```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;
}
```

```cpp
/*
 * CvFloodFill.h
 *
 *  Created on: 22 mars 2012
 *      Author: davidroussel
 */

#ifndef CVGFLOODFILL_H_
#define CVGFLOODFILL_H_

#include "CvProcessor.h"

/**
 * Class to process source image with gaussian filters
 */
class CvFloodFill: virtual public CvProcessor
{
    public:
        /**
         * Image Display type
         */
        typedef enum
        {
            INPUT_IM = 0, //!< Input image
            MASK_IM, //!< Flood fill mask
            MERGED_IM, //!< Mixes Input dans flood fill image
            NBDISPLAY_IM //!< Number of elements in this enum
        } ImageDisplay;

        /**
         * Flood Fill mode
         */
        typedef enum
        {
            /**
             * Fixed range, lower and upper threshold for pixel aggregation criteria
             * are absolute
             */
            FIXED_RANGE = 1,
            /**
             * Floating range, lower and upper threshold of pixel aggregation criteria
             * are floating (absolute compared to the value of the current pixel to
             * aggregate neighbors with).
             */
            FLOATING_RANGE = 2,
            NBFILLING_MODES
        } FloodFillMode;

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

        /**
         * Image displayed
         */
        Mat displayImage;

        /**
         * True when display image changed since last update
         */
        bool displayImageChanged;

        /**
         * Mask image
         */
        Mat mask;

        /**
         * Merged Source and flood fill image
         */
        Mat merged;

        /**
         * Image display mode.
         */
        ImageDisplay displayMode;

        // ------------------------------------------------------------------------
        // Flood parameters
        // ------------------------------------------------------------------------
        /**
         * indicates a manual seed has been provided and initialSeed can be
         * used to flood the image.
         * seeded can be reset to false in order to cancel current flood.
         * @warning reset seeded to false should also cause flooded to be
         * reset to false.
         * [initial value is false]
         */
        bool seeded;

        /**
         * Indicates image has been flooded at least once and barycenter
         * of the flooded area (centerSeed) can be used to seed the flood of
         * the next image.
         * flooded is reset when a new manual seed is provided
         * [initial value is false]
         */
        bool flooded;

        /**
         * flooding fill mode [default value is 1]
         *  - 1: Fixed range flood fill mode
         *  - 2: Gradient (floating range) floodfill mode
         */
        FloodFillMode ffillMode;

        /**
         * Lower difference for pixel aggregation [default value is 20]
         */
        int loDiff;

        /**
         * upper difference for pixel aggregation [default value is 20]
         */
        int upDiff;

        /**
         * pixel connectivity for flood fill [default value is 4]
         */
        int connectivity;

        /**
         * Is source image color ?  [default value is true]
         */
        bool isColor;

        /**
         * New mask value (old mask values are thresholded to 128)
         *  [default value is 255]
         */
        int newMaskVal;

        /**
         * Flood flags computed from conenctivity, newMaskVal and ffillMode such
         * that
         * @code
         *  flags = connectivity + (newMaskVal << 8) +
         *  (ffillMode == 1 ? CV_FLOODFILL_FIXED_RANGE : 0);
         * @endcode
         */
        int floodFlags;

        /**
         * Number of pixels in the flooded area
         */
        int floodArea;

        /**
         * The initial seed for flood fill obtained from user click
         *  [default value is (-1, -1) when initial seed is not set]
         */
        Point initialSeed;

        /**
         * The seed for next image obtained from flooded area barycenter
         * [default value is (-1, -1) when initial seed is not set]
         */
        Point centerSeed;

        /**
         * Show/Hides seed point in source image and merged image
         */
        bool showSeed;

        /**
         * Flood color to use in the merged image
         */
        Scalar floodColor;

        /**
         * Flooded area bounding box
         */
        Rect floodBoundingBox;

        /**
```

```
181         * Show/Hides flooded area bounding box in source image
182         */
183        bool showBoundingBox;
184
185        // ------------------------------------------------------------------
186        // Utility methods
187        // ------------------------------------------------------------------
188        /**
189         * Setup attributes when source image is changed
190         * @param image source Image
191         * @param completeSetup is true when used to change source image,
192         * and false when used in constructor
193         */
194        virtual void setup(Mat *image, bool completeSetup);
195
196        /**
197         * Cleanup attributes before changing source image or cleaning class
198         * before destruction
199         */
200        virtual void cleanup();
201
202        /**
203         * Compute barycenter of last flooded mask
204         * @param mask the mask image where mask pixels are set to newMaskVal
205         * @param center the barycenter point computed here
206         * @param threshold the value to use as threshold to find flooded area
207         * pixels in the mask image
208         * @note One can also use the OpenCV function moments(...) to compute
209         * all moments up to the third order but we only need m00 (number of
210         * pixels of the flooded area) and m01 & m10 to compute flooded area
211         * center = (m10/m00, m01/m00).
212         * @return true if there is some flooded pixels in the mask to compute
213         * a barycenter, false otherwise.
214         * @note we need to know if flooding has failed as we should reset
215         * seeded and flooded variables accordingly
216         */
217        template<typename T>
218        bool computeFloodCenter(const Mat & mask,
219                                Point & center,
220                                const T threshold = numeric_limits<T>::max());
221    public:
222        /**
223         * Flood fill class constructor
224         * @param sourceImage
225         */
226        CvFloodFill(Mat *sourceImage);
227
228        /**
229         * Flood fill class destructor
230         */
231        virtual ~CvFloodFill();
232
233        /**
234         * Flood fill update:
235         *  - Copy source image to merged image
236         *  - if image has already been flooded compute Flood barycenter
237         *      - if flood has succeded set seed as the barycenter
238         *      - else reset seeded and flooded states
239         *  - else
240         *      - if image has been seed manually then use this seed
241         *  - clears mask with zeros
242         *  - if there is a seed
243         *      - flood fill the image
244         *      - if flood area counts some pixels then sets flooded state
245         *      - if flooded
246         *          - if show bounding box is on then draw bouding box rectangle
247         *            in source image
248         *          - if show seed is on then draw seed in source image and
249         *            merged image
250         *  - according to displayMode set displayImage
251         *
252         */
253        virtual void update();
254
255        // ------------------------------------------------------------------
256        // ImageDisplay image related methods
257        // ------------------------------------------------------------------
258 //       /**
259 //        * Gets displayImageChanged current status when display image is changed
260 //        * @return the current displayImageChanged value
261 //        */
262 //       bool isDisplayImageChanged();
263
264        /**
265         * Get current display mode
266         * @return the current display mode
267         */
268        ImageDisplay getDisplayMode() const;
269
270        /**
```



```
271         * Sets a new display mode
272         * @param displayMode the new display mode to set
273         */
274        virtual void setDisplayMode(const ImageDisplay displayMode);
275
276        /**
277         * Gets Image reference corresponding to the current displayMode and
278         * edgeMode
279         * @return Image reference corresponding to the current displayMode and
280         * edgeMode
281         */
282        const Mat & getDisplayImage() const;
283
284        /**
285         * Gets Image pointer corresponding to the current displayMode and
286         * edgeMode
287         * @return Image reference corresponding to the current displayMode and
288         * edgeMode
289         */
290        Mat * getDisplayImagePtr();
291
292        // ------------------------------------------------------------------
293        // Flood related methods
294        // ------------------------------------------------------------------
295        /**
296         * Clears flood and reset seede and flooded values to false;
297         */
298        void clearFlood();
299
300        /**
301         * Gets the seeded status of the image
302         * @return true if there is a seed, false otherwise
303         */
304        bool isSeeded() const;
305
306        /**
307         * Gets the flooded status of the image
308         * @return true if image has been flooded, false otherwise
309         */
310        bool isFlooded() const;
311
312        /**
313         * Gets the current flooding mode (absolute or relative)
314         * @return the current floodin mode
315         */
316        FloodFillMode getFfillMode() const;
317
318        /**
319         * Sets a new flooding mode (absolute or relative
320         * @param ffillMode the new flooding mode
321         */
322        virtual void setFfillMode(const FloodFillMode ffillMode);
323
324        /**
325         * Gets the lower difference in pixel values for flooding
326         * @return the current lower difference for flooding
327         */
328        int getLoDiff() const;
329
330        /**
331         * Sets a new lower difference in pixels values for flooding
332         * @param loDiff the new lower difference for flooding
333         */
334        virtual void setLoDiff(const int loDiff);
335
336        /**
337         * Gets loDiff pointer.
338         * OpenCV trackbars require direct access to values
339         * @return the address of loDiff attribute
340         */
341        int * getLoDiffPtr();
342
343        /**
344         * Gets the upper difference in pixel values for flooding
345         * @return the current upper difference for flooding
346         */
347        int getUpDiff() const;
348
349        /**
350         * Sets a new upper difference in pixels values for flooding
351         * @param upDiff the new upper difference for flooding
352         */
353        virtual void setUpDiff(const int upDiff);
354
355        /**
356         * Gets upDiff pointer.
357         * OpenCV trackbars require direct access to values
358         * @return the address of upDiff attribute
359         */
360        int * getUpDiffPtr();
```

```
361
362         /**
363          * Gets the current connectivity for pixels neighbors for flooding
364          * @return the current connectivity for pixels neighbors for flooding
365          */
366         int getConnectivity() const;
367
368         /**
369          * Sets a new connectivity for pixels neighbors for flooding
370          * @param connectivity the new connectivity for pixels neighbors
371          * for flooding
372          */
373         virtual void setConnectivity(const int connectivity);
374
375         /**
376          * Gets the current color status of the source image
377          * @return the current color status of the source image
378          */
379         bool getIsColor() const;
380
381         /**
382          * Gets the current initial seed
383          * @return the current initial seed
384          * @note use isSeeded to check if there is an initial seed
385          */
386         const Point & getInitialSeed() const;
387
388         /**
389          * Sets an new initial seed
390          * @param initialSeed the new initial seed
391          */
392         virtual void setInitialSeed(const Point & initialSeed);
393
394         /**
395          * Gets the current flooded area center for future seed
396          * @return the current flooded area center
397          * @note use isFlooded to check image has been flooded
398          */
399         const Point & getCenterSeed() const;
400
401         /**
402          * Gets current show/hide seed point status
403          * @return the current show/hide seed point status
404          */
405         bool isShowSeed() const;
406
407         /**
408          * Sets new show/hide seed point status
409          * @param showSeed the new show/hide seed point status
410          */
411         virtual void setShowSeed(const bool showSeed);
412
413         /**
414          * Generates a new random color for floodColor
415          */
416         void newFloodColor();
417
418         /**
419          * Gets the current flooded area bounding box
420          * @return the current flooded area bounding box
421          * @note use isFlooded to check image has been flooded
422          */
423         const Rect & getFloodBoundingBox() const;
424
425         /**
426          * Gets the current show/hide bounding box status
427          * @return the current show/hide bounding box status
428          */
429         bool isShowBoundingBox() const;
430
431         /**
432          * Set the show/hide bounding box status
433          * @param showBoundingBox the new show/hide bounding box status
434          */
435         virtual void setShowBoundingBox(const bool showBoundingBox);
436 };
437
438 #endif /* CVGFLOODFILL_H_ */
```



```
1  /*
2   * CvFloodFill.cpp
3   *
4   *  Created on: 26 fÃ©vr. 2012
5   *      Author: davidroussel
6   */
7
8  #include <ctime>        // for clock
9  #include <iostream>     // for cerr
10 using namespace std;
11
12 #include <opencv2/imgproc/imgproc.hpp>
13
14 #include <assert.h>
15
16 #include "CvFloodFill.h"
17
18 /*
19  * Flood fill class constructor
20  * @param sourceImage
21  */
22 CvFloodFill::CvFloodFill(Mat * sourceImage) :
23     CvProcessor(sourceImage),
24     dim(sourceImage->size()),
25     // mask is 2 pixel wider and taller
26     mask(Size(sourceImage->cols + 2, sourceImage->rows + 2), CV_8UC1),
27     merged(dim, type),
28     displayMode(INPUT_IM),
29     seeded(false),
30     flooded(false),
31     ffillMode(FIXED_RANGE),
32     loDiff(20),
33     upDiff(20),
34     connectivity(4),
35     isColor(sourceImage->channels() > 1 ? true : false),
36     newMaskVal(255),
37     floodFlags(connectivity + (newMaskVal << 8) +
38                (ffillMode == 1 ? CV_FLOODFILL_FIXED_RANGE : 0)),
39     floodArea(0),
40     initialSeed(-1, -1),
41     centerSeed(-1, -1),
42     floodBoundingBox(0,0,0,0),
43     showBoundingBox(false)
44
45 {
46     setup(sourceImage, false);
47
48     newFloodColor();
49
50     // Adds named image to additional images map
51     addImage("display", &displayImage);
52 }
53
54 /*
55  * Gaussian filtering class destructor
56  */
57 CvFloodFill::~CvFloodFill()
58 {
59     cleanup();
60 }
61
62 /*
63  * Setup attributes when source image is changed
64  * @param image source Image
65  * @param completeSetup is true when used to change source image,
66  * and false when used in constructor
67  */
68 void CvFloodFill::setup(Mat *image, bool completeSetup)
69 {
70     assert (image != NULL);
71
72     CvProcessor::setup(image, completeSetup);
73
74     if (completeSetup) // complete setup
75     {
76         dim = sourceImage->size();
77         seeded = false;
78         flooded = false;
79         isColor = (sourceImage->channels() > 1 ? true : false);
80         floodArea = 0;
81         initialSeed = Point(-1,-1);
82         centerSeed = Point(-1,-1);
83         floodBoundingBox = Rect(0,0,0,0);
84         showBoundingBox = false;
85         mask.create(Size(image->cols + 2, image->rows + 2), CV_8UC1);
86         merged.create(dim, type);
87         displayMode = INPUT_IM;
88     }
89     else // setup during constructor only
90     {
```

```
91
92      }
93
94      // in any cases
95  }
96
97  /*
98   * Cleanup attributes before changing source image or cleaning class
99   * before destruction
100  */
101 void CvFloodFill::cleanup()
102 {
103     merged.release();
104     mask.release();
105     displayImage.release();
106
107     // super cleanup
108     CvProcessor::cleanup();
109 }
110
111 /*
112  * Gets displayImageChanged current status when display image is changed
113  * @return the current displayImageChanged value
114  */
115 //bool CvFloodFill::isDisplayImageChanged()
116 //{
117 //  return displayImageChanged;
118 //}
119
120 /*
121  * Get current display mode
122  * @return the current display mode
123  */
124 CvFloodFill::ImageDisplay CvFloodFill::getDisplayMode() const
125 {
126     return displayMode;
127 }
128
129 /*
130  * Sets a new display mode
131  * @param displayMode the new display mode to set
132  */
133 void CvFloodFill::setDisplayMode(const ImageDisplay displayMode)
134 {
135     if ((displayMode ≥ INPUT_IM) ∧ (displayMode < NBDISPLAY_IM))
136     {
137         this→displayMode = displayMode;
138     }
139     else
140     {
141         cerr << "display mode out of range: " << displayMode << endl;
142     }
143 }
144
145 /*
146  * Gets Image reference corresponding to the current displayMode and
147  * edgeMode
148  * @return Image reference corresponding to the current displayMode and
149  * edgeMode
150  */
151 const Mat & CvFloodFill::getDisplayImage() const
152 {
153     return displayImage;
154 }
155
156 /*
157  * Gets Image pointer corresponding to the current displayMode and
158  * edgeMode
159  * @return Image reference corresponding to the current displayMode and
160  * edgeMode
161  */
162 Mat * CvFloodFill::getDisplayImagePtr()
163 {
164     return &displayImage;
165 }
166
167 /*
168  * Gets the seeded status of the image
169  * @return true if there is a seed, false otherwise
170  */
171 bool CvFloodFill::isSeeded() const
172 {
173     return seeded;
174 }
175
176 /*
177  * Gets the flooded status of the image
178  * @return true if image has been flooded, false otherwise
179  */
180 bool CvFloodFill::isFlooded() const
```



```
181 {
182     return flooded;
183 }
184
185 /*
186  * Gets the current flooding mode (absolute or relative)
187  * @return the current floodin mode
188  */
189 CvFloodFill::FloodFillMode CvFloodFill::getFfillMode() const
190 {
191     return ffillMode;
192 }
193
194 /*
195  * Sets a new flooding mode (absolute or relative
196  * @param ffillMode the new flooding mode
197  */
198 void CvFloodFill::setFfillMode(const FloodFillMode ffillMode)
199 {
200     if(ffillMode < NBFILLING_MODES)
201     {
202         this→ffillMode = ffillMode;
203         floodFlags = connectivity + (newMaskVal << 8) +
204                      (ffillMode ≡ FIXED_RANGE ? CV_FLOODFILL_FIXED_RANGE : 0);
205
206     }
207
208     resetMeanProcessTime();
209 }
210
211 /*
212  * Gets the lower difference in pixel values for flooding
213  * @return the current lower difference for flooding
214  */
215 int CvFloodFill::getLoDiff() const
216 {
217     return loDiff;
218 }
219
220 /*
221  * Sets a new lower difference in pixels values for flooding
222  * @param loDiff the new lower difference for flooding
223  */
224 void CvFloodFill::setLoDiff(const int loDiff)
225 {
226     if((loDiff ≥ 0) ∧ (loDiff ≤ 255))
227     {
228         this→loDiff = loDiff;
229     }
230
231     resetMeanProcessTime();
232 }
233
234 /*
235  * Gets loDiff pointer.
236  * OpenCV trackbars require direct access to values
237  * @return
238  */
239 int * CvFloodFill::getLoDiffPtr()
240 {
241     return &loDiff;
242 }
243
244 /*
245  * Gets the upper difference in pixel values for flooding
246  * @return the current upper difference for flooding
247  */
248 int CvFloodFill::getUpDiff() const
249 {
250     return upDiff;
251 }
252
253 /*
254  * Sets a new upper difference in pixels values for flooding
255  * @param upDiff the new upper difference for flooding
256  */
257 void CvFloodFill::setUpDiff(const int upDiff)
258 {
259     if((upDiff ≥ 0) ∧ (upDiff ≤ 255))
260     {
261         this→upDiff = upDiff;
262     }
263
264     resetMeanProcessTime();
265 }
266
267 /*
268  * Gets upDiff pointer.
269  * OpenCV trackbars require direct access to values
270  * @return the address of upDiff attribute
```

```
 */
int * CvFloodFill::getUpDiffPtr()
{
    return &upDiff;
}

/*
 * Gets the current connectivity for pixels neighbors for flooding
 * @return the current connectivity for pixels neighbors for flooding
 */
int CvFloodFill::getConnectivity() const
{
    return connectivity;
}

/*
 * Sets a new connectivity for pixels neighbors for flooding
 * @param connectivity the new connectivity for pixels neighbors
 * for flooding
 */
void CvFloodFill::setConnectivity(const int connectivity)
{
    if ((connectivity == 4) || (connectivity == 8))
    {
        this->connectivity = connectivity;
        floodFlags = connectivity + (newMaskVal << 8) +
                     (ffillMode == FIXED_RANGE ? CV_FLOODFILL_FIXED_RANGE : 0);
    }

    resetMeanProcessTime();
}

/*
 * Gets the current color status of the source image
 * @return the current color status of the source image
 */
bool CvFloodFill::getIsColor() const
{
    return isColor;
}

/*
 * Gets the current initial seed
 * @return the current initial seed
 * @note use isSeeded to check if there is an initial seed
 */
const Point &CvFloodFill::getInitialSeed() const
{
    return initialSeed;
}

/*
 * Sets an new initial seed
 * @param initialSeed the new initial seed
 */
void CvFloodFill::setInitialSeed(const Point & initialSeed)
{
    this->initialSeed = initialSeed;
    seeded = true;
}

/*
 * Gets the current flooded area center for future seed
 * @return the current flooded area center
 * @note use isFlooded to check image has been flooded
 */
const Point & CvFloodFill::getCenterSeed() const
{
    return centerSeed;
}

/*
 * Gets current show/hide seed point status
 * @return the current show/hide seed point status
 */
bool CvFloodFill::isShowSeed() const
{
    return showSeed;
}

/*
 * Sets new show/hide seed point status
 * @param showSeed the new show/hide seed point status
 */
void CvFloodFill::setShowSeed(const bool showSeed)
{
    this->showSeed = showSeed;
}

/*
```



```
 * Generates a new random color for floodColor
 */
void CvFloodFill::newFloodColor()
{
    // random numbers for new seed color
    int b = (unsigned)theRNG() & 255;
    int g = (unsigned)theRNG() & 255;
    int r = (unsigned)theRNG() & 255;
    floodColor = isColor ? Scalar(b, g, r) : Scalar(r*0.299 + g*0.587 + b*0.114);
}

/*
 * Gets the current flooded area bounding box
 * @return the current flooded area bounding box
 * @note use isFlooded to check image has been flooded
 */
const Rect & CvFloodFill::getFloodBoundingBox() const
{
    return floodBoundingBox;
}

/*
 * Gets the current show/hide bounding box status
 * @return the current show/hide bounding box status
 */
bool CvFloodFill::isShowBoundingBox() const
{
    return showBoundingBox;
}

/*
 * Set the show/hide bounding box status
 * @param showBoundingBox the new show/hide bounding box status
 */
void CvFloodFill::setShowBoundingBox(const bool showBoundingBox)
{
    this->showBoundingBox = showBoundingBox;
}

/*
 * Flood fill update:
 *  - Copy source image to merged image
 *  - if image has already been flooded compute Flood barycenter
 *      - if flood has succeded set seed as the barycenter
 *      - else reset seeded and flooded states
 *  - else
 *      - if image has been seed manually then use this seed
 *  - clears mask with zeros
 *  - if there is a seed
 *      - flood fill the image
 *      - if flood area counts some pixels then sets flooded state
 *      - if flooded
 *          - if show bounding box is on then draw bouding box rectangle
 *              in source image
 *          - if show seed is on then draw seed in source image and
 *              merged image
 *  - according to displayMode set displayImage
 *
 */
void CvFloodFill::update()
{
    // copy source image to merged image
    sourceImage->copyTo(merged);

    clock_t start, end;

    start = clock();

    // if image has been flooded once, then compute flood barycenter as the
    // seed for next flood
    Point seed;
    if (flooded)
    {
        // TODO ComplÃ©ter la mÃ©thode computeFloodCenter pour calculer
        // le barycentre centre de la zone inondÃ©e (flooded) Ã  partir de l'image
        // du masque qui contient des valeurs Ã  255 dans cette zone et 0 ailleurs
        bool res = computeFloodCenter<uchar>(mask, centerSeed, (uchar)newMaskVal);
        if (res)
        {
            // set seed for flooding
            seed = centerSeed;
        }
        else
        {
            seeded = false;
            flooded = false;
        }
    }
    else
    {
```

```
451         if (seeded)
452         {
453             seed = initialSeed;
454         }
455     }
456
457     // update mask with zeros : clears mask
458     mask = Scalar(0);
459
460     // flood
461     if (seeded) // We can flood the image
462     {
463         // Check seed is inside image otherwise floodFill will crash
464         seed.x ≥ 0 ? seed.x : seed.x = 0;
465         seed.x < dim.width ? seed.x : seed.x = dim.width - 1;
466         seed.y ≥ 0 ? seed.y : seed.y = 0;
467         seed.y < dim.height ? seed.y : seed.y = dim.height - 1;
468
469         // Flood image from seed point
470         // merged --> mask
471         // with seed point : seed
472         // with flood color : floodColor
473         // update flood bounding box : floodBoundingBox
474         // Lo diff set as Scalar(loDiff, loDiff, loDiff)
475         // Up diff set as Scalar(upDiff, upDiff, upDiff)
476         // use already computed floodFlags
477         // TODO floodArea = floodFill(...);
478
479         // if floodArea contains some pixels then flooded is true now
480         flooded = (floodArea > 0 ? true : false);
481
482         // if image has been flooded and showBoundingBox is true then
483         // draw flooded area bounding box in source image
484         if (flooded)
485         {
486             if (showBoundingBox) // Draws bounding box in source image with flood color
487             {
488                 int topLeftX = floodBoundingBox.x;
489                 int topleftY = floodBoundingBox.y;
490                 int boxWidth = floodBoundingBox.width;
491                 int boxHeight = floodBoundingBox.height;
492                 Point p1 = Point(topLeftX, topleftY);
493                 Point p2 = Point(topLeftX + boxWidth,
494                                  topleftY + boxHeight);
495                 rectangle(*sourceImage, // image to draw in (sourceImage)
496                           p1, // top left point
497                           p2, // bottom right point
498                           floodColor, // draw color : flood color
499                           3, // line width
500                           CV_AA); // Line Type (better with AA)
501             }
502         }
503
504         // if showSeed is true then Shows seed point in source image
505         // and merged image as a small circle with red color
506         if (showSeed)
507         {
508             circle(*sourceImage, // image to draw in
509                    seed, // center : seed points
510                    3, // radius
511                    Scalar(0, 0, 255), // draw color
512                    2, // Line width
513                    CV_AA); // Line type (better with AA)
514             circle(merged, seed, 3, Scalar(0, 0, 255), 2, CV_AA);
515         }
516     }
517
518     end = clock();
519     processTime = end - start;
520     meanProcessTime += processTime;
521
522     // ----------------------------------------------------------------------
523     // select image to display ...
524     // ----------------------------------------------------------------------
525     uchar * previousImageData = displayImage.data;
526
527     switch (displayMode)
528     {
529         case INPUT_IM:
530             displayImage = *sourceImage;
531             break;
532         case MASK_IM:
533             displayImage = mask;
534             break;
535         case MERGED_IM:
536             displayImage = merged;
537             break;
538         default:
539             if (verboseLevel ≥ CvProcessor::VERBOSE_WARNINGS)
540             {
```



```
541                 cerr << "unknown display image index " << displayMode << endl;
542             }
543             displayImage = *sourceImage;
544             break;
545     }
546
547     // Sets display image changed  status
548     // This status will be used in the QcvProcessor descendant
549     if (previousImageData ≠ displayImage.data)
550     {
551         displayImageChanged = true;
552     }
553     else
554     {
555         displayImageChanged = false;
556     }
557 }
558
559 /*
560  * Clears flood and reset seed and flooded values to false;
561  */
562 void CvFloodFill::clearFlood()
563 {
564     flooded = false;
565     seeded = false;
566     floodArea = 0;
567     initialSeed = Point(-1,-1);
568     centerSeed = Point(-1,-1);
569     floodBoundingBox = Rect(0,0,0,0);
570
571     resetMeanProcessTime();
572 }
573
574 /*
575  * Compute barycenter of last computed flood
576  * @param mask the mask image where mask pixels are set to newMaskVal
577  * @param center the barycenter point computed here
578  * @param threshold the value to use as threshold to find flooded area
579  * pixels in the mask image
580  * @note One can also use the OpenCV function moments(...) to compute
581  * all moments up to the third order but we only need m00 (number of
582  * pixels of the flooded area) and m01 & m10 to compute flooded area
583  * center = (m10/m00, m01/m00).
584  * @return true if some pixels have been flooded, false otherwise.
585  * @note we need to know if flooding has failed as we should reset
586  * seeded and flooded variables accordingly
587  */
588 template<typename T>
589 bool CvFloodFill::computeFloodCenter(const Mat & mask,
590                                      Point & center,
591                                      const T threshold)
592 {
593     // first check mask is single channel
594     if (mask.channels() ≡ 1)
595     {
596         long pixelCount = 0;     // m00
597         long lineCount = 0;      // m01
598         long colCount = 0;       // m10
599
600         for (int i = 0; i < mask.rows; i++)
601         {
602             for (int j = 0; j < mask.cols; j++)
603             {
604                 if (mask.at<T>(i,j) ≥ threshold)
605                 {
606                     // TODO ComplÃ©ter ...
607                     // update pixelCount : +1
608                     // update lineCount : +i
609                     // update colCount : +j
610                 }
611             }
612         }
613
614         if (pixelCount > 0)
615         {
616             lineCount/=pixelCount;  // m01 / m00
617             colCount/=pixelCount;   // m10 / m00
618
619             center.x = (int)colCount;
620             center.y = (int)lineCount;
621
622             return true;
623         }
624         else
625         {
626             return false;
627         }
628     }
629     else
630     {
```

```
631             cerr << "CvFloodFill::computeFloodCenter : mask image has "
632                  << mask.channels() << " channels, flood center computation aborted"
633                  << endl;
634             return false;
635         }
636 }
637
```



```
1  /*
2   * QcvProcessor.h
3   *
4   *  Created on: 19 fÃ©vr. 2012
5   *      Author: davidroussel
6   */
7
8  #ifndef QCVPROCESSOR_H_
9  #define QCVPROCESSOR_H_
10
11 #include <QObject>
12 #include <QDebug>
13 #include <QString>
14 #include <QRegExp>
15 #include <QMutex>
16 #include <QThread>
17 #include "CvProcessor.h"
18 Q_DECLARE_METATYPE(CvProcessor::ProcessTime)
19
20 /**
21  * Qt flavored class to process a source image with OpenCV 2+
22  */
23 class QcvProcessor : public QObject, public virtual CvProcessor
24 {
25     Q_OBJECT
26
27     protected:
28
29         /**
30          * Default timeout to show messages
31          */
32         static int defaultTimeOut;
33
34         /**
35          * Number format used to format numbers into QStrings
36          */
37         static QString numberFormat;
38
39         /**
40          * The regular expression used to validate new number formats
41          * @see #setNumberFormat
42          */
43         static QRegExp numberRegExp;
44
45         /**
46          * format used to format Mean/Std time values : <mean> Â± <std>
47          */
48         static QString meanStdFormat;
49
50         /**
51          * format used to format Min/Max time values : <min> / <max>
52          */
53         static QString minMaxFormat;
54
55         /**
56          * The Source image mutex in order to avoid concurrent access to
57          * the source image (typically the source image may be currently
58          * modified by the capture for instance)
59          */
60         QMutex * sourceLock;
61
62         /**
63          * the thread in which this processor should run
64          */
65         QThread * updateThread;
66
67         /**
68          * Message to send when something changes
69          */
70         QString message;
71
72         /**
73          * String used to store formatted process time value
74          */
75         QString processTimeString;
76
77         /**
78          * String used to store formatted min/max time values
79          */
80         QString processMinMaxTimeString;
81
82     public:
83
84         /**
85          * QcvProcessor constructor
86          * @param image the source image
87          * @param imageLock the mutex for concurrent access to the source image.
88          * In order to avoid concurrent access to the same image
89          * @param updateThread the thread in which this processor should run
90          * @param parent parent QObject
```

```
91          */
92         QcvProcessor(Mat * image,
93                      QMutex * imageLock = NULL,
94                      QThread * updateThread = NULL,
95                      QObject * parent = NULL);
96
97         /**
98          * QcvProcessor destructor
99          */
100        virtual ~QcvProcessor();
101
102        /**
103         * Sets new number format
104         * @param format the new number format
105         * @pre format string should look like "%8.1f" or at least not be longer
106         * than 10 chars since format is a 10 chars array.
107         * @post id format string is valid and shorter than 10 chars
108         * it has been applied as the new format string.
109         */
110        static void setNumberFormat(const char * format);
111
112        /**
113         * Get the format c-string for numbers
114         * @return the format string for numbers (e.g.: "%5.2f")
115         */
116        static const char * getNumberFormat();
117
118        /**
119         * Get the format c-string for std dev of numbers
120         * @return the format string for numbers (e.g.: " Â± %4.2f")
121         */
122        static const char * getStdFormat();
123
124        /**
125         * Get the format c-string for min / max of numbers
126         * @return the format string for numbers (e.g.: "%5.2f / %5.2f")
127         */
128        static const char * getMinMaxFormat();
129
130        /**
131         * Send to debug stream (for showing processor attributes values)
132         * @param dbg the debug stream to send to
133         * @return a reference to the output stream
134         */
135        virtual QDebug & toDBStream(QDebug & dbg) const;
136
137        /**
138         * Friend QDebug output operator
139         * @param dbg the debug stream
140         * @param proc the QcvProcessor to send to debug stream
141         * @return the debug stream
142         */
143        friend QDebug & operator <<(QDebug & dbg, const QcvProcessor & proc);
144
145    public slots:
146        /**
147         * Update computed images slot and sends updated signal
148         */
149        virtual void update();
150
151        /**
152         * Changes source image slot.
153         * Attributes needs to be cleaned up then set up again
154         * @param image the new source Image
155         * @throw CvProcessorException#NULL IMAGE when new source image is NULL
156         * @post Various signals are emitted:
157         *  - imageChanged(sourceImage)
158         *  - imageCchanged()
159         *  - if image size changed then imageSizeChanged() is emitted
160         *  - if image color space changed then imageColorsChanged() is emitted
161         */
162        virtual void setSourceImage(Mat * image) throw (CvProcessorException);
163
164        /**
165         * Sets Time per feature processing time unit (reimplemented as a slot).
166         * @param value the time per feature value (true or false)
167         */
168        virtual void setTimePerFeature(const bool value);
169
170        /**
171         * Reset mean and std process time in order to re-start computing
172         * (reimplemented as a slot)
173         * new mean and std process time values.
174         */
175        virtual void resetMeanProcessTime();
176
177    signals:
178        /**
179         * Signal emitted when update is complete
180         */
```



```
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_ */
```

```cpp
/*
 * QCvProcessor.cpp
 *
 *  Created on: 19 fÃ©vr. 2012
 *      Author: davidroussel
 */

#include <QRegExpValidator>
#include <QMetaType>
#include <QDebug>
#include "QcvProcessor.h"

/*
 * Proto instantiation of CvProcessor template method
 * Stream & CvProcessor::toStream_Impl<Stream>(Stream &) const with concrete
 * type Qdebug
 */
template QDebug & CvProcessor::toStream_Impl<QDebug>(QDebug &) const;

/*
 * Default timeout to show messages
 */
int QcvProcessor::defaultTimeOut = 5000;

/*
 * Number format used to format numbers into QStrings
 */
QString QcvProcessor::numberFormat = QString::fromUtf8("%7.0f");

/*
 * The regular expression used to validate new number formats
 * @see #setNumberFormat
 */
QRegExp QcvProcessor::numberRegExp("%[+- 0#]*[0-9]*(.][0-9]+)?[efEF]");

/*
 * format used to format Mean/Std time values : <mean> Â± <std>
 */
QString QcvProcessor::meanStdFormat = numberFormat + QString::fromUtf8(" Â± %5.0f");

/*
 * format used to format Min/Max time values : <min> / <max>
 */
QString QcvProcessor::minMaxFormat = numberFormat +  QString::fromUtf8("/") +
                                     numberFormat;

/*
 * 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::QcvProcessor(Mat * image,
                           QMutex * imageLock,
                           QThread * updateThread,
                           QObject * parent) :
    CvProcessor(image), // <-- virtual base class constructor first
    QObject(parent),
    sourceLock(imageLock),
    updateThread(updateThread),
    message(),
    processTimeString()
{
    if (updateThread ≠ NULL)
    {
        this→moveToThread(updateThread);

        connect(this, SIGNAL(finished()), updateThread, SLOT(quit()),
                Qt::DirectConnection);

        updateThread→start();
    }
}

/*
 * QcvProcessor destructor
 */
QcvProcessor::~QcvProcessor()
{
    // Lock might be already destroyed in source object so don't try to unlock

    message.clear();
    processTimeString.clear();

    emit finished();

    if (updateThread ≠ NULL)
    {
```



```cpp
        // Wait until update thread has received the "finished" signal through
        // "quit" slot
        updateThread→wait();
    }
}

/*
 * Sets new number format
 * @param format the new number format
 */
void QcvProcessor::setNumberFormat(const char * format)
{
    /*
     * The format string should validate the following regex
     * %[+- 0#]*[0-9]*([.][0-9]+)?[efEF]
     */
    QRegExpValidator validator(numberRegExp, NULL);

    QString qFormat(format);
    int pos = 0;
    if (validator.validate(qFormat,pos) ≡ QValidator::Acceptable)
    {
        numberFormat = format;
        meanStdFormat = format + QString::fromUtf8(" Â± ") + format;
        minMaxFormat = format + QString::fromUtf8("/") + format;
    }
    else
    {
        qWarning("QcvProcessor::setNumberFormat(%s) : invalid format", format);
    }
}

/*
 * Send to stream (for showing processor attributes values)
 * @param dbg the debug stream to send to
 * @return a reference to the output stream
 */
QDebug & QcvProcessor::toDBStream(QDebug & dbg) const
{
    return toStream_Impl<QDebug>(dbg);
}

/*
 * Friend QDebug output operator
 * @param dbg the debug stream
 * @param proc the QcvProcessor to send to debug stream
 * @return the debug stream
 */
QDebug & operator << (QDebug & dbg, const QcvProcessor & proc)
{
    proc.toDBStream(dbg.nospace());
    return dbg.space();
}

/*
 * Update computed images slot and sends updated signal
 * required
 */
void QcvProcessor::update()
{
    /*
     * Important note : CvProcessor::update() should NOT be called here
     * since it should be called in QcvXXXprocessor subclasses such that
     * QcvXXXProcessor::update method should contain :
     *  - call to CvXXXProcessor::update() (not QCvXXXProcessor)
     *  - emit signals from QcvXXXProcessor
     *  - call to QcvProcessor::update() (this method) to
     *      - emit updated signal
     *      - emit standard process time strings signals
     *  - or
     *      - emit updated signal in QcvXXXProcessor
     *      - customize your processtimes and emit time strings signals
     */
    emit updated();
    processTimeString.sprintf(meanStdFormat.toStdString().c_str(),
                              getMeanProcessTime(0), getStdProcessTime(0));
//  processMinMaxTimeString.sprintf(minMaxFormat.toStdString().c_str(),
//                                  getMinProcessTime(0), getMaxProcessTime(0));
    emit processTimeUpdated(processTimeString);
//  emit processTimeMinMaxUpdated(processMinMaxTimeString);
    emit processTimeUpdated(&meanProcessTime);
}

/*
 * Changes source image slot.
 * Attributes needs to be cleaned up then set up again
 * @param image the new source Image
 * @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
 */
void QcvProcessor::setSourceImage(Mat *image)
    throw (CvProcessorException)
{
    Size previousSize(sourceImage->size());
    int previousNbChannels(nbChannels);

    if (sourceLock != NULL)
    {
        sourceLock->lock();
        // qDebug() << "QcvProcessor::setSourceImage: lock";
    }

    CvProcessor::setSourceImage(image);

    if (sourceLock != NULL)
    {
        // qDebug() << "QcvProcessor::setSourceImage: unlock";
        sourceLock->unlock();
    }

    emit imageChanged(sourceImage);

    emit imageChanged();

    if ((previousSize.width != image->cols) ∨
        (previousSize.height != image->rows))
    {
        emit imageSizeChanged();
    }

    if (previousNbChannels != nbChannels)
    {
        emit imageColorsChanged();
    }

    // Force update
    update();
}

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

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


/*
 * Get the format c-string for numbers
 * @return the format string for numbers (e.g.: "%5.2f")
 */
const char * QcvProcessor::getNumberFormat()
{
    return numberFormat.toStdString().c_str();
}

/*
 * Get the format c-string for std dev of numbers
 * @return the format string for numbers (e.g.: " Â± %4.2f")
 */
const char * QcvProcessor::getStdFormat()
{
    return meanStdFormat.toLocal8Bit().data();
}

/*
 * Get the format c-string for min / max of numbers
 * @return the format string for numbers (e.g.: "%5.2f / %5.2f")
 */
const char * QcvProcessor::getMinMaxFormat()
{
    return minMaxFormat.toLocal8Bit().data();
}
```



```
/*
 * QcvFloodFill.h
 *
 *  Created on: 25 fÃ©vr. 2012
 *      Author: davidroussel
 */

#ifndef QCVFLOODFILL_H_
#define QCVFLOODFILL_H_

#include <QMutex>
#include <QPoint>

#include "QcvProcessor.h"
#include "CvFloodFill.h"

/**
 * Qt oriented CvProcessor example
 */
class QcvFloodFill : public QcvProcessor, public CvFloodFill
{
    Q_OBJECT

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

    public:
        /**
         * QcvFloodFill constructor
         * @param inFrame the input frame from capture
         * @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 object
         */
        QcvFloodFill(Mat * inFrame,
                     QMutex * imageLock = NULL,
                     QThread * updateThread = NULL,
                     QObject * parent = NULL);

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

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

        /**
         * Select image to set in displayImage and sends notification message
         * @param index select the index to select display image
         */
        void setDisplayMode(const ImageDisplay index);

        /**
         * Sets a new flooding mode (absolute or relative) with notification
         * @param ffillMode the new flooding mode
         */
        void setFfillMode(const FloodFillMode ffillMode);

        /**
         * Sets a new lower difference in pixels values for flooding
         * @param loDiff the new lower difference for flooding
         */
        void setLoDiff(const int loDiff);

        /**
         * Sets a new upper difference in pixels values for flooding
         * @param upDiff the new upper difference for flooding
         */
        void setUpDiff(const int upDiff);

        /**
         * Sets a new connectivity for pixels neighbors for flooding with
         * notification
         * @param connectivity the new connectivity for pixels neighbors
         * for flooding
         */
        void setConnectivity(const int connectivity);

        /**
         * Sets an new initial seed
```

```
91          * @param initialSeed the new initial seed
92          */
93         void setInitialSeed(const Point & initialSeed);
94
95         /**
96          * Sets new show/hide seed point status with notification
97          * @param showSeed the new show/hide seed point status
98          */
99         void setShowSeed(const bool showSeed);
100
101        /**
102         * Set the show/hide bounding box status with notification
103         * @param showBoundingBox the new show/hide bounding box status
104         */
105        void setShowBoundingBox(const bool showBoundingBox);
106
107        /**
108         * Slot to clear current flood when left or right mouse button is
109         * pressed (should be connected to QcvMatWidget::pressPoint signal)
110         * Later relase event will evt trigger new seed for flood
111         * @param p the point the event occured
112         * @param button the pressed button
113         */
114        void clearFloodPoint(const QPoint & p, const Qt::MouseButton & button);
115
116        /**
117         * Slot to set intialSeed point (should be connected to
118         * QcvMatWidget::releasePoint signal)
119         * @param p the initial seed point
120         * @param button the button pressed and released
121         */
122        void setSeedPoint(const QPoint & p, const Qt::MouseButton & button);
123
124};
125
126#endif /* QCVFLOODFILL_H_ */
```



```
1  /*
2   * QcvFloodFill.cpp
3   *
4   *  Created on: 25 fÃ©vr. 2012
5   *      Author: davidroussel
6   */
7
8  #include "QcvFloodFill.h"
9
10 /*
11  * QcvFloodFill constructor
12  * @param inFrame the input frame from capture
13  * @param imageLock the mutex for concurrent access to the source image.
14  * In order to avoid concurrent access to the same image
15  * @param updateThread the thread in which this processor should run
16  * @param parent parent QObject
17  */
18 QcvFloodFill::QcvFloodFill(Mat * inFrame,
19                            QMutex * imageLock,
20                            QThread * updateThread,
21                            QObject * parent) :
22     CvProcessor(inFrame), // <-- virtual base class constructor first
23     QcvProcessor(inFrame, imageLock, updateThread, parent),
24     CvFloodFill(inFrame),
25     selfLock(updateThread ≠ NULL ? new QMutex() :
26                                    (imageLock ≠ NULL ? imageLock : NULL))
27
28 {
29     numberFormat = QString::fromUtf8("%5.0f");
30     meanStdFormat = numberFormat + QString::fromUtf8(" Â± %4.0f Âµs");
31     minMaxFormat = numberFormat + QString::fromUtf8("/") + numberFormat +
32                    QString::fromUtf8(" Âµs");
33 }
34
35 /*
36  * QcvFloodFill destructor
37  */
38 QcvFloodFill::~QcvFloodFill()
39 {
40     if (selfLock ≠ NULL)
41     {
42         selfLock→lock();
43         selfLock→unlock();
44         delete selfLock;
45     }
46 }
47
48 /*
49  * Update computed images and sends displayImageChanged signal if
50  * required
51  */
52 void QcvFloodFill::update()
53 {
54     bool hasSourceLock = (sourceLock ≠ NULL) ∧ (sourceLock ≠ selfLock);
55     if (hasSourceLock)
56     {
57         sourceLock→lock();
58         // qDebug() << "QcvFloodFill::update : lock";
59     }
60
61     bool hasLock = selfLock ≠ NULL;
62     if (hasLock)
63     {
64         selfLock→lock();
65     }
66
67     CvFloodFill::update();
68
69     if (hasLock)
70     {
71         selfLock→unlock();
72     }
73
74     if (hasSourceLock)
75     {
76         // qDebug() << "QcvFloodFill::update : unlock";
77         sourceLock→unlock();
78     }
79
80     // at the end of update, if displayImageChanged is true then display
81     // image has changed
82     if (displayImageChanged)
83     {
84         emit imageChanged(&displayImage);
85     }
86
87     /*
88      * emit updated signal
89      */
90     QcvProcessor::update();
```

```
91  }
92
93  /*
94   * Select image to set in displayImage and sends notification message
95   * @param select the index to select display image
96   */
97  void QcvFloodFill::setDisplayMode(const ImageDisplay index)
98  {
99      bool hasLock = selfLock ≠ NULL;
100     if (hasLock)
101     {
102         selfLock→lock();
103     }
104
105     CvFloodFill::setDisplayMode(index);
106
107     if (hasLock)
108     {
109         selfLock→unlock();
110     }
111
112     message.clear();
113     message.append(tr("Display Image set to: "));
114     switch (index)
115     {
116         case INPUT_IM:
117             message.append(tr("Input"));
118             break;
119         case MASK_IM:
120             message.append(tr("Mask"));
121             break;
122         case MERGED_IM:
123             message.append(tr("Merged Mask/Input"));
124             break;
125         case NBDISPLAY_IM:
126         default:
127             message.append(tr("Unknown"));
128             break;
129     }
130
131     emit sendMessage(message, defaultTimeOut);
132 }
133
134 /*
135  * Sets a new flooding mode (absolute or relative) with notification
136  * @param ffillMode the new flooding mode
137  */
138 void QcvFloodFill::setFfillMode(const FloodFillMode ffillMode)
139 {
140     bool hasLock = selfLock ≠ NULL;
141     if (hasLock)
142     {
143         selfLock→lock();
144     }
145
146     CvFloodFill::setFfillMode(ffillMode);
147
148     if (hasLock)
149     {
150         selfLock→unlock();
151     }
152
153     message.clear();
154     message.append(tr("Flood type set to "));
155     switch (getFfillMode())
156     {
157         case FIXED_RANGE:
158             message.append(tr("Fixed"));
159             break;
160         case FLOATING_RANGE:
161             message.append(tr("Floating"));
162             break;
163         default:
164             message.append(tr("Unknown"));
165             break;
166     }
167
168     message.append(tr(" range threshold"));
169
170     emit sendMessage(message, defaultTimeOut);
171 }
172
173 /*
174  * Sets a new lower difference in pixels values for flooding
175  * @param loDiff the new lower difference for flooding
176  */
177 void QcvFloodFill::setLoDiff(const int loDiff)
178 {
179     bool hasLock = selfLock ≠ NULL;
180     if (hasLock)
```



```
181     {
182         selfLock→lock();
183     }
184
185     CvFloodFill::setLoDiff(loDiff);
186
187     if (hasLock)
188     {
189         selfLock→unlock();
190     }
191
192 }
193
194 /*
195  * Sets a new upper difference in pixels values for flooding
196  * @param upDiff the new upper difference for flooding
197  */
198 void QcvFloodFill::setUpDiff(const int upDiff)
199 {
200     bool hasLock = selfLock ≠ NULL;
201     if (hasLock)
202     {
203         selfLock→lock();
204     }
205
206     CvFloodFill::setUpDiff(upDiff);
207
208     if (hasLock)
209     {
210         selfLock→unlock();
211     }
212
213 }
214
215 /*
216  * Sets a new connectivity for pixels neighbors for flooding with
217  * notification
218  * @param connectivity the new connectivity for pixels neighbors
219  * for flooding
220  */
221 void QcvFloodFill::setConnectivity(const int connectivity)
222 {
223     bool hasLock = selfLock ≠ NULL;
224     if (hasLock)
225     {
226         selfLock→lock();
227     }
228
229     CvFloodFill::setConnectivity(connectivity);
230
231     if (hasLock)
232     {
233         selfLock→unlock();
234     }
235
236     message.clear();
237
238     message.append(tr("Pixel connectivity set to "));
239     message.append(QString::number(getConnectivity()));
240     message.append(tr(" neighbors"));
241
242     emit sendMessage(message, defaultTimeOut);
243 }
244
245 /*
246  * Sets an new initial seed
247  * @param initialSeed the new initial seed
248  */
249 void QcvFloodFill::setInitialSeed(const Point & initialSeed)
250 {
251     bool hasLock = selfLock ≠ NULL;
252     if (hasLock)
253     {
254         selfLock→lock();
255     }
256
257     CvFloodFill::setInitialSeed(initialSeed);
258
259     if (hasLock)
260     {
261         selfLock→unlock();
262     }
263 }
264
265 /*
266  * Sets new show/hide seed point status with notification
267  * @param showSeed the new show/hide seed point status
268  */
269 void QcvFloodFill::setShowSeed(const bool showSeed)
270 {
```

```
271     bool hasLock = selfLock ≠ NULL;
272     if (hasLock)
273     {
274         selfLock→lock();
275     }
276
277     CvFloodFill::setShowSeed(showSeed);
278
279     if (hasLock)
280     {
281         selfLock→unlock();
282     }
283
284     message.clear();
285
286     message.append(tr("Show seed point is "));
287     if(isShowSeed())
288     {
289         message.append(tr("on"));
290     }
291     else
292     {
293         message.append(tr("off"));
294     }
295
296     emit sendMessage(message, defaultTimeOut);
297 }
298
299 /*
300  * Set the show/hide bounding box status with notification
301  * @param showBoundingBox the new show/hide bounding box status
302  */
303 void QcvFloodFill::setShowBoundingBox(const bool showBoundingBox)
304 {
305     bool hasLock = selfLock ≠ NULL;
306     if (hasLock)
307     {
308         selfLock→lock();
309     }
310
311     CvFloodFill::setShowBoundingBox(showBoundingBox);
312
313     if (hasLock)
314     {
315         selfLock→unlock();
316     }
317
318     message.clear();
319
320     message.append(tr("Show bouding box is "));
321     if(isShowBoundingBox())
322     {
323         message.append(tr("on"));
324     }
325     else
326     {
327         message.append(tr("off"));
328     }
329
330     emit sendMessage(message, defaultTimeOut);
331 }
332
333 /*
334  * Slot to clear current flood when left or right mouse button is
335  * pressed (should be connected to QcvMatWigdet::pressPoint signal)
336  * Later relase event will evt trigger new seed for flood
337  * @param p the point the event occured
338  * @param button the pressed button
339  */
340 void QcvFloodFill::clearFloodPoint(const QPoint &, const Qt::MouseButton & button)
341 {
342     // if button is left or right
343     if ((button ≡ Qt::LeftButton) ∨ (button ≡ Qt::RightButton))
344     {
345         bool hasLock = selfLock ≠ NULL;
346         if (hasLock)
347         {
348             selfLock→lock();
349         }
350
351         CvFloodFill::clearFlood();
352
353         if (hasLock)
354         {
355             selfLock→unlock();
356         }
357     }
358 }
359
360 /*
```



```
361  * Slot to set intialSeed point
362  * @param p the initial seed point
363  * @param button the button pressed
364  */
365 void QcvFloodFill::setSeedPoint(const QPoint & p, const Qt::MouseButton & button)
366 {
367     // if button left initiate new seed
368     if (button ≡ Qt::LeftButton)
369     {
370         int px = (p.x() > 0 ? p.x() : 0);
371         int py = (p.y() > 0 ? p.y() : 0);
372         setInitialSeed(Point(px, py));
373     }
374 }
```

```
/*
 * QcvMatWidget.h
 *
 *  Created on: 28 fÃ©vr. 2011
 *^H      Author: davidroussel
 */

#ifndef QCVMATWIDGET_H_
#define QCVMATWIDGET_H_

#include <QWidget>
#include <QHBoxLayout>
#include <QMouseEvent>
#include <QPoint>

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

/**
 * Abstract widget to show OpenCV Mat image into QT.
 * Should be refined in
 *  - QcvMatWidgetLabel
 *  - QcvMatWidgetImage
 *  - QcvMatWidgetGL
 */
class QcvMatWidget : public QWidget
{
    Q_OBJECT

    public:
        /**
         * Mouse sensivity of the image widget
         */
        typedef enum
        {
            /**
             * Sensitive to no mouse click or drag
             */
            MOUSE_NONE = 0,
            /**
             * Sensitive to mouse clicks
             */
            MOUSE_CLICK = 1,
            /**
             * Sensitive to mouse drag
             */
            MOUSE_DRAG = 2,
            /**
             * Sensitive to mouse click and drag
             */
            MOUSE_CLICK_AND_DRAG = 3
        } MouseSense;

    protected:
        /**
         * The widget layout
         */
        QHBoxLayout * layout;

        /**
         * The OpenCV BGR or gray image
         */
        Mat * sourceImage;

        /**
         * The OpenCV RGB image converted from gray or BGR OpenCV image
         */
        Mat displayImage;

        /**
         * Default size when no image has been set
         */
        static QSize defaultSize;

        /**
         * the aspect ratio ofthe image to draw
         */
        double aspectRatio;

        /**
         * Default aspect ratio when image is not set yet
         */
        static double defaultAspectRatio;

        /**
         * Indicate a mouse button is currently pressed within the widget
         */
        bool mousePressed;

        /**
```



```
         * Indicate a mouse is moved after a button has been pressed
         */
        bool mouseMoved;

        /**
         * Mouse sensivity
         */
        MouseSense mouseSense;

        /**
         * mouse pressed location
         */
        QPoint pressedPoint;

        /**
         * Mouse pressed button
         */
        Qt::MouseButton pressedButton;

        /**
         * mouse drag location
         */
        QPoint draggedPoint;

        /**
         * mouse release location
         */
        QPoint releasedPoint;

        /**
         * Selection rectangle
         */
        QRect selectionRect;

        /**
         * Drawing color
         */
        static const Scalar drawingColor;

        /**
         * Drawing width
         */
        static const int drawingWidth;

//      size_t count;

        /**
         * Pixel scale used to draw images.
         * Used in OpenGL contexts in order to draw images with the right pixel
         * scale on Hi DPI devices (such as retina screens)
         */
        float pixelScale;

    public:

        /**
         * OpenCV QT Widget default constructor
         * @param parent parent widget
         * @param mouseSense mouse sensivity
         */
        QcvMatWidget(QWidget *parent = NULL,
                     MouseSense mouseSense = MOUSE_NONE);

        /**
         * OpenCV QT Widget constructor
         * @param sourceImage the source image
         * @param parent parent widget
         * @param mouseSense mouse sensivity
         * @pre sourceImage is not NULL
         */
        QcvMatWidget(Mat * sourceImage,
                     QWidget *parent = NULL,
                     MouseSense mouseSense = MOUSE_NONE);

        /**
         * OpenCV Widget destructor.
         * Releases displayImage.
         */
        virtual ~QcvMatWidget(void);

//^H     ^H  /**
//^H     ^H   * Widget minimum size is set to the contained image size
//^H     ^H   * @return le size of the image within
//^H     ^H   */
//^H     ^H  QSize minimumSize() const;

        /**
         * Size hint (because size depends on sourceImage properties)
         * @return size obtained from sourceImage or defaultSize if sourceImage
         * is not set yet
```

```
181          */
182         QSize sizeHint() const;
183 
184         /**
185          * Gets Mat widget mouse clickable status
186          * @return true if widget is sensitive to mouse click
187          */
188         bool isMouseClickable() const;
189 
190         /**
191          * Gets Mat widget mouse dragable status
192          * @return true if widget is sensitive to mouse drag
193          */
194         bool isMouseDragable() const;
195 
196     protected:
197 
198         /**
199          * paint event reimplemented to draw content (in this case only
200          * draw in display image since final rendering method is not yet available)
201          * @param event the paint event
202          */
203         virtual void paintEvent(QPaintEvent * event);
204 
205         /**
206          * Widget setup
207          * @post new Layout has been created and set for this widget
208          */
209         void setup();
210 
211         /**
212          * Converts BGR or Gray source image to RGB display image
213          * @pre sourceImage is not NULL
214          * @post BGR or Gray source image has been converted to RGB displayimage
215          * @see #sourceImage
216          * @see #displayImage
217          */
218         void convertImage();
219 
220         /**
221          * Callback called when mouse button pressed event occurs.
222          * reimplemented to send pressPoint signal when left mouse button is
223          * pressed
224          * @param event mouse event
225          */
226         void mousePressEvent(QMouseEvent *event);
227 
228         /**
229          * Callback called when mouse move event occurs.
230          * reimplemented to send dragPoint signal when mouse is dragged
231          * (after left mouse button has been pressed)
232          * @param event mouse event
233          */
234         void mouseMoveEvent(QMouseEvent *event);
235 
236         /**
237          * Callback called when mouse button released event occurs.
238          * reimplemented to send releasePoint signal when left mouse button is
239          * released
240          * @param event mouse event
241          */
242         void mouseReleaseEvent(QMouseEvent *event);
243 
244         /**
245          * Draw Cross
246          * @param p the cross center
247          */
248         virtual void drawCross(const QPoint & p);
249 
250         /**
251          * Draw rectangle
252          * @param r the rectangle to draw
253          */
254         virtual void drawRectangle(const QRect & r);
255 
256 //       /*
257 //        * paint event reimplemented to draw content
258 //        * @param event the paint event
259 //        */
260 //       virtual void paintEvent(QPaintEvent * event) = 0;
261 
262         /**
263          * Modifiy selectionRect using two points
264          * @param p1 first point
265          * @param p2 second point
266          */
267         void selectionRectFromPoints(const QPoint & p1, const QPoint & p2);
268 
269     public slots:
270         /**
```



```
271          * Sets new source image
272          * @param sourceImage the new source image
273          * @pre sourceImage is not NULL
274          * @post new sourceImage has been set and aspectRatio has been updated
275          */
276         virtual void setSourceImage(Mat * sourceImage);
277 
278         /**
279          * Update slot customized to include convertImage before actually
280          * updating
281          * @post sourceImage have been converted to RGB and widget updated
282          */
283         virtual void update();
284 
285         /**
286          * Recompute pixel scale according to screen pixel scale.
287          * Slot triggered by a screenChanged(QScreen*) emitted by the containing
288          * window handle.
289          * Used with Hi DPI devices (such as retina screens).
290          * @post pixel scale have been updated according to
291          * devicePixelRatioF provided by the QPaintDevice super class
292          */
293         virtual void screenChanged();
294 
295     signals:
296 
297         /**
298          * Signal sent to transmit the point in the widget where a mouse
299          * button has been pressed
300          * @param p the point where any mouse button has been pressed
301          * @param button the button pressed
302          */
303         void pressPoint(const QPoint & p, const Qt::MouseButton & button);
304 
305         /**
306          * Signal sent to transmit the point in the widget where mouse cursor is
307          * currently dragged to (which suppose a mouse button has been
308          * previously pressed)
309          * @param p the point where the mouse cursor is dragged to
310          */
311         void dragPoint(const QPoint & p);
312 
313         /**
314          * Signal sent to transmit the point in the widget where a mouse
315          * button has been released
316          * @param p the point where left mouse button has been released
317          * @param button the button pressed
318          */
319         void releasePoint(const QPoint & p, const Qt::MouseButton & button);
320 
321         /**
322          * Signal sent to transmit the rectangle selection when mouse button
323          * has been clicked, dragged and released
324          * @param r the rectangle selection
325          * @param button the button pressed during dragging
326          */
327         void releaseSelection(const QRect & r, const Qt::MouseButton & button);
328 };
329 
330 #endif /* QCVMATWIDGET_H_ */
```

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

#include <opencv2/imgproc.hpp>

#include "QcvMatWidget.h"

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

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

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

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

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

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

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

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

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



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

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

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

    this->sourceImage = sourceImage;

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

}

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

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

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

```
    }
}

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

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

        emit pressPoint(pressedPoint, pressedButton);
    }
}

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

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

        selectionRectFromPoints(pressedPoint, draggedPoint);

        emit dragPoint(draggedPoint);
    }
}

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

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

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

    x1 = x0 - 2*offset;
```



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

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

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

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

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

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

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

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

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



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


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

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



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

jul 31, 16 0:05 **QcvMatWidgetLabel.hpp** Page 1/1

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

jul 31, 16 18:14 **QcvMatWidgetLabel.cpp** Page 1/1

```
1   //#include <iostream>
2   #include <QtDebug>
3   #include "QcvMatWidgetLabel.h"
4
5   using namespace std;
6
7   /*
8    * OpenCV QT Widget default constructor
9    * @param parent parent widget
10   */
11  QcvMatWidgetLabel::QcvMatWidgetLabel(QWidget *parent,
12                                       MouseSense mouseSense) :
13      QcvMatWidget(parent, mouseSense),
14      imageLabel(new QLabel())
15  {
16      setup();
17  }
18
19  /*
20   * OpenCV QT Widget constructor
21   * @param the source OpenCV qImage
22   * @param parent parent widget
23   */
24  QcvMatWidgetLabel::QcvMatWidgetLabel(Mat * sourceImage,
25                                       QWidget *parent,
26                                       MouseSense mouseSense) :
27      QcvMatWidget(sourceImage, parent, mouseSense),
28      imageLabel(new QLabel())
29  {
30      setup();
31  }
32
33  /*
34   * Widget setup
35   * @pre imageLabel has been allocated
36   */
37  void QcvMatWidgetLabel::setup()
38  {
39      layout->addWidget(imageLabel,0,Qt::AlignCenter);
40  }
41
42  /*
43   * OpenCV Widget destructor.
44   */
45  QcvMatWidgetLabel::~QcvMatWidgetLabel(void)
46  {
47      delete imageLabel;
48  }
49
50  /*
51   * paint event reimplemented to draw content
52   * @param event the paint event
53   */
54  void QcvMatWidgetLabel::paintEvent(QPaintEvent * event)
55  {
56  //  qDebug("QcvMatWidgetLabel::paintEvent");
57      QcvMatWidget::paintEvent(event);
58
59      if (displayImage.data != NULL)
60      {
61          // Builds QImage from RGB image data
62          // and sets image as Label pixmap
63          imageLabel->setPixmap(QPixmap::fromImage(QImage((uchar *) displayImage.data,
64                                                          displayImage.cols,
65                                                          displayImage.rows,
66                                                          displayImage.step,
67                                                          QImage::Format_RGB888)));
68      }
69      else
70      {
71          qWarning("QcvMatWidgetLabel::paintEvent : image.data is NULL");
72      }
73  }
```

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

#ifndef QOPENCVWIDGETQGL_H_
#define QOPENCVWIDGETQGL_H_

#include <QGLWidget>

#include "QcvMatWidget.h"
#include "QGLImageRender.h"

/**
 * OpenCV Widget for QT with QGLWidget display
 */
class QcvMatWidgetGL: public QcvMatWidget
{
    private:
        /**
         * QGLWidget to draw in
         */
        QGLImageRender * gl;

    public:

        /**
         * OpenCV QT Widget default constructor
         * @param parent parent widget
         * @param mouseSense mouse sensivity
         */
        QcvMatWidgetGL(QWidget *parent = NULL,
                       MouseSense mouseSense = MOUSE_NONE);

        /**
         * OpenCV QT Widget constructor
         * @param sourceImage the source image
         * @param parent parent widget
         * @param mouseSense mouse sensivity
         */
        QcvMatWidgetGL(Mat * sourceImage,
                       QWidget *parent = NULL,
                       MouseSense mouseSense = MOUSE_NONE);

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

        /**
         * OpenCV Widget destructor.
         */
        virtual ~QcvMatWidgetGL();

    protected:
        /*
         * paint event reimplemented to draw content
         * @param event the paint event
         */
        void paintEvent(QPaintEvent * event);
};

#endif /* QOPENCVWIDGETQGL_H_ */
```



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

#include "QcvMatWidgetGL.h"

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

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

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

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

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

    convertImage();

    gl = new QGLImageRender(displayImage, GL_RGB, &pixelScale, this);

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

/*
 * paint event reimplemented to draw content
 * @param event the paint event
 */
void QcvMatWidgetGL::paintEvent(QPaintEvent * event)
{
    QcvMatWidget::paintEvent(event);
    gl→update();
}
```

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

#ifndef QCVMATWIDGETIMAGE_H_
#define QCVMATWIDGETIMAGE_H_

#include <QImage>
#include <QPainter>

#include "QcvMatWidget.h"

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

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

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

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

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

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

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

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

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

    private:
        /**
         * Setup widget (defines size policy)
         */
        void setup();

    protected:
        /**
         * paint event reimplemented to draw content
         * @param event the paint event
```



```
         */
        void paintEvent(QPaintEvent * event);

};

#endif /* QCVMATWIDGETIMAGE_H_ */
```

```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);
```



```cpp

    // re-setup geometry since height x width may have changed
    updateGeometry();
}

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

/*
 * aspect ratio method
 * @param w width
 * @return the required height fo r this width
 */
int QcvMatWidgetImage::heightForWidth(int w) const
{
//  qDebug ("height  = %d for width  = %d called", (int)((double)w/aspectRatio), w);
    return (int)((double)w/aspectRatio);
}

/*
 * Minimum size hint according to aspect ratio and min height of 100
 * @return minimum size hint
 */
//QSize QcvMatWidgetImage::minimumSizeHint () const
//{
//  // qDebug("min size called");
//  // return QSize((int)(100.0*aspectRatio), 100);
//  return sizeHint();
//}


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

    // evt draws in image directly
    QcvMatWidget::paintEvent(event);

    if (displayImage.data ≠ NULL)
    {
        // then draw image
        QPainter painter(this);
        painter.setRenderHint(QPainter::SmoothPixmapTransform, true);
        if (event ≡ NULL)
        {
            painter.drawImage(0, 0, *qImage);
        }
        else // partial repaint
        {
            painter.drawImage(event→rect(), *qImage);
        }
    }
    else
    {
        qWarning("QcvMatWidgetImage::paintEvent : image.data is NULL");
    }
}
```

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

#ifndef QGLIMAGERENDER_H_
#define QGLIMAGERENDER_H_

#include <QGLWidget>
#include <QSize>
#include <QSizePolicy>

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

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

        /**
         * The pixel format:
         *  - GL_RGB for RGB converted images
         *  - GL_BGR for OpenCV natural format
         */
        GLenum pixelFormat;

        /**
         * Pixel scale pointer from container
         */
        float * pixelScale;

    public:
        /**
         * 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(const Mat & image,
                       const GLenum format = GL_RGB,
                       float * pixelScale = NULL,
                       QWidget *parent = NULL);

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

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

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

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

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



```

#endif /* QGLIMAGERENDER_H_ */
```

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

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

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

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

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

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

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

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

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

    glClear(GL_COLOR_BUFFER_BIT);

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



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

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

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

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

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

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

#ifndef QCVVIDEOCAPTURE_H_
#define QCVVIDEOCAPTURE_H_

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

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

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

    private:

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

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

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

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

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

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

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

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



```

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

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

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

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

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

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

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

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

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

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

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

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

        /**
         * Number of frames to test frame rate
         */
        static size_t defaultFrameNumberTest;

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

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

    public:
        /**
         * QcvVideoCapture constructor.
         * Opens the default camera (0)
```

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

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

        /**
         * QcvVideoCapture constructor from file name
         * @param fileName video file to open
         * @param flipVideo mirror image
         * @param gray convert image to gray
         * @param skip indicates capture can skip an image. When the capture
         * result has not been processed yet, or when false that capture should
         * wait for the result to be processed before grabbing a new image.
         * This only applies when #updateThread is not NULL.
         * @param width desired width or 0 to keep capture width
         * @param height desired height or 0 to keep capture height
         * @param updateThread the thread used to run this capture
         * @param parent the parent QObject
         */
        QcvVideoCapture(const QString & fileName,
                        const bool flipVideo = false,
                        const bool gray = false,
                        const bool skip = true,
                        const unsigned int  width = 0,
                        const unsigned int height = 0,
                        QThread * updateThread = NULL,
                        QObject * parent = NULL);

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

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

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

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



```
         * called before #grabTest
         */
        bool isDirectResizeable() const;

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

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

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

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

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

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

    public slots:
        /**
         * Open new device Id
         * @param deviceId device number to open
         * @param width desired width or 0 to keep capture width
         * @param height desired height or 0 to keep capture height
         * @return true if device has been opened and checked and timer launched
         */
        bool open(const int deviceId,
                  const unsigned int width = 0,
                  const unsigned int height = 0);

        /**
         * Open new video file
         * @param fileName video file to open
         * @param width desired width or 0 to keep capture width
         * @param height desired height or 0 to keep capture height
         * @return true if video has been opened and timer launched
         */
        bool open(const QString & fileName,
                  const unsigned int width = 0,
                  const unsigned int height = 0);
        /**
         * Sets video flipping
         * @param flipVideo flipped video or not
         */
        void setFlipVideo(const bool flipVideo);

        /**
         * Sets video conversion to gray
         * @param grayConversion the gray conversion status
         */
        void setGray(const bool grayConversion);

        /**
         * Sets #imageDisplay size according to preferred width and height
         * @param size new desired size to set
         * @param alreadyLocked mutex lock has already been aquired so setSize does not have
         * to acquire the lock
         * @pre a first image have been grabbed
         */
        void setSize(const QSize & size);

    private:
        /**
         * Performs a grab test to fill #image.
         * if capture is opened then tries to grab and if grab succeeds then
         * tries to retrieve image from grab and sets image size.
         * @return true if capture is opened and successfully grabbed a first
```

```
361             * frame into #image, false otherwise
362             * @post Moreover this method determines if direct resizing is allowed
363             * on this capture instance by trying to set
364             * CV_CAP_PROP_FRAME_WIDTH and CV_CAP_PROP_FRAME_HEIGHT.
365             */
366            bool grabTest();
367
368            /**
369             * Get or compute interval between two frames in ms and sets the
370             * frameRate attribute.
371             * Tries to get CV_CAP_PROP_FPS from capture and if not available
372             * computes times between frames by grabbing defaultNumberTest images
373             * @return interval between two frames
374             * @param message message passed to grabInterval and display ahead of
375             * the framerate computed during grabInterval
376             * @pre capture is already instanciated
377             * @post message indicating frame rate has been emitted and interval
378             * between two frames has been returned
379             */
380            int grabInterval(const QString & message);
381
382            /**
383             * Sets #imageDisplay size according to preferred width and height
384             * @param width desired width
385             * @param height desired height
386             * @pre a first image have been grabbed
387             */
388            void setSize(const unsigned int width,
389                         const unsigned int height);
390
391            /**
392             * Tries to set capture size directly on capture by setting properties.
393             *  - CV_CAP_PROP_FRAME_WIDTH to set frame width
394             *  - CV_CAP_PROP_FRAME_HEIGHT to set frame height
395             * @param width the width property to set on capture
396             * @param height the height property to set on capture
397             * @return true if capture is opened and if width and height have been
398             * set successfully through @code capture.set(...) @endcode. Returns
399             * false otherwise.
400             * @post if at least width or height have been set successfully, capture
401             * image is released then updated again so it will have the right
402             * dimensions.
403             * @warning if mutex lock can't be obtained to ensure atomic access to
404             * capture object, then we start recursing until we obtain that lock,
405             * which is gross and should be fixed !!!
406             */
407            bool setDirectSize(const unsigned int width, const unsigned int height);
408
409        protected slots:
410            /**
411             * update slot triggerred by timer : Grabs a new image and sends updated()
412             * signal iff new image has been grabbed, otherwise there is no more
413             * images to grab so kills timer.
414             * @note If lock on OpenCV capture object can not be obtained then
415             * capture is skipped. This is not critical since update is called
416             * regularly by the #timer, so we'll try updating image next time.
417             */
418            void update();
419
420        signals:
421            /**
422             * Signal emitted when a new image has been grabbed
423             */
424            void updated();
425
426            /**
427             * Signal emitted when capture is released
428             */
429            void finished();
430
431            /**
432             * Signal to send update message when something changes
433             * @param message the message
434             * @param timeout number of ms the message should be displayed
435             */
436            void messageChanged(const QString & message, int timeout = 0);
437
438            /**
439             * Signal to send when image has changed after opening new device or
440             * setting new display size
441             * @param image the new image to send
442             */
443            void imageChanged(Mat * image);
444
445            /**
446             * Signal emitted when timer is started whith a new delay
447             * @param delay the new timer delay value
448             */
449            void timerChanged(const int delay);
450
```



```
451            /**
452             * Signal to send when video capture is restarted (typically when
453             * playing video file and reaching the end of the file, the capture
454             * will try to go back to the beginning and play it again from start).
455             */
456            void restarted();
457    };
458
459    #endif /* QCVVIDEOCAPTURE_H_ */
460
```

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

#include <QElapsedTimer>
#include <QDebug>

#include "QcvVideoCapture.h"

#include <opencv2/imgproc/imgproc.hpp>

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

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

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

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

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



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

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

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

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

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

    if (grabTest())
    {
```

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



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

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



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

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



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

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



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

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



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

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

#ifndef CAPTUREFACTORY_H_
#define CAPTUREFACTORY_H_

#include <QString>
#include <QStringList>
#include <QThread>
#include "QcvVideoCapture.h"

/**
 * Capture Factory creates QcvVideoCapture from arguments list
 */
class CaptureFactory
{
    private:
        /**
         * The capture instance to create
         */
        QcvVideoCapture *capture;

        /**
         * Device number to open. Generally :
         *  - 0 is internal or fisrt camera
         *  - 1 is external or second camera
         */
        int deviceNumber;

        /**
         * Indicates capture opens camera or file.
         * Default value is true
         */
        bool liveVideo;

        /**
         * Video should be flipped horizontally for mirror effect
         * Default value is false
         */
        bool flippedVideo;

        /**
         * Video should be converted to gray during capture.
         * Default value is false
         */
        bool grayVideo;

        /**
         * Capture can skip capturing new image when previous image has not
         * been processed yet, or can wait for the previous image to be
         * processed before grabbing a new image.
         */
        bool skipImages;

        /**
         * Video preferred width (evt resize video)
         * Default value is 0 which means no preferred width
         */
        int preferredWidth;

        /**
         * Video preferred height (evt resize video)
         * Default value is 0 which means no preferred height
         */
        int preferredHeight;

        /**
         * Path to video file
         */
        QString videoPath;

    public:
        /**
         * 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(const QStringList & argList);

        /**
```



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

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

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

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

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

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

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

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

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

        /**
         * Provide capture instanciated according to values
         * extracted from argument lists
         * @param updateThread the thread to run this capture or NULL if this
         * capture run in the current thread
         * @return the new capture instance
         */
        QcvVideoCapture * getCaptureInstance(QThread * updatethread = NULL);
};

#endif /* CAPTUREFACTORY_H_ */
```

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

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

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

        if (currentArg == "-d" || currentArg =="--device")
        {
            // Next argument should be device number integer
            if (it.hasNext())
            {
                QString deviceString(it.next());
                bool convertOk;
                deviceNumber = deviceString.toInt(&convertOk,10);
                if (!convertOk || deviceNumber < 0)
                {
                    qWarning("Warning: Invalid device number %d",deviceNumber);
                    deviceNumber = 0;
                }
                liveVideo = true;
            }
            else
            {
                qWarning("Warning: device tag found with no following device number");
            }
        }
        else if (currentArg == "-v" || currentArg == "--video")
        {
            // Next argument should be a path name to video file or URL
            if (it.hasNext())
            {
                videoPath = it.next();
                liveVideo = false;
            }
            else
            {
                qWarning("file tag found with no following filename");
            }
        }
        else if (currentArg == "-m" || currentArg == "--mirror")
        {
            flippedVideo = true;
        }
        else if (currentArg == "-g" || currentArg == "--gray")
        {
            grayVideo = true;
        }
        else if (currentArg == "-k" || currentArg == "--skip")
        {
            skipImages = true;
        }
        else if (currentArg == "-s" || currentArg == "--size")
        {
```



```
            if (it.hasNext())
            {
                // search for <width>x<height>
                QString sizeString = it.next();
                int xIndex = sizeString.indexOf(QChar('x'), 0,
                    Qt::CaseInsensitive);
                if (xIndex != -1)
                {
                    QString widthString = sizeString.left(xIndex);
                    preferredWidth = widthString.toUInt();
                    qDebug("preferred width is %d", preferredWidth);

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

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

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

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

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

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

/*
 * Set gray conversion
```

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

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

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

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

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

    qDebug() << "Opening ";
    if (liveVideo)
    {
        // Live video feed
        qDebug() << "Live Video ... from camera # " << deviceNumber;
        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;
}
```



```
#ifndef MEANVALUE_H
#define MEANVALUE_H

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

/**
 * Mean and std value for type T values expressed in type R
 * @tparam T the type of value to compute mean and std with
 * @tparam R the type of value of mean and std computation
 * @example
 * @code
 *  MeanValue<clock_t, double>
 * @endcode
 * @author David Roussel
 * @date 2014/05/31
 */
template <typename T, typename R = T>
class MeanValue
{
    private:
        /**
         * Elements sum
         * @warning this implementation can lead to sum overflow
         */
        T sum;

        /**
         * Element square sum (used to get std)
         * @warning this implementation can lead to sum2 overflow
         */
        T sum2;

        /**
         * Number of elements counted so far
         */
        size_t count;

        /**
         * Minimum recorded value
         */
        T minValue;

        /**
         * Maximum recorded value
         */
        T maxValue;

        /**
         * Value to reset minimum value to
         * (a high value so that next value will have reasonable chances to be
         * less than this value)
         */
        const T resetMinValue;

        /**
         * Value to reset maximum value to
         * (a low value so that next value will have reasonable chances to be
         * greater than this value)
         */
        const T resetMaxValue;
    public:
        /**
         * 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
         */
        MeanValue(const T & initialValue = static_cast<T>(0),
                  const T & initialMinimum = static_cast<T>(numeric_limits<T>::max()));

        /**
         * Copy constructor
         * @param mv the other mean value to copy
         */
        MeanValue(const MeanValue<T, R> & mv);

        /**
         * Move constructor
         * @param mv the other mean value to copy
         */
        MeanValue(MeanValue<T, R> && mv);

        /**
         * Destructor
         */
        virtual ~MeanValue();
```

```
91
92          /**
93           * Function call operator
94           * @param value value to add to the values sum and values square sum
95           * @post elements count has been increased
96           */
97          void operator ()(const T & value);
98
99          /**
100          * Self increment operator
101          * @param value value to add to the values sum and values square sum
102          * @post elements count has been increased
103          * @note does the same thing as Function call operator
104          */
105         void operator +=(const T & value);
106
107         /**
108          * Copy operator from another mean value
109          * @param mv the mean value to copy
110          * @return a reference to the current mean value
111          */
112         MeanValue<T, R> & operator =(const MeanValue<T, R> & mv);
113
114         /**
115          * Move operator from another mean value
116          * @param mv the mean value to move
117          * @return a reference to the current mean value
118          */
119         MeanValue<T, R> & operator =(MeanValue<T, R> && mv);
120
121         /**
122          * Cast operator to result type
123          * @return the mean value
124          */
125         operator R() const;
126
127         /**
128          * Compute mean value : E(X) = sum/nbElements
129          * @return the mean value of all added elements.
130          */
131         R mean() const;
132
133         /**
134          * Compute standard deviation of values : sqrt(E(X^2) - E(X)^2)
135          * @return the standard deviation of all added elements.
136          */
137         R std() const;
138
139         /**
140          * Minimum recorded value accessor
141          * @return the minimum recorded value (until reset)
142          */
143         T min() const;
144
145         /**
146          * Maximum recorded value accessor
147          * @return the maximum recorded value (until reset)
148          */
149         T max() const;
150
151         /**
152          * Reset added values, square values and count to 0, and reset
153          * min & max values to their default values
154          */
155         void reset();
156 };
157
158 /**
159  * Output operator for MeanValue
160  * @param out the output stream
161  * @param mv the MeanValue to print on the output stream
162  * @return a reference to the current output stream
163  * @post put mean value Â± std value on the stream
164  */
165 template <typename T, typename R>
166 ostream & operator <<(ostream & out, const MeanValue<T, R> & mv);
167
168 #endif // MEANVALUE_H
```



```
1   #include <cmath>
2   #include <opencv2/core/core.hpp>    // for MeanValue<cv::Mat, cv::Mat> specialization
3
4   #include "MeanValue.h"
5
6   /*
7    * Constructor.
8    * Initialize sum & sum2 to T(0) and count to 0
9    * @param initialValue [optional] a T specimen can be provided in order
10   * to initialise sum and sum2 by copying the specimen
11   * @param initialMinimum [optional] initial value of minimum and minimum
12   * reset value
13   */
14  template <typename T, typename R>
15  MeanValue<T, R>::MeanValue(const T & initialValue,
16                             const T & initialMinimum) :
17      sum(initialValue),
18      sum2(initialValue),
19      count(0),
20      minValue(initialMinimum),
21      maxValue(initialValue),
22      resetMinValue(initialMinimum),
23      resetMaxValue(initialValue)
24  {
25  }
26
27  /*
28   * Copy constructor
29   * @param mv the other mean value to copy
30   */
31  template <typename T, typename R>
32  MeanValue<T, R>::MeanValue(const MeanValue<T, R> & mv) :
33      sum(mv.sum),
34      sum2(mv.sum2),
35      count(mv.count),
36      minValue(mv.minValue),
37      maxValue(mv.maxValue),
38      resetMinValue(mv.resetMinValue),
39      resetMaxValue(mv.resetMaxValue)
40  {
41  }
42
43  /*
44   * Move constructor
45   * @param mv the other mean value to copy
46   */
47  template <typename T, typename R>
48  MeanValue<T, R>::MeanValue(MeanValue<T, R> && mv) :
49      sum(mv.sum),
50      sum2(mv.sum2),
51      count(mv.count),
52      minValue(mv.minValue),
53      maxValue(mv.maxValue),
54      resetMinValue(mv.resetMinValue),
55      resetMaxValue(mv.resetMaxValue)
56  {
57  }
58
59  /*
60   * Destructor
61   */
62  template <typename T, typename R>
63  MeanValue<T, R>::~MeanValue()
64  {
65  }
66
67  /*
68   * Function call operator
69   * @param value value to add to the values sum and values square sum
70   * @post elements count has been increased
71   */
72  template <typename T, typename R>
73  void MeanValue<T, R>::operator ()(const T & value)
74  {
75      sum += value;
76      sum2 += value * value;
77      count++;
78      if (value > maxValue)
79      {
80          maxValue = value;
81      }
82      if (value < minValue)
83      {
84          minValue = value;
85      }
86  }
87
88  /*
89   * Self increment operator
90   * @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 "QcvFloodFill.h"

namespace Ui {
    class MainWindow;
}

/**
 * OpenCV/Qt capture input main window
 */
class MainWindow : public QMainWindow
{
    Q_OBJECT

    public:

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

        /**
         * MainWindow constructor.
         * @param capture the capture QObject to capture frames from devices
         * or video files
         * @param processor the colorspace class to compute various components
         * on various color spaces
         * @param parent parent widget
         */
        explicit MainWindow(QcvVideoCapture * capture,
                            QcvFloodFill * 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 Color space object to compute color components
         */
        QcvFloodFill * 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 setupImageWidget(const RenderMode mode);

        /**
         * Setup UI according to capture settings when app launches
         */
        void setupUIfromCapture();

        /**
         * Setup UI according to processor settings when app launches
         */
        void setupUIfromProcessor();

    private slots:

        /**
         * Setup processor from current UI settings when processor source image
         * changes
         */
        void setupProcessorfromUI();

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

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

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

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

```

```
181         /**
182          * Menu action when flip image is selected.
183          * Sets capture to change flip status which leads to reverse
184          * image horizontally
185          */
186         void on_actionFlip_triggered();
187
188         /**
189          * Menu action when gray image is selected.
190          * Sets capture to convert source image to gray
191          */
192         void on_actionGray_triggered();
193
194         /**
195          * Menu action when original image size is selected.
196          * Sets capture not to resize image
197          */
198         void on_actionOriginalSize_triggered();
199
200         /**
201          * Menu action when constrained image size is selected.
202          * Sets capture resize to preferred width and height
203          */
204         void on_actionConstrainedSize_triggered();
205
206         /**
207          * Menu action to replace current image rendering widget by a
208          * QcvMatWidgetImage instance.
209          */
210         void on_actionRenderImage_triggered();
211
212         /**
213          * Menu action to replace current image rendering widget by a
214          * QcvMatWidgetLabel with pixmap instance.
215          */
216         void on_actionRenderPixmap_triggered();
217
218         /**
219          * Menu action to replace current image rendering widget by a
220          * QcvMatWidgetGL instance.
221          */
222         void on_actionRenderOpenGL_triggered();
223
224
225         /**
226          * Original size radioButton action.
227          * Sets capture resize to off
228          */
229         void on_radioButtonOrigSize_clicked();
230
231         /**
232          * Custom size radioButton action.
233          * Sets capture resize to preferred width and height
234          */
235         void on_radioButtonCustomSize_clicked();
236
237         /**
238          * Width spinbox value change.
239          * Changes the preferred width and if custom size is selected apply
240          * this custom width
241          * @param value the desired width
242          */
243         void on_spinBoxWidth_valueChanged(int value);
244
245         /**
246          * Height spinbox value change.
247          * Changes the preferred height and if custom size is selected apply
248          * this custom height
249          * @param value the desired height
250          */
251         void on_spinBoxHeight_valueChanged(int value);
252
253         /**
254          * Flip capture image horizontally.
255          * changes capture flip status
256          */
257         void on_checkBoxFlip_clicked();
258
259         /**
260          * Convert capture image to gray
261          */
262         void on_checkBoxGray_clicked();
263
264         /**
265          * Select input image for display
266          */
267         void on_radioButtonInput_clicked();
268
269         /**
270          * Select mask image for display
```



```
271          */
272         void on_radioButtonMask_clicked();
273
274         /**
275          * Select merged image for display
276          */
277         void on_radioButtonMerged_clicked();
278
279         /**
280          * Select absolute threshold mode for flood fill
281          */
282         void on_radioButtonAbsThreshold_clicked();
283
284         /**
285          * Select floating threshold mode for flood fill
286          */
287         void on_radioButtonRelThreshold_clicked();
288
289         /**
290          * Clears current floor
291          */
292         void on_pushButtonClearFlood_clicked();
293
294         /**
295          * Generate new color for flood
296          */
297         void on_pushButtonNewColor_clicked();
298
299         /**
300          * Show/hides flood bouding box in source image
301          */
302         void on_checkBoxBBox_clicked();
303
304         /**
305          * Show/hides flood center
306          */
307         void on_checkBoxCenter_clicked();
308
309         /**
310          * Changes loDiff value
311          * @param value the new loDiff value
312          */
313         void on_spinBoxLoDiff_valueChanged(int value);
314
315         /**
316          * Changes upDiff value
317          * @param value the new upDiff value
318          */
319         void on_spinBoxUpDiff_valueChanged(int value);
320
321         /**
322          * Selects 4 pixels connectivity for flooding
323          */
324         void on_radioButton4Connect_clicked();
325
326         /**
327          * Selects 4 pixels connectivity for flooding
328          */
329         void on_radioButton8Connect_clicked();
330
331         /**
332          * Link the two Diff sliders and spinBox together
333          * @param checked the new link state
334          */
335         void on_checkBoxLink_clicked(bool checked);
336 };
337
338 #endif // MAINWINDOW_H
```

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

#include <QObject>
#include <QFileDialog>
#include <QDebug>
#include <assert.h>

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

/*
 * MainWindow constructor
 * @param capture the capture QObject to capture frames from devices
 * or video files
 * @param parent parent widget
 */
MainWindow::MainWindow(QcvVideoCapture * capture,
                       QcvFloodFill * processor,
                       QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow),
    capture(capture),
    processor(processor),
    preferredWidth(640),
    preferredHeight(480)
{
    ui->setupUi(this);
    ui->scrollArea->setBackgroundRole(QPalette::Mid);

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

    assert(processor != NULL);

    // ----------------------------------------------------------------------
    // Signal/Slot connections
    // ----------------------------------------------------------------------
    // Replace QcvMatWidget instance with QcvMatWidgetImage instance and
    // sets widgetImage source for the first time
    setupImageWidget(RENDER_IMAGE);

    // Connects Mainwindow messages to status bar
    connect(this, SIGNAL(sendMessage(QString,int)),
            ui->statusBar, SLOT(showMessage(QString,int)));

    // Connects capture status messages to statusBar
    connect(capture, SIGNAL(messageChanged(QString,int)),
            ui->statusBar, SLOT(showMessage(QString,int)));

    // Connects processor status messages to statusBar
    connect(processor, SIGNAL(sendMessage(QString,int)),
            ui->statusBar, SLOT(showMessage(QString,int)));

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

    connect(processor, SIGNAL(processTimeUpdated(QString)),
            ui->labelTime, SLOT(setText(QString)));

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

    // ----------------------------------------------------------------------
    // UI setup according to capture and processor options
    // ----------------------------------------------------------------------
    setupUIfromCapture();

    setupUIfromProcessor();
}

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



```cpp

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

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

    // Pressed mouse button in image widget clears current flood in processor
    disconnect(ui->widgetImage, SIGNAL(pressPoint(QPoint,Qt::MouseButton)),
               processor, SLOT(clearFloodPoint(QPoint,Qt::MouseButton)));

    // Released left mouse button in image widget creates a new flood seed
    disconnect(ui->widgetImage, SIGNAL(releasePoint(QPoint,Qt::MouseButton)),
               processor, SLOT(setSeedPoint(QPoint,Qt::MouseButton)));

    // remove widget in scroll area
    QWidget * w = ui->scrollArea->takeWidget();

    if (w == ui->widgetImage)
    {
        // delete removed widget
        delete ui->widgetImage;

        // create new widget
        Mat * image = processor->getImagePtr("display");

        switch (mode)
        {
            case RENDER_PIXMAP:
                ui->widgetImage = new QcvMatWidgetLabel(image,
                                                        ui->scrollArea,
                                                        QcvMatWidget::MOUSE_CLICK);
                break;
            case RENDER_GL:
                ui->widgetImage = new QcvMatWidgetGL(image,
                                                     ui->scrollArea,
                                                     QcvMatWidget::MOUSE_CLICK);
                break;
            case RENDER_IMAGE:
            default:
                ui->widgetImage = new QcvMatWidgetImage(image,
                                                        ui->scrollArea,
                                                        QcvMatWidget::MOUSE_CLICK);
                break;
        }

        if (ui->widgetImage != NULL)
        {
            ui->widgetImage->setObjectName(QString::fromUtf8("widgetImage"));

            // add it to the scroll area
            ui->scrollArea->setWidget(ui->widgetImage);

            connect(processor, SIGNAL(updated()),
                    ui->widgetImage, SLOT(update()));

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

            // Pressed mouse button in image widget clears current flood in processor
            connect(ui->widgetImage, SIGNAL(pressPoint(QPoint,Qt::MouseButton)),
                    processor, SLOT(clearFloodPoint(QPoint,Qt::MouseButton)),
                    Qt::DirectConnection);

            // Released left mouse button in image widget creates a new flood seed
            connect(ui->widgetImage, SIGNAL(releasePoint(QPoint,Qt::MouseButton)),
                    processor, SLOT(setSeedPoint(QPoint,Qt::MouseButton)),
                    Qt::DirectConnection);


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

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

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

        QSize size = capture->getSize();
        qDebug("Capture->size is %dx%d", size.width(), size.height());
        preferredWidth = size.width();
        preferredHeight = size.height();

    }
    else
    {
        /*
         * Initial Size radio buttons configuration
         */
        ui->radioButtonCustomSize->setChecked(false);
        ui->radioButtonOrigSize->setChecked(true);

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

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

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

    // Sets gray checkbox and menu item states
    bool grayed = capture->isGray();
    ui->actionGray->setChecked(grayed);
    ui->checkBoxGray->setChecked(grayed);
}

/*
 * Setup UI according to processor settings when app launches
 */
void MainWindow::setupUIfromProcessor()
{
```



```
    // Sets selected image for display
    switch (processor->getDisplayMode())
    {
        case CvFloodFill::INPUT_IM:
            ui->radioButtonInput->setChecked(true);
            break;
        case CvFloodFill::MASK_IM:
            ui->radioButtonMask->setChecked(true);
            break;
        case CvFloodFill::MERGED_IM:
            ui->radioButtonMerged->setChecked(true);
            break;
        case CvFloodFill::NBDISPLAY_IM:
        default:
            // Do nothing
            break;
    }

    // Sets flooding mode radio buttons
    switch (processor->getFfillMode())
    {
        case CvFloodFill::FIXED_RANGE:
            ui->radioButtonAbsThreshold->setChecked(true);
            break;
        case CvFloodFill::FLOATING_RANGE:
            ui->radioButtonRelThreshold->setChecked(true);
            break;
        default:
        break;
    }

    // Sets show Bounding box
    ui->checkBoxBBox->setChecked(processor->isShowBoundingBox());

    // Sets Show center
    ui->checkBoxCenter->setChecked(processor->isShowSeed());

    // Set pixel connectivity radio buttons
    if (processor->getConnectivity() == 4)
    {
        ui->radioButton4Connect->setChecked(true);
    }
    else
    {
        ui->radioButton8Connect->setChecked(true);
    }

    // Sets LoDiff slider and spinBox
    ui->spinBoxLoDiff->setValue(processor->getLoDiff());

    // Sets upDiff slider and spinBox
    ui->spinBoxUpDiff->setValue(processor->getUpDiff());

}

/*
 * Setup processor from current UI settings when processor source image
 * changes
 */
void MainWindow::setupProcessorfromUI()
{
    if (ui->radioButtonInput->isChecked())
    {
        processor->setDisplayMode(CvFloodFill::INPUT_IM);
    }

    if (ui->radioButtonMask->isChecked())
    {
        processor->setDisplayMode(CvFloodFill::MASK_IM);
    }

    if (ui->radioButtonMerged->isChecked())
    {
        processor->setDisplayMode(CvFloodFill::MERGED_IM);
    }

    if (ui->radioButtonAbsThreshold->isChecked())
    {
        processor->setFfillMode(CvFloodFill::FIXED_RANGE);
    }

    if (ui->radioButtonRelThreshold->isChecked())
    {
        processor->setFfillMode(CvFloodFill::FLOATING_RANGE);
    }

    processor->setShowBoundingBox(ui->checkBoxBBox->isChecked());
    processor->setShowSeed(ui->checkBoxCenter->isChecked());

    if (ui->radioButton4Connect->isChecked())
```

```
361     {
362         processor->setConnectivity(4);
363     }
364
365     if (ui->radioButton8Connect->isChecked())
366     {
367         processor->setConnectivity(8);
368     }
369
370     processor->setLoDiff(ui->spinBoxLoDiff->value());
371     processor->setUpDiff(ui->spinBoxUpDiff->value());
372 }
373
374 /*
375  * Menu action when Sources->camera 0 is selected
376  * Sets capture to open device 0. If device is not available
377  * menu item is set to inactive.
378  */
379 void MainWindow::on_actionCamera_0_triggered()
380 {
381     int width = 0;
382     int height = 0;
383
384     if (ui->radioButtonCustomSize->isChecked())
385     {
386         width = preferredWidth;
387         height = preferredHeight;
388     }
389
390     qDebug("Opening device 0 ...");
391 //  if (!capture->open(0, width, height))
392 //  {
393 //      qWarning("Unable to open device 0");
394 //      // disable menu item if camera 0 does not exist
395 //      ui->actionCamera_0->setDisabled(true);
396 //  }
397     emit deviceChanged(0, width, height);
398 }
399
400 /*
401  * Menu action when Sources->camera 1 is selected
402  * Sets capture to open device 0. If device is not available
403  * menu item is set to inactive
404  */
405 void MainWindow::on_actionCamera_1_triggered()
406 {
407     int width = 0;
408     int height = 0;
409
410     if (ui->radioButtonCustomSize->isChecked())
411     {
412         width = preferredWidth;
413         height = preferredHeight;
414     }
415
416     qDebug("Opening device 1 ...");
417 //  if (!capture->open(1, width, height))
418 //  {
419 //      qWarning("Unable to open device 1");
420 //      // disable menu item if camera 1 does not exist
421 //      ui->actionCamera_1->setDisabled(true);
422 //  }
423     emit deviceChanged(1, width, height);
424 }
425
426 /*
427  * Menu action when Sources->file is selected.
428  * Opens file dialog and tries to open selected file (is not empty),
429  * then sets capture to open the selected file
430  */
431 void MainWindow::on_actionFile_triggered()
432 {
433     int width = 0;
434     int height = 0;
435
436     if (ui->radioButtonCustomSize->isChecked())
437     {
438         width = preferredWidth;
439         height = preferredHeight;
440     }
441
442     QString fileName =
443             QFileDialog::getOpenFileName(this,
444                                          tr("Open Video"),
445                                          "./",
446                                          tr("Video Files (*.avi *.mkv *.mp4 *.m4v)"),
447                                          NULL,
448                                          QFileDialog::ReadOnly);
449
450     qDebug("Opening file %s ...", fileName.toStdString().c_str());
```



```
451
452     if (fileName.length() > 0)
453     {
454 //      if (!capture->open(fileName))
455 //      {
456 //          qWarning("Unable to open device file : %s",
457 //                   fileName.toStdString().c_str());
458 //      }
459 //      // setupProcessorFromUI(); // already done from connection
460         emit fileChanged(fileName, width, height);
461     }
462     else
463     {
464         qWarning("empty file name");
465     }
466 }
467
468 /*
469  * Menu action to qui application
470  */
471 void MainWindow::on_actionQuit_triggered()
472 {
473     this->close();
474 }
475
476 /*
477  * Menu action when flip image is selected.
478  * Sets capture to change flip status which leads to reverse
479  * image horizontally
480  */
481 void MainWindow::on_actionFlip_triggered()
482 {
483     emit flipChanged(¬capture->isFlipVideo());
484     /*
485      * There is no need to update ui->checkBoxFlip since it is connected
486      * to ui->actionFlip through signals/slots
487      */
488 }
489
490 /*
491  * Menu action when gray image is selected.
492  * Sets capture to convert source image to gray
493  */
494 void MainWindow::on_actionGray_triggered()
495 {
496     bool isGray = ¬capture->isGray();
497
498     emit grayChanged(isGray);
499 }
500
501 /*
502  * Menu action when original image size is selected.
503  * Sets capture not to resize image
504  */
505 void MainWindow::on_actionOriginalSize_triggered()
506 {
507
508     ui->actionConstrainedSize->setChecked(false);
509
510     emit sizeChanged(QSize(0, 0));
511 }
512
513 /*
514  * Menu action when constrained image size is selected.
515  * Sets capture resize to preferred width and height
516  */
517 void MainWindow::on_actionConstrainedSize_triggered()
518 {
519     ui->actionOriginalSize->setChecked(false);
520
521     emit sizeChanged(QSize(preferredWidth, preferredHeight));
522 }
523
524 /*
525  * Menu action to replace current image rendering widget by a
526  * QcvMatWidgetImage instance.
527  */
528 void MainWindow::on_actionRenderImage_triggered()
529 {
530     setupImageWidget(RENDER_IMAGE);
531 }
532
533 /*
534  * Menu action to replace current image rendering widget by a
535  * QcvMatWidgetLabel with pixmap instance.
536  */
537 void MainWindow::on_actionRenderPixmap_triggered()
538 {
539     setupImageWidget(RENDER_PIXMAP);
540 }
```

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



```
631 /*
632  * Select mask image for display
633  */
634 void MainWindow::on_radioButtonMask_clicked()
635 {
636     processor->setDisplayMode(CvFloodFill::MASK_IM);
637 }
638
639 /*
640  * Select merged image for display
641  */
642 void MainWindow::on_radioButtonMerged_clicked()
643 {
644     processor->setDisplayMode(CvFloodFill::MERGED_IM);
645 }
646
647 /*
648  * Select absolute threshold mode for flood fill
649  */
650 void MainWindow::on_radioButtonAbsThreshold_clicked()
651 {
652     processor->setFfillMode(CvFloodFill::FIXED_RANGE);
653 }
654
655 /*
656  * Select floating threshold mode for flood fill
657  */
658 void MainWindow::on_radioButtonRelThreshold_clicked()
659 {
660     processor->setFfillMode(CvFloodFill::FLOATING_RANGE);
661 }
662
663 /*
664  * Clears current floor
665  */
666 void MainWindow::on_pushButtonClearFlood_clicked()
667 {
668     processor->clearFlood();
669 }
670
671 /*
672  * Generate new color for flood
673  */
674 void MainWindow::on_pushButtonNewColor_clicked()
675 {
676     processor->newFloodColor();
677 }
678
679 /*
680  * Show/hides flood bouding box in source image
681  */
682 void MainWindow::on_checkBoxBBox_clicked()
683 {
684     processor->setShowBoundingBox(ui->checkBoxBBox->isChecked());
685 }
686
687 /*
688  * Show/hides flood center
689  */
690 void MainWindow::on_checkBoxCenter_clicked()
691 {
692     processor->setShowSeed(ui->checkBoxCenter->isChecked());
693 }
694
695 /*
696  * Changes loDiff value
697  * @param value the new loDiff value
698  */
699 void MainWindow::on_spinBoxLoDiff_valueChanged(int value)
700 {
701     processor->setLoDiff(value);
702 }
703
704 /*
705  * Changes upDiff value
706  * @param value the new upDiff value
707  */
708 void MainWindow::on_spinBoxUpDiff_valueChanged(int value)
709 {
710     processor->setUpDiff(value);
711 }
712
713 /*
714  * Selects 4 pixels connectivity for flooding
715  */
716 void MainWindow::on_radioButton4Connect_clicked()
717 {
718     processor->setConnectivity(4);
719 }
720
```

```
721 /*
722  * Selects 4 pixels connectivity for flooding
723  */
724 void MainWindow::on_radioButton8Connect_clicked()
725 {
726     processor->setConnectivity(8);
727 }
728
729 /*
730  * Link the two Diff sliders and spinBox together
731  * @param checked the new link state
732  */
733 void MainWindow::on_checkBoxLink_clicked(bool checked)
734 {
735     if(checked)
736     {
737         // connect spinner spinBoxLoDiff  to spinBoxUpDiff
738         connect(ui->spinBoxLoDiff, SIGNAL(valueChanged(int)),
739                 ui->spinBoxUpDiff, SLOT(setValue(int)));
740         // Disable spinBoxUpDiff & horizontalSliderUpDiff
741         ui->spinBoxUpDiff->setEnabled(false);
742         ui->horizontalSliderUpDiff->setEnabled(false);
743     }
744     else
745     {
746         // disconnect spinner spinBoxLoDiff  to spinBoxUpDiff
747         disconnect(ui->spinBoxLoDiff, SIGNAL(valueChanged(int)),
748                 ui->spinBoxUpDiff, SLOT(setValue(int)));
749         // Enable spinBoxUpDiff & horizontalSliderUpDiff
750         ui->spinBoxUpDiff->setEnabled(true);
751         ui->horizontalSliderUpDiff->setEnabled(true);
752     }
753 }
```



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

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



```
181
182     bool sameThread = capThread ≡ procThread;
183
184     if (capThread ≠ NULL)
185     {
186         delete capThread;
187     }
188
189     if (procThread ≠ NULL ∧ ¬sameThread)
190     {
191         delete procThread;
192     }
193
194     return retVal;
195 }
196
197 /*
198  * Usage function shown just before launching QApp
199  * @param name the name of the program (argv[0])
200  */
201 void usage(char * name)
202 {
203     cout << "usage : " << basename(name) << " "
204          << "[-d | --device] <device number> "
205          << "[-v | --video] <video file> "
206          << "[-s | --size] <width>x<height> "
207          << "[-m | --mirror] " << endl
208          << "Keys : " << endl
209          << "\t i: color input image" << endl
210          << "\t r: toggle between image sizes" << endl
211          << "\t m: mask image" << endl
212          << "\t e: merged input/mask image" << endl
213          << "\t x: clears current flood" << endl
214          << "\t n: sets new color for flood" << endl
215          << "\t f: toggle between absolute or relative threshold" << endl
216          << "\t b: show/hides flooded area bounding box in source image" << endl
217          << "\t s: show/hides seed point" << endl
218          << "\t ESCAPE | CTRL-Q quits" << endl
219          << "Mouse : " << endl
220          << "\t left button : selects new seed point" << endl
221          << "\t right button : clears current seed" << endl;
222 }
```