPassFilterSize[channel] = lowPassFilterSize[channel];
645         }
646         else if (filterSize < filterMinSize)
647         {
648             highPassFilterSize[channel] = filterMinSize;
649         }
650         else
651         {
652             highPassFilterSize[channel] = filterSize;
653         }
654     }
655 }
656
657 // ------------------------------------------------------------------
658 // Utility methods
659 // ------------------------------------------------------------------
660 /*
661  * Modify image to obtain reverse frequencies on the Fourier transform
662  * (low frequencies at the center of the image and high frequencies on
663  * the border), or modify image obtained from reverse Fourier transform
664  * with reversed frequencies.
665  * @param imgIn source image
666  * @param imgOut destination image
667  */
668 template <typename T>
669 void CvDFT::frequencyShift(Mat & imgIn, Mat & imgOut)
670 {
671     int i, j;
672
673     for (i = 0; i < imgIn.rows; i++)
674     {
675         for (j = 0; j < imgIn.cols; j++)
676         {
677             /*
678              * Performance issue : using pow(-1.0, i + j) makes frequencyShift
679              * use up to 42.6 % of CPU time of update loop whereas using
680              * ((i+j)%2 == 0 ? 1.0 : -1.0) reduces this to 5.8 % of CPU time.
681              */
682             // imgOut.at<T> (i, j) = imgIn.at<T> (i, j) * (T)pow(-1.0, i + j);
683             imgOut.at<T>(i, j) = imgIn.at<T>(i, j) * (T) ((i + j) % 2 ≡ 0 ? 1.0 : -1.0);
684         }
685     }
686 }
687
688 /*
689  * Computes a 2D gaussian on image
690  * @param image output (and/or input) image
691  * @param x0 x center
692  * @param y0 y center
693  * @param sigma gaussian width (at half height). If sigma <= 0 no
694  * output is performed
695  * @param amp amplitude
696  */
697 template <typename T>
698 void CvDFT::gaussian2D(Mat & image, double x0, double y0, double sigma,
699     double amp)
700 {
701     if (sigma > 0.0)
702     {
703         // 2 * sigma^2
704         // TODO complÃ©ter
705         // double sigmaFactor = ...
706
707         for (int i = 0; i < image.rows; i++)
708         {
709             // yterms in the gaussian (y - y0)^2 / (2 * sigma^2)
710             // TODO complÃ©ter
711             // double yterms = ...
712
713             for (int j = 0; j < image.cols; j++)
714             {
715                 // xterms in the gaussian (x - x0)^2 / (2 * sigma^2)
716                 // TODO complÃ©ter
717                 // double xterms = ...
718
719                 // Gaussian = (T) (amp * exp(-(xterms + yterms))
720                 // TODO remplacer
```

```
721                 image.at<T> (i, j) = (T)(amp);
722                 // TODO par
723                 // image.at<T> (i, j) = (T)(amp * exp(...));
724             }
725         }
726     }
727 //  else
728 //  {
729 //      clog << "gaussian2D no output" << endl;
730 //  }
731 }
732
733 /*
734  * Computes a 2D sinc on image
735  * @param image output (and/or input) image
736  * @param x0 x center
737  * @param y0 y center
738  * @param sigma width (at half height). If sigma <= 0 no
739  * output is performed
740  * @param amp amplitude
741  */
742 template <typename T>
743 void CvDFT::sinc2D(Mat & image, double x0, double y0, double sigma,
744     double amp)
745 {
746     // h : mid height width
747     double h = 0.6033 * M_PI / sigma;
748     double espilon = 1e-9;
749     double xterms, yterms, valTerm;
750
751     if (sigma > 0.0)
752     {
753         for (int i = 0; i < image.rows; i++)
754         {
755             // yterms in sinc (y-y0)^2
756             // TODO ComplÃ©ter ...
757             // double yterms = ...
758
759             for (int j = 0; j < image.cols; j++)
760             {
761                 // xterms in sinc (x-x0)^2
762                 // TODO ComplÃ©ter ...
763                 // double xterms = ...
764
765                 // TODO Remplacer ...
766                 double valTerm = 0;
767                 // TODO par value term in sin(value)/value : h * (xterms + yterms)^(1/2)
768                 // double valTerm =
769
770                 if (abs(valTerm) > espilon)
771                 {
772                     // TODO Remplacer ...
773                     image.at<T> (i, j) = (T)(amp);
774                     // TODO Par amp * sin(...)/...
775                     // image.at<T> (i, j) = (T)(...);
776                 }
777                 else
778                 {
779                     // Sinc for 0 value is amp (avoid divide by 0)
780                     image.at<T> (i, j) = (T)amp;
781                 }
782             }
783         }
784     }
785 //  else
786 //  {
787 //      clog << "sinc2D no output" << endl;
788 //  }
789 }
790
791
792 /*
793  * Log scale T valued image
794  * @param imgIn input image
795  * @param imgOut output image
796  * @param scaleFactor such as
797  * \f$ imgOut = scaleFactor \times \log(1 + imgIn)\f$
798  */
799 template <typename T>
800 void CvDFT::logScaleImg(const Mat & imgIn, Mat & imgOut,
801     const T scaleFactor)
802 {
803     MatConstIterator_<T> inIt = imgIn.begin<T>();
804     MatConstIterator_<T> inItEnd = imgIn.end<T>();
805     MatIterator_<T> outIt = imgOut.begin<T>();
806     for (; inIt ≠ inItEnd; ++inIt, ++outIt)
807     {
808         (*outIt) = scaleFactor * (T)log(1.0 + (*inIt));
809     }
810 }
```



```
811
812 template <typename T>
813 void CvDFT::reverseValues(const Mat & imgIn, Mat & imgOut, const T value)
814 {
815     // input image iterators
816     MatConstIterator_<T> inIt = imgIn.begin<T>();
817     MatConstIterator_<T> inItEnd = imgIn.end<T>();
818     // output image iterators
819     MatIterator_<T> outIt = imgOut.begin<T>();
820     MatIterator_<T> outItEnd = imgOut.end<T>();
821     for (; (inIt ≠ inItEnd) ∧ (outIt ≠ outItEnd); ++inIt, ++outIt)
822     {
823         (*outIt) = value - (*inIt);
824     }
825 }
```

```
/*
 * QcvDFT.h
 *
 *  Created on: 22 fÃ©vr. 2012
 *      Author: davidroussel
 */

#ifndef QCVDFT_H_
#define QCVDFT_H_

#include <QMutex>

#include "QcvProcessor.h"
#include "CvDFT.h"

class QcvDFT: public QcvProcessor, public CvDFT
{
    Q_OBJECT

    protected:
        /**
         * Self lock for operation from multiple threads
         * @note May be NULL if there is no update thread
         */
        QMutex * selfLock;
    public:

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

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

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

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

        /**
         * Filter type setting with notification
         * @param filterType ne new filter type
         */
        void setFilterType(FilterType filterType);

        /**
         * Setting filtering status with notification
         * @param filtering ne new filtering status
         */
        void setFiltering(bool filtering);

        /**
         * Setting the log scale factor
         * @param logScaleFactor the new log scale factor
         */
        void setLogScaleFactor(double logScaleFactor);

        /**
         * Low pass filter size setting
         * @param channel channel index. If channel index == number of channels
         * then set value for all channels
         * @param filterSize the new value of low pass filter size.
         * @note filterSize is limited to range
         * [highPassFilterSize...filterMaxSize]
         */
        void setLowPassFilterSize(const int channel,
```



```
                                  const int filterSize);

        /**
         * High pass filter size setting
         * @param channel channel index. If channel index == number of channels
         * then set value for all channels
         * @param filterSize the new value of high pass filter size.
         * @note filterSize is limited to range
         * [filterMinSize...lowPassFilterSize]
         */
        void setHighPassFilterSize(const int channel,
                                   const int filterSize);

    signals:

//      /**
//       * Signal sent when source image changes to adjust max filter sizes
//       */
//      void dftSizeChanged();

        /**
         * Signal sent when input dftSize square image has been reallocated
         * @param image the new in square image
         */
        void squareImageChanged(Mat * image);

        /**
         * Signal sent when spectrum image has been reallocated
         * @param image the new spectrum image
         */
        void spectrumImageChanged(Mat * image);

        /**
         * Signal sent when inverse image has been reallocated
         * @param image the new inverse image
         */
        void inverseImageChanged(Mat * image);
};

#endif /* QCVDFT_H_ */
```

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

#include "QcvDFT.h"


/*
 * QcvDFT 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
 */
QcvDFT::QcvDFT(Mat *image,
               QMutex * imageLock,
               QThread * updateThread,
               QObject *parent) :
    CvProcessor(image), // <-- virtual base class constructor first
    QcvProcessor(image, imageLock, updateThread, parent),
    CvDFT(image),
    selfLock(updateThread ≠ NULL ? new QMutex() :
                                   (imageLock ≠ NULL ? imageLock : NULL))
{
}


/*
 * QcvDFT destructor
 */
QcvDFT::~QcvDFT()
{
    message.clear();
    if (selfLock ≠ NULL)
    {
        // wait for update completion
        selfLock→lock();
        selfLock→unlock();
        delete selfLock;
    }
}

/*
 * Update computed images slot and sends updated signal
 * required
 */
void QcvDFT::update()
{
    bool hasSourceLock = (sourceLock ≠ NULL) ∧ (sourceLock ≠ selfLock);
    if (hasSourceLock)
    {
        sourceLock→lock();
        // qDebug() << "QcvDFT::update : lock";
    }
    bool hasLock = selfLock ≠ NULL;
    if (hasLock)
    {
        selfLock→lock();
    }
    /*
     * Update DFT images
     */
    CvDFT::update();

    if (hasLock)
    {
        selfLock→unlock();
    }
    if (hasSourceLock)
    {
        // qDebug() << "QcvDFT::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 QcvDFT::setSourceImage(Mat *image)
    throw (CvProcessorException)
```



```
{
    Size previousSize(sourceImage→size());
    int previousNbChannels(nbChannels);
    Size previousDftSize(dftSize);
    bool hasLock = selfLock ≠ NULL;
    if (hasLock)
    {
        selfLock→lock();
    }

    CvProcessor::setSourceImage(image);

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

    emit imageChanged(sourceImage);

    emit imageChanged();

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

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

    emit squareImageChanged(&inFrameSquare);

    emit spectrumImageChanged(&spectrumMagnitudeImage);

    emit inverseImageChanged(&inverseImage);

    if ((previousDftSize.width ≠ dftSize.width) ∨
        (previousDftSize.height ≠ dftSize.height))
    {
        emit imageSizeChanged();
        emit sendText(QString::number(optimalDFTSize));
    }

    // Force update
    // update();
}

/*
 * Filter type setting with notification
 * @param filterType ne new filter type
 */
void QcvDFT::setFilterType(FilterType filterType)
{
    bool hasLock = selfLock ≠ NULL;
    if (hasLock)
    {
        selfLock→lock();
    }

    CvDFT::setFilterType(filterType);

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

    message.clear();
    message.append(tr("Filter type set to "));
    switch (filterType)
    {
        case BOX_FILTER:
            message.append(tr("Box"));
            break;
        case GAUSS_FILTER:
            message.append(tr("Gaussian"));
            break;
        case SINC_FILTER:
            message.append(tr("Sinus Cardinal"));
            break;
        default:
            message.append(tr("Unknown"));
            break;
    }

    emit sendMessage(message, defaultTimeOut);
}

/*
```

```
181  * Setting filtering status with notification
182  * @param filtering ne new filtering status
183  */
184 void QcvDFT::setFiltering(bool filtering)
185 {
186     bool hasLock = selfLock ≠ NULL;
187     if (hasLock)
188     {
189         selfLock→lock();
190     }
191
192     CvDFT::setFiltering(filtering);
193
194     if (hasLock)
195     {
196         selfLock→unlock();
197     }
198
199     message.clear();
200
201     message.append(tr("frequency filtering is "));
202
203     if (filtering)
204     {
205         message.append(tr("on"));
206     }
207     else
208     {
209         message.append(tr("off"));
210     }
211
212     emit sendMessage(message, defaultTimeOut);
213 }
214
215 /*
216  * Setting the log scale factor
217  * @param logScaleFactor the new log scale factor
218  */
219 void QcvDFT::setLogScaleFactor(double logScaleFactor)
220 {
221     bool hasLock = selfLock ≠ NULL;
222     if (hasLock)
223     {
224         selfLock→lock();
225     }
226
227     CvDFT::setLogScaleFactor(logScaleFactor);
228
229     if (hasLock)
230     {
231         selfLock→unlock();
232     }
233 }
234
235 /*
236  * Low pass filter size setting
237  * @param channel channel index. If channel index == number of channels
238  * then set value for all channels
239  * @param filterSize the new value of low pass filter size.
240  * @note filterSize is limited to range
241  * [highPassFilterSize...filterMaxSize]
242  */
243 void QcvDFT::setLowPassFilterSize(const int channel, const int filterSize)
244 {
245     bool hasLock = selfLock ≠ NULL;
246     if (hasLock)
247     {
248         selfLock→lock();
249     }
250
251     CvDFT::setLowPassFilterSize(channel, filterSize);
252
253     if (hasLock)
254     {
255         selfLock→unlock();
256     }
257 }
258
259 /*
260  * High pass filter size setting
261  * @param channel channel index. If channel index == number of channels
262  * then set value for all channels
263  * @param filterSize the new value of high pass filter size.
264  * @note filterSize is limited to range
265  * [filterMinSize...lowPassFilterSize]
266  */
267 void QcvDFT::setHighPassFilterSize(const int channel,const int filterSize)
268 {
269     bool hasLock = selfLock ≠ NULL;
270     if (hasLock)
```



```
271     {
272         selfLock→lock();
273     }
274
275     CvDFT::setHighPassFilterSize(channel, filterSize);
276
277     if (hasLock)
278     {
279         selfLock→unlock();
280     }
281 }
```

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

#include <opencv2/imgproc.hpp>

#include "QcvMatWidget.h"

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

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

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

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

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

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

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

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

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



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

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

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

    this->sourceImage = sourceImage;

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

}

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

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

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

```
181     }
182 }
183
184 /*
185  * Callback called when mouse button pressed event occurs.
186  * reimplemented to send pressPoint signal when left mouse button is
187  * pressed
188  * @param event mouse event
189  */
190 void QcvMatWidget::mousePressEvent(QMouseEvent *event)
191 {
192     if (mouseSense > MOUSE_NONE)
193     {
194 //      qDebug("mousePressEvent(%d, %d) with button %d",
195 //             event->pos().x(), event->pos().y(), event->button());
196         mousePressed = true;
197         pressedPoint = event->pos();
198         pressedButton = event->button();
199
200         if((event->button() == Qt::LeftButton) & (mouseSense & MOUSE_DRAG))
201         {
202             // initialise selection rect
203             selectionRect.setTopLeft(pressedPoint);
204             selectionRect.setBottomRight(pressedPoint);
205         }
206
207         emit pressPoint(pressedPoint, pressedButton);
208     }
209 }
210
211 /*
212  * Callback called when mouse move event occurs.
213  * reimplemented to send dragPoint signal when mouse is dragged
214  * (after left mouse button has been pressed)
215  * @param event mouse event
216  */
217 void QcvMatWidget::mouseMoveEvent(QMouseEvent *event)
218 {
219     mouseMoved = true;
220     draggedPoint = event->pos();
221
222     if ((mouseSense & MOUSE_DRAG) & mousePressed)
223     {
224 //      qDebug("mouseMoveEvent(%d, %d) with button %d",
225 //             event->pos().x(), event->pos().y(), event->button());
226
227         selectionRectFromPoints(pressedPoint, draggedPoint);
228
229         emit dragPoint(draggedPoint);
230     }
231 }
232
233 /*
234  * Callback called when mouse button released event occurs.
235  * reimplemented to send releasePoint signal when left mouse button is
236  * released
237  * @param event mouse event
238  */
239 void QcvMatWidget::mouseReleaseEvent(QMouseEvent *event)
240 {
241     if ((mouseSense > MOUSE_NONE) & mousePressed)
242     {
243 //      qDebug("mouseReleaseEvent(%d, %d) with button %d",
244 //             event->pos().x(), event->pos().y(), event->button());
245         mousePressed = false;
246         mouseMoved = false;
247         releasedPoint = event->pos();
248         emit releasePoint(releasedPoint, pressedButton);
249
250         if ((event->button() == Qt::LeftButton) & (mouseSense & MOUSE_DRAG))
251         {
252             selectionRectFromPoints(pressedPoint, releasedPoint);
253             emit releaseSelection(selectionRect, event->button());
254         }
255     }
256 }
257
258 /*
259  * Draw Cross
260  * @param p the cross center
261  */
262 void QcvMatWidget::drawCross(const QPoint & p)
263 {
264     int x0 = p.x();
265     int y0 = p.y();
266     int x1, x2, x3, x4;
267     int y1, y2, y3, y4;
268     int offset = 10;
269
270     x1 = x0 - 2*offset;
```



```
271     x2 = x0 - offset;
272     x3 = x0 + offset;
273     x4 = x0 + 2*offset;
274     y1 = y0 - 2*offset;
275     y2 = y0 - offset;
276     y3 = y0 + offset;
277     y4 = y0 + 2*offset;
278
279     Point p1a(x1, y0);
280     Point p1b(x2, y0);
281     Point p2a(x3, y0);
282     Point p2b(x4, y0);
283     Point p3a(x0, y1);
284     Point p3b(x0, y2);
285     Point p4a(x0, y3);
286     Point p4b(x0, y4);
287
288     line(displayImage, p1a, p1b, drawingColor, drawingWidth, CV_AA);
289     line(displayImage, p2a, p2b, drawingColor, drawingWidth, CV_AA);
290     line(displayImage, p3a, p3b, drawingColor, drawingWidth, CV_AA);
291     line(displayImage, p4a, p4b, drawingColor, drawingWidth, CV_AA);
292 }
293
294 /*
295  * Draw rectangle
296  * @param r the rectangle to draw
297  */
298 void QcvMatWidget::drawRectangle(const QRect & r)
299 {
300     int x1 = r.left();
301     int x2 = r.right();
302     int y1 = r.top();
303     int y2 = r.bottom();
304
305     Point p1(x1, y1);
306     Point p2(x2, y2);
307
308     rectangle(displayImage, p1, p2, drawingColor, drawingWidth, CV_AA);
309 }
310
311 /*
312  * Modifiv selectionRect using two points
313  * @param p1 first point
314  * @param p2 second point
315  */
316 void QcvMatWidget::selectionRectFromPoints(const QPoint & p1, const QPoint & p2)
317 {
318     int left, right, top, bottom;
319     if (p1.x() < p2.x())
320     {
321         left = p1.x();
322         right = p2.x();
323     }
324     else
325     {
326         left = p2.x();
327         right = p1.x();
328     }
329
330     if (p1.y() < p2.y())
331     {
332         top = p1.y();
333         bottom = p2.y();
334     }
335     else
336     {
337         top = p2.y();
338         bottom = p1.y();
339     }
340
341     selectionRect.setLeft(left);
342     selectionRect.setRight(right);
343     selectionRect.setTop(top);
344     selectionRect.setBottom(bottom);
345 }
346
347
348
349 /*
350  * Widget minimum size is set to the contained image size
351  * @return le size of the image within
352  */
353 //QSize QcvMatWidget::minimumSize() const
354 //{
355 //  return sizeHint();
356 //}
357
358
359 /*
360  * Size hint (because size depends on sourceImage properties)
```

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



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

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

using namespace std;

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

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

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

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

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

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



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

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

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

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

    setup();
}

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

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

    /*
     * Customize layout
     */

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

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

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

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



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

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

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

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

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

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

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

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

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

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

    glClear(GL_COLOR_BUFFER_BIT);

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



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

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

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

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

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

```
1   /*
2    * QcvVideoCapture.cpp
3    *
4    *  Created on: 29 janv. 2012
5    *      Author: davidroussel
6    */
7
8   #include <QElapsedTimer>
9   #include <QDebug>
10
11  #include "QcvVideoCapture.h"
12
13  #include <opencv2/imgproc/imgproc.hpp>
14
15  /*
16   * default time interval between refresh
17   */
18  int QcvVideoCapture::defaultFrameDelay = 33;
19
20  /*
21   * Number of frames to test frame rate
22   */
23  size_t QcvVideoCapture::defaultFrameNumberTest = 5;
24
25  /*
26   * Default message showing time (at least 2000 ms)
27   */
28  int QcvVideoCapture::messageDelay = 5000;
29
30  /*
31   * QcvVideoCapture constructor.
32   * Opens the default camera (0)
33   * @param flipVideo mirror image status
34   * @param gray convert image to gray status
35   * @param skip indicates capture can skip an image. When the capture
36   * result has not been processed yet, or when false that capture should
37   * wait for the result to be processed before grabbing a new image.
38   * This only applies when #updateThread is not NULL.
39   * @param width desired width or 0 to keep capture width
40   * @param height desired height or 0 to keep capture height
41   * otherwise capture is updated in the current thread.
42   * @param updateThread the thread used to run this capture
43   * @param parent the parent QObject
44   */
45  QcvVideoCapture::QcvVideoCapture(const bool flipVideo,
46                                   const bool gray,
47                                   const bool skip,
48                                   const unsigned int width,
49                                   const unsigned int height,
50                                   QThread * updateThread,
51                                   QObject * parent) :
52      QcvVideoCapture(0, flipVideo, gray, skip, width, height, updateThread,
53                      parent)
54  {
55  }
56
57  /*
58   * QcvVideoCapture constructor with device Id
59   * @param deviceId the id of the camera to open
60   * @param flipVideo mirror image
61   * @param gray convert image to gray
62   * @param skip indicates capture can skip an image. When the capture
63   * result has not been processed yet, or when false that capture should
64   * wait for the result to be processed before grabbing a new image.
65   * This only applies when #updateThread is not NULL.
66   * @param width desired width or 0 to keep capture width
67   * @param height desired height or 0 to keep capture height
68   * @param updateThread the thread used to run this capture
69   * @param parent the parent QObject
70   */
71  QcvVideoCapture::QcvVideoCapture(const int deviceId,
72                                   const bool flipVideo,
73                                   const bool gray,
74                                   const bool skip,
75                                   const unsigned int width,
76                                   const unsigned int height,
77                                   QThread * updateThread,
78                                   QObject * parent) :
79      QObject(parent),
80      filename(),
81      capture(deviceId),
82      timer(new QTimer(updateThread != NULL ? this : NULL)),
83      updateThread(updateThread),
84      mutex(QMutex::NonRecursive),
85      lockLevel(0),
86      liveVideo(true),
87      flipVideo(flipVideo),
88      resize(false),
89      directResize(false),
90      gray(gray),
```



```
91      skip(skip),
92      size(0, 0),
93      originalSize(0, 0),
94      frameRate(0.0),
95      statusMessage()
96  {
97      if (updateThread != NULL)
98      {
99          moveToThread(this->updateThread);
100         connect(this, SIGNAL(finished()), updateThread, SLOT(quit()),
101                 Qt::DirectConnection);
102     }
103
104     timer->setSingleShot(false);
105     connect(timer, SIGNAL(timeout()), SLOT(update()));
106
107     if (grabTest())
108     {
109         setSize(width, height);
110         QString message("Camera ");
111         message.append(QString::number(deviceId));
112         message.append(" ");
113         int delay = grabInterval(message);
114         if (updateThread != NULL)
115         {
116             updateThread->start();
117         }
118         timer->start(delay);
119         qDebug("timer started with %d ms delay", delay);
120         emit timerChanged(delay);
121     }
122     else
123     {
124         qDebug() << "QcvVideoCapture::QcvVideoCapture(" << deviceId
125                  << "): grab test failed";
126     }
127 }
128
129 /*
130  * QcvVideoCapture constructor from file name
131  * @param fileName video file to open
132  * @param flipVideo mirror image
133  * @param gray convert image to gray
134  * @param skip indicates capture can skip an image. When the capture
135  * result has not been processed yet, or when false that capture should
136  * wait for the result to be processed before grabbing a new image.
137  * This only applies when #updateThread is not NULL.
138  * @param width desired width or 0 to keep capture width
139  * @param height desired height or 0 to keep capture height
140  * @param updateThread the thread used to run this capture
141  * @param parent the parent QObject
142  */
143 QcvVideoCapture::QcvVideoCapture(const QString & fileName,
144                                  const bool flipVideo,
145                                  const bool gray,
146                                  const bool skip,
147                                  const unsigned int width,
148                                  const unsigned int height,
149                                  QThread * updateThread,
150                                  QObject * parent) :
151     QObject(parent),
152     filename(fileName),
153     capture(fileName.toStdString()),
154     timer(new QTimer(updateThread != NULL ? this : NULL)),
155     updateThread(updateThread),
156     mutex(QMutex::NonRecursive),
157     lockLevel(0),
158     liveVideo(false),
159     flipVideo(flipVideo),
160     resize(false),
161     directResize(false),
162     gray(gray),
163     skip(skip),
164     size(0, 0),
165     originalSize(0, 0),
166     frameRate(0.0),
167     statusMessage()
168 {
169     if (updateThread != NULL)
170     {
171         moveToThread(this->updateThread);
172         connect(this, SIGNAL(finished()), updateThread, SLOT(quit()),
173                 Qt::DirectConnection);
174     }
175
176     timer->setSingleShot(false);
177     connect(timer, SIGNAL(timeout()), SLOT(update()));
178
179     if (grabTest())
180     {
```

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



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

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



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

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



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

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



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

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



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

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

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

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

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

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

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

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

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

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

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

/*
 * Set gray conversion
```

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

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

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

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

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

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

    return capture;
}
```



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

#include "MeanValue.h"

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

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

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

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

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

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

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

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

    return *this;
}

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

    return *this;
}

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

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

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



```

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

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

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

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

    return out;
}

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

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

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



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

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

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

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

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

/**
 * OpenCV/Qt Histograms and LUT main window
 */
class MainWindow : public QMainWindow
{
    Q_OBJECT

    public:

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

        /**
         * MainWindow constructor.
         * @param capture the capture QObject to capture frames from devices
         * or video files
         * @param processor Fourier transform and filter processor
         * @param parent parent widget
         */
        explicit MainWindow(QcvVideoCapture * capture,
                            QcvDFT * processor,
                            QWidget *parent = NULL);

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

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

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

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

        /**
         * Signal to send for opening a video file in the capture
         * @param fileName video file to open
         * @param width desired width or 0 to keep capture width
         * @param height desired height or 0 to keep capture height
         * @return true if video has been opened and timer launched
         */
```



```
        void fileChanged(const QString & fileName,
                         const unsigned int width,
                         const unsigned int height);

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

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

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

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

        /**
         * The Fourier Transform and filter processor
         */
        QcvDFT * processor;

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

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

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

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

        /**
         * Set filters spinBoxes and sliders link state
         * @param linked the link status
         * @post When link is on all sliders/spinboxes of low pass and high pass
         * filters are linked together, moving/changing one changes the others.
         * When link os off sliders/spinboxes are not linked together
         */
        void setLinkedFilterSizes(bool linked);

    private slots:

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

        /**
         * Setup filter min/max and evt values according to source image size.
         * Filter max size is \f$\frac{FFTSize}{\sqrt{2}}\f$
         */
        void setupFilterSizes();

        /**
         * Setup filters sliders/spinboxes availability according to the number
         * of channels in the source image
         */
        void setupFiltersAvailability();

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

```
181         void on_actionCamera_0_triggered();
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 on_actionCamera_1_triggered();
189
190         /**
191          * Menu action when Sources->file is selected.
192          * Opens file dialog and tries to open selected file (is not empty),
193          * then sets capture to open the selected file
194          */
195         void on_actionFile_triggered();
196
197         /**
198          * Menu action to quit application.
199          */
200         void on_actionQuit_triggered();
201
202         /**
203          * Menu action when flip image is selected.
204          * Sets capture to change flip status which leads to reverse
205          * image horizontally
206          */
207         void on_actionFlip_triggered();
208
209         /**
210          * Menu action when gray image is selected.
211          * Sets capture to change gray status which leads convert captured image
212          * to gray or not
213          */
214         void on_actionGray_triggered();
215
216         /**
217          * Menu action when original image size is selected.
218          * Sets capture not to resize image
219          */
220         void on_actionOriginalSize_triggered();
221
222         /**
223          * Menu action when constrained image size is selected.
224          * Sets capture resize to preferred width and height
225          */
226         void on_actionConstrainedSize_triggered();
227
228         /**
229          * Menu action to replace current image rendering widget by a
230          * QcvMatWidgetImage instance.
231          */
232         void on_actionRenderImage_triggered();
233
234         /**
235          * Menu action to replace current image rendering widget by a
236          * QcvMatWidgetLabel with pixmap instance.
237          */
238         void on_actionRenderPixmap_triggered();
239
240         /**
241          * Menu action to replace current image rendering widget by a
242          * QcvMatWidgetGL instance.
243          */
244         void on_actionRenderOpenGL_triggered();
245
246         /**
247          * Original size radioButton action.
248          * Sets capture resize to off
249          */
250         void on_radioButtonOrigSize_clicked();
251
252         /**
253          * Custom size radioButton action.
254          * Sets capture resize to preferred width and height
255          */
256         void on_radioButtonCustomSize_clicked();
257
258         /**
259          * Width spinbox value change.
260          * Changes the preferred width and if custom size is selected apply
261          * this custom width
262          * @param value the desired width
263          */
264         void on_spinBoxWidth_valueChanged(int value);
265
266         /**
267          * Height spinbox value change.
268          * Changes the preferred height and if custom size is selected apply
269          * this custom height
270          * @param value the desired height
```



```
271          */
272         void on_spinBoxHeight_valueChanged(int value);
273
274         /**
275          * Flip capture image horizontally.
276          * changes capture flip status
277          */
278         void on_checkBoxFlip_clicked();
279
280         /**
281          * convert capture image to gray level.
282          * changes cpature gray conversion status
283          */
284         void on_checkBoxGray_clicked();
285
286         /**
287          * Changes logscale factor for spectrum
288          * @param value the new logscale factor
289          */
290         void on_spinBoxMag_valueChanged(int value);
291
292         /**
293          * Sets filtering on/off
294          */
295         void on_checkBoxFiltering_clicked();
296
297         /**
298          * Sets Filter mode to box
299          */
300         void on_radioButtonFilterBox_clicked();
301
302         /**
303          * Sets filter mode to gaussian
304          */
305         void on_radioButtonFilterGauss_clicked();
306
307         /**
308          * Sets filter mode to sinus cardinal
309          */
310         void on_radioButtonFilterSinc_clicked();
311
312         /**
313          * Changes low pass filter size for red component
314          * @param value the new low pass filter size for red component
315          * @note low pass filters are lower bounded by their respective high
316          * pass filters
317          */
318         void on_spinBoxRedLP_valueChanged(int value);
319
320         /**
321          * Changes low pass filter size for green component
322          * @param value the new low pass filter size for green component
323          * @note low pass filters are lower bounded by their respective high
324          * pass filters
325          */
326         void on_spinBoxGreenLP_valueChanged(int value);
327
328         /**
329          * Changes low pass filter size for blue component
330          * @param value the new low pass filter size for blue component
331          * @note low pass filters are lower bounded by their respective high
332          * pass filters
333          */
334         void on_spinBoxBlueLP_valueChanged(int value);
335
336         /**
337          * Changes high pass filter size for red component
338          * @param value the new high pass filter size for red component
339          * @note high pass filters are upper bounded by their respective low
340          * pass filters
341          */
342         void on_spinBoxRedHP_valueChanged(int value);
343
344         /**
345          * Changes high pass filter size for green component
346          * @param value the new high pass filter size for green component
347          * @note high pass filters are upper bounded by their respective low
348          * pass filters
349          */
350         void on_spinBoxGreenHP_valueChanged(int value);
351
352         /**
353          * Changes high pass filter size for blue component
354          * @param value the new high pass filter size for blue component
355          * @note high pass filters are upper bounded by their respective low
356          * pass filters
357          */
358         void on_spinBoxBlueHP_valueChanged(int value);
359
360         /**
```

```
361          * Changes spinboxes/sliders link mode from low pass pane
362          */
363         void on_checkBoxLinkLP_clicked();
364
365         /**
366          * Changes spinboxes/sliders link mode from high pass pane
367          */
368         void on_checkBoxLinkHP_clicked();
369 };
370
371 #endif // MAINWINDOW_H
```



```
1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3
4  #include <QObject>
5  #include <QFileDialog>
6  #include <QWindow>
7  #include <QDebug>
8  #include <assert.h>
9
10 #include "QcvMatWidgetImage.h"
11 #include "QcvMatWidgetLabel.h"
12 #include "QcvMatWidgetGL.h"
13
14 /*
15  * MainWindow constructor.
16  * @param capture the capture QObject to capture frames from devices
17  * or video files
18  * @param processor Fourier transform and filter processor
19  * @param parent parent widget
20  */
21 MainWindow::MainWindow(QcvVideoCapture * capture,
22                        QcvDFT * processor,
23                        QWidget *parent) :
24     QMainWindow(parent),
25     ui(new Ui::MainWindow),
26     capture(capture),
27     processor(processor),
28     preferredWidth(341),
29     preferredHeight(256)
30 {
31     ui->setupUi(this);
32     ui->scrollAreaSource->setBackgroundRole(QPalette::Mid);
33     ui->scrollAreaSpectrum->setBackgroundRole(QPalette::Mid);
34     ui->scrollAreaInverse->setBackgroundRole(QPalette::Mid);
35
36     // ----------------------------------------------------------------------
37     // Assertions
38     // ----------------------------------------------------------------------
39     assert(capture ≠ NULL);
40
41     assert(processor ≠ NULL);
42
43     // ----------------------------------------------------------------------
44     // Special widgets initialisation
45     // ----------------------------------------------------------------------
46     // Replace QcvMatWidget instances with QcvMatWidgetImage instances
47     // sets image widget sources for the first time
48     // connects processor->update to image Widgets->updated
49     // connects processor->image_changed to image_widgets->setSourceImage
50     setRenderingMode(RENDER_IMAGE);
51
52     ui->labelFFTSizeValue->setText(QString::number(processor->getOptimalDftSize()));
53
54     // Setup filter sizes according to image size
55     setupFilterSizes();
56
57     // Setup spin and sliders availability according to image colors
58     setupFiltersAvailability();
59
60     // ----------------------------------------------------------------------
61     // rest of Signal/Slot connections
62     // ----------------------------------------------------------------------
63     // processor->sendText --> labelFFTSizeValue->setText when source image
64     // changes, fft size might also change
65
66     connect(processor, SIGNAL(sendText(QString)),
67             ui->labelFFTSizeValue, SLOT(setText(QString)));
68
69     // Capture, processor and this messages to status bar
70     connect(capture, SIGNAL(messageChanged(QString,int)),
71             ui->statusBar, SLOT(showMessage(QString,int)));
72
73     connect(processor, SIGNAL(sendMessage(QString,int)),
74             ui->statusBar, SLOT(showMessage(QString,int)));
75
76     connect(this, SIGNAL(sendMessage(QString,int)),
77             ui->statusBar, SLOT(showMessage(QString,int)));
78
79     // When Processor source image changes, some attributes are reinitialised
80     // So we have to set them up again according to current UI values
81     connect(processor, SIGNAL(imageChanged()),
82             this, SLOT(setupProcessorFromUI()));
83
84     // When processor images size changes we need to update filter sizes
85     // (min, max & value)
86     connect(processor, SIGNAL(imageSizeChanged()),
87             this, SLOT(setupFilterSizes()));
88
89     // When processor image color channels changes we need to update filters
90     // available sliders and spin boxes
```

```
91      connect(processor, SIGNAL(imageColorsChanged()),
92              this, SLOT(setupFiltersAvailability()));
93
94      // Connects UI requests to capture
95      connect(this, SIGNAL(sizeChanged(const QSize &)),
96              capture, SLOT(setSize(const QSize &)), Qt::DirectConnection);
97      connect(this, SIGNAL(deviceChanged(int,uint,uint)),
98              capture, SLOT(open(int,uint,uint)), Qt::DirectConnection);
99      connect(this, SIGNAL(fileChanged(QString,uint,uint)),
100             capture, SLOT(open(QString,uint,uint)), Qt::DirectConnection);
101     connect(this, SIGNAL(flipChanged(bool)),
102             capture, SLOT(setFlipVideo(bool)), Qt::DirectConnection);
103     connect(this, SIGNAL(grayChanged(bool)),
104             capture, SLOT(setGray(bool)), Qt::DirectConnection);
105
106     // ----------------------------------------------------------------------
107     // UI setup according to capture options
108     // ----------------------------------------------------------------------
109     // Sets size radioButton states
110     if (capture->isResized())
111     {
112         /*
113          * Initial Size radio buttons configuration
114          */
115         ui->radioButtonOrigSize->setChecked(false);
116         ui->radioButtonCustomSize->setChecked(true);
117         /*
118          * Initial Size menu items configuration
119          */
120         ui->actionOriginalSize->setChecked(false);
121         ui->actionConstrainedSize->setChecked(true);
122
123         QSize size = capture->getSize();
124         qDebug("Capture->size is %dx%d", size.width(), size.height());
125         preferredWidth = size.width();
126         preferredHeight = size.height();
127     }
128     else
129     {
130         /*
131          * Initial Size radio buttons configuration
132          */
133         ui->radioButtonCustomSize->setChecked(false);
134         ui->radioButtonOrigSize->setChecked(true);
135
136         /*
137          * Initial Size menu items configuration
138          */
139         ui->actionConstrainedSize->setChecked(false);
140         ui->actionOriginalSize->setChecked(true);
141     }
142
143     // Sets spinboxes preferred size
144     ui->spinBoxWidth->setValue(preferredWidth);
145     ui->spinBoxHeight->setValue(preferredHeight);
146
147     // Sets flipCheckbox and menu item states
148     bool flipped = capture->isFlipVideo();
149     ui->actionFlip->setChecked(flipped);
150     ui->checkBoxFlip->setChecked(flipped);
151
152     // Sets grayCheckbox and menu item states
153     bool gray = capture->isGray();
154     ui->actionGray->setChecked(gray);
155     ui->checkBoxGray->setChecked(gray);
156
157     // ----------------------------------------------------------------------
158     // UI setup according to DFTProcessor options
159     // ----------------------------------------------------------------------
160     // Setting up log scale spinbox value and boundaries
161     ui->spinBoxMag->setValue((int)processor->getLogScaleFactor());
162     ui->spinBoxMag->setMinimum((int)processor->minLogScaleFactor);
163     ui->spinBoxMag->setMaximum((int)processor->maxLogScaleFactor);
164
165     // Setting up filtering checkbox
166     ui->checkBoxFiltering->setChecked(processor->isFiltering());
167
168     // Setting up filtering type
169     CvDFT::FilterType type = processor->getFilterType();
170     switch (type)
171     {
172         case CvDFT::BOX_FILTER:
173             ui->radioButtonFilterBox->setChecked(true);
174             break;
175         case CvDFT::GAUSS_FILTER:
176             ui->radioButtonFilterGauss->setChecked(true);
177             break;
178         case CvDFT::SINC_FILTER:
179             ui->radioButtonFilterSinc->setChecked(true);
180             break;
```



```
181         default:
182             break;
183     }
184 }
185
186 /*
187  * MainWindow destructor
188  */
189 MainWindow::~MainWindow()
190 {
191     delete ui;
192 }
193
194 /*
195  * Menu action when Sources->camera 0 is selected
196  * Sets capture to open device 0. If device is not available
197  * menu item is set to inactive.
198  */
199 void MainWindow::on_actionCamera_0_triggered()
200 {
201     int width = 0;
202     int height = 0;
203
204     if (ui->radioButtonCustomSize->isChecked())
205     {
206         width = preferredWidth;
207         height = preferredHeight;
208     }
209
210     qDebug("Opening device 0 ...");
211 //  if (!capture->open(0, width, height))
212 //  {
213 //      qWarning("Unable to open device 0");
214 //      // disable menu item if camera 0 does not exist
215 //      ui->actionCamera_0->setDisabled(true);
216 //  }
217     emit deviceChanged(0, width, height);
218 }
219
220 /*
221  * Menu action when Sources->camera 1 is selected
222  * Sets capture to open device 0. If device is not available
223  * menu item is set to inactive
224  */
225 void MainWindow::on_actionCamera_1_triggered()
226 {
227     int width = 0;
228     int height = 0;
229
230     if (ui->radioButtonCustomSize->isChecked())
231     {
232         width = preferredWidth;
233         height = preferredHeight;
234     }
235
236     qDebug("Opening device 1 ...");
237 //  if (!capture->open(1, width, height))
238 //  {
239 //      qWarning("Unable to open device 1");
240 //      // disable menu item if camera 1 does not exist
241 //      ui->actionCamera_1->setDisabled(true);
242 //  }
243     emit deviceChanged(1, width, height);
244 }
245
246 /*
247  * Menu action when Sources->file is selected.
248  * Opens file dialog and tries to open selected file (is not empty),
249  * then sets capture to open the selected file
250  */
251 void MainWindow::on_actionFile_triggered()
252 {
253     int width = 0;
254     int height = 0;
255
256     if (ui->radioButtonCustomSize->isChecked())
257     {
258         width = preferredWidth;
259         height = preferredHeight;
260     }
261
262     QString fileName =
263             QFileDialog::getOpenFileName(this,
264                                          tr("Open Video"),
265                                          "./",
266                                          tr("Video Files (*.avi *.mkv *.mp4 *.m4v)"),
267                                          NULL,
268                                          QFileDialog::ReadOnly);
269
270     qDebug("Opening file %s ...", fileName.toStdString().c_str());
```

```
271
272     if (fileName.length() > 0)
273     {
274 //      if (!capture->open(fileName))
275 //      {
276 //          qWarning("Unable to open device file : %s",
277 //                   fileName.toStdString().c_str());
278 //      }
279 //      // setupProcessorFromUI(); // already done from connection
280         emit fileChanged(fileName, width, height);
281     }
282     else
283     {
284         qWarning("empty file name");
285     }
286 }
287
288 /*
289  * Menu action to qui application
290  */
291 void MainWindow::on_actionQuit_triggered()
292 {
293     this->close();
294 }
295
296 /*
297  * Menu action when flip image is selected.
298  * Sets capture to change flip status which leads to reverse
299  * image horizontally
300  */
301 void MainWindow::on_actionFlip_triggered()
302 {
303     emit flipChanged(¬capture->isFlipVideo());
304     /*
305      * There is no need to update ui->checkBoxFlip since it is connected
306      * to ui->actionFlip through signals/slots
307      */
308 }
309
310 /*
311  * Menu action when gray image is selected.
312  * Sets capture to change gray status which leads convert captured image
313  * to gray or not
314  */
315 void MainWindow::on_actionGray_triggered()
316 {
317     bool isGray = ¬capture->isGray();
318
319     emit grayChanged(isGray);
320 }
321
322 /*
323  * Menu action when original image size is selected.
324  * Sets capture not to resize image
325  */
326 void MainWindow::on_actionOriginalSize_triggered()
327 {
328
329     ui->actionConstrainedSize->setChecked(false);
330
331     emit sizeChanged(QSize(0, 0));
332 }
333
334 /*
335  * Menu action when constrained image size is selected.
336  * Sets capture resize to preferred width and height
337  */
338 void MainWindow::on_actionConstrainedSize_triggered()
339 {
340     ui->actionOriginalSize->setChecked(false);
341
342     emit sizeChanged(QSize(preferredWidth, preferredHeight));
343 }
344
345 /*
346  * Changes widgetImage nature according to desired rendering mode.
347  * Possible values for mode are:
348  *  - IMAGE: widgetImage is assigned to a QcvMatWidgetImage instance
349  *  - PIXMAP: widgetImage is assigned to a QcvMatWidgetLabel instance
350  *  - GL: widgetImage is assigned to a QcvMatWidgetGL instance
351  * @param mode
352  */
353 void MainWindow::setRenderingMode(const RenderMode mode)
354 {
355     // Disconnect signals from slots first
356     disconnect(processor, SIGNAL(updated()),
357                ui->sourceImage, SLOT(update()));
358     disconnect(processor, SIGNAL(updated()),
359                ui->spectrumImage, SLOT(update()));
360     disconnect(processor, SIGNAL(updated()),
```



```
361                ui->inverseImage, SLOT(update()));
362
363     disconnect(processor, SIGNAL(squareImageChanged(Mat*)),
364                ui->sourceImage, SLOT(setSourceImage(Mat*)));
365     disconnect(processor, SIGNAL(spectrumImageChanged(Mat*)),
366                ui->spectrumImage, SLOT(setSourceImage(Mat*)));
367     disconnect(processor, SIGNAL(inverseImageChanged(Mat*)),
368                ui->inverseImage, SLOT(setSourceImage(Mat*)));
369
370     QWindow * currentWindow = windowHandle();
371     if (mode ≡ RENDER_GL)
372     {
373         disconnect(currentWindow,
374                    SIGNAL(screenChanged(QScreen *)),
375                    ui->sourceImage,
376                    SLOT(screenChanged()));
377         disconnect(currentWindow,
378                    SIGNAL(screenChanged(QScreen*)),
379                    ui->spectrumImage,
380                    SLOT(screenChanged()));
381         disconnect(currentWindow,
382                    SIGNAL(screenChanged(QScreen*)),
383                    ui->inverseImage,
384                    SLOT(screenChanged()));
385     }
386
387     // remove widgets in scroll areas
388     QWidget * wSource = ui->scrollAreaSource->takeWidget();
389     QWidget * wSpectrum = ui->scrollAreaSpectrum->takeWidget();
390     QWidget * wInverse = ui->scrollAreaInverse->takeWidget();
391
392     if ((wSource ≡ ui->sourceImage) ∧
393         (wSpectrum ≡ ui->spectrumImage) ∧
394         (wInverse ≡ ui->inverseImage))
395     {
396         // delete removed widgets
397         delete ui->sourceImage;
398         delete ui->spectrumImage;
399         delete ui->inverseImage;
400
401         // create new widget
402         Mat * sourceMat = processor->getImagePtr("square");
403         Mat * spectrumMat = processor->getImagePtr("spectrum");
404         Mat * inverseMat = processor->getImagePtr("inverse");
405
406         switch (mode)
407         {
408             case RENDER_PIXMAP:
409                 ui->sourceImage = new QcvMatWidgetLabel(sourceMat);
410                 ui->spectrumImage = new QcvMatWidgetLabel(spectrumMat);
411                 ui->inverseImage = new QcvMatWidgetLabel(inverseMat);
412                 break;
413             case RENDER_GL:
414                 ui->sourceImage = new QcvMatWidgetGL(sourceMat);
415                 ui->spectrumImage = new QcvMatWidgetGL(spectrumMat);
416                 ui->inverseImage = new QcvMatWidgetGL(inverseMat);
417                 break;
418             case RENDER_IMAGE:
419             default:
420                 ui->sourceImage = new QcvMatWidgetImage(sourceMat);
421                 ui->spectrumImage = new QcvMatWidgetImage(spectrumMat);
422                 ui->inverseImage = new QcvMatWidgetImage(inverseMat);
423                 break;
424         }
425
426         if ((ui->sourceImage ≠ NULL) ∧
427             (ui->spectrumImage ≠ NULL) ∧
428             (ui->inverseImage ≠ NULL))
429         {
430             // Name the new images widgets with same name as in UI files
431             ui->sourceImage->setObjectName(QString::fromUtf8("sourceImage"));
432             ui->spectrumImage->setObjectName(QString::fromUtf8("spectrumImage"));
433             ui->inverseImage->setObjectName(QString::fromUtf8("inverseImage"));
434
435             // add to scroll areas
436             ui->scrollAreaSource->setWidget(ui->sourceImage);
437             ui->scrollAreaSpectrum->setWidget(ui->spectrumImage);
438             ui->scrollAreaInverse->setWidget(ui->inverseImage);
439
440             // Reconnect signals to slots
441             connect(processor, SIGNAL(updated()),
442                     ui->sourceImage, SLOT(update()));
443             connect(processor, SIGNAL(updated()),
444                     ui->spectrumImage, SLOT(update()));
445             connect(processor, SIGNAL(updated()),
446                     ui->inverseImage, SLOT(update()));
447
448             connect(processor, SIGNAL(squareImageChanged(Mat*)),
449                     ui->sourceImage, SLOT(setSourceImage(Mat*)));
450             connect(processor, SIGNAL(spectrumImageChanged(Mat*)),
```

```
451                     ui→spectrumImage, SLOT(setSourceImage(Mat*)));
452             connect(processor, SIGNAL(inverseImageChanged(Mat*)),
453                     ui→inverseImage, SLOT(setSourceImage(Mat*)));
454 
455             if (mode == RENDER_GL)
456             {
457                 connect(currentWindow,
458                         SIGNAL(screenChanged(QScreen *)),
459                         ui→sourceImage,
460                         SLOT(screenChanged()));
461                 connect(currentWindow,
462                         SIGNAL(screenChanged(QScreen *)),
463                         ui→spectrumImage,
464                         SLOT(screenChanged()));
465                 connect(currentWindow,
466                         SIGNAL(screenChanged(QScreen *)),
467                         ui→inverseImage,
468                         SLOT(screenChanged()));
469             }
470 
471             // Sends message to status bar and sets menu checks
472             message.clear();
473             message.append(tr("Render more set to "));
474             switch (mode)
475             {
476                 case RENDER_IMAGE:
477                     ui→actionRenderPixmap→setChecked(false);
478                     ui→actionRenderOpenGL→setChecked(false);
479                     message.append(tr("QImage"));
480                     break;
481                 case RENDER_PIXMAP:
482                     ui→actionRenderImage→setChecked(false);
483                     ui→actionRenderOpenGL→setChecked(false);
484                     message.append(tr("QPixmap in QLabel"));
485                     break;
486                 case RENDER_GL:
487                     ui→actionRenderImage→setChecked(false);
488                     ui→actionRenderPixmap→setChecked(false);
489                     message.append(tr("QGLWidget"));
490                     break;
491                 default:
492                 break;
493             }
494             emit sendMessage(message, 5000);
495         }
496         else
497         {
498             qDebug("MainWindow::on_actionRenderXXX some new widget is null");
499         }
500     }
501     else
502     {
503         qDebug("MainWindow::on_actionRenderXXX removed widget is not in ui->");
504     }
505 }
506 
507 /*
508  * Re setup processor from UI settings when source changes
509  */
510 void MainWindow::setupProcessorFromUI()
511 {
512     processor→setLogScaleFactor((double)ui→spinBoxMag→value());
513 
514     if (ui→radioButtonFilterBox→isChecked())
515     {
516         processor→setFilterType(CvDFT::BOX_FILTER);
517     }
518     else if (ui→radioButtonFilterGauss→isChecked())
519     {
520         processor→setFilterType(CvDFT::GAUSS_FILTER);
521     }
522     else
523     {
524         processor→setFilterType(CvDFT::SINC_FILTER);
525     }
526 
527     processor→setFiltering(ui→checkBoxFiltering→isChecked());
528 
529     processor→setLowPassFilterSize(CvDFT::BLUE, ui→spinBoxBlueLP→value());
530     processor→setLowPassFilterSize(CvDFT::GREEN, ui→spinBoxGreenLP→value());
531     processor→setLowPassFilterSize(CvDFT::RED, ui→spinBoxRedLP→value());
532 
533     processor→setHighPassFilterSize(CvDFT::BLUE, ui→spinBoxBlueHP→value());
534     processor→setHighPassFilterSize(CvDFT::GREEN, ui→spinBoxGreenHP→value());
535     processor→setHighPassFilterSize(CvDFT::RED, ui→spinBoxRedHP→value());
536 }
537 
538 /*
539  * Setup filter min/max and evt values according to source image size.
540  * Filter max size is \$f\frac{FFTSize}{\sqrt{2}}\$f
```



```
541  */
542 void MainWindow::setupFilterSizes()
543 {
544     qDebug("Setting up filter sizes and values");
545 
546     int maxFilterValue = (int)processor→getMaxFilterSize();
547     int minFilterValue = (int)processor→getMinFilterSize();
548     int nbChannels = processor→getNbChannels();
549 
550     // spin boxes in same order as openCV components
551     QSpinBox * lowPassSpinBoxes[3] =
552     {
553         ui→spinBoxBlueLP,
554         ui→spinBoxGreenLP,
555         ui→spinBoxRedLP,
556     };
557 
558     // sliders in same order as openCV components
559     QSlider * lowPassSliders[3] =
560     {
561         ui→horizontalSliderBLP,
562         ui→horizontalSliderGLP,
563         ui→horizontalSliderRLP
564     };
565 
566     for (int i=0; i < 3; i++)
567     {
568         lowPassSpinBoxes[i]→setMaximum(maxFilterValue);
569         lowPassSpinBoxes[i]→setMinimum(minFilterValue);
570         lowPassSliders[i]→setMaximum(maxFilterValue);
571         lowPassSliders[i]→setMinimum(minFilterValue);
572 
573         if (i < nbChannels)
574         {
575             lowPassSpinBoxes[i]→setValue(processor→getLowPassFilterSize(i));
576         }
577         else
578         {
579             lowPassSpinBoxes[i]→setValue(maxFilterValue);
580         }
581     }
582 
583     QSpinBox * highPassSpinBoxes[3] =
584     {
585         ui→spinBoxBlueHP,
586         ui→spinBoxGreenHP,
587         ui→spinBoxRedHP
588     };
589 
590     QSlider * highPassSliders[3] =
591     {
592         ui→horizontalSliderBHP,
593         ui→horizontalSliderGHP,
594         ui→horizontalSliderRHP
595     };
596 
597     for (int i=0; i < 3; i++)
598     {
599         highPassSpinBoxes[i]→setMaximum(maxFilterValue);
600         highPassSpinBoxes[i]→setMinimum(minFilterValue);
601         highPassSliders[i]→setMaximum(maxFilterValue);
602         highPassSliders[i]→setMinimum(minFilterValue);
603 
604         if (i < nbChannels)
605         {
606             highPassSpinBoxes[i]→setValue(processor→getHighPassFilterSize(i));
607         }
608         else
609         {
610             highPassSpinBoxes[i]→setValue(minFilterValue);
611         }
612     }
613 }
614 
615 /*
616  * Setup filters sliders/spinboxes availability according to the number
617  * of channels in the source image
618  */
619 void MainWindow::setupFiltersAvailability()
620 {
621     int nbChannels = processor→getNbChannels();
622 
623     qDebug("Setting up filters availability with %d channels", nbChannels);
624 
625     QSpinBox * lowPassSpinBoxes[3] =
626     {
627         ui→spinBoxBlueLP,
628         ui→spinBoxGreenLP,
629         ui→spinBoxRedLP
630     };
```

```
631
632     QSlider * lowPassSliders[3] =
633     {
634         ui->horizontalSliderBLP,
635         ui->horizontalSliderGLP,
636         ui->horizontalSliderRLP
637     };
638
639     for (int i=0; i < 3; i++)
640     {
641         if (i < nbChannels)
642         {
643             lowPassSpinBoxes[i]->setEnabled(true);
644             lowPassSliders[i]->setEnabled(true);
645
646             // Re set processor filter values
647             processor->setLowPassFilterSize(i, lowPassSpinBoxes[i]->value());
648         }
649         else
650         {
651             lowPassSpinBoxes[i]->setEnabled(false);
652             lowPassSliders[i]->setEnabled(false);
653         }
654     }
655
656     QSpinBox * highPassSpinBoxes[3] =
657     {
658         ui->spinBoxBlueHP,
659         ui->spinBoxGreenHP,
660         ui->spinBoxRedHP
661     };
662
663     QSlider * highPassSliders[3] =
664     {
665         ui->horizontalSliderBHP,
666         ui->horizontalSliderGHP,
667         ui->horizontalSliderRHP
668     };
669
670     for (int i=0; i < 3; i++)
671     {
672         if (i < nbChannels)
673         {
674             highPassSpinBoxes[i]->setEnabled(true);
675             highPassSliders[i]->setEnabled(true);
676
677             // Re set processor filter values
678             processor->setHighPassFilterSize(i, highPassSpinBoxes[i]->value());
679         }
680         else
681         {
682             highPassSpinBoxes[i]->setEnabled(false);
683             highPassSliders[i]->setEnabled(false);
684         }
685     }
686
687     if (nbChannels == 1)
688     {
689         setLinkedFilterSizes(false);
690         ui->checkBoxLinkLP->setEnabled(false);
691         ui->checkBoxLinkHP->setEnabled(false);
692
693         ui->labelBlueLP->setText(tr("Gray"));
694         ui->labelBlueHP->setText(tr("Gray"));
695     }
696     else
697     {
698         ui->checkBoxLinkLP->setEnabled(true);
699         ui->checkBoxLinkHP->setEnabled(true);
700
701         ui->labelBlueLP->setText(tr("Blue"));
702         ui->labelBlueHP->setText(tr("Blue"));
703     }
704 }
705
706
707 /*
708  * Set filters spinBoxes and sliders link state
709  * @param linked the link status
710  * @post When link is on all sliders/spinboxes of low pass and high pass
711  * filters are linked together, moving/changing one changes the others.
712  * When link os off sliders/spinboxes are not linked together
713  */
714 void MainWindow::setLinkedFilterSizes(bool linked)
715 {
716     if (linked)
717     {
718         qDebug("Linking Sliders together");
719         // check all link checkboxes since we don't know which one lead here
720         ui->checkBoxLinkLP->setChecked(true);
```



```
721         ui->checkBoxLinkHP->setChecked(true);
722
723         // set blue/green values to red value
724         int redLPValue = ui->spinBoxRedLP->value();
725         ui->spinBoxGreenLP->setValue(redLPValue);
726         ui->spinBoxBlueLP->setValue(redLPValue);
727         int redHPValue = ui->spinBoxRedHP->value();
728         ui->spinBoxGreenHP->setValue(redHPValue);
729         ui->spinBoxBlueHP->setValue(redHPValue);
730
731         // link all filter sizes sliders and spinboxes together
732         // only spinboxes are affected since sliders and spinboxes
733         // arer already connected together
734
735         // red -> green
736         connect(ui->spinBoxRedLP, SIGNAL(valueChanged(int)),
737                 ui->spinBoxGreenLP, SLOT(setValue(int)));
738         connect(ui->spinBoxRedHP, SIGNAL(valueChanged(int)),
739                 ui->spinBoxGreenHP, SLOT(setValue(int)));
740
741         // red -> blue
742         connect(ui->spinBoxRedLP, SIGNAL(valueChanged(int)),
743                 ui->spinBoxBlueLP, SLOT(setValue(int)));
744         connect(ui->spinBoxRedHP, SIGNAL(valueChanged(int)),
745                 ui->spinBoxBlueHP, SLOT(setValue(int)));
746
747         // green -> red
748         connect(ui->spinBoxGreenLP, SIGNAL(valueChanged(int)),
749                 ui->spinBoxRedLP, SLOT(setValue(int)));
750         connect(ui->spinBoxGreenHP, SIGNAL(valueChanged(int)),
751                 ui->spinBoxRedHP, SLOT(setValue(int)));
752
753         // green -> blue
754         connect(ui->spinBoxGreenLP, SIGNAL(valueChanged(int)),
755                 ui->spinBoxBlueLP, SLOT(setValue(int)));
756         connect(ui->spinBoxGreenHP, SIGNAL(valueChanged(int)),
757                 ui->spinBoxBlueHP, SLOT(setValue(int)));
758
759         // blue -> red
760         connect(ui->spinBoxBlueLP, SIGNAL(valueChanged(int)),
761                 ui->spinBoxRedLP, SLOT(setValue(int)));
762         connect(ui->spinBoxBlueHP, SIGNAL(valueChanged(int)),
763                 ui->spinBoxRedHP, SLOT(setValue(int)));
764
765         // blue -> green
766         connect(ui->spinBoxBlueLP, SIGNAL(valueChanged(int)),
767                 ui->spinBoxGreenLP, SLOT(setValue(int)));
768         connect(ui->spinBoxBlueHP, SIGNAL(valueChanged(int)),
769                 ui->spinBoxGreenHP, SLOT(setValue(int)));
770     }
771     else
772     {
773         qDebug("Unlink sliders from each other");
774         // uncheck all link checkboxes since we don't know which one lead here
775         ui->checkBoxLinkLP->setChecked(false);
776         ui->checkBoxLinkHP->setChecked(false);
777
778         // unlink all filter sizes sliders and spinboxes from each other
779         // only spinboxes are affected since sliders and spinboxes
780         // arer already connected together
781
782         // red -> green
783         disconnect(ui->spinBoxRedLP, SIGNAL(valueChanged(int)),
784                    ui->spinBoxGreenLP, SLOT(setValue(int)));
785         disconnect(ui->spinBoxRedHP, SIGNAL(valueChanged(int)),
786                    ui->spinBoxGreenHP, SLOT(setValue(int)));
787
788         // red -> blue
789         disconnect(ui->spinBoxRedLP, SIGNAL(valueChanged(int)),
790                    ui->spinBoxBlueLP, SLOT(setValue(int)));
791         disconnect(ui->spinBoxRedHP, SIGNAL(valueChanged(int)),
792                    ui->spinBoxBlueHP, SLOT(setValue(int)));
793
794         // green -> red
795         disconnect(ui->spinBoxGreenLP, SIGNAL(valueChanged(int)),
796                    ui->spinBoxRedLP, SLOT(setValue(int)));
797         disconnect(ui->spinBoxGreenHP, SIGNAL(valueChanged(int)),
798                    ui->spinBoxRedHP, SLOT(setValue(int)));
799
800         // green -> blue
801         disconnect(ui->spinBoxGreenLP, SIGNAL(valueChanged(int)),
802                    ui->spinBoxBlueLP, SLOT(setValue(int)));
803         disconnect(ui->spinBoxGreenHP, SIGNAL(valueChanged(int)),
804                    ui->spinBoxBlueHP, SLOT(setValue(int)));
805
806         // blue -> red
807         disconnect(ui->spinBoxBlueLP, SIGNAL(valueChanged(int)),
808                    ui->spinBoxRedLP, SLOT(setValue(int)));
809         disconnect(ui->spinBoxBlueHP, SIGNAL(valueChanged(int)),
810                    ui->spinBoxRedHP, SLOT(setValue(int)));
```

```
811
812         // blue -> green
813         disconnect(ui->spinBoxBlueLP, SIGNAL(valueChanged(int)),
814                    ui->spinBoxGreenLP, SLOT(setValue(int)));
815         disconnect(ui->spinBoxBlueHP, SIGNAL(valueChanged(int)),
816                    ui->spinBoxGreenHP, SLOT(setValue(int)));
817     }
818 }
819
820 /*
821  * Menu action to replace current image rendering widget by a
822  * QcvMatWidgetImage instance.
823  */
824 void MainWindow::on_actionRenderImage_triggered()
825 {
826     qDebug("Setting image rendering to: images");
827     setRenderingMode(RENDER_IMAGE);
828 }
829
830 /*
831  * Menu action to replace current image rendering widget by a
832  * QcvMatWidgetLabel with pixmap instance.
833  */
834 void MainWindow::on_actionRenderPixmap_triggered()
835 {
836     qDebug("Setting image rendering to: pixmaps");
837     setRenderingMode(RENDER_PIXMAP);
838 }
839
840 /*
841  * Menu action to replace current image rendering widget by a
842  * QcvMatWidgetGL instance.
843  */
844 void MainWindow::on_actionRenderOpenGL_triggered()
845 {
846     qDebug("Setting image rendering to: opengl");
847     setRenderingMode(RENDER_GL);
848 }
849
850 /*
851  * Original size radioButton action.
852  * Sets capture resize to off
853  */
854 void MainWindow::on_radioButtonOrigSize_clicked()
855 {
856     ui->actionConstrainedSize->setChecked(false);
857     emit sizeChanged(QSize(0, 0));
858 }
859
860 /*
861  * Custom size radioButton action.
862  * Sets capture resize to preferred width and height
863  */
864 void MainWindow::on_radioButtonCustomSize_clicked()
865 {
866     ui->actionOriginalSize->setChecked(false);
867     emit sizeChanged(QSize(preferredWidth, preferredHeight));
868 }
869
870 /*
871  * Width spinbox value change.
872  * Changes the preferred width and if custom size is selected apply
873  * this custom width
874  * @param value the desired width
875  */
876 void MainWindow::on_spinBoxWidth_valueChanged(int value)
877 {
878     preferredWidth = value;
879     if (ui->radioButtonCustomSize->isChecked())
880     {
881         emit sizeChanged(QSize(preferredWidth, preferredHeight));
882     }
883 }
884
885 /*
886  * Height spinbox value change.
887  * Changes the preferred height and if custom size is selected apply
888  * this custom height
889  * @param value the desired height
890  */
891 void MainWindow::on_spinBoxHeight_valueChanged(int value)
892 {
893     preferredHeight = value;
894     if (ui->radioButtonCustomSize->isChecked())
895     {
896         emit sizeChanged(QSize(preferredWidth, preferredHeight));
897     }
898 }
899
900 /*
```



```
901  * Flip capture image horizontally.
902  * changes capture flip status
903  */
904 void MainWindow::on_checkBoxFlip_clicked()
905 {
906     /*
907      * There is no need to update ui->actionFlip since it is connected
908      * to ui->checkBoxFlip through signals/slots
909      */
910     emit flipChanged(ui->checkBoxFlip->isChecked());
911 }
912
913 /*
914  * convert capture image to gray level.
915  * changes cpature gray conversion status
916  */
917 void MainWindow::on_checkBoxGray_clicked()
918 {
919     bool isGray = ui->checkBoxGray->isChecked();
920     emit grayChanged(isGray);
921 }
922
923 /*
924  * Changes logscale factor for spectrum
925  * @param value the new logscale factor
926  */
927 void MainWindow::on_spinBoxMag_valueChanged(int value)
928 {
929     processor->setLogScaleFactor((double)value);
930
931     double realScale = processor->getLogScaleFactor();
932
933     ui->spinBoxMag->setValue((int)realScale);
934 }
935
936 /*
937  * Sets filtering on/off
938  */
939 void MainWindow::on_checkBoxFiltering_clicked()
940 {
941     processor->setFiltering(ui->checkBoxFiltering->isChecked());
942 }
943
944 /*
945  * Sets Filter mode to box
946  */
947 void MainWindow::on_radioButtonFilterBox_clicked()
948 {
949     processor->setFilterType(CvDFT::BOX_FILTER);
950 }
951
952 /*
953  * Sets filter mode to gaussian
954  */
955 void MainWindow::on_radioButtonFilterGauss_clicked()
956 {
957     processor->setFilterType(CvDFT::GAUSS_FILTER);
958 }
959
960 /*
961  * Sets filter mode to sinus cardinal
962  */
963 void MainWindow::on_radioButtonFilterSinc_clicked()
964 {
965     processor->setFilterType(CvDFT::SINC_FILTER);
966 }
967
968 /*
969  * Changes low pass filter size for red component
970  * @param value the new low pass filter size for red component
971  * @note low pass filters are lower bounded by their respective high
972  * pass filters
973  */
974 void MainWindow::on_spinBoxRedLP_valueChanged(int value)
975 {
976     processor->setLowPassFilterSize(CvDFT::RED, value);
977     // low pass filter might be lower bounded by high pass filter
978     ui->spinBoxRedLP->setValue(processor->getLowPassFilterSize(CvDFT::RED));
979 }
980
981 /*
982  * Changes low pass filter size for green component
983  * @param value the new low pass filter size for green component
984  * @note low pass filters are lower bounded by their respective high
985  * pass filters
986  */
987 void MainWindow::on_spinBoxGreenLP_valueChanged(int value)
988 {
989     processor->setLowPassFilterSize(CvDFT::GREEN, value);
990     // low pass filter might be lower bounded by high pass filter
```

```
    ui->spinBoxGreenLP->setValue(processor->getLowPassFilterSize(CvDFT::GREEN));
}

/*
 * Changes low pass filter size for blue component
 * @param value the new low pass filter size for blue component
 * @note low pass filters are lower bounded by their respective high
 * pass filters
 */
void MainWindow::on_spinBoxBlueLP_valueChanged(int value)
{
    processor->setLowPassFilterSize(CvDFT::BLUE, value);
    // low pass filter might be lower bounded by high pass filter
    ui->spinBoxBlueLP->setValue(processor->getLowPassFilterSize(CvDFT::BLUE));
}

/*
 * Changes high pass filter size for red component
 * @param value the new high pass filter size for red component
 * @note high pass filters are upper bounded by their respective low
 * pass filters
 */
void MainWindow::on_spinBoxRedHP_valueChanged(int value)
{
    processor->setHighPassFilterSize(CvDFT::RED, value);
    // high pass filter might be upper bounded by low pass filter
    ui->spinBoxRedHP->setValue(processor->getHighPassFilterSize(CvDFT::RED));
}

/*
 * Changes high pass filter size for green component
 * @param value the new high pass filter size for green component
 * @note high pass filters are upper bounded by their respective low
 * pass filters
 */
void MainWindow::on_spinBoxGreenHP_valueChanged(int value)
{
    processor->setHighPassFilterSize(CvDFT::GREEN, value);
    // high pass filter might be upper bounded by low pass filter
    ui->spinBoxGreenHP->setValue(processor->getHighPassFilterSize(CvDFT::GREEN));
}

/*
 * Changes high pass filter size for blue component
 * @param value the new high pass filter size for blue component
 * @note high pass filters are upper bounded by their respective low
 * pass filters
 */
void MainWindow::on_spinBoxBlueHP_valueChanged(int value)
{
    processor->setHighPassFilterSize(CvDFT::BLUE, value);
    // high pass filter might be upper bounded by low pass filter
    ui->spinBoxBlueHP->setValue(processor->getHighPassFilterSize(CvDFT::BLUE));
}

/*
 * Changes spinboxes/sliders link mode from low pass pane
 */
void MainWindow::on_checkBoxLinkLP_clicked()
{
    setLinkedFilterSizes(ui->checkBoxLinkLP->isChecked());
}

/*
 * Changes spinboxes/sliders link mode from low pass pane
 */
void MainWindow::on_checkBoxLinkHP_clicked()
{
    setLinkedFilterSizes(ui->checkBoxLinkHP->isChecked());
}
```



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

using namespace std;

#include "QcvVideoCapture.h"
#include "CaptureFactory.h"
#include "QcvDFT.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 + QT5
 * @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[])
{
    CvProcessor::VerboseLevel verboseLevel = CvProcessor::VERBOSE_WARNINGS; // verbose up to notif
    // CvProcessor::VerboseLevel verboseLevel = CvProcessor::VERBOSE_ACTIVITY; // verbose all

    // -----------------------------------------------------------------------
    // 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");
            }
        }
        else if (currentArg == "-v" || currentArg == "--verbose")
        {
            // next argument should be a verbose level (from 0 to 4)
            if (it.hasNext())
            {
                QString verboseLevelString(it.next());
                bool convertOk;
                int newVerboseLevel = verboseLevelString.toUInt(&convertOk, 10);
                if (!convertOk ||
                    newVerboseLevel < 0 ||
                    newVerboseLevel > (int)CvProcessor::NBVERBOSELEVEL)
                {
                    qWarning("Invalid verbose level %d", newVerboseLevel);
                }
                else
                {
                    verboseLevel = (CvProcessor::VerboseLevel)newVerboseLevel;
                }
            }
            else
            {
                // by default set it to max verbose
```

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



```
181
182     // ----------------------------------------------------------------------
183     // Cleanup & return
184     // ----------------------------------------------------------------------
185     delete processor;
186     delete capture;
187
188     bool sameThread = capThread ≡ procThread;
189
190     if (capThread ≠ NULL)
191     {
192         delete capThread;
193     }
194
195     if (procThread ≠ NULL ∧ ¬sameThread)
196     {
197         delete procThread;
198     }
199
200     return retVal;
201 }
202
203 /*
204  * Usage function shown just before launching QApp
205  * @param name the name of the program (argv[0])
206  */
207 void usage(char * name)
208 {
209     cout << "usage : " << basename(name) << " "
210          << "[-d | --device] <device number> "
211          << "[-v | --video] <video file> "
212          << "[-s | --size] <width>x<height> "
213          << "[-m | --mirror]"
214          << "[-t | --threads] <number of threads>"
215          << endl;
216 }
```