```
/*
 * CvProcessor.h
 *
 *  Created on: 21 févr. 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;
```



```

        /**
         * 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_ */
```

```cpp
/*
 * CvProcessor.cpp
 *
 *  Created on: 21 févr. 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;
    }
```



```cpp
    }

    // 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:
```

```
                cout << "no messages";
                break;
            case VERBOSE_ERRORS:
                cout << "unrecoverable errors only";
                break;
            case VERBOSE_WARNINGS:
                cout << "errors and warnings";
                break;
            case VERBOSE_NOTIFICATIONS:
                cout << "errors, warnings and notifications";
                break;
            case VERBOSE_ACTIVITY:
                cout << "All messages";
                break;
            case NBVERBOSELEVEL:
            default:
                cout << "Unknown verobse mode (unchanged)";
                break;
        }
        cout << endl;
    }

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

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

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

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



```
    }

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

    /*
     * Reset mean and std process time in order to re-start computing
     * new mean and std process time values.
     */
    void CvProcessor::resetMeanProcessTime()
    {
        meanProcessTime.reset();
    }


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

    /*
     * Sets Time per feature processing time unit
     * @param value the time per feature value (true or false)
     */
    void CvProcessor::setTimePerFeature(const bool value)
    {
        timePerFeature = value;
    }

    /*
     * Send to stream (for showing processor attributes values)
     * @param out the stream to send to
     * @return a reference to the output stream
     */
    ostream & CvProcessor::toStream(ostream & out) const
    {
        return toStream_Impl<ostream>(out);
    }

    /*
     * Send to output stream operator
     * @param out the output stream to send to
     * @param proc the processor to send to the output stream
     * @return a reference to the output stream used
     */
    ostream & operator <<(ostream & out, const CvProcessor & proc)
    {
        return proc.toStream(out);
    }

    /*
     * Proto instantiation of CvProcessor template method
     * Stream & CvProcessor::toStream_Impl<Stream>(Stream &) const with concrete
     * type ostream
     */
    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
/*
 * CvColorSpaces.h
 *
 *  Created on: 25 févr. 2012
 *      Author: davidroussel
 */

#ifndef CVCOLORSPACES_H_
#define CVCOLORSPACES_H_

#include <vector>
using namespace std;

#include "CvProcessor.h"
#include "Palette.h"

/**
 * Class to process source image into several color spaces such as RGB, HSV and
 * YCbCr
 */
class CvColorSpaces : public virtual CvProcessor
{
    public:
        /**
         * Indices of colors to show in color components
         */
        typedef enum
        {
            BINDEX = 0, //!< index for blue
            GINDEX,     //!< index for green
            RINDEX,     //!< index for red
            MAXINDEX,   //!< index for maximum of RGB (or BGR)
            HINDEX,     //!< index for hue
            CbINDEX,    //!< index for cb
            CrINDEX,    //!< index for cr
            NbShows     //!< NbShows
        } ShowColor;

        /**
         * Image type selected for display
         */
        typedef enum
        {
            INPUT = 0,  //!< color input image is selected for display
            GRAY,       //!< gray input image is selected for display
            RED,        //!< red component from BGR is selected for display
            GREEN,      //!< green component from BGR is selected for display
            BLUE,       //!< blue component from BGR is selected for display
            MAX_BGR,    //!< Maximum of R, G and B components
            XYZ_X,      //!< X component of XYZ space
            XYZ_Y,      //!< Y component of XYZ space
            XYZ_Z,      //!< Z component of XYZ space
            HUE,        //!< Hue component from HSV is selected for display
            SATURATION, //!< Saturation component from HSV is selected for display
            VALUE,      //!< Lightness component from HSV is selected for display
            Y,          //!< Lightness component from YCrCb is selected for display
            Cr,         //!< Green/Magenta Cr component from YCrCb is selected for display
            Cb,         //!< Yellow/Blue Cb component from YCrCb is selected for display
            NbSelected
        } Display;

        /**
         * Hue image display mode
         */
        typedef enum
        {
            HUECOLOR=0,     //!< Normal Hue mode
            HUESATURATE,    //!< Hue*Saturation mode
            HUEVALUE,       //!< Hue*Value mode
            HUEGRAY,        //!< Gray mode
            NBHUES          //!< Number of Hue display modes
        } HueDisplay;

    protected :

        /**
         * Image displayed
         */
        Mat displayImage;

        /**
         * Gray converted image
         */
        Mat inFrameGray;

        /**
         * BGR individual channels
         */
        vector<Mat> bgrChannels;
```



```cpp
        /**
         * BGR colored images built from individual channels and palettes
         */
        Mat bgrColoredChannels[3];

        /**
         * Maximum of B & G channels
         */
        Mat maxBGChannels;

        /**
         * Maximum of maxBGChannels and R channel
         */
        Mat maxBGRChannels;

        /**
         * Colored maximum of B & G channels
         */
        Mat maxBGChannelsColor;

        /**
         * Colored Maximum of maxBGChannels and R channel
         */
        Mat maxBGRChannelsColor;

        /**
         * XYZ floating point converted image
         */
        Mat inFrameXYZ;

        /**
         * XYZ floating point channels
         */
        Mat xyzGrayChannels[3];

        /**
         * XYZ channels normalized to 0..255
         */
        Mat xyzDisplayChannels[3];

        /**
         * HSV converted image
         */
        Mat inFrameHSV;

        /**
         * HSV individual channels
         */
        vector<Mat> hsvChannels;

        /**
         * Hue colored image built from hue component and hsv palette
         */
        Mat hueColorImage;

        /**
         * Hue Mix channels to build hue colored display image
         */
        Mat hueMixChannels[3];

        /**
         * Hue image built from hueMixChannels
         */
        Mat hueMixImage;

        /**
         * Hue colored mixed image normalized from hueMixImage
         */
        Mat hueMixedColorImage;

        /**
         * Mix mode to create hue colored image
         */
        HueDisplay hueDisplay;

        /**
         * YCbCr converted image
         */
        Mat inFrameYCrCb;

        /**
         * YCbCr channels
         */
        vector<Mat> yCrCbChannels;

        /**
         * Cr colored image
         */
        Mat crColoredImage;

```

```
181         /**
182          * Cb colored image
183          */
184         Mat cbColoredImage;
185 
186         /**
187          * Palette to build colored red component image
188          */
189         Palette redMap;
190 
191         /**
192          * Palette to build colored green component image
193          */
194         Palette greenMap;
195 
196         /**
197          * Palette to build colored blue component image
198          */
199         Palette blueMap;
200 
201         /**
202          * Pointers to RGB palettes
203          * pointing respectively to
204          *  - blueMap
205          *  - greenMap
206          *  - redMap
207          */
208         Palette * bgrMap[3];
209 
210         /**
211          * Palette for hue component
212          */
213         Palette hMap;
214 
215         /**
216          * Palette for Cb component
217          */
218         Palette cbMap;
219 
220         /**
221          * Palette for Cr component
222          */
223         Palette crMap;
224 
225         /**
226          * Booleans to choose to display channels as grayscale
227          * or colored images
228          */
229         bool showColorChannel[NbShows];
230 
231         /**
232          * Selected image type to display
233          */
234         Display imageDisplayIndex;
235 
236         /**
237          * True when display image changed since last update
238          */
239         bool displayImageChanged;
240 
241     public :
242         /**
243          * Color spaces constructor
244          * @param inFrame input image
245          */
246         CvColorSpaces(Mat * inFrame);
247 
248         /**
249          * Color spaces destructor
250          */
251         virtual ~CvColorSpaces();
252 
253         /**
254          * Update compute selected image for display according to
255          * selected parameters such as imageDisplayIndex, showColorChannel,
256          * and eventually hueDisplay
257          */
258         virtual void update();
259 
260         /**
261          * Get currently selected image index
262          * @return the currently selected image for display index
263          */
264         Display getDisplayImageIndex();
265 
266         /**
267          * Select image to set in displayImage :
268          *  - INPUT selects input image for display
269          *  - GRAY selects gray converted input image for display
270          *  - RED selects BGR red component image for display
```



```
271          *  - GREEN selects BGR green component image for display
272          *  - BLUE selects BGR blue component image for display
273          *  - HUE selects HSV hue component image for display
274          *  - SATURATION selects HSV saturation component image for display
275          *  - VALUE selects HSV value component image for display
276          *  - Y selects YCrCb Y component image for display
277          *  - Cr selects YCrCb Cr component image for display
278          *  - Cb selects YCrCb Cb component image for display
279          * @param index select the index to select display image
280          */
281         virtual void setDisplayImageIndex(const Display index);
282 
283         /**
284          * Get the color display status for specific channels (such as red,
285          * green, blue, hue ...)
286          * @param c the channel to get color display status:
287          *  - BINDEX color display status for blue component
288          *  - GINDEX color display status for green component
289          *  - RINDEX color display status for red component
290          *  - HINDEX color display status for hue component
291          *  - CbNDEX color display status for Cb component
292          *  - CrNDEX color display status for Cr component
293          * @return the color display status of selected component
294          */
295         bool getColorChannel(const ShowColor c);
296 
297         /**
298          * Sets the color display status of selected component
299          * @param c the selected component:
300          *  - BINDEX color display status for blue component
301          *  - GINDEX color display status for green component
302          *  - RINDEX color display status for red component
303          *  - HINDEX color display status for hue component
304          *  - CbNDEX color display status for Cb component
305          *  - CrNDEX color display status for Cr component
306          * @param value the value to set on the selected component
307          */
308         virtual void setColorChannel(const ShowColor c, const bool value);
309 
310         /**
311          * Get currently selected hue display mode
312          * @return the currenlty selected hue display mode
313          */
314         HueDisplay getHueDisplaymode();
315 
316         /**
317          * Select hue display mode :
318          *  - HUECOLOR Normal Hue mode
319          *  - HUESATURATE Hue*Saturatin mode
320          *  - HUEVALUE Hue*Value mode
321          *  - HUEGRAY Gray mode
322          * @param mode the mode so select
323          */
324         virtual void setHueDisplayMode(const HueDisplay mode);
325 
326         /**
327          * Gets the image selected for display
328          * @return the display image
329          */
330         Mat & getDisplayImage();
331 
332     protected:
333         // ------------------------------------------------------------------
334         // Setup and cleanup attributes
335         // ------------------------------------------------------------------
336         /**
337          * Setup internal attributes according to source image
338          * @param sourceImage a new source image
339          * @param fullSetup full setup is needed when source image is changed
340          * @pre sourceimage is not NULL
341          * @note this method should be reimplemented in sub classes
342          */
343         virtual void setup(Mat * sourceImage, bool fullSetup = true);
344 
345         /**
346          * Clean up internal attributes before changing source image or
347          * cleaning up class before destruction
348          * @note this method should be reimplemented in sub classes
349          */
350         virtual void cleanup();
351 
352         /**
353          * Show Min and Max values and locations for a matrix
354          * @param m the matrix to consider
355          */
356         static void showMinMaxLoc(const Mat & m);
357 
358         /**
359          * Compute Maximum of color images by comparing pixel norm
360          * rather than a per channel max like the openCV max function
```

```
361         * @param src1 the first color (or gray) image
362         * @param src2 the second color (or gray) image
363         * @param dst the color (or gray) destination
364         * @pre the norm max is only computed if arguments are of type CV_8UC3,
365         * otherwise ordinary max is performed
366         */
367        static void normMax(const Mat& src1, const Mat& src2, Mat& dst);
368 };
369
370 #endif /* CVCOLORSPACES_H_ */
```



```
1  /*
2   * CvColorSpaces.cpp
3   *
4   *  Created on: 8 févr. 2012
5   *      Author: davidroussel
6   */
7  #include <cassert>  // for assert
8  #include <iostream> // for cerr
9  using namespace std;
10
11 #include <opencv2/imgproc/imgproc.hpp> // for cvtColor
12
13 #include "mapRed.h"
14 #include "mapGreen.h"
15 #include "mapBlue.h"
16 #include "mapHSV.h"
17 #include "mapCb.h"
18 #include "mapCr.h"
19
20 #include "CvColorSpaces.h"
21
22 /*
23  * Color spaces constructor
24  * @param sourceImage input image
25  */
26 CvColorSpaces::CvColorSpaces(Mat * sourceImage) :
27     CvProcessor(sourceImage),
28     inFrameGray(sourceImage->size(), CV_8UC1),
29     maxBGChannels(sourceImage->size(), CV_8UC1),
30     maxBGRChannels(sourceImage->size(), CV_8UC1),
31     maxBGChannelsColor(sourceImage->size(), CV_8UC3),
32     maxBGRChannelsColor(sourceImage->size(), CV_8UC3),
33     inFrameXYZ(sourceImage->size(), CV_64FC3),
34     inFrameHSV(sourceImage->size(), CV_8UC3),
35     hueColorImage(sourceImage->size(), CV_8UC3),
36     hueMixImage(sourceImage->size(), CV_8UC3),
37     hueMixedColorImage(sourceImage->size(), CV_8UC3),
38     hueDisplay(HUECOLOR),
39     inFrameYCrCb(sourceImage->size(), CV_8UC3),
40     crColoredImage(sourceImage->size(), CV_8UC3),
41     cbColoredImage(sourceImage->size(), CV_8UC3),
42     redMap(mapRed),
43     greenMap(mapGreen),
44     blueMap(mapBlue),
45     hMap(mapHSV),
46     cbMap(mapCb),
47     crMap(mapCr),
48     imageDisplayIndex(INPUT),
49     displayImageChanged(false)
50 {
51     setup(sourceImage, false);
52     addImage("display", &displayImage);
53 }
54
55 /*
56  * Color spaces destructor
57  */
58 CvColorSpaces::~CvColorSpaces()
59 {
60     cleanup();
61 }
62
63 /*
64  * Setup internal attributes according to source image
65  * @param sourceImage a new source image
66  * @param fullSetup full setup is needed when source image is changed
67  * @pre sourceImage is not NULL
68  * @note this method should be reimplemented in sub classes
69  */
70 void CvColorSpaces::setup(Mat * sourceImage, bool fullSetup)
71 {
72 //  clog << "CvColorSpaces::"<< (fullSetup ? "full " : "") <<"setup" << endl;
73
74     assert(sourceImage != NULL);
75
76     CvProcessor::setup(sourceImage, fullSetup);
77
78     // Full setup starting point
79     if (fullSetup) // only when sourceImage changes
80     {
81         inFrameGray.create(sourceImage->size(), CV_8UC1);
82         maxBGChannels.create(sourceImage->size(), CV_8UC1);
83         maxBGRChannels.create(sourceImage->size(), CV_8UC1);
84         maxBGChannelsColor.create(sourceImage->size(), CV_8UC3);
85         maxBGRChannelsColor.create(sourceImage->size(), CV_8UC3);
86         inFrameXYZ.create(sourceImage->size(), CV_64FC3),
87         inFrameHSV.create(sourceImage->size(), CV_8UC3);
88         hueColorImage.create(sourceImage->size(), CV_8UC3);
89         hueMixImage.create(sourceImage->size(), CV_8UC3);
90         hueMixedColorImage.create(sourceImage->size(), CV_8UC3);
```

```
91          inFrameYCrCb.create(sourceImage→size(), CV_8UC3);
92          crColoredImage.create(sourceImage→size(), CV_8UC3);
93          cbColoredImage.create(sourceImage→size(), CV_8UC3);
94          processTime = 0;
95      }
96      else // only at construction
97      {
98          bgrMap[0] = &blueMap;
99          bgrMap[1] = &greenMap;
100         bgrMap[2] = &redMap;
101
102         for (size_t i = 0; i < (size_t) NbShows; i++)
103         {
104             showColorChannel[i] = true;
105         }
106     }
107
108     // Partial setup starting point (in both cases)
109     for (int i=0; i < 3; i++)
110     {
111         bgrChannels.push_back(Mat(sourceImage→size(), CV_8UC1));
112         bgrColoredChannels[i].create(sourceImage→size(), CV_8UC3);
113         xyzGrayChannels[i].create(sourceImage→size(), CV_64FC1);
114         xyzDisplayChannels[i].create(sourceImage→size(), CV_8UC1);
115         hsvChannels.push_back(Mat(sourceImage→size(), CV_8UC1));
116         hueMixChannels[i].create(sourceImage→size(), CV_8UC1);
117         yCrCbChannels.push_back(Mat(sourceImage→size(), CV_8UC1));
118     }
119 }
120
121 /*
122  * Clean up images before changing source image or terminating
123  * CvColorSpaces
124  */
125 void CvColorSpaces::cleanup()
126 {
127 //  clog << "CvColorSpaces::cleanup()" << endl;
128
129     cbColoredImage.release();
130     crColoredImage.release();
131     for (size_t i = 0; i < yCrCbChannels.size(); i++)
132     {
133         yCrCbChannels[i].release();
134     }
135     yCrCbChannels.clear();
136     inFrameYCrCb.release();
137
138     hueMixedColorImage.release();
139     hueMixImage.release();
140     for (size_t i = 0; i < 3; i++)
141     {
142         hueMixChannels[i].release();
143     }
144     hueColorImage.release();
145     for (size_t i = 0; i < hsvChannels.size(); i++)
146     {
147         hsvChannels[i].release();
148     }
149     hsvChannels.clear();
150     inFrameHSV.release();
151
152     for (size_t i = 0; i < bgrChannels.size(); i++)
153     {
154         bgrChannels[i].release();
155         bgrColoredChannels[i].release();
156         xyzGrayChannels[i].release();
157         xyzDisplayChannels[i].release();
158     }
159     bgrChannels.clear();
160
161     inFrameXYZ.release();
162
163     maxBGRChannelsColor.release();
164
165     maxBGChannelsColor.release();
166
167     maxBGRChannels.release();
168
169     maxBGChannels.release();
170
171     inFrameGray.release();
172
173     displayImage.release();
174
175     CvProcessor::cleanup();
176 }
177
178
179 /*
180  * Update compute selected image for display according to
```



```
181  * selected parameters such as imageDisplayIndex, showColorChannel,
182  * and eventually hueDisplay
183  * @return true if display image has changed, false otherwise
184  */
185 void CvColorSpaces::update()
186 {
187     clock_t start, end;
188     start = clock();
189     // ------------------------------------------------------------------
190     // Compute needed images
191     // ------------------------------------------------------------------
192     switch (imageDisplayIndex)
193     {
194         case INPUT:
195             // Ain't got nothin to do here : input image doesn't need to be processed
196             break;
197
198         // ------------------------------------------------------------------
199         // Gray level conversion
200         // ------------------------------------------------------------------
201         case GRAY:
202             // Converts to gray
203             // sourceImage -> inFramegray
204             // TODO Compléter ...
205             break;
206
207         // ------------------------------------------------------------------
208         // RGB Decomposition
209         // ------------------------------------------------------------------
210         case RED:
211         case GREEN:
212         case BLUE:
213         case MAX_BGR:
214             // Split BGR channels : sourceImage -> bgrChannels
215             // TODO Compléter ...
216
217             // TODO Build colored image from channels : red channel leads to a
218             // red colored image, and so on ...
219             // by applying bgrMap[x] on bgrChannels[x] to produce
220             // bgrColoredChannels[x]
221             // bgrChannels[i] -> bgrColoredChannels[i]
222             for (size_t i = 0; i < bgrChannels.size(); i++)
223             {
224                 // TODO Compléter ...
225             }
226
227             if (imageDisplayIndex == MAX_BGR)
228             {
229                 if (¬showColorChannel[MAXINDEX])
230                 {
231                     // Compute maximum of BGR channels using cv::max
232                     // bgrChannels[0 & 1] -> maxBGChannels
233                     // bgrChannels[2] & maxBGChannels -> maxBGRChannels
234                     // TODO à compléter ...
235                 }
236                 else
237                 {
238                     // Compute colored maximum of BGR channels using normMax
239                     // bgrColoredChannels[0 & 1] -> maxBGChannelsColor
240                     // bgrColoredChannels[2] & maxBGChannelsColor -> maxBGRChannelsColor
241                     // TODO à compléter normMax puis utiliser normMax ...
242                 }
243             }
244
245             /*
246              * TODO What are the characteristics of blue component vs
247              * green or red ?
248              * Tip: use gray images instead of colored images to compare
249              * Answer below:
250              *
251              */
252
253             break;
254
255         // ------------------------------------------------------------------
256         // XYZ conversion
257         // ------------------------------------------------------------------
258         case XYZ_X:
259         case XYZ_Y:
260         case XYZ_Z:
261             // Converts to XYZ : sourceImage -> inFrameXYZ
262             // TODO à compléter ...
263
264             // Splits inFrameXYZ to channels xyzGrayChannels
265             // TODO à compléter ...
266
267             // Converts floating point channels to display channels
268             // xyzGrayChannels[...] -> xyzDisplayChannels[...]
269             for( size_t i=0; i < 3; i++)
270             {
```

```
271                 // TODO à compléter ...
272             }
273 
274             /*
275              * TODO What component X, Y or Z looks more like luminance to you ?
276              * Answer below:
277              *
278              */
279 
280             break;
281         // -----------------------------------------------------------------
282         // HSV conversion
283         // -----------------------------------------------------------------
284         case HUE:
285         case SATURATION:
286         case VALUE:
287             // Converts to HSV : sourceImage -> inFrameHSV
288             // TODO à compléter ...
289 
290             // Split HSV channels : inFrameHSV -> hsvChannels
291             // TODO à compléter ...
292 
293             // evt show min/max of H component : should be [0...179]°
294             // showMinMaxLoc(hsvChannels[0]);
295 
296             // Normalize hue from 0 to 255 because hsv colormap (hMap)
297             // applied below expects value within 0 to 255 range
298             // hsvChannels[0] -> hsvChannels[0]
299             // TODO à compléter ...
300 
301             // Build colored Hue image : hsvChannels[0] -> hueColorImage
302             // TODO à compléter ...
303 
304             // Build Mixed Hue Color and (Saturation or Value) image
305             if ((hueDisplay > HUECOLOR) ^ (hueDisplay < HUEGRAY))
306             {
307                 // Creates a 3 channel image from saturation or value channel
308                 // depending on huDisplay value
309                 // hsvChannels -> hueMixChannels
310                 // TODO à compléter ...
311 
312                 // merge mix channels into color image
313                 // hueMixChannels --> hueMixImage
314                 // TODO à compléter ...
315 
316                 // Build colored Hue image \times Saturation or Value
317                 // hueColorImage x hueMixImage -> hueMixedColorImage
318                 // TODO à compléter ...
319             }
320             break;
321 
322             /*
323              * TODO To what other component the V component of HSV space looks
324              * like ?
325              * Tip: Use gray images instead of colored images to compare.
326              * Answer below:
327              *
328              */
329 
330         // -----------------------------------------------------------------
331         // YCbCr conversion
332         // -----------------------------------------------------------------
333         case Y:
334         case Cr:
335         case Cb:
336             // Converts to YCrCb : sourceImage -> inFrameYCrCb
337             // TODO à compléter ...
338 
339             // Split YCrCb channels : inFrameYCrCb -> yCrCbChannels
340             // TODO à compléter ...
341 
342             // Apply palette on cr & cb components
343             // crmap, yCrCbChannels[1] -> crColoredImage
344             // TODO à compléter ...
345             // cbmap, yCrCbChannels[2] -> cbColoredImage
346             // TODO à compléter ...
347             break;
348         default:
349             cerr << "unknown image display index" << imageDisplayIndex << endl;
350             break;
351         /*
352          * TODO How does the Y component compares to the gray component ?
353          * Answer below :
354          */
355 
356         /*
357          * TODO What can you tell about the details in Cr or Cb components vs
358          * the details in the Y component ?
359          * Answer below :
360          */
```



```
361     }
362 
363     // -----------------------------------------------------------------------
364     // select image to display ...
365     // -----------------------------------------------------------------------
366 
367     uchar * previousImageData = displayImage.data;
368 
369     switch (imageDisplayIndex)
370     {
371         case INPUT:
372             displayImage = *sourceImage;
373             break;
374         case GRAY:
375             displayImage = inFrameGray;
376             break;
377         case RED:
378             if (showColorChannel[RINDEX])
379             {
380                 displayImage = bgrColoredChannels[RINDEX];
381             }
382             else
383             {
384                 displayImage = bgrChannels[RINDEX];
385             }
386             break;
387         case GREEN:
388             if (showColorChannel[GINDEX])
389             {
390                 displayImage = bgrColoredChannels[GINDEX];
391             }
392             else
393             {
394                 displayImage = bgrChannels[GINDEX];
395             }
396             break;
397         case BLUE:
398             if (showColorChannel[BINDEX])
399             {
400                 displayImage = bgrColoredChannels[BINDEX];
401             }
402             else
403             {
404                 displayImage = bgrChannels[BINDEX];
405             }
406             break;
407         case MAX_BGR:
408             if (showColorChannel[MAXINDEX])
409             {
410                 displayImage = maxBGRChannelsColor;
411             }
412             else
413             {
414                 displayImage = maxBGRChannels;
415             }
416             break;
417         case XYZ_X:
418             displayImage = xyzDisplayChannels[0];
419             break;
420         case XYZ_Y:
421             displayImage = xyzDisplayChannels[1];
422             break;
423         case XYZ_Z:
424             displayImage = xyzDisplayChannels[2];
425             break;
426         case HUE:
427             switch (hueDisplay)
428             {
429                 case HUECOLOR:
430                     displayImage = hueColorImage;
431                     break;
432                 case HUESATURATE:
433                 case HUEVALUE:
434                     displayImage = hueMixedColorImage;
435                     break;
436                 case HUEGRAY:
437                     displayImage = hsvChannels[0];
438                     break;
439                 case NBHUES:
440                 default:
441                     cerr << "unknown Hue display mode "<< hueDisplay
442                          << endl;
443                     break;
444             }
445             break;
446         case SATURATION:
447             displayImage = hsvChannels[1];
448             break;
449         case VALUE:
450             displayImage = hsvChannels[2];
```

```
451             break;
452         case Y:
453             displayImage = yCrCbChannels[0];
454             break;
455         case Cr:
456             if (showColorChannel[CrINDEX])
457             {
458                 displayImage = crColoredImage;
459             }
460             else
461             {
462                 displayImage = yCrCbChannels[1];
463             }
464             break;
465         case Cb:
466             if (showColorChannel[CbINDEX])
467             {
468                 displayImage = cbColoredImage;
469             }
470             else
471             {
472                 displayImage = yCrCbChannels[2];
473             }
474             break;
475         default:
476             cerr << "unknown display image index " << imageDisplayIndex << endl;
477             displayImage = *sourceImage;
478             break;
479     }
480
481     displayImageChanged = previousImageData ≠ displayImage.data;
482
483     end = clock();
484     processTime = (end - start);
485     meanProcessTime += processTime;
486     if (displayImageChanged)
487     {
488         resetMeanProcessTime();
489     }
490 }
491
492 /*
493  * Gets the image selected for display
494  * @return the display image
495  */
496 Mat & CvColorSpaces::getDisplayImage()
497 {
498     return displayImage;
499 }
500
501 /*
502  * Get currently selected image index
503  * @return the currently selected image for display index
504  */
505 CvColorSpaces::Display CvColorSpaces::getDisplayImageIndex()
506 {
507     return imageDisplayIndex;
508 }
509
510 /*
511  * Select image to set in displayImage :
512  *  - INPUT selects input image for display
513  *  - GRAY selects gray converted input image for display
514  *  - RED selects BGR red component image for display
515  *  - GREEN selects BGR green component image for display
516  *  - BLUE selects BGR blue component image for display
517  *  - HUE selects HSV hue component image for display
518  *  - SATURATION selects HSV saturation component image for display
519  *  - VALUE selects HSV value component image for display
520  *  - Y selects YCrCb Y component image for display
521  *  - Cr selects YCrCb Cr component image for display
522  *  - Cb selects YCrCb Cb component image for display
523  * @param select the index to select display image
524  */
525 void CvColorSpaces::setDisplayImageIndex(const Display index)
526 {
527     if (index < NbSelected)
528     {
529         imageDisplayIndex = index;
530         processTime = 0;
531     }
532     else
533     {
534         cerr << "CvColorSpaces::setDisplayImageIndex : index " << index
535              << " out of bounds" << endl;
536     }
537 }
538
539 /*
540  * Get the color display status for specific channels (such as red,
```



```
541  * green, blue, hue ...)
542  * @param c the channel to get color display status:
543  *  - BINDEX color display status for blue component
544  *  - GINDEX color display status for green component
545  *  - RINDEX color display status for red component
546  *  - MAXINDEX color display for max of RGB
547  *  - HINDEX color display status for hue component
548  *  - CbNDEX color display status for Cb component
549  *  - CrNDEX color display status for Cr component
550  * @return the color display status of selected component
551  */
552 bool CvColorSpaces::getColorChannel(const ShowColor c)
553 {
554     return showColorChannel[c];
555 }
556
557 /*
558  * Sets the color display status of selected component
559  * @param c the selected component:
560  *  - BINDEX color display status for blue component
561  *  - GINDEX color display status for green component
562  *  - RINDEX color display status for red component
563  *  - MAXINDEX color display for max of RGB
564  *  - HINDEX color display status for hue component
565  *  - CbNDEX color display status for Cb component
566  *  - CrNDEX color display status for Cr component
567  * @param value the value to set on the selected component
568  */
569 void CvColorSpaces::setColorChannel(const ShowColor c, const bool value)
570 {
571     if ( c < NbShows)
572     {
573         showColorChannel[c] = value;
574         processTime = 0;
575     }
576     else
577     {
578         cerr << "CvColorSpaces::setColorChannel : index " << c
579              << " out of bounds" << endl;
580     }
581 }
582
583 /*
584  * Get currently selected hue display mode
585  * @return the currenlty selected hue display mode
586  */
587 CvColorSpaces::HueDisplay CvColorSpaces::getHueDisplaymode()
588 {
589     return hueDisplay;
590 }
591
592 /*
593  * Select hue display mode :
594  *  - HUECOLOR Normal Hue mode
595  *  - HUESATURATE Hue*Saturatin mode
596  *  - HUEVALUE Hue*Value mode
597  *  - HUEGRAY Gray mode
598  * @param mode the mode so select
599  */
600 void CvColorSpaces::setHueDisplayMode(const HueDisplay mode)
601 {
602     if (mode < NBHUES)
603     {
604         hueDisplay = mode;
605         processTime = 0;
606     }
607     else
608     {
609         cerr << "CvColorSpaces::setHueDisplayMode : index " << mode
610              << " out of bounds" << endl;
611     }
612 }
613
614 /*
615  * Show Min and Max values and locations for a matrix
616  * @param m the matrix to consider
617  */
618 void CvColorSpaces::showMinMaxLoc(const Mat & m)
619 {
620     // search for min & max value locations
621     double minVal, maxVal;
622     Point minLoc, maxLoc;
623     minMaxLoc(m,&minVal, &maxVal, &minLoc, &maxLoc);
624     clog << "sat values : min = " << minVal << " at (" << minLoc.x
625          << "," << minLoc.y << ") max = " << maxVal << " at ("
626          << maxLoc.x << "," << maxLoc.y << ")" << endl;
627 }
628
629 /*
630  * Compute Maximum of color images by computing a channel wide norm
```

```
631  * to find which is the greatest rather than mixing channels
632  * @param src1 the first color (or gray) image
633  * @param src2 the second color (or gray) image
634  * @param dst the color (or gray) destination
635  */
636 void CvColorSpaces::normMax(const Mat& src1, const Mat& src2, Mat& dst)
637 {
638     // first check if src1, src2 && dst have the same sizes and type
639     if ( (src1.rows ≡ src2.rows) ∧
640          (src1.rows ≡ dst.rows) ∧
641          (src1.cols ≡ src2.cols) ∧
642          (src1.cols ≡ dst.cols) ∧
643          (src1.type() ≡ src2.type()) ∧
644          (src1.type() ≡ dst.type()) )
645     {
646         if (src1.type() ≡ CV_8UC3)
647         {
648             // Compute max by pixel norm rather than mixing pixels
649             for(int i = 0; i < src1.rows; ++i)
650             {
651                 for (int j = 0; j < src1.cols; ++j)
652                 {
653                     /*
654                      * TODO compute pixel norms from src1 & src2 using
655                      * ddot (scalar product) on each pixel
656                      * the result (dst) is the pixel with the greatest norm
657                      */
658                 }
659             }
660         }
661         else
662         {
663             // compute max the regular way with max function
664             cv::max(src1, src2, dst);
665         }
666     }
667     else
668     {
669         // Do nothing
670         cerr << "CvColorSpaces::normMax : incompatible images" << endl;
671     }
672 }
673
674
```



```
1  /*
2   * QcvProcessor.h
3   *
4   *  Created on: 19 févr. 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> ± <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.: " ± %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 févr. 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> ± <std>
 */
QString QcvProcessor::meanStdFormat = numberFormat + QString::fromUtf8("±%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("±") + 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.: " ± %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();
}
```



```
/*
 * QcvColorSpaces.h
 *
 *  Created on: 25 févr. 2012
 *      Author: davidroussel
 */

#ifndef QCVCOLORSPACES_H_
#define QCVCOLORSPACES_H_

#include <QMutex>

#include "QcvProcessor.h"
#include "CvColorSpaces.h"

/**
 * Qt oriented Colorspaces
 */
class QcvColorSpaces : public QcvProcessor, public CvColorSpaces
{
    Q_OBJECT

    protected:
        /**
         * Self lock for operations in multiple threads
         * @note may be NULL if not multithreaded
         */
        QMutex * selfLock;

    public:
        /**
         * QcvColorSpaces constructor
         * @param inFrame the input frame from capture
         * @param imageLock the mutex on source image
         * @param updateThread the thread in which this processor runs
         * @param parent object
         */
        QcvColorSpaces(Mat * inFrame,
                       QMutex * imageLock = NULL,
                       QThread * updateThread = NULL,
                       QObject * parent = NULL);

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

    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 setDisplayImageIndex(const Display index);

        /**
         * Sets the color display status of selected component and sends
         * notification message
         * @param c the selected component:
         * @param value the value to set on the selected component
         */
        void setColorChannel(const ShowColor c, const bool value);

        /**
         * Select hue display mode and sends notification message
         * @param mode the mode so select
         */
        void setHueDisplayMode(const HueDisplay mode);
};

#endif /* QCVCOLORSPACES_H_ */
```

```cpp
/*
 * QcvColorSpaces.cpp
 *
 *  Created on: 25 févr. 2012
 *      Author: davidroussel
 */

#include <QDebug>
#include "QcvColorSpaces.h"

/*
 * QcvColorSpaces constructor
 * @param inFrame the input frame from capture
 * @param imageLock the mutex on source image
 * @param updateThread the thread in which this processor runs
 * @param parent object
 */
QcvColorSpaces::QcvColorSpaces(Mat * inFrame,
                               QMutex * imageLock,
                               QThread * updateThread,
                               QObject * parent) :
    CvProcessor(inFrame),
    QcvProcessor(inFrame, imageLock, updateThread, parent),
    CvColorSpaces(inFrame),
    selfLock(updateThread ≠ NULL ? new QMutex() :
                                   (imageLock ≠ NULL ? imageLock : NULL))
{
    QcvProcessor::numberFormat = QString::fromUtf8("%6.0f");
    QcvProcessor::meanStdFormat = QString::fromUtf8("%5.0f");
}

/*
 * QcvColorSpaces destructor
 */
QcvColorSpaces::~QcvColorSpaces()
{
    if (selfLock ≠ NULL)
    {
        selfLock→lock();
        selfLock→unlock();
        delete selfLock;
    }
}

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

    CvColorSpaces::update();

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

    if (hasSourceLock)
    {
        // qDebug() << "QcvColorSpaces::update : unlock";
        sourceLock→unlock();
    }

    if (displayImageChanged)
    {
        emit imageChanged(&displayImage);
    }

    QcvProcessor::update(); // emits updated signal
}

/*
 * Select image to set in displayImage and sends notification message
 * @param select the index to select display image
 */
void QcvColorSpaces::setDisplayImageIndex(const Display index)
{
    bool hasLock = selfLock ≠ NULL;
    if (hasLock)
    {
        selfLock→lock();
    }

    CvColorSpaces::setDisplayImageIndex(index);

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

    message.clear();
    message.append(tr("Display Image set to: "));
    switch (index)
    {
        case INPUT:
            message.append(tr("Input"));
            break;
        case GRAY:
            message.append(tr("Gray level"));
            break;
        case RED:
            message.append(tr("Red component of RGB space"));
            break;
        case GREEN:
            message.append(tr("Green component of RGB space"));
            break;
        case BLUE:
            message.append(tr("Blue component of RGB space"));
            break;
        case MAX_BGR:
            message.append(tr("Maximum of RGB components"));
            break;
        case XYZ_X:
            message.append(tr("X component of XYZ space"));
            break;
        case XYZ_Y:
            message.append(tr("Y component of XYZ space"));
            break;
        case XYZ_Z:
            message.append(tr("Z component of XYZ space"));
            break;
        case HUE:
            message.append(tr("Hue component of HSV space"));
            break;
        case SATURATION:
            message.append(tr("Saturation component of HSV space"));
            break;
        case VALUE:
            message.append(tr("Value component of HSV space"));
            break;
        case Y:
            message.append(tr("Y component of YCbCr space"));
            break;
        case Cr:
            message.append(tr("Cr component of YCbCr space"));
            break;
        case Cb:
            message.append(tr("Cb component of YCbCr space"));
            break;
        case NbSelected:
        default:
            message.append(tr("Unknown"));
            break;
    }

    emit sendMessage(message, defaultTimeOut);
}

/*
 * Sets the color display status of selected component and sends
 * notification message
 * @param c the selected component:
 * @param value the value to set on the selected component
 */
void QcvColorSpaces::setColorChannel(const ShowColor c, const bool value)
{
    bool hasLock = selfLock ≠ NULL;
    if (hasLock)
    {
        selfLock→lock();
    }

    CvColorSpaces::setColorChannel(c, value);

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

```
181
182     message.clear();
183     message.append(tr("Setting "));
184     switch (c)
185     {
186         case BINDEX:
187             message.append(tr("blue"));
188             break;
189         case GINDEX:
190             message.append(tr("green"));
191             break;
192         case RINDEX:
193             message.append(tr("red"));
194             break;
195         case HINDEX:
196             message.append(tr("hue"));
197             break;
198         case CbINDEX:
199             message.append(tr("Cb"));
200             break;
201         case CrINDEX:
202             message.append(tr("Cr"));
203             break;
204         case NbShows:
205         default:
206             message.append(tr("unknown"));
207             break;
208     }
209     message.append(tr("component show as colored to: "));
210     if (value)
211     {
212         message.append(tr("on"));
213     }
214     else
215     {
216         message.append(tr("off"));
217     }
218
219     emit sendMessage(message, defaultTimeOut);
220
221 }
222
223 /*
224  * Select hue display mode and sends notification message
225  * @param mode the mode so select
226  */
227 void QcvColorSpaces::setHueDisplayMode(const HueDisplay mode)
228 {
229     bool hasLock = selfLock ≠ NULL;
230     if (hasLock)
231     {
232         selfLock→lock();
233     }
234
235     CvColorSpaces::setHueDisplayMode(mode);
236
237     if (hasLock)
238     {
239         selfLock→unlock();
240     }
241
242     message.clear();
243     message.append(tr("Setting hue color display as: "));
244     switch (mode)
245     {
246         case HUECOLOR:
247             message.append(tr("hue only"));
248             break;
249         case HUESATURATE:
250             message.append(tr("hue x saturation"));
251             break;
252         case HUEVALUE:
253             message.append(tr("hue x value"));
254             break;
255         case HUEGRAY:
256             message.append(tr("hue as gray"));
257             break;
258         case NBHUES:
259         default:
260             message.append(tr("unknown"));
261             break;
262     }
263
264     emit sendMessage(message, defaultTimeOut);
265 }
```



```
1  /*
2   * QcvMatWidget.h
3   *
4   *  Created on: 28 févr. 2011
5   *^H      Author: davidroussel
6   */
7
8  #ifndef QCVMATWIDGET_H_
9  #define QCVMATWIDGET_H_
10
11 #include <QWidget>
12 #include <QHBoxLayout>
13 #include <QMouseEvent>
14 #include <QPoint>
15
16 #include <opencv2/core/mat.hpp>
17 using namespace cv;
18
19 /**
20  * Abstract widget to show OpenCV Mat image into QT.
21  * Should be refined in
22  *  - QcvMatWidgetLabel
23  *  - QcvMatWidgetImage
24  *  - QcvMatWidgetGL
25  */
26 class QcvMatWidget : public QWidget
27 {
28     Q_OBJECT
29
30     public:
31         /**
32          * Mouse sensivity of the image widget
33          */
34         typedef enum
35         {
36             /**
37              * Sensitive to no mouse click or drag
38              */
39             MOUSE_NONE = 0,
40             /**
41              * Sensitive to mouse clicks
42              */
43             MOUSE_CLICK = 1,
44             /**
45              * Sensitive to mouse drag
46              */
47             MOUSE_DRAG = 2,
48             /**
49              * Sensitive to mouse click and drag
50              */
51             MOUSE_CLICK_AND_DRAG = 3
52         } MouseSense;
53
54     protected:
55         /**
56          * The widget layout
57          */
58         QHBoxLayout * layout;
59
60         /**
61          * The OpenCV BGR or gray image
62          */
63         Mat * sourceImage;
64
65         /**
66          * The OpenCV RGB image converted from gray or BGR OpenCV image
67          */
68         Mat displayImage;
69
70         /**
71          * Default size when no image has been set
72          */
73         static QSize defaultSize;
74
75         /**
76          * the aspect ratio ofthe image to draw
77          */
78         double aspectRatio;
79
80         /**
81          * Default aspect ratio when image is not set yet
82          */
83         static double defaultAspectRatio;
84
85         /**
86          * Indicate a mouse button is currently pressed within the widget
87          */
88         bool mousePressed;
89
90         /**
```

```
91          * Indicate a mouse is moved after a button has been pressed
92          */
93         bool mouseMoved;
94
95         /**
96          * Mouse sensivity
97          */
98         MouseSense mouseSense;
99
100        /**
101         * mouse pressed location
102         */
103        QPoint pressedPoint;
104
105        /**
106         * Mouse pressed button
107         */
108        Qt::MouseButton pressedButton;
109
110        /**
111         * mouse drag location
112         */
113        QPoint draggedPoint;
114
115        /**
116         * mouse release location
117         */
118        QPoint releasedPoint;
119
120        /**
121         * Selection rectangle
122         */
123        QRect selectionRect;
124
125        /**
126         * Drawing color
127         */
128        static const Scalar drawingColor;
129
130        /**
131         * Drawing width
132         */
133        static const int drawingWidth;
134
135//      size_t count;
136
137        /**
138         * Pixel scale used to draw images.
139         * Used in OpenGL contexts in order to draw images with the right pixel
140         * scale on Hi DPI devices (such as retina screens)
141         */
142        float pixelScale;
143
144    public:
145
146        /**
147         * OpenCV QT Widget default constructor
148         * @param parent parent widget
149         * @param mouseSense mouse sensivity
150         */
151        QcvMatWidget(QWidget *parent = NULL,
152                     MouseSense mouseSense = MOUSE_NONE);
153
154        /**
155         * OpenCV QT Widget constructor
156         * @param sourceImage the source image
157         * @param parent parent widget
158         * @param mouseSense mouse sensivity
159         * @pre sourceImage is not NULL
160         */
161        QcvMatWidget(Mat * sourceImage,
162                     QWidget *parent = NULL,
163                     MouseSense mouseSense = MOUSE_NONE);
164
165        /**
166         * OpenCV Widget destructor.
167         * Releases displayImage.
168         */
169        virtual ~QcvMatWidget(void);
170
171//^H     ^H  /**
172//^H     ^H   * Widget minimum size is set to the contained image size
173//^H     ^H   * @return le size of the image within
174//^H     ^H   */
175//^H     ^H  QSize minimumSize() const;
176
177        /**
178         * Size hint (because size depends on sourceImage properties)
179         * @return size obtained from sourceImage or defaultSize if sourceImage
180         * 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 Grav source image to RGB display image
213         * @pre sourceImage is not NULL
214         * @post BGR or Grav 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_ */
```



```
1  /*
2   * QcvMatWidget.cpp
3   *
4   *  Created on: 28 févr. 2011
5   *      Author: davidroussel
6   */
7  #include <QtDebug>
8 
9  #include <opencv2/imgproc.hpp>
10 
11 #include "QcvMatWidget.h"
12 
13 /*
14  * Default size when no image has been set
15  */
16 QSize QcvMatWidget::defaultSize(640, 480);
17 
18 /*
19  * Default aspect ratio when image is not set yet
20  */
21 double QcvMatWidget::defaultAspectRatio = 4.0/3.0;
22 
23 /*
24  * Drawing color
25  */
26 const Scalar QcvMatWidget::drawingColor(0xFF,0xCC,0x00,0x88);
27 
28 /*
29  * Drawing width
30  */
31 const int QcvMatWidget::drawingWidth(3);
32 
33 /*
34  * OpenCV QT Widget default constructor
35  * @param parent parent widget
36  * @param mouseSense mouse sensivity
37  */
38 QcvMatWidget::QcvMatWidget(QWidget *parent,
39                            MouseSense mouseSense) :
40     QWidget(parent),
41     sourceImage(NULL),
42     aspectRatio(defaultAspectRatio),
43     mousePressed(false),
44     mouseSense(mouseSense),
45 //  count(0)
46     pixelScale(devicePixelRatioF())
47 {
48     setup();
49 }
50 
51 /*
52  * OpenCV QT Widget constructor
53  * @param the source image
54  * @param parent parent widget
55  * @param mouseSense mouse sensivity
56  */
57 QcvMatWidget::QcvMatWidget(Mat * sourceImage,
58                            QWidget *parent,
59                            MouseSense mouseSense) :
60     QWidget(parent),
61     sourceImage(sourceImage),
62     aspectRatio((double)sourceImage->cols / (double)sourceImage->rows),
63     mousePressed(false),
64     mouseSense(mouseSense),
65 //  count(0)
66     pixelScale(devicePixelRatioF())
67 {
68     setup();
69 }
70 
71 /*
72  * OpenCV Widget destructor.
73  * Releases displayImage.
74  */
75 QcvMatWidget::~QcvMatWidget()
76 {
77     displayImage.release();
78 }
79 
80 /*
81  * paint event reimplemented to draw content (in this case only
82  * draw in display image since final rendering method is not yet available)
83  * @param event the paint event
84  */
85 void QcvMatWidget::paintEvent(QPaintEvent * event)
86 {
87     Q_UNUSED(event);
88 
89     if (displayImage.data != NULL)
90     {
```

```
91          // evt draw in image
92          if (mousePressed)
93          {
94              // if MOUSE_CLICK only draws a cross
95              if (mouseSense > MOUSE_NONE)
96              {
97                  if (¬(mouseSense & MOUSE_DRAG))
98                  {
99                      if (mouseMoved)
100                     {
101                         drawCross(draggedPoint);
102                     }
103                     else
104                     {
105                         drawCross(pressedPoint);
106                     }
107                 }
108                 else  // else if MOUSE_DRAG starts drawing a rectangle
109                 {
110                     drawRectangle(selectionRect);
111                 }
112             }
113         }
114     }
115     else
116     {
117         qWarning("QcvMatWidget::paintEvent : image.data is NULL");
118     }
119 }
120
121 /*
122  * Widget setup
123  */
124 void QcvMatWidget::setup()
125 {
126     layout = new QHBoxLayout();
127     layout→setContentsMargins(0,0,0,0);
128     setLayout(layout);
129 }
130
131 /*
132  * Sets new source image
133  * @param sourceImage the new source image
134  */
135 void QcvMatWidget::setSourceImage(Mat * sourceImage)
136 {
137     // qDebug("QcvMatWidget::setSourceImage");
138
139     this→sourceImage = sourceImage;
140
141     // re-setup geometry since height x width may have changed
142     aspectRatio = (double)sourceImage→cols / (double)sourceImage→rows;
143     // qDebug ("aspect ratio changed to %4.2f", aspectRatio);
144
145 }
146
147 /*
148  * Converts BGR or Gray source image to RGB display image
149  * @see #sourceImage
150  * @see #displayImage
151  */
152 void QcvMatWidget::convertImage()
153 {
154 //  qDebug("Convert image");
155
156     int depth = sourceImage→depth();
157     int channels = sourceImage→channels();
158
159     // Converts any image type to RGB format
160     switch (depth)
161     {
162         case CV_8U:
163             switch (channels)
164             {
165                 case 1: // gray level image
166                     cvtColor(*sourceImage, displayImage,CV_GRAY2RGB);
167                     break;
168                 case 3: // Color image (OpenCV produces BGR images)
169                     cvtColor(*sourceImage, displayImage, CV_BGR2RGB);
170                     break;
171                 default:
172                     qFatal("This number of channels (%d) is not supported",
173                            channels);
174                     break;
175             }
176             break;
177         default:
178             qFatal("This image depth (%d) is not implemented in QcvMatWidget",
179                    depth);
180             break;
```



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

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



```
361   * @return size obtained from sourceImage
362   */
363  QSize QcvMatWidget::sizeHint() const
364  {
365      if (sourceImage != NULL)
366      {
367          return QSize(sourceImage->cols, sourceImage->rows);
368      }
369      else
370      {
371          return defaultSize;
372      }
373  }
374
375  /*
376   * Gets Mat widget mouse clickable status
377   * @return true if widget is sensitive to mouse click
378   */
379  bool QcvMatWidget::isMouseClickable() const
380  {
381      return (mouseSense & MOUSE_CLICK);
382  }
383
384  /*
385   * Gets Mat widget mouse dragable status
386   * @return true if widget is sensitive to mouse drag
387   */
388  bool QcvMatWidget::isMouseDragable() const
389  {
390      return (mouseSense & MOUSE_DRAG);
391  }
392
393  /*
394   * Update slot customized to include convertImage before actually
395   * updating
396   */
397  void QcvMatWidget::update()
398  {
399  //  count++;
400  //  qDebug() << "QcvMatWidget::update " << count;
401  //  std::cerr << "{o";
402      convertImage();
403      QWidget::update();
404  //  std::cerr << "}";
405  }
406
407  /*
408   * Recompute pixel scale according to screen pixel scale.
409   * Used with Hi DPI devices (such as retina screens)
410   * @post pixel scale have been updated according to
411   * devicePixelRatioF provided by the QPaintDevice super class
412   */
413  void QcvMatWidget::screenChanged()
414  {
415      pixelScale = devicePixelRatioF();
416  //  qDebug() << "Pixel scale updated to" << pixelScale;
417  }
418
419  // ----------------------------------------------------------------------------
420  // convertImage old algorithm
421  // ----------------------------------------------------------------------------
422  //  int cvIndex, cvLineStart;
423  //  // switch between bit depths
424  //  switch (displayImage.depth())
425  //  {
426  //      case CV_8U:
427  //          switch (displayImage.channels())
428  //          {
429  //              case 1: // Gray 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
```



```
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
```

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

using namespace std;

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

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

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

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

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

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



```cpp
/*
 * QcvMatWidgetGL.h
 *
 *  Created on: 28 févr. 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_ */
```

```cpp
/*
 * QcvMatWidgetGL.cpp
 *
 *  Created on: 28 févr. 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();
}
```



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

#ifndef QCVMATWIDGETIMAGE_H_
#define QCVMATWIDGETIMAGE_H_

#include <QImage>
#include <QPainter>

#include "QcvMatWidget.h"

/**
 * OpenCV Widget for QT with a QPainter to draw image
 */
class QcvMatWidgetImage: public QcvMatWidget
{
    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
```

```
91          */
92         void paintEvent(QPaintEvent * event);
93
94 };
95
96 #endif /* QCVMATWIDGETIMAGE_H_ */
```



```
1  /*
2   * QcvMatWidgetImage.cpp
3   *
4   *  Created on: 31 janv. 2012
5   *      Author: davidroussel
6   */
7
8  #include "QcvMatWidgetImage.h"
9  #include <QPaintEvent>
10 #include <QSizePolicy>
11 #include <QDebug>
12
13 /*
14  * Default Constructor
15  * @param parent parent widget
16  */
17 QcvMatWidgetImage::QcvMatWidgetImage(QWidget *parent,
18                                      MouseSense mouseSense) :
19     QcvMatWidget(parent, mouseSense),
20     qImage(NULL)
21 {
22     setup();
23 }
24
25 /*
26  * Constructor
27  * @param sourceImage source image
28  * @param parent parent widget
29  */
30 QcvMatWidgetImage::QcvMatWidgetImage(Mat * sourceImage,
31                                      QWidget *parent,
32                                      MouseSense mouseSense) :
33     QcvMatWidget(sourceImage, parent, mouseSense),
34     qImage(NULL)
35 {
36     setSourceImage(sourceImage);
37
38     setup();
39 }
40
41 /*
42  * Setup widget (defines size policy)
43  */
44 void QcvMatWidgetImage::setup()
45 {
46 //  qDebug("QcvMatWidgetImage::Setup");
47
48     /*
49      * Customize size policy
50      */
51     QSizePolicy qsp(QSizePolicy::Fixed, QSizePolicy::Fixed);
52     // sets height depends on width (also need to reimplement heightForWidth())
53     qsp.setHeightForWidth(true);
54     setSizePolicy(qsp);
55
56     /*
57      * Customize layout
58      */
59
60     // size policy has changed to call updateGeometry
61     updateGeometry();
62 }
63
64 /*
65  * Destructor.
66  */
67 QcvMatWidgetImage::~QcvMatWidgetImage()
68 {
69     if (qImage ≠ NULL)
70     {
71         delete qImage;
72     }
73 }
74
75 /*
76  * Sets new source image
77  * @param sourceImage the new source image
78  */
79 void QcvMatWidgetImage::setSourceImage(Mat * sourceImage)
80 {
81     if (qImage ≠ NULL)
82     {
83         delete qImage;
84     }
85     // setup and convert image
86     QcvMatWidget::setSourceImage(sourceImage);
87     convertImage();
88     qImage = new QImage((uchar *) displayImage.data, displayImage.cols,
89         displayImage.rows, displayImage.step,
90         QImage::Format_RGB888);
```

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



```
1   /*
2    * QGLImageRender.h
3    *
4    *  Created on: 28 févr. 2011
5    *      Author: davidroussel
6    */
7
8   #ifndef QGLIMAGERENDER_H_
9   #define QGLIMAGERENDER_H_
10
11  #include <QGLWidget>
12  #include <QSize>
13  #include <QSizePolicy>
14
15  #include <opencv2/core/mat.hpp>
16  using namespace cv;
17
18  /**
19   * A Class allowing to draw OpenCV Mat images using OpenGL
20   */
21  class QGLImageRender: public QGLWidget
22  {
23      private:
24          /**
25           * The RGB image to draw
26           */
27          Mat image;
28
29          /**
30           * The pixel format:
31           *  - GL_RGB for RGB converted images
32           *  - GL_BGR for OpenCV natural format
33           */
34          GLenum pixelFormat;
35
36          /**
37           * Pixel scale pointer from container
38           */
39          float * pixelScale;
40
41      public:
42          /**
43           * QGLImageRender Constructor
44           * @param image the RGB image to draw in the pixel buffer
45           * @param format pixel format
46           * @param pixelScale pixel scale pointer from container
47           * @param parent the parent widget
48           */
49          QGLImageRender(const Mat & image,
50                         const GLenum format = GL_RGB,
51                         float * pixelScale = NULL,
52                         QWidget *parent = NULL);
53
54          /**
55           * QGLImageRender destructor
56           */
57          virtual ~QGLImageRender();
58
59          /**
60           * Size hint
61           * @return Qsize containing size hint
62           */
63          QSize sizeHint () const;
64
65          /**
66           * Minimum Size hint
67           * @return QSize containing the minimum size hint
68           */
69          QSize minimumSizeHint() const;
70
71          /**
72           * Size Policy for this widget
73           * @return A No resize at all policy
74           */
75          QSizePolicy sizePolicy () const;
76
77      protected :
78          /**
79           * Initialise GL drawing (called once on each QGLContext)
80           */
81          void initializeGL();
82          /**
83           * Paint GL : called whenever the widget needs to be painted
84           */
85          void paintGL();
86          /**
87           * Resize GL : called whenever the widget has been resized
88           */
89          void resizeGL(int width, int height);
90  };
```

```
91
92  #endif /* QGLIMAGERENDER_H_ */
```



```
1   /*
2    * QGLImageRender.cpp
3    *
4    *  Created on: 28 févr. 2011
5    *      Author: davidroussel
6    */
7   #include <QDebug>
8   #ifdef __APPLE__
9       #include <gl.h>
10      #include <glu.h>
11  #else
12      #include <GL/gl.h>
13      #include <GL/glu.h>
14  #endif
15  #include "QGLImageRender.h"
16
17  /*
18   * QGLImageRender Constructor
19   * @param image the RGB image to draw in the pixel buffer
20   * @param format pixel format
21   * @param pixelScale pixel scale pointer from container
22   * @param parent the parent widget
23   */
24  QGLImageRender::QGLImageRender(const Mat & image,
25                                 const GLenum format,
26                                 float * pixelScale,
27                                 QWidget *parent) :
28      QGLWidget(parent),
29      image(image),
30      pixelFormat(format),
31      pixelScale(pixelScale)
32  {
33      if (¬doubleBuffer())
34      {
35          qWarning("QGLImageRender::QGLImageRender caution : no double buffer");
36      }
37      if (this→image.data ≡ NULL)
38      {
39          qWarning("QGLImageRender::QGLImageRender caution : image data is null");
40      }
41      if (this→pixelScale ≡ NULL)
42      {
43          qCritical("QGLImageRender::QGLImageRender caution : pixel scale is null");
44      }
45  }
46
47  QGLImageRender::~QGLImageRender()
48  {
49      image.release();
50  }
51
52  void QGLImageRender::initializeGL()
53  {
54  //  qDebug("GL init ...");
55      glClearColor(0.0, 0.0, 0.0, 0.0);
56  //  glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
57  }
58
59  void QGLImageRender::resizeGL(int width, int height)
60  {
61  //  qDebug("GL resizeGL ...");
62
63      glViewport(0, 0, (GLsizei) width, (GLsizei) height);
64
65      glMatrixMode(GL_PROJECTION);
66      glLoadIdentity();
67      if (image.data ≠ NULL)
68      {
69          glOrtho(0, (GLdouble) image.cols, 0, (GLdouble) image.rows, 1.0, -1.0);
70      }
71
72      glMatrixMode(GL_MODELVIEW);
73      glLoadIdentity();
74  }
75
76  void QGLImageRender::paintGL()
77  {
78  //  qDebug("GL drawing pixels ...");
79
80      glClear(GL_COLOR_BUFFER_BIT);
81
82      if (image.data ≠ NULL)
83      {
84          /* apply the right translate so the image drawing starts top left */
85          glRasterPos4f(0.0f,(GLfloat)(image.rows), 0.0f, 1.0f);
86          /*
87           * for hi dpi displays
88           *  typically pixelScale =
89           *  - 1.0 for normal displays
90           *  - 2.0 for hidpi displays
```

```
91          */
92          glPixelZoom(*pixelScale, -(*pixelScale));
93
94          glDrawPixels(image.cols, image.rows, pixelFormat,
95                       GL_UNSIGNED_BYTE, image.data);
96          // In any circumstance you should NOT use glFlush or swapBuffers() here
97      }
98      else
99      {
100         qWarning("Nothing to draw");
101     }
102 }
103
104 QSize QGLImageRender::sizeHint () const
105 {
106     return minimumSizeHint();
107 }
108
109 QSize QGLImageRender::minimumSizeHint() const
110 {
111     if (image.data != NULL)
112     {
113         return QSize(image.cols, image.rows);
114     }
115     else
116     {
117         qWarning("QGLImageRender::minimumSizeHint : probably invalid sizeHint");
118         return QSize(320,240);
119     }
120 }
121
122 QSizePolicy QGLImageRender::sizePolicy () const
123 {
124     return QSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
125 }
```



```
1  /*
2   * QcvVideoCapture.h
3   *
4   *  Created on: 29 janv. 2012
5   *      Author: davidroussel
6   */
7
8  #ifndef QCVVIDEOCAPTURE_H_
9  #define QCVVIDEOCAPTURE_H_
10
11 #include <QObject>
12 #include <QSize>
13 #include <QTimer>
14 #include <QThread>
15 #include <QMutex>
16
17 #include <opencv2/highgui/highgui.hpp>
18 using namespace cv;
19
20 /**
21  * Qt Class for capturing videos from cameras of files with OpenCV.
22  * QcvVideoCapture opens streams and refresh itself automatically.
23  * When frame has been refreshed a signal is emitted.
24  */
25 class QcvVideoCapture: public QObject
26 {
27     Q_OBJECT
28
29     private:
30
31         /**
32          * file name used to open video file.
33          * Used to reopen video file when video is finished.
34          */
35         QString filename;
36
37         /**
38          * Video capture instance
39          * @warning capture is regularly updated by a timer, but can also be
40          * manipulated by other methods (such as #setDirectSize). So capture
41          * access for new images should be protected by a mutex to ensure
42          * atomic access to capture object at a time.
43          */
44         VideoCapture capture;
45
46         /**
47          * refresh timer
48          */
49         QTimer * timer;
50
51         /**
52          * Independant thread to update capture.
53          * If independant thread is required, then update method is called
54          * from within this thread. Otherwise, update method is called from
55          * main thread.
56          */
57         QThread * updateThread;
58
59         /**
60          * Mutex lock to ensure atomic access capture grabbing new image.
61          * @warning if QcvVideoCapture object is not updated in the
62          * #updateThread, then trying to lock mutex multiple times with
63          * mutex.lock() will lead to a deadlock, so if this object has no
64          * #updateThread (if #updateThread == NULL) we should use
65          * mutex.tryLock() instead and give up when lock can't be obtained with
66          * tryLock(). For instance when tryLock into #update method fails, this
67          * means that capture object is locked in some other method, so we don't
68          * grab any new image this time and hope, we'll be able to do it next
69          * time #update will be called.
70          */
71         QMutex mutex;
72
73         /**
74          * Mutex lock state memory to avoid locking the mutex multiple times
75          * across multiple methods. When a mutex.lock() is performed locked
76          * should be set to true until mutex.unlock(). Hence, if a method
77          * requiring lock is performed, a second lock is avoided by checking
78          * this attribute.
79          */
80         size_t lockLevel;
81
82         /**
83          * Image Matrix to obtain from capture
84          */
85         Mat image;
86
87         /**
88          * image resized (if required)
89          */
90         Mat imageResized;
```

```
91
92          /**
93           * [resized] image flipped (if required)
94           */
95          Mat imageFlipped;
96
97          /**
98           * Image converted for display:
99           *  - scaled
100          *  - flipped horizontally
101          *  - converted to gray
102          */
103         Mat imageDisplay;
104
105         /**
106          * Live video indication (from cam)
107          */
108         bool liveVideo;
109
110         /**
111          * flipVideo to mirror image
112          */
113         bool flipVideo;
114
115         /**
116          * scale image to preferred width and height
117          */
118         bool resize;
119
120         /**
121          * scaling is performed into capture rather than through cv::resize
122          * function
123          */
124         bool directResize;
125
126         /**
127          * image converted to gray
128          */
129         bool gray;
130
131         /**
132          * Allow capture to skip an image capture when lock can't be acquired
133          * before grabbing a new image. Otherwise we'll wait until the lock
134          * is acquired before grabbing an new image. The lock might be acquired
135          * by another lenghty thread/processor during image processing.
136          */
137         bool skip;
138
139         /**
140          * Current Image size (might be different from natural capture image
141          * size)
142          */
143         QSize size;
144
145         /**
146          * Capture natural image size (without resizing)
147          */
148         QSize originalSize;
149
150         /**
151          * Capture frame rate obtained either by getting the CV_CAP_PROP_FPS
152          * VideoCapture property or by computing capture time on several images
153          * @see #grabInterval
154          */
155         double frameRate;
156
157         /**
158          * default time interval between refresh
159          */
160         static int defaultFrameDelay;
161
162         /**
163          * Number of frames to test frame rate
164          */
165         static size_t defaultFrameNumberTest;
166
167         /**
168          * Status message to send when something changes
169          */
170         QString statusMessage;
171
172         /**
173          * Default message showing time (at least 2000 ms)
174          */
175         static int messageDelay;
176
177     public:
178         /**
179          * QcvVideoCapture constructor.
180          * Opens the default camera (0)
```



```
181          * @param flipVideo mirror image status
182          * @param gray convert image to gray status
183          * @param skip indicates capture can skip an image. When the capture
184          * result has not been processed yet, or when false that capture should
185          * wait for the result to be processed before grabbing a new image.
186          * This only applies when #updateThread is not NULL.
187          * @param width desired width or 0 to keep capture width
188          * @param height desired height or 0 to keep capture height
189          * otherwise capture is updated in the current thread.
190          * @param updateThread the thread used to run this capture
191          * @param parent the parent QObject
192          */
193         QcvVideoCapture(const bool flipVideo = false,
194                         const bool gray = false,
195                         const bool skip = true,
196                         const unsigned int  width = 0,
197                         const unsigned int height = 0,
198                         QThread * updateThread = NULL,
199                         QObject * parent = NULL);
200
201         /**
202          * QcvVideoCapture constructor with device Id
203          * @param deviceId the id of the camera to open
204          * @param flipVideo mirror image
205          * @param gray convert image to gray
206          * @param skip indicates capture can skip an image. When the capture
207          * result has not been processed yet, or when false that capture should
208          * wait for the result to be processed before grabbing a new image.
209          * This only applies when #updateThread is not NULL.
210          * @param width desired width or 0 to keep capture width
211          * @param height desired height or 0 to keep capture height
212          * @param updateThread the thread used to run this capture
213          * @param parent the parent QObject
214          */
215         QcvVideoCapture(const int deviceId,
216                         const bool flipVideo = false,
217                         const bool gray = false,
218                         const bool skip = true,
219                         const unsigned int  width = 0,
220                         const unsigned int height = 0,
221                         QThread * updateThread = NULL,
222                         QObject * parent = NULL);
223
224         /**
225          * QcvVideoCapture constructor from file name
226          * @param fileName video file to open
227          * @param flipVideo mirror image
228          * @param gray convert image to gray
229          * @param skip indicates capture can skip an image. When the capture
230          * result has not been processed yet, or when false that capture should
231          * wait for the result to be processed before grabbing a new image.
232          * This only applies when #updateThread is not NULL.
233          * @param width desired width or 0 to keep capture width
234          * @param height desired height or 0 to keep capture height
235          * @param updateThread the thread used to run this capture
236          * @param parent the parent QObject
237          */
238         QcvVideoCapture(const QString & fileName,
239                         const bool flipVideo = false,
240                         const bool gray = false,
241                         const bool skip = true,
242                         const unsigned int  width = 0,
243                         const unsigned int height = 0,
244                         QThread * updateThread = NULL,
245                         QObject * parent = NULL);
246
247         /**
248          * QcvVideoCapture destructor.
249          * releases video capture and image
250          */
251         virtual ~QcvVideoCapture();
252
253         /**
254          * Size accessor
255          * @return the image size
256          */
257         const QSize & getSize() const;
258
259         /**
260          * Gets resize state.
261          * @return true if imageDisplay have been resized to preferred width and
262          * height, false otherwise
263          */
264         bool isResized() const;
265
266         /**
267          * Gets direct resize state.
268          * @return true if image can be resized directly into capture.
269          * @note direct resize capabilities are tested into #grabTest which is
270          * called in all constructors. So #isDirectResizeable should not be
```

```
271         * called before #grabTest
272         */
273        bool isDirectResizeable() const;
274
275        /**
276         * Gets video flipping status
277         * @return flipped video status
278         */
279        bool isFlipVideo() const;
280
281        /**
282         * Gets video gray converted status
283         * @return the converted to gray status
284         */
285        bool isGray() const;
286
287        /**
288         * Gets the image skipping policy
289         * @return true if new image can be skipped when previous one has not
290         * been processed yet, false otherwise.
291         */
292        bool isSkippable() const;
293
294        /**
295         * Gets the current frame rate
296         * @return the current frame rate
297         */
298        double getFrameRate() const;
299
300        /**
301         * Image accessor
302         * @return the image to display
303         */
304        Mat * getImage();
305
306        /**
307         * The source image mutex
308         * @return  the mutex used on image access
309         */
310        QMutex * getMutex();
311
312    public slots:
313        /**
314         * Open new device Id
315         * @param deviceId device number to open
316         * @param width desired width or 0 to keep capture width
317         * @param height desired height or 0 to keep capture height
318         * @return true if device has been opened and checked and timer launched
319         */
320        bool open(const int deviceId,
321                  const unsigned int width = 0,
322                  const unsigned int height = 0);
323
324        /**
325         * Open new video file
326         * @param fileName video file to open
327         * @param width desired width or 0 to keep capture width
328         * @param height desired height or 0 to keep capture height
329         * @return true if video has been opened and timer launched
330         */
331        bool open(const QString & fileName,
332                  const unsigned int width = 0,
333                  const unsigned int height = 0);
334        /**
335         * Sets video flipping
336         * @param flipVideo flipped video or not
337         */
338        void setFlipVideo(const bool flipVideo);
339
340        /**
341         * Sets video conversion to gray
342         * @param grayConversion the gray conversion status
343         */
344        void setGray(const bool grayConversion);
345
346        /**
347         * Sets #imageDisplay size according to preferred width and height
348         * @param size new desired size to set
349         * @param alreadyLocked mutex lock has already been aquired so setSize does not have
350         * to acquire the lock
351         * @pre a first image have been grabbed
352         */
353        void setSize(const QSize & size);
354
355    private:
356        /**
357         * Performs a grab test to fill #image.
358         * if capture is opened then tries to grab and if grab succeeds then
359         * tries to retrieve image from grab and sets image size.
360         * @return true if capture is opened and successfully grabbed a first
```



```
361         * frame into #image. false otherwise
362         * @post Moreover this method determines if direct resizing is allowed
363         * on this capture instance by trying to set
364         * CV_CAP_PROP_FRAME_WIDTH and CV_CAP_PROP_FRAME_HEIGHT.
365         */
366        bool grabTest();
367
368        /**
369         * Get or compute interval between two frames in ms and sets the
370         * frameRate attribute.
371         * Tries to get CV_CAP_PROP_FPS from capture and if not available
372         * computes times between frames by grabbing defaultNumberTest images
373         * @return interval between two frames
374         * @param message message passed to grabInterval and display ahead of
375         * the framerate computed during grabInterval
376         * @pre capture is already instanciated
377         * @post message indicating frame rate has been emitted and interval
378         * between two frames has been returned
379         */
380        int grabInterval(const QString & message);
381
382        /**
383         * Sets #imageDisplay size according to preferred width and height
384         * @param width desired width
385         * @param height desired height
386         * @pre a first image have been grabbed
387         */
388        void setSize(const unsigned int width,
389                     const unsigned int height);
390
391        /**
392         * Tries to set capture size directly on capture by setting properties.
393         *  - CV_CAP_PROP_FRAME_WIDTH to set frame width
394         *  - CV_CAP_PROP_FRAME_HEIGHT to set frame height
395         * @param width the width property to set on capture
396         * @param height the height property to set on capture
397         * @return true if capture is opened and if width and height have been
398         * set successfully through @code capture.set(...) @endcode. Returns
399         * false otherwise.
400         * @post if at least width or height have been set successfully, capture
401         * image is released then updated again so it will have the right
402         * dimensions.
403         * @warning if mutex lock can't be obtained to ensure atomic access to
404         * capture object, then we start recursing until we obtain that lock,
405         * which is gross and should be fixed !!!
406         */
407        bool setDirectSize(const unsigned int width, const unsigned int height);
408
409    protected slots:
410        /**
411         * update slot 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
```



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

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



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

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



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

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



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

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



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

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



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

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



```
1  /*
2   * CaptureFactory.h
3   *
4   *  Created on: 11 févr. 2012
5   *      Author: davidroussel
6   */
7
8  #ifndef CAPTUREFACTORY_H_
9  #define CAPTUREFACTORY_H_
10
11 #include <QString>
12 #include <QStringList>
13 #include <QThread>
14 #include "QcvVideoCapture.h"
15
16 /**
17  * Capture Factory creates QcvVideoCapture from arguments list
18  */
19 class CaptureFactory
20 {
21     private:
22         /**
23          * The capture instance to create
24          */
25         QcvVideoCapture *capture;
26
27         /**
28          * Device number to open. Generally :
29          *  - 0 is internal or fisrt camera
30          *  - 1 is external or second camera
31          */
32         int deviceNumber;
33
34         /**
35          * Indicates capture opens camera or file.
36          * Default value is true
37          */
38         bool liveVideo;
39
40         /**
41          * Video should be flipped horizontally for mirror effect
42          * Default value is false
43          */
44         bool flippedVideo;
45
46         /**
47          * Video should be converted to gray during capture.
48          * Default value is false
49          */
50         bool grayVideo;
51
52         /**
53          * Capture can skip capturing new image when previous image has not
54          * been processed yet, or can wait for the previous image to be
55          * processed before grabbing a new image.
56          */
57         bool skipImages;
58
59         /**
60          * Video preferred width (evt resize video)
61          * Default value is 0 which means no preferred width
62          */
63         int preferredWidth;
64
65         /**
66          * Video preferred height (evt resize video)
67          * Default value is 0 which means no preferred height
68          */
69         int preferredHeight;
70
71         /**
72          * Path to video file
73          */
74         QString videoPath;
75
76     public:
77         /**
78          * Capture Factory constructor.
79          * Arguments can be
80          *  - [-d | --device] <device number> : camera number
81          *  - [-f | --file] <filename> : video file name
82          *  - [-m | --mirror] : flip image horizontally
83          *  - [-g | --gray] : convert to gray level
84          *  - [-s | --size] <width>x<height>: preferred width and height
85          * @param argList program the argument list provided as a list of
86          * strings
87          */
88         CaptureFactory(const QStringList & argList);
89
90         /**
```

```
91          * Capture factory destructor
92          */
93         virtual ~CaptureFactory();
94
95         /**
96          * Set the capture to live (webcam) or file source
97          * @param live the video source
98          */
99         void setLiveVideo(const bool live);
100
101        /**
102         * Set device number to use when instanciating the capture with
103         * live video.
104         * @param deviceNumber the device number to use
105         */
106        void setDeviceNumber(const int deviceNumber);
107
108        /**
109         * Set path to video file when #liveVideo is false
110         * @param path the path to the video file source
111         */
112        void setFile(const QString & path);
113
114        /**
115         * Set video horizontal flip state (useful for selfies)
116         * @param flipped the horizontal flip state
117         */
118        void setFlipped(const bool flipped);
119
120        /**
121         * Set gray conversion
122         * @param gray the gray conversion state
123         */
124        void setGray(const bool gray);
125
126        /**
127         * Set video grabbing skippable. When true, grabbing is skipped when
128         * previously grabbed image has not been processed yet. Otherwise,
129         * grabbing new image wait for the previous image to be processed.
130         * This only applies if capture is run in a separate thread.
131         * @param skip the video grabbing skippable state
132         */
133        void setSkippable(const bool skip);
134
135        /**
136         * Set video size (independently of video source actual size)
137         * @param width the desired image width
138         * @param height the desired image height
139         */
140        void setSize(const size_t width, const size_t height);
141
142        /**
143         * Set video size (independently of video source actual size)
144         * @param size the desired video size
145         */
146        void setSize(const QSize & size);
147
148        /**
149         * Provide capture instanciated according to values
150         * extracted from argument lists
151         * @param updateThread the thread to run this capture or NULL if this
152         * capture run in the current thread
153         * @return the new capture instance
154         */
155        QcvVideoCapture * getCaptureInstance(QThread * updatethread = NULL);
156 };
157
158 #endif /* CAPTUREFACTORY_H_ */
```



```
1  /*
2   * CaptureFactory.cpp
3   *
4   *  Created on: 11 févr. 2012
5   *      Author: davidroussel
6   */
7
8  #include <cstdlib>  // for NULL
9  #include <QDebug>
10 #include <QFile>
11 #include <QtGlobal>
12 #include <QStringListIterator>
13 #include "CaptureFactory.h"
14
15 /*
16  * Capture Factory constructor.
17  * Arguments can be
18  *  - [-d | --device] <device number> : camera number
19  *  - [-f | --file] <filename> : video file name
20  *  - [-m | --mirror] : flip image horizontally
21  *  - [-g | --gray] : convert to gray level
22  *  - [-s | --size] <width>x<height>: preferred width and height
23  * @param argList program the argument list provided as a list of
24  * strings
25  */
26 CaptureFactory::CaptureFactory(const QStringList & argList) :
27     capture(NULL),
28     deviceNumber(0),
29     liveVideo(true),
30     flippedVideo(false),
31     grayVideo(false),
32     skipImages(false),
33     preferredWidth(0),
34     preferredHeight(0),
35     videoPath()
36 {
37     // C++ Like iterator
38     // for (QStringList::const_iterator it = argList.begin(); it != argList.end(); ++it)
39     // Java like iterator (because we use hasNext multiple times)
40     for (QListIterator<QString> it(argList); it.hasNext(); )
41     {
42         QString currentArg(it.next());
43
44         if (currentArg == "-d" || currentArg =="--device")
45         {
46             // Next argument should be device number integer
47             if (it.hasNext())
48             {
49                 QString deviceString(it.next());
50                 bool convertOk;
51                 deviceNumber = deviceString.toInt(&convertOk,10);
52                 if (!convertOk || deviceNumber < 0)
53                 {
54                     qWarning("Warning: Invalid device number %d",deviceNumber);
55                     deviceNumber = 0;
56                 }
57                 liveVideo = true;
58             }
59             else
60             {
61                 qWarning("Warning: device tag found with no following device number");
62             }
63         }
64         else if (currentArg == "-v" || currentArg == "--video")
65         {
66             // Next argument should be a path name to video file or URL
67             if (it.hasNext())
68             {
69                 videoPath = it.next();
70                 liveVideo = false;
71             }
72             else
73             {
74                 qWarning("file tag found with no following filename");
75             }
76         }
77         else if (currentArg == "-m" || currentArg == "--mirror")
78         {
79             flippedVideo = true;
80         }
81         else if (currentArg == "-g" || currentArg == "--gray")
82         {
83             grayVideo = true;
84         }
85         else if (currentArg == "-k" || currentArg == "--skip")
86         {
87             skipImages = true;
88         }
89         else if (currentArg == "-s" || currentArg == "--size")
90         {
```

```
91              if (it.hasNext())
92              {
93                  // search for <width>x<height>
94                  QString sizeString = it.next();
95                  int xIndex = sizeString.indexOf(QChar('x'), 0,
96                          Qt::CaseInsensitive);
97                  if (xIndex ≠ -1)
98                  {
99                      QString widthString = sizeString.left(xIndex);
100                     preferredWidth = widthString.toUInt();
101                     qDebug("preferred width is %d", preferredWidth);
102
103                     QString heightString = sizeString.remove(0, xIndex+1);
104                     preferredHeight = heightString.toUInt();
105                     qDebug("preferred height is %d", preferredHeight);
106                 }
107                 else
108                 {
109                     qWarning("invalid <width>x<height>");
110                 }
111             }
112             else
113             {
114                 qWarning("size not found after --size");
115             }
116         }
117     }
118 }
119
120 /*
121  * Capture factory destructor
122  */
123 CaptureFactory::~CaptureFactory()
124 {
125 }
126
127 /*
128  * Set the capture to live (webcam) or file source
129  * @param live the video source
130  */
131 void CaptureFactory::setLiveVideo(const bool live)
132 {
133     liveVideo = live;
134 }
135
136 /*
137  * Set device number to use when instanciating the capture with
138  * live video.
139  * @param deviceNumber the device number to use
140  */
141 void CaptureFactory::setDeviceNumber(const int deviceNumber)
142 {
143     if (deviceNumber ≥ 0)
144     {
145         this→deviceNumber = deviceNumber;
146     }
147     else
148     {
149         qWarning("CaptureFactory::setDeviceNumber: invalid number %d", deviceNumber);
150     }
151 }
152
153 /*
154  * Set path to video file when #liveVideo is false
155  * @param path the path to the video file source
156  */
157 void CaptureFactory::setFile(const QString & path)
158 {
159     if (QFile::exists(path))
160     {
161         videoPath = path;
162     }
163     else
164     {
165         qWarning() << QObject::tr("CaptureFactory::setFile: path") << path
166                    << QObject::tr(" does not exist");
167     }
168 }
169
170 /*
171  * Set video horizontal flip state (useful for selfies)
172  * @param flipped the horizontal flip state
173  */
174 void CaptureFactory::setFlipped(const bool flipped)
175 {
176     flippedVideo = flipped;
177 }
178
179 /*
180  * Set gray conversion
```



```
181  * @param gray the gray conversion state
182  */
183 void CaptureFactory::setGray(const bool gray)
184 {
185     grayVideo = gray;
186 }
187
188 /*
189  * Set video grabbing skippable. When true, grabbing is skipped when
190  * previously grabbed image has not been processed yet. Otherwise,
191  * grabbing new image wait for the previous image to be processed.
192  * This only applies if capture is run in a separate thread.
193  * @param skip the video grabbing skippable state
194  */
195 void CaptureFactory::setSkippable(const bool skip)
196 {
197     skipImages = skip;
198 }
199
200 /*
201  * Set video size (independently of video source actual size)
202  * @param width the desired image width
203  * @param height the desired image height
204  */
205 void CaptureFactory::setSize(const size_t width, const size_t height)
206 {
207     preferredWidth = (int)width;
208     preferredHeight = (int)height;
209 }
210
211 /*
212  * Set video size (independently of video source actual size)
213  * @param size the desired video size
214  */
215 void CaptureFactory::setSize(const QSize & size)
216 {
217     preferredWidth = size.width();
218     preferredHeight = size.height();
219 }
220
221 /*
222  * Provide capture instanciated according to values
223  * extracted from argument lists
224  * @param updateThread the thread to run this capture or NULL if this
225  * capture run in the current thread
226  * @return the new capture instance
227  */
228 QcvVideoCapture * CaptureFactory::getCaptureInstance(QThread * updateThread)
229 {
230     // ------------------------------------------------------------------------
231     // Opening Video Capture
232     // ------------------------------------------------------------------------
233     if (liveVideo)
234     {
235         qDebug() << "opening device # " << deviceNumber;
236     }
237     else
238     {
239         qDebug() << "opening video file " << videoPath;
240     }
241
242     qDebug() << "Opening ";
243     if (liveVideo)
244     {
245         // Live video feed
246         qDebug() << "Live Video ... from camera # " << deviceNumber;
247         capture = new QcvVideoCapture(deviceNumber,
248                                       flippedVideo,
249                                       grayVideo,
250                                       skipImages,
251                                       preferredWidth,
252                                       preferredHeight,
253                                       updateThread);
254     }
255     else
256     {
257         // Video file or stream
258         qDebug() << videoPath << " ... ";
259         capture = new QcvVideoCapture(videoPath,
260                                       flippedVideo,
261                                       grayVideo,
262                                       skipImages,
263                                       preferredWidth,
264                                       preferredHeight,
265                                       updateThread);
266     }
267
268     return capture;
269 }
```

```
/*
 * Palette.h
 *
 *  Created on: 13 sept. 2010
 *      Author: David Roussel
 */

#ifndef PALETTE_H_
#define PALETTE_H_

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

#include <vector>
using namespace std;

/**
 * Palette loads colormap from files or static arrays and apply it to a single
 * channel image (8 bits single channel image : CV_8UC1) in order to rebuild a
 * BGR image featuring the colors in the palette.
 * A Colormap is composed of 256 RGB values that should be applied for each
 * level (from 0 to 255) of the single channel image.
 * colormap is applied
 * @warning colormap are stored in RGB order, but OpenCV images are stored in
 * BGR order.
 */
class Palette
{
    protected:
        /**
         * RGB colormap
         *  - Red colormap is first component
         *  - Green colormap is second component
         *  - Blue colormap is third component
         */
        vector<Mat> colormap;

        /**
         * Minimum value in the palette.
         * In order to check invalid values in the palette
         */
        int minValue;

        /**
         * Maximum value in the palette.
         * In order to check invalid values in the palette
         */
        int maxValue;

        /**
         * BGR BGRChannels of the resulting image
         */
        vector<Mat> BGRChannels;

        /**
         * Checks if channels have been allocated yet.
         * Channels may be allocated only when they are not or when they
         * does not fit the image dimension provided in applyPalette methods.
         * In this case, if BGRChannels have been allocated they are released and
         * recreated.
         */
        bool channelsAllocated;

        /**
         * Number of elements in the colormap : 256
         */
        static const size_t CMAPSIZE;

        /**
         * Number of components in the color image
         */
        static const size_t COMPSIZE;

    public:
        /**
         * Constructor from bidimensional array
         * @param map bidimensional array containing palette values
         * @param min  minimum value in the palette (default is 0)
         * @param max maximum value in the palette (default is 255)
         */
        Palette(uchar map[][3], int min = 0, int max = 255);

        /**
         * Constructor from file name.
         * List of operations :
         *  - opens the file
         *  - if file is correctly opened, then reads each line (ignoring lines
         *  starting with a "#" which indicates a comment line)
         *  - each line should contain 3 bytes : e.g. 127 0 255
         * @param filename the name of the file to read
```



```
         * @param min minimum value in the palette (default is 0)
         * @param max maximum value in the palette (default is 255)
         */
        Palette(const char * const filename, int min=0, int max = 255);

        /**
         * Palette destructor.
         * Relese all images and clear vectors
         */
        virtual ~Palette();

        /**
         * Apply the colormap on the single channel source image to build
         * a destination 3 channels color image.
         * @param src source mono-channel image
         * @param dst destination BGR-BGRChannels image
         */
        void applyPalette(const Mat & src, Mat & dst);
};

#endif /* PALETTE_H_ */
```

```cpp
/*
 * Palette.cpp
 *
 *  Created on: 13 sept. 2010
 *      Author: David Roussel
 */

#include <iostream>      // pour cout & cerr
#include <fstream>       // pour l'ifstream
#include <string>        // pour les string
using namespace std;

#include "Palette.h"

const size_t Palette::CMAPSIZE = 256;

const size_t Palette::COMPSIZE = 3;

/*
 * Constructor from bidimensional array
 * @param map bidimensional array containing palette values
 * @param minimum value in the palette (default is 0)
 * @param maximum value in the palette (default is 255)
 */
Palette::Palette(unsigned char map[][3], int min, int max) :
    colormap(COMPSIZE),
    minValue(min),
    maxValue(max),
    BGRChannels(COMPSIZE),
    channelsAllocated(false)
{
    // initialize colormaps
    for (size_t i=0; i < colormap.size(); i++)
    {
        colormap[i].create(CMAPSIZE,1,CV_8UC1);
    }

    // fill colormap with values
    for (size_t c = 0; c < COMPSIZE; c++)
    {
        for (size_t i=0; i < CMAPSIZE; i++)
        {
            colormap[c].at<uchar>(i, 0) = map[i][c];
        }
    }
}

/*
 * Constructor from file name.
 * List of operations :
 *  - opens the file
 *  - if file is correctly opened, then reads each line (ignoring lines
 *  starting with a "#" which indicates a comment line)
 *  - each line should contain 3 bytes : e.g. 127 0 255
 * @param filename the name of the file to read
 * @param minimum value in the palette (default is 0)
 * @param maximum value in the palette (default is 255)
 */
Palette::Palette(const char * const filename, int min, int max) :
    colormap(COMPSIZE),
    minValue(min),
    maxValue(max),
    BGRChannels(COMPSIZE),
    channelsAllocated(false)
{
    // initialize colormaps
    for (size_t i=0; i < colormap.size(); i++)
    {
        colormap[i].create(CMAPSIZE,1,CV_8UC1);
    }

    unsigned int lineCount = 0;
    unsigned int dataLineCount = 0;

    if (filename ≠ NULL)
    {
        ifstream inputFile(filename);

        if (inputFile.is_open())
        {
            string currentLine;
            istringstream lineStream;
            size_t searchComment;
            int readValues[COMPSIZE];

            while (¬inputFile.eof())
            {
                getline(inputFile, currentLine);

                lineCount++;
```



```cpp

                if (currentLine.length() > 0)
                {
                    // checks for # character at the beginning of the line
                    searchComment = currentLine.find('#');

                    if ((searchComment ≡ string::npos) ∧
                        ((int)searchComment ≠ 0))
                        // no leading comment found : data line
                    {
                        // set current line into input string stream
                        lineStream.str(currentLine);

                        for (size_t i=0; i < COMPSIZE; i++)
                        {
                            // reads single value from input string stream
                            lineStream >> readValues[i];
                            if (lineStream.fail())
                            {
                                cerr << "Error reading RGB value index " << i
                                     <<" at line " << lineCount << endl;
                                exit(EXIT_FAILURE);
                            }
                            else
                            {
                                // checks invalid values
                                if (readValues[i] > maxValue)
                                {
                                    readValues[i] = maxValue;
                                }
                                if (readValues[i] < minValue)
                                {
                                    readValues[i] = minValue;
                                }
                            }

                            // Fill colormap with value
                            colormap[i].at<uchar>((int)dataLineCount,0)
                                = (uchar)readValues[i];
                        }

                        lineStream.clear();

//                      cout << "line " << lineCount << "[" << dataLineCount
//                           << "] contains : \"" << currentLine << "\" data are "
//                           << readValues[0] << ", " << readValues[1]
//                           << ", " << readValues[2] << endl;

                        dataLineCount++;
                    }
//                  else // comment found at pos 0
//                  {
//                      cout << "comment line at line " << lineCount
//                           << " : " << currentLine << endl;
//                  }
                }
//              else // empty line : skip
//              {
//                  cout << "empty line at line " << lineCount << endl;
//              }
            }

            if (dataLineCount ≠ CMAPSIZE)
            {
                cerr << "Wrong number of datalines in the colormap : "
                     << dataLineCount << endl;
                exit(EXIT_FAILURE);
            }
//          else
//          {
//              cout << "Correctly read "<< CMAPSIZE <<" data lines" << endl;
//          }

            inputFile.close();
        }
        else // inputFile is not opened
        {
            cerr << "Palette::Palette(" << filename << ") : unable to open file"
                 << endl;
            exit(EXIT_FAILURE);
        }
    }
    else // filename is NULL
    {
        cerr << "Palette::Palette(NULL filename) : empty file name" << endl;
        exit(EXIT_FAILURE);
    }
}

/*
```

```
 * Palette destructor.
 * Relese all images and clear vectors
 */
Palette::~Palette()
{
    // Release matrices
    for (size_t i=0; i < colormap.size(); i++)
    {
        colormap[i].release();
        BGRChannels[i].release();
    }

    // Clear vectors
    colormap.clear();
    BGRChannels.clear();
}

/*
 * Apply the colormap on the single channel source image to build
 * a destination 3 channels color image.
 * @param src source mono-channel image
 * @param dst destination BGR-BGRChannels image
 */
void Palette::applyPalette(const Mat & src, Mat & dst)
{
    const size_t BGR2RGB[CMAPSIZE] = {2,1,0};

    // checks if source has only one channel
    if (src.channels() == 1)
    {
        if (¬channelsAllocated) // BGRChannels should be allocated first
        {
            for (size_t i=0; i < BGRChannels.size(); i++)
            {
                BGRChannels[i].create(src.size(),CV_8UC1);
            }
            channelsAllocated = true;
        }

        if (src.size() ≠ BGRChannels[0].size()) // BGRChannels should be reallocated
        {
            for (size_t i=0; i < BGRChannels.size(); i++)
            {
                BGRChannels[i].release();
                BGRChannels[i].create(src.size(),CV_8UC1);
            }
        }

        // Apply Look Up Table on each channel
        for (size_t i=0; i < COMPSIZE; i++)
        {
            LUT(src,colormap[BGR2RGB[i]],BGRChannels[i]);
        }

        // then merge all the BGRChannels into a BGR image
        merge(BGRChannels, dst);
    }
    else // source has multiple channels
    {
        cerr << "Palette::applyColormap(...) : source has " << src.channels()
             << " channels" << endl;
    }
}
```



```
#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 ± 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 ± std value on the stream
 */
template <typename T, typename R>
ostream & operator <<(ostream & out, const MeanValue<T, R> & mv)
{
    out << mv.mean() << "±" << 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 "QcvColorSpaces.h"

namespace Ui {
    class MainWindow;
}

/**
 * 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;

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

    public:
        /**
         * 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,
                            QcvColorSpaces * 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 video flip
         */
        void flipChanged(const bool flip);

        /**
         * Signal to send when gray source image request changes
```



```
         * @param gray 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
         */
        QcvColorSpaces * processor;

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

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

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

        /**
         * Mean process time string
         */
        QString meanTimeString;

        /**
         * Std process time string
         */
        QString stdTimeString;

        /**
         * 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();

        /**
         * Updates mean and std of process time
         * @param The updated process time (MeanValue<clock_t, double>)
         */
        void on_processTimeupdated(const CvProcessor::ProcessTime * pt);

        /**
         * 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
```

```
181         */
182        void on_actionCamera_1_triggered();
183
184        /**
185         * Menu action when Sources->file is selected.
186         * Opens file dialog and tries to open selected file (is not empty),
187         * then sets capture to open the selected file
188         */
189        void on_actionFile_triggered();
190
191        /**
192         * Menu action to quit application.
193         */
194        void on_actionQuit_triggered();
195
196        /**
197         * Menu action when flip image is selected.
198         * Sets capture to change flip status which leads to reverse
199         * image horizontally
200         */
201        void on_actionFlip_triggered();
202
203        /**
204         * Menu action when original image size is selected.
205         * Sets capture not to resize image
206         */
207        void on_actionOriginalSize_triggered();
208
209        /**
210         * Menu action when constrained image size is selected.
211         * Sets capture resize to preferred width and height
212         */
213        void on_actionConstrainedSize_triggered();
214
215        /**
216         * Menu action to replace current image rendering widget by a
217         * QcvMatWidgetImage instance.
218         */
219        void on_actionRenderImage_triggered();
220
221        /**
222         * Menu action to replace current image rendering widget by a
223         * QcvMatWidgetLabel with pixmap instance.
224         */
225        void on_actionRenderPixmap_triggered();
226
227        /**
228         * Menu action to replace current image rendering widget by a
229         * QcvMatWidgetGL instance.
230         */
231        void on_actionRenderOpenGL_triggered();
232
233
234        /**
235         * Original size radioButton action.
236         * Sets capture resize to off
237         */
238        void on_radioButtonOrigSize_clicked();
239
240        /**
241         * Custom size radioButton action.
242         * Sets capture resize to preferred width and height
243         */
244        void on_radioButtonCustomSize_clicked();
245
246        /**
247         * Width spinbox value change.
248         * Changes the preferred width and if custom size is selected apply
249         * this custom width
250         * @param value the desired width
251         */
252        void on_spinBoxWidth_valueChanged(int value);
253
254        /**
255         * Height spinbox value change.
256         * Changes the preferred height and if custom size is selected apply
257         * this custom height
258         * @param value the desired height
259         */
260        void on_spinBoxHeight_valueChanged(int value);
261
262        /**
263         * Flip capture image horizontally.
264         * changes capture flip status
265         */
266        void on_checkBoxFlip_clicked();
267
268        /**
269         * Select input image for display
270         */
```



```
271        void on_radioButtonInput_clicked();
272
273        /**
274         * Select Gray image for display
275         */
276        void on_radioButtonGray_clicked();
277
278        /**
279         * Select red component of RGB space for display
280         */
281        void on_radioButtonRed_clicked();
282
283        /**
284         * Select green component of RGB space for display
285         */
286        void on_radioButtonGreen_clicked();
287
288        /**
289         * Select blue component of RGB space for display
290         */
291        void on_radioButtonBlue_clicked();
292
293        /**
294         * Select hue component of HSV space for display
295         */
296        void on_radioButtonHue_clicked();
297
298        /**
299         * Select saturation component of HSV space for display
300         */
301        void on_radioButtonSaturation_clicked();
302
303        /**
304         * Select value component of HSV space for display
305         */
306        void on_radioButtonValue_clicked();
307
308        /**
309         * Select Y component of YCbCr space for display
310         */
311        void on_radioButtonY_clicked();
312
313        /**
314         * Select Cr component of YCbCr space for display
315         */
316        void on_radioButtonCr_clicked();
317
318        /**
319         * Select Cb component of YCbCr space for display
320         */
321        void on_radioButtonCb_clicked();
322
323        /**
324         * Select component display as colored image
325         */
326        void on_radioButtonChColor_clicked();
327
328        /**
329         * Select componet display as gray image
330         */
331        void on_radioButtonChGray_clicked();
332
333        /**
334         * Select hue component display as hue alone
335         */
336        void on_radioButtonMixHue_clicked();
337
338        /**
339         * Select hue component display as hue x saturation value
340         */
341        void on_radioButtonMixHueSat_clicked();
342
343        /**
344         * Select hue component display as hue x value value
345         */
346        void on_radioButtonMixHueVal_clicked();
347
348        /**
349         * Select X component for display
350         */
351        void on_radioButtonXYZ_X_clicked();
352
353        /**
354         * Select Y component for display
355         */
356        void on_radioButtonXYZ_Y_clicked();
357
358        /**
359         * Select Z component for display
360         */
```

```
361         void on_radioButtonXYZ_Z_clicked();
362
363         /**
364          * Select Maximum of RGB as display
365          */
366         void on_radioButtonMaxBGR_clicked();
367 };
368
369 #endif // MAINWINDOW_H
```



```
1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3
4  #include <QObject>
5  #include <QFileDialog>
6  #include <QWindow>
7  #include <QDebug>
8  #include <assert.h>
9
10 #include "QcvMatWidgetImage.h"
11 #include "QcvMatWidgetLabel.h"
12 #include "QcvMatWidgetGL.h"
13
14 /*
15  * MainWindow constructor
16  * @param capture the capture QObject to capture frames from devices
17  * or video files
18  * @param parent parent widget
19  */
20 MainWindow::MainWindow(QcvVideoCapture * capture,
21                        QcvColorSpaces * processor,
22                        QWidget * parent)
23     : QMainWindow(parent),
24       ui(new Ui::MainWindow),
25       capture(capture),
26       processor(processor),
27       preferredWidth(640),
28       preferredHeight(480)
29 {
30     ui->setupUi(this);
31     ui->scrollArea->setBackgroundRole(QPalette::Mid);
32
33     // ----------------------------------------------------------------------
34     // Assertions
35     // ----------------------------------------------------------------------
36     assert(capture ≠ NULL);
37
38     assert(processor ≠ NULL);
39
40     // ----------------------------------------------------------------------
41     // Signal/Slot connections
42     // ----------------------------------------------------------------------
43     // Replace QcvMatWidget instance with QcvMatWidgetImage instance and
44     // sets widgetImage source for the first time
45     setupImageWidget(RENDER_IMAGE);
46
47     // Connects Mainwindow messages to status bar
48     connect(this,
49             SIGNAL(sendMessage(QString, int) ),
50             ui->statusBar,
51             SLOT(showMessage(QString, int) ));
52
53     // Connects capture status messages to statusBar
54     connect(capture,
55             SIGNAL(messageChanged(QString, int) ),
56             ui->statusBar,
57             SLOT(showMessage(QString, int) ));
58
59     // Connects processor status messages to statusBar
60     connect(processor,
61             SIGNAL(sendMessage(QString, int) ),
62             ui->statusBar,
63             SLOT(showMessage(QString, int) ));
64
65     // When Processor source image changes, some attributes are reinitialised
66     // So we have to set them up again according to current UI values
67     connect(processor,
68             SIGNAL(imageChanged()),
69             this,
70             SLOT(setupProcessorfromUI()));
71
72     // Connects processor time to UI time label
73     //  connect(processor, SIGNAL(processTimeUpdated(QString)),
74     //          ui->labelProcessTimeValue, SLOT(setText(QString)));
75     connect(processor,
76             SIGNAL(processTimeUpdated(const CvProcessor::ProcessTime*)),
77             this,
78             SLOT(on_processTimeupdated(const CvProcessor::ProcessTime*)));
79
80     // Connects UI requests to capture
81     connect(this,
82             SIGNAL(sizeChanged(const QSize &) ),
83             capture,
84             SLOT(setSize(const QSize &) ),
85             Qt::DirectConnection);
86     connect(this,
87             SIGNAL(deviceChanged(int, uint, uint)),
88             capture,
89             SLOT(open(int, uint, uint)),
90             Qt::DirectConnection);
```

```
91      connect(this,
92              SIGNAL(fileChanged(QString, uint, uint)),
93              capture,
94              SLOT(open(QString, uint, uint)),
95              Qt::DirectConnection);
96      connect(this,
97              SIGNAL(flipChanged(bool) ),
98              capture,
99              SLOT(setFlipVideo(bool) ),
100             Qt::DirectConnection);
101     // ------------------------------------------------------------------------
102     // UI setup according to capture and processor options
103     // ------------------------------------------------------------------------
104     setupUIfromCapture();
105
106     setupUIfromProcessor();
107 }
108
109 /*
110  * MainWindow destructor
111  */
112 MainWindow::~MainWindow()
113 {
114     delete ui;
115 }
116
117 /*
118  * Changes widgetImage nature according to desired rendering mode.
119  * Possible values for mode are:
120  *  - IMAGE: widgetImage is assigned to a QcvMatWidgetImage instance
121  *  - PIXMAP: widgetImage is assigned to a QcvMatWidgetLabel instance
122  *  - GL: widgetImage is assigned to a QcvMatWidgetGL instance
123  * @param mode
124  */
125 void MainWindow::setupImageWidget(const RenderMode mode)
126 {
127     // Disconnect first
128     disconnect(processor, SIGNAL(updated()), ui->widgetImage, SLOT(update()));
129
130     disconnect(processor,
131                SIGNAL(imageChanged(Mat *) ),
132                ui->widgetImage,
133                SLOT(setSourceImage(Mat *) ));
134
135     QWindow * currentWindow = windowHandle();
136     if (mode == RENDER_GL)
137     {
138         disconnect(currentWindow,
139                    SIGNAL(screenChanged(QScreen *)),
140                    ui->widgetImage,
141                    SLOT(screenChanged()));
142     }
143
144     // remove widget in scroll area
145     QWidget * w = ui->scrollArea->takeWidget();
146
147     if (w == ui->widgetImage)
148     {
149         // delete removed widget
150         delete ui->widgetImage;
151
152         // create new widget
153         Mat * image = processor->getImagePtr("display");
154         switch (mode)
155         {
156             case RENDER_PIXMAP:
157                 ui->widgetImage = new QcvMatWidgetLabel(image);
158                 break;
159             case RENDER_GL:
160                 ui->widgetImage = new QcvMatWidgetGL(image);
161                 break;
162             case RENDER_IMAGE:
163             default:
164                 ui->widgetImage = new QcvMatWidgetImage(image);
165                 break;
166         }
167
168         if (ui->widgetImage != NULL)
169         {
170             ui->widgetImage->setObjectName(QString::fromUtf8("widgetImage"));
171
172             // add it to the scroll area
173             ui->scrollArea->setWidget(ui->widgetImage);
174
175             connect(
176                 processor, SIGNAL(updated()), ui->widgetImage, SLOT(update()));
177
178             connect(processor,
179                     SIGNAL(imageChanged(Mat *) ),
180                     ui->widgetImage,
```



```
181                     SLOT(setSourceImage(Mat *) ));
182
183             if (mode == RENDER_GL)
184             {
185                 connect(currentWindow,
186                         SIGNAL(screenChanged(QScreen *)),
187                         ui->widgetImage,
188                         SLOT(screenChanged()));
189             }
190
191             // Sends message to status bar and sets menu checks
192             message.clear();
193             message.append(tr("Render mode set to "));
194             switch (mode)
195             {
196                 case RENDER_IMAGE:
197                     ui->actionRenderPixmap->setChecked(false);
198                     ui->actionRenderOpenGL->setChecked(false);
199                     message.append(tr("QImage"));
200                     break;
201                 case RENDER_PIXMAP:
202                     ui->actionRenderImage->setChecked(false);
203                     ui->actionRenderOpenGL->setChecked(false);
204                     message.append(tr("QPixmap in QLabel"));
205                     break;
206                 case RENDER_GL:
207                     ui->actionRenderImage->setChecked(false);
208                     ui->actionRenderPixmap->setChecked(false);
209                     message.append("QGLWidget");
210                     break;
211                 default:
212                     break;
213             }
214             emit sendMessage(message, 5000);
215         }
216         else
217         {
218             qDebug("MainWindow::on_actionRenderXXX new widget is null");
219         }
220     }
221     else
222     {
223         qDebug(
224             "MainWindow::on_actionRenderXXX removed widget is not imageWidget");
225     }
226 }
227
228 /*
229  * Setup UI according to capture settings when app launches
230  */
231 void MainWindow::setupUIfromCapture()
232 {
233     // ------------------------------------------------------------------------
234     // UI setup according to capture options
235     // ------------------------------------------------------------------------
236     // Sets size radioButton states
237     if (capture->isResized())
238     {
239         /*
240          * Initial Size radio buttons configuration
241          */
242         ui->radioButtonOrigSize->setChecked(false);
243         ui->radioButtonCustomSize->setChecked(true);
244         /*
245          * Initial Size menu items configuration
246          */
247         ui->actionOriginalSize->setChecked(false);
248         ui->actionConstrainedSize->setChecked(true);
249
250         QSize size = capture->getSize();
251         qDebug("Capture->size is %dx%d", size.width(), size.height());
252         preferredWidth = size.width();
253         preferredHeight = size.height();
254     }
255     else
256     {
257         /*
258          * Initial Size radio buttons configuration
259          */
260         ui->radioButtonCustomSize->setChecked(false);
261         ui->radioButtonOrigSize->setChecked(true);
262
263         /*
264          * Initial Size menu items configuration
265          */
266         ui->actionConstrainedSize->setChecked(false);
267         ui->actionOriginalSize->setChecked(true);
268     }
269
270     // Sets spinboxes preferred size
```

```
271     ui->spinBoxWidth->setValue(preferredWidth);
272     ui->spinBoxHeight->setValue(preferredHeight);
273
274     // Sets flipCheckbox and menu item states
275     bool flipped = capture->isFlipVideo();
276     ui->actionFlip->setChecked(flipped);
277     ui->checkBoxFlip->setChecked(flipped);
278 }
279
280 /*
281  * Setup UI according to processor settings when app launches
282  */
283 void MainWindow::setupUIfromProcessor()
284 {
285     // Sets selected image for display
286     switch (processor->getDisplayImageIndex())
287     {
288         case CvColorSpaces::INPUT:
289             ui->radioButtonInput->setChecked(true);
290             break;
291         case CvColorSpaces::GRAY:
292             ui->radioButtonGray->setChecked(true);
293             break;
294         case CvColorSpaces::RED:
295             ui->radioButtonRed->setChecked(true);
296             break;
297         case CvColorSpaces::GREEN:
298             ui->radioButtonGreen->setChecked(true);
299             break;
300         case CvColorSpaces::BLUE:
301             ui->radioButtonBlue->setChecked(true);
302             break;
303         case CvColorSpaces::HUE:
304             ui->radioButtonHue->setChecked(true);
305             break;
306         case CvColorSpaces::SATURATION:
307             ui->radioButtonSaturation->setChecked(true);
308             break;
309         case CvColorSpaces::VALUE:
310             ui->radioButtonValue->setChecked(true);
311             break;
312         case CvColorSpaces::Y:
313             ui->radioButtonY->setChecked(true);
314             break;
315         case CvColorSpaces::Cr:
316             ui->radioButtonCr->setChecked(true);
317             break;
318         case CvColorSpaces::Cb:
319             ui->radioButtonCb->setChecked(true);
320             break;
321         case CvColorSpaces::NbSelected:
322         default:
323             // Do nothing
324             break;
325     }
326
327     // By default set radio button gray channel to checked
328     ui->radioButtonChGray->setChecked(true);
329
330     // if at least one showColor index is true then set radiobutton color
331     // channel to true
332     for (size_t i = 0; i < CvColorSpaces::NbShows; i++)
333     {
334         if (processor->getColorChannel((CvColorSpaces::ShowColor) i))
335         {
336             ui->radioButtonChColor->setChecked(true);
337             break;
338         }
339     }
340
341     // Sets Hue mix mode
342     switch (processor->getHueDisplaymode())
343     {
344         case CvColorSpaces::HUECOLOR:
345             ui->radioButtonMixHue->setChecked(true);
346             break;
347         case CvColorSpaces::HUESATURATE:
348             ui->radioButtonMixHueSat->setChecked(true);
349             break;
350         case CvColorSpaces::HUEVALUE:
351             ui->radioButtonMixHueVal->setChecked(true);
352             break;
353         case CvColorSpaces::HUEGRAY:
354             ui->radioButtonChGray->setChecked(true);
355             break;
356         default:
357             break;
358     }
359 }
360
```



```
361 /*
362  * Setup processor from current UI settings when processor source image
363  * changes
364  */
365 void MainWindow::setupProcessorfromUI()
366 {
367     if (ui->radioButtonInput->isChecked())
368     {
369         processor->setDisplayImageIndex(CvColorSpaces::INPUT);
370     }
371
372     if (ui->radioButtonGray->isChecked())
373     {
374         processor->setDisplayImageIndex(CvColorSpaces::GRAY);
375     }
376
377     if (ui->radioButtonRed->isChecked())
378     {
379         processor->setDisplayImageIndex(CvColorSpaces::RED);
380     }
381
382     if (ui->radioButtonGreen->isChecked())
383     {
384         processor->setDisplayImageIndex(CvColorSpaces::GREEN);
385     }
386
387     if (ui->radioButtonBlue->isChecked())
388     {
389         processor->setDisplayImageIndex(CvColorSpaces::BLUE);
390     }
391
392     if (ui->radioButtonHue->isChecked())
393     {
394         processor->setDisplayImageIndex(CvColorSpaces::HUE);
395     }
396
397     if (ui->radioButtonSaturation->isChecked())
398     {
399         processor->setDisplayImageIndex(CvColorSpaces::SATURATION);
400     }
401
402     if (ui->radioButtonValue->isChecked())
403     {
404         processor->setDisplayImageIndex(CvColorSpaces::VALUE);
405     }
406
407     if (ui->radioButtonY->isChecked())
408     {
409         processor->setDisplayImageIndex(CvColorSpaces::Y);
410     }
411
412     if (ui->radioButtonCr->isChecked())
413     {
414         processor->setDisplayImageIndex(CvColorSpaces::Cr);
415     }
416
417     if (ui->radioButtonCb->isChecked())
418     {
419         processor->setDisplayImageIndex(CvColorSpaces::Cb);
420     }
421
422     if (ui->radioButtonChColor->isChecked())
423     {
424         for (size_t i = 0; i < CvColorSpaces::NbShows; i++)
425         {
426             processor->setColorChannel((CvColorSpaces::ShowColor) i, true);
427         }
428         if (ui->radioButtonMixHue->isChecked())
429         {
430             processor->setHueDisplayMode(CvColorSpaces::HUECOLOR);
431         }
432         else if (ui->radioButtonMixHueSat->isChecked())
433         {
434             processor->setHueDisplayMode(CvColorSpaces::HUESATURATE);
435         }
436         else
437         {
438             processor->setHueDisplayMode(CvColorSpaces::HUEVALUE);
439         }
440     }
441
442     if (ui->radioButtonChGray->isChecked())
443     {
444         for (size_t i = 0; i < CvColorSpaces::NbShows; i++)
445         {
446             processor->setColorChannel((CvColorSpaces::ShowColor) i, false);
447         }
448         processor->setHueDisplayMode(CvColorSpaces::HUEGRAY);
449     }
450 }
```

```
451
452 /*
453  * Updates mean and std of process time
454  * @param The updated process time (MeanValue<clock_t, double>)
455  */
456 void MainWindow::on_processTimeupdated(const CvProcessor::ProcessTime * pt)
457 {
458     meanTimeString.sprintf("%6.0f", pt->mean() / 1000.0);
459     stdTimeString.sprintf("%5.0f", pt->std() / 1000.0);
460     ui->labelProcessTimeValue->setText(meanTimeString);
461     ui->labelProcessTimeStd->setText(stdTimeString);
462 }
463
464 /*
465  * Menu action when Sources->camera 0 is selected
466  * Sets capture to open device 0. If device is not available
467  * menu item is set to inactive.
468  */
469 void MainWindow::on_actionCamera_0_triggered()
470 {
471     int width = 0;
472     int height = 0;
473
474     if (ui->radioButtonCustomSize->isChecked())
475     {
476         width = preferredWidth;
477         height = preferredHeight;
478     }
479
480     qDebug("Opening device 0 ...");
481     //  if (!capture->open(0, width, height))
482     //  {
483     //      qWarning("Unable to open device 0");
484     //      // disable menu item if camera 0 does not exist
485     //      ui->actionCamera_0->setDisabled(true);
486     //  }
487     emit(deviceChanged(0, width, height));
488 }
489
490 /*
491  * Menu action when Sources->camera 1 is selected
492  * Sets capture to open device 0. If device is not available
493  * menu item is set to inactive
494  */
495 void MainWindow::on_actionCamera_1_triggered()
496 {
497     int width = 0;
498     int height = 0;
499
500     if (ui->radioButtonCustomSize->isChecked())
501     {
502         width = preferredWidth;
503         height = preferredHeight;
504     }
505
506     qDebug("Opening device 1 ...");
507     //  if (!capture->open(1, width, height))
508     //  {
509     //      qWarning("Unable to open device 1");
510     //      // disable menu item if camera 1 does not exist
511     //      ui->actionCamera_1->setDisabled(true);
512     //  }
513
514     emit deviceChanged(1, width, height);
515 }
516
517 /*
518  * Menu action when Sources->file is selected.
519  * Opens file dialog and tries to open selected file (is not empty),
520  * then sets capture to open the selected file
521  */
522 void MainWindow::on_actionFile_triggered()
523 {
524     int width = 0;
525     int height = 0;
526
527     if (ui->radioButtonCustomSize->isChecked())
528     {
529         width = preferredWidth;
530         height = preferredHeight;
531     }
532
533     QString fileName = QFileDialog::getOpenFileName(
534         this,
535         tr("Open Video"),
536         "./",
537         tr("Video Files (*.avi *.m4v *.mkv *.mp4)"),
538         NULL,
539         QFileDialog::ReadOnly);
540
```



```
541     // qDebug("Opening file %s ...", fileName.toStdString().c_str());
542
543     if (fileName.length() > 0)
544     {
545         //      if (!capture->open(fileName, width, height))
546         //      {
547         //          qWarning("Unable to open device file : %s",
548         //                   fileName.toStdString().c_str());
549         //      }
550
551         emit fileChanged(fileName, width, height);
552     }
553     else
554     {
555         qWarning("empty file name");
556     }
557 }
558
559 /*
560  * Menu action to qui application
561  */
562 void MainWindow::on_actionQuit_triggered()
563 {
564     this->close();
565 }
566
567 /*
568  * Menu action when flip image is selected.
569  * Sets capture to change flip status which leads to reverse
570  * image horizontally
571  */
572 void MainWindow::on_actionFlip_triggered()
573 {
574     emit flipChanged(¬capture->isFlipVideo());
575
576     /*
577      * There is no need to update ui->checkBoxFlip since it is connected
578      * to ui->actionFlip through signals/slots
579      */
580 }
581
582 /*
583  * Menu action when original image size is selected.
584  * Sets capture not to resize image
585  */
586 void MainWindow::on_actionOriginalSize_triggered()
587 {
588     ui->actionConstrainedSize->setChecked(false);
589     emit sizeChanged(QSize(0, 0));
590 }
591
592 /*
593  * Menu action when constrained image size is selected.
594  * Sets capture resize to preferred width and height
595  */
596 void MainWindow::on_actionConstrainedSize_triggered()
597 {
598     ui->actionOriginalSize->setChecked(false);
599     emit sizeChanged(QSize(preferredWidth, preferredHeight));
600 }
601
602 /*
603  * Menu action to replace current image rendering widget by a
604  * QcvMatWidgetImage instance.
605  */
606 void MainWindow::on_actionRenderImage_triggered()
607 {
608     setupImageWidget(RENDER_IMAGE);
609 }
610
611 /*
612  * Menu action to replace current image rendering widget by a
613  * QcvMatWidgetLabel with pixmap instance.
614  */
615 void MainWindow::on_actionRenderPixmap_triggered()
616 {
617     setupImageWidget(RENDER_PIXMAP);
618 }
619
620 /*
621  * Menu action to replace current image rendering widget by a
622  * QcvMatWidgetGL instance.
623  */
624 void MainWindow::on_actionRenderOpenGL_triggered()
625 {
626     setupImageWidget(RENDER_GL);
627 }
628
629 /*
630  * Original size radioButton action.
```

```
631  * Sets capture resize to off
632  */
633 void MainWindow::on_radioButtonOrigSize_clicked()
634 {
635     ui->actionConstrainedSize->setChecked(false);
636     emit sizeChanged(QSize(0, 0));
637 }
638
639 /*
640  * Custom size radioButton action.
641  * Sets capture resize to preferred width and height
642  */
643 void MainWindow::on_radioButtonCustomSize_clicked()
644 {
645     ui->actionOriginalSize->setChecked(false);
646     emit sizeChanged(QSize(preferredWidth, preferredHeight));
647 }
648
649 /*
650  * Width spinbox value change.
651  * Changes the preferred width and if custom size is selected apply
652  * this custom width
653  * @param value the desired width
654  */
655 void MainWindow::on_spinBoxWidth_valueChanged(int value)
656 {
657     preferredWidth = value;
658     if (ui->radioButtonCustomSize->isChecked())
659     {
660         emit sizeChanged(QSize(preferredWidth, preferredHeight));
661     }
662 }
663
664 /*
665  * Height spinbox value change.
666  * Changes the preferred height and if custom size is selected apply
667  * this custom height
668  * @param value the desired height
669  */
670 void MainWindow::on_spinBoxHeight_valueChanged(int value)
671 {
672     preferredHeight = value;
673     if (ui->radioButtonCustomSize->isChecked())
674     {
675         emit sizeChanged(QSize(preferredWidth, preferredHeight));
676     }
677 }
678
679 /*
680  * Flip capture image horizontally.
681  * changes capture flip status
682  */
683 void MainWindow::on_checkBoxFlip_clicked()
684 {
685     /*
686      * There is no need to update ui->actionFlip since it is connected
687      * to ui->checkBoxFlip through signals/slots
688      */
689     emit flipChanged(ui->checkBoxFlip->isChecked());
690 }
691
692 /*
693  * Select input image for display
694  */
695 void MainWindow::on_radioButtonInput_clicked()
696 {
697     processor->setDisplayImageIndex(CvColorSpaces::INPUT);
698 }
699
700 /*
701  * Select Gray image for display
702  */
703 void MainWindow::on_radioButtonGray_clicked()
704 {
705     processor->setDisplayImageIndex(CvColorSpaces::GRAY);
706 }
707
708 /*
709  * Select red component of RGB space for display
710  */
711 void MainWindow::on_radioButtonRed_clicked()
712 {
713     processor->setDisplayImageIndex(CvColorSpaces::RED);
714 }
715
716 /*
717  * Select green component of RGB space for display
718  */
719 void MainWindow::on_radioButtonGreen_clicked()
720 {
```



```
721     processor->setDisplayImageIndex(CvColorSpaces::GREEN);
722 }
723
724 /*
725  * Select blue component of RGB space for display
726  */
727 void MainWindow::on_radioButtonBlue_clicked()
728 {
729     processor->setDisplayImageIndex(CvColorSpaces::BLUE);
730 }
731
732 /*
733  * Select hue component of HSV space for display
734  */
735 void MainWindow::on_radioButtonHue_clicked()
736 {
737     processor->setDisplayImageIndex(CvColorSpaces::HUE);
738 }
739
740 /*
741  * Select saturation component of HSV space for display
742  */
743 void MainWindow::on_radioButtonSaturation_clicked()
744 {
745     processor->setDisplayImageIndex(CvColorSpaces::SATURATION);
746 }
747
748 /*
749  * Select value component of HSV space for display
750  */
751 void MainWindow::on_radioButtonValue_clicked()
752 {
753     processor->setDisplayImageIndex(CvColorSpaces::VALUE);
754 }
755
756 /*
757  * Select Y component of YCbCr space for display
758  */
759 void MainWindow::on_radioButtonY_clicked()
760 {
761     processor->setDisplayImageIndex(CvColorSpaces::Y);
762 }
763
764 /*
765  * Select Cr component of YCbCr space for display
766  */
767 void MainWindow::on_radioButtonCr_clicked()
768 {
769     processor->setDisplayImageIndex(CvColorSpaces::Cr);
770 }
771
772 /*
773  * Select Cb component of YCbCr space for display
774  */
775 void MainWindow::on_radioButtonCb_clicked()
776 {
777     processor->setDisplayImageIndex(CvColorSpaces::Cb);
778 }
779
780 /*
781  * Select component display as colored image
782  */
783 void MainWindow::on_radioButtonChColor_clicked()
784 {
785     for (size_t i = 0; i < CvColorSpaces::NbShows; i++)
786     {
787         processor->setColorChannel((CvColorSpaces::ShowColor) i, true);
788     }
789     if (ui->radioButtonMixHue->isChecked())
790     {
791         processor->setHueDisplayMode(CvColorSpaces::HUECOLOR);
792     }
793     else if (ui->radioButtonMixHueSat->isChecked())
794     {
795         processor->setHueDisplayMode(CvColorSpaces::HUESATURATE);
796     }
797     else
798     {
799         processor->setHueDisplayMode(CvColorSpaces::HUEVALUE);
800     }
801 }
802
803 /*
804  * Select componet display as gray image
805  */
806 void MainWindow::on_radioButtonChGray_clicked()
807 {
808     for (size_t i = 0; i < CvColorSpaces::NbShows; i++)
809     {
810         processor->setColorChannel((CvColorSpaces::ShowColor) i, false);
```

```
811         }
812         processor->setHueDisplayMode(CvColorSpaces::HUEGRAY);
813     }
814
815     /*
816      * Select hue component display as hue alone
817      */
818     void MainWindow::on_radioButtonMixHue_clicked()
819     {
820         processor->setHueDisplayMode(CvColorSpaces::HUECOLOR);
821     }
822
823     /*
824      * Select hue component display as hue x saturation value
825      */
826     void MainWindow::on_radioButtonMixHueSat_clicked()
827     {
828         processor->setHueDisplayMode(CvColorSpaces::HUESATURATE);
829     }
830
831     /*
832      * Select hue component display as hue x value value
833      */
834     void MainWindow::on_radioButtonMixHueVal_clicked()
835     {
836         processor->setHueDisplayMode(CvColorSpaces::HUEVALUE);
837     }
838
839     void MainWindow::on_radioButtonXYZ_X_clicked()
840     {
841         processor->setDisplayImageIndex(CvColorSpaces::XYZ_X);
842     }
843
844     void MainWindow::on_radioButtonXYZ_Y_clicked()
845     {
846         processor->setDisplayImageIndex(CvColorSpaces::XYZ_Y);
847     }
848
849     void MainWindow::on_radioButtonXYZ_Z_clicked()
850     {
851         processor->setDisplayImageIndex(CvColorSpaces::XYZ_Z);
852     }
853
854     void MainWindow::on_radioButtonMaxBGR_clicked()
855     {
856         processor->setDisplayImageIndex(CvColorSpaces::MAX_BGR);
857     }
```



```
1   #ifndef RED_MAP_
2   #define RED_MAP_
3
4   /**
5    * Color map for RGB red component color image
6    */
7   unsigned char mapRed[256][3] =
8   {
9       {0, 0, 0},
10      {1, 0, 0},
11      {2, 0, 0},
12      {3, 0, 0},
13      {4, 0, 0},
14      {5, 0, 0},
15      {6, 0, 0},
16      {7, 0, 0},
17      {8, 0, 0},
18      {9, 0, 0},
19      {10, 0, 0},
20      {11, 0, 0},
21      {12, 0, 0},
22      {13, 0, 0},
23      {14, 0, 0},
24      {15, 0, 0},
25      {16, 0, 0},
26      {17, 0, 0},
27      {18, 0, 0},
28      {19, 0, 0},
29      {20, 0, 0},
30      {21, 0, 0},
31      {22, 0, 0},
32      {23, 0, 0},
33      {24, 0, 0},
34      {25, 0, 0},
35      {26, 0, 0},
36      {27, 0, 0},
37      {28, 0, 0},
38      {29, 0, 0},
39      {30, 0, 0},
40      {31, 0, 0},
41      {32, 0, 0},
42      {33, 0, 0},
43      {34, 0, 0},
44      {35, 0, 0},
45      {36, 0, 0},
46      {37, 0, 0},
47      {38, 0, 0},
48      {39, 0, 0},
49      {40, 0, 0},
50      {41, 0, 0},
51      {42, 0, 0},
52      {43, 0, 0},
53      {44, 0, 0},
54      {45, 0, 0},
55      {46, 0, 0},
56      {47, 0, 0},
57      {48, 0, 0},
58      {49, 0, 0},
59      {50, 0, 0},
60      {51, 0, 0},
61      {52, 0, 0},
62      {53, 0, 0},
63      {54, 0, 0},
64      {55, 0, 0},
65      {56, 0, 0},
66      {57, 0, 0},
67      {58, 0, 0},
68      {59, 0, 0},
69      {60, 0, 0},
70      {61, 0, 0},
71      {62, 0, 0},
72      {63, 0, 0},
73      {64, 0, 0},
74      {65, 0, 0},
75      {66, 0, 0},
76      {67, 0, 0},
77      {68, 0, 0},
78      {69, 0, 0},
79      {70, 0, 0},
80      {71, 0, 0},
81      {72, 0, 0},
82      {73, 0, 0},
83      {74, 0, 0},
84      {75, 0, 0},
85      {76, 0, 0},
86      {77, 0, 0},
87      {78, 0, 0},
88      {79, 0, 0},
89      {80, 0, 0},
90      {81, 0, 0},
```

```
91      {82, 0, 0},
92      {83, 0, 0},
93      {84, 0, 0},
94      {85, 0, 0},
95      {86, 0, 0},
96      {87, 0, 0},
97      {88, 0, 0},
98      {89, 0, 0},
99      {90, 0, 0},
100     {91, 0, 0},
101     {92, 0, 0},
102     {93, 0, 0},
103     {94, 0, 0},
104     {95, 0, 0},
105     {96, 0, 0},
106     {97, 0, 0},
107     {98, 0, 0},
108     {99, 0, 0},
109     {100, 0, 0},
110     {101, 0, 0},
111     {102, 0, 0},
112     {103, 0, 0},
113     {104, 0, 0},
114     {105, 0, 0},
115     {106, 0, 0},
116     {107, 0, 0},
117     {108, 0, 0},
118     {109, 0, 0},
119     {110, 0, 0},
120     {111, 0, 0},
121     {112, 0, 0},
122     {113, 0, 0},
123     {114, 0, 0},
124     {115, 0, 0},
125     {116, 0, 0},
126     {117, 0, 0},
127     {118, 0, 0},
128     {119, 0, 0},
129     {120, 0, 0},
130     {121, 0, 0},
131     {122, 0, 0},
132     {123, 0, 0},
133     {124, 0, 0},
134     {125, 0, 0},
135     {126, 0, 0},
136     {127, 0, 0},
137     {128, 0, 0},
138     {129, 0, 0},
139     {130, 0, 0},
140     {131, 0, 0},
141     {132, 0, 0},
142     {133, 0, 0},
143     {134, 0, 0},
144     {135, 0, 0},
145     {136, 0, 0},
146     {137, 0, 0},
147     {138, 0, 0},
148     {139, 0, 0},
149     {140, 0, 0},
150     {141, 0, 0},
151     {142, 0, 0},
152     {143, 0, 0},
153     {144, 0, 0},
154     {145, 0, 0},
155     {146, 0, 0},
156     {147, 0, 0},
157     {148, 0, 0},
158     {149, 0, 0},
159     {150, 0, 0},
160     {151, 0, 0},
161     {152, 0, 0},
162     {153, 0, 0},
163     {154, 0, 0},
164     {155, 0, 0},
165     {156, 0, 0},
166     {157, 0, 0},
167     {158, 0, 0},
168     {159, 0, 0},
169     {160, 0, 0},
170     {161, 0, 0},
171     {162, 0, 0},
172     {163, 0, 0},
173     {164, 0, 0},
174     {165, 0, 0},
175     {166, 0, 0},
176     {167, 0, 0},
177     {168, 0, 0},
178     {169, 0, 0},
179     {170, 0, 0},
180     {171, 0, 0},
```



```
181     {172, 0, 0},
182     {173, 0, 0},
183     {174, 0, 0},
184     {175, 0, 0},
185     {176, 0, 0},
186     {177, 0, 0},
187     {178, 0, 0},
188     {179, 0, 0},
189     {180, 0, 0},
190     {181, 0, 0},
191     {182, 0, 0},
192     {183, 0, 0},
193     {184, 0, 0},
194     {185, 0, 0},
195     {186, 0, 0},
196     {187, 0, 0},
197     {188, 0, 0},
198     {189, 0, 0},
199     {190, 0, 0},
200     {191, 0, 0},
201     {192, 0, 0},
202     {193, 0, 0},
203     {194, 0, 0},
204     {195, 0, 0},
205     {196, 0, 0},
206     {197, 0, 0},
207     {198, 0, 0},
208     {199, 0, 0},
209     {200, 0, 0},
210     {201, 0, 0},
211     {202, 0, 0},
212     {203, 0, 0},
213     {204, 0, 0},
214     {205, 0, 0},
215     {206, 0, 0},
216     {207, 0, 0},
217     {208, 0, 0},
218     {209, 0, 0},
219     {210, 0, 0},
220     {211, 0, 0},
221     {212, 0, 0},
222     {213, 0, 0},
223     {214, 0, 0},
224     {215, 0, 0},
225     {216, 0, 0},
226     {217, 0, 0},
227     {218, 0, 0},
228     {219, 0, 0},
229     {220, 0, 0},
230     {221, 0, 0},
231     {222, 0, 0},
232     {223, 0, 0},
233     {224, 0, 0},
234     {225, 0, 0},
235     {226, 0, 0},
236     {227, 0, 0},
237     {228, 0, 0},
238     {229, 0, 0},
239     {230, 0, 0},
240     {231, 0, 0},
241     {232, 0, 0},
242     {233, 0, 0},
243     {234, 0, 0},
244     {235, 0, 0},
245     {236, 0, 0},
246     {237, 0, 0},
247     {238, 0, 0},
248     {239, 0, 0},
249     {240, 0, 0},
250     {241, 0, 0},
251     {242, 0, 0},
252     {243, 0, 0},
253     {244, 0, 0},
254     {245, 0, 0},
255     {246, 0, 0},
256     {247, 0, 0},
257     {248, 0, 0},
258     {249, 0, 0},
259     {250, 0, 0},
260     {251, 0, 0},
261     {252, 0, 0},
262     {253, 0, 0},
263     {254, 0, 0},
264     {255, 0, 0}
265 };
266
267 #endif // RED_MAP_
```

```
#ifndef GREEN_MAP_
#define GREEN_MAP_

/**
 * Color map for RGB green component color image
 */
unsigned char mapGreen[256][3] =
{
    {0, 0, 0},
    {0, 1, 0},
    {0, 2, 0},
    {0, 3, 0},
    {0, 4, 0},
    {0, 5, 0},
    {0, 6, 0},
    {0, 7, 0},
    {0, 8, 0},
    {0, 9, 0},
    {0, 10, 0},
    {0, 11, 0},
    {0, 12, 0},
    {0, 13, 0},
    {0, 14, 0},
    {0, 15, 0},
    {0, 16, 0},
    {0, 17, 0},
    {0, 18, 0},
    {0, 19, 0},
    {0, 20, 0},
    {0, 21, 0},
    {0, 22, 0},
    {0, 23, 0},
    {0, 24, 0},
    {0, 25, 0},
    {0, 26, 0},
    {0, 27, 0},
    {0, 28, 0},
    {0, 29, 0},
    {0, 30, 0},
    {0, 31, 0},
    {0, 32, 0},
    {0, 33, 0},
    {0, 34, 0},
    {0, 35, 0},
    {0, 36, 0},
    {0, 37, 0},
    {0, 38, 0},
    {0, 39, 0},
    {0, 40, 0},
    {0, 41, 0},
    {0, 42, 0},
    {0, 43, 0},
    {0, 44, 0},
    {0, 45, 0},
    {0, 46, 0},
    {0, 47, 0},
    {0, 48, 0},
    {0, 49, 0},
    {0, 50, 0},
    {0, 51, 0},
    {0, 52, 0},
    {0, 53, 0},
    {0, 54, 0},
    {0, 55, 0},
    {0, 56, 0},
    {0, 57, 0},
    {0, 58, 0},
    {0, 59, 0},
    {0, 60, 0},
    {0, 61, 0},
    {0, 62, 0},
    {0, 63, 0},
    {0, 64, 0},
    {0, 65, 0},
    {0, 66, 0},
    {0, 67, 0},
    {0, 68, 0},
    {0, 69, 0},
    {0, 70, 0},
    {0, 71, 0},
    {0, 72, 0},
    {0, 73, 0},
    {0, 74, 0},
    {0, 75, 0},
    {0, 76, 0},
    {0, 77, 0},
    {0, 78, 0},
    {0, 79, 0},
    {0, 80, 0},
    {0, 81, 0},
```



```
    {0, 82, 0},
    {0, 83, 0},
    {0, 84, 0},
    {0, 85, 0},
    {0, 86, 0},
    {0, 87, 0},
    {0, 88, 0},
    {0, 89, 0},
    {0, 90, 0},
    {0, 91, 0},
    {0, 92, 0},
    {0, 93, 0},
    {0, 94, 0},
    {0, 95, 0},
    {0, 96, 0},
    {0, 97, 0},
    {0, 98, 0},
    {0, 99, 0},
    {0, 100, 0},
    {0, 101, 0},
    {0, 102, 0},
    {0, 103, 0},
    {0, 104, 0},
    {0, 105, 0},
    {0, 106, 0},
    {0, 107, 0},
    {0, 108, 0},
    {0, 109, 0},
    {0, 110, 0},
    {0, 111, 0},
    {0, 112, 0},
    {0, 113, 0},
    {0, 114, 0},
    {0, 115, 0},
    {0, 116, 0},
    {0, 117, 0},
    {0, 118, 0},
    {0, 119, 0},
    {0, 120, 0},
    {0, 121, 0},
    {0, 122, 0},
    {0, 123, 0},
    {0, 124, 0},
    {0, 125, 0},
    {0, 126, 0},
    {0, 127, 0},
    {0, 128, 0},
    {0, 129, 0},
    {0, 130, 0},
    {0, 131, 0},
    {0, 132, 0},
    {0, 133, 0},
    {0, 134, 0},
    {0, 135, 0},
    {0, 136, 0},
    {0, 137, 0},
    {0, 138, 0},
    {0, 139, 0},
    {0, 140, 0},
    {0, 141, 0},
    {0, 142, 0},
    {0, 143, 0},
    {0, 144, 0},
    {0, 145, 0},
    {0, 146, 0},
    {0, 147, 0},
    {0, 148, 0},
    {0, 149, 0},
    {0, 150, 0},
    {0, 151, 0},
    {0, 152, 0},
    {0, 153, 0},
    {0, 154, 0},
    {0, 155, 0},
    {0, 156, 0},
    {0, 157, 0},
    {0, 158, 0},
    {0, 159, 0},
    {0, 160, 0},
    {0, 161, 0},
    {0, 162, 0},
    {0, 163, 0},
    {0, 164, 0},
    {0, 165, 0},
    {0, 166, 0},
    {0, 167, 0},
    {0, 168, 0},
    {0, 169, 0},
    {0, 170, 0},
    {0, 171, 0},
```

```
    {0, 172, 0},
    {0, 173, 0},
    {0, 174, 0},
    {0, 175, 0},
    {0, 176, 0},
    {0, 177, 0},
    {0, 178, 0},
    {0, 179, 0},
    {0, 180, 0},
    {0, 181, 0},
    {0, 182, 0},
    {0, 183, 0},
    {0, 184, 0},
    {0, 185, 0},
    {0, 186, 0},
    {0, 187, 0},
    {0, 188, 0},
    {0, 189, 0},
    {0, 190, 0},
    {0, 191, 0},
    {0, 192, 0},
    {0, 193, 0},
    {0, 194, 0},
    {0, 195, 0},
    {0, 196, 0},
    {0, 197, 0},
    {0, 198, 0},
    {0, 199, 0},
    {0, 200, 0},
    {0, 201, 0},
    {0, 202, 0},
    {0, 203, 0},
    {0, 204, 0},
    {0, 205, 0},
    {0, 206, 0},
    {0, 207, 0},
    {0, 208, 0},
    {0, 209, 0},
    {0, 210, 0},
    {0, 211, 0},
    {0, 212, 0},
    {0, 213, 0},
    {0, 214, 0},
    {0, 215, 0},
    {0, 216, 0},
    {0, 217, 0},
    {0, 218, 0},
    {0, 219, 0},
    {0, 220, 0},
    {0, 221, 0},
    {0, 222, 0},
    {0, 223, 0},
    {0, 224, 0},
    {0, 225, 0},
    {0, 226, 0},
    {0, 227, 0},
    {0, 228, 0},
    {0, 229, 0},
    {0, 230, 0},
    {0, 231, 0},
    {0, 232, 0},
    {0, 233, 0},
    {0, 234, 0},
    {0, 235, 0},
    {0, 236, 0},
    {0, 237, 0},
    {0, 238, 0},
    {0, 239, 0},
    {0, 240, 0},
    {0, 241, 0},
    {0, 242, 0},
    {0, 243, 0},
    {0, 244, 0},
    {0, 245, 0},
    {0, 246, 0},
    {0, 247, 0},
    {0, 248, 0},
    {0, 249, 0},
    {0, 250, 0},
    {0, 251, 0},
    {0, 252, 0},
    {0, 253, 0},
    {0, 254, 0},
    {0, 255, 0}
};

#endif // GREEN_MAP_
```



```
#ifndef BLUE_MAP_
#define BLUE_MAP_

/**
 * Color map for RGB blue component color image
 */
unsigned char mapBlue[256][3] =
{
    {0, 0, 0},
    {0, 0, 1},
    {0, 0, 2},
    {0, 0, 3},
    {0, 0, 4},
    {0, 0, 5},
    {0, 0, 6},
    {0, 0, 7},
    {0, 0, 8},
    {0, 0, 9},
    {0, 0, 10},
    {0, 0, 11},
    {0, 0, 12},
    {0, 0, 13},
    {0, 0, 14},
    {0, 0, 15},
    {0, 0, 16},
    {0, 0, 17},
    {0, 0, 18},
    {0, 0, 19},
    {0, 0, 20},
    {0, 0, 21},
    {0, 0, 22},
    {0, 0, 23},
    {0, 0, 24},
    {0, 0, 25},
    {0, 0, 26},
    {0, 0, 27},
    {0, 0, 28},
    {0, 0, 29},
    {0, 0, 30},
    {0, 0, 31},
    {0, 0, 32},
    {0, 0, 33},
    {0, 0, 34},
    {0, 0, 35},
    {0, 0, 36},
    {0, 0, 37},
    {0, 0, 38},
    {0, 0, 39},
    {0, 0, 40},
    {0, 0, 41},
    {0, 0, 42},
    {0, 0, 43},
    {0, 0, 44},
    {0, 0, 45},
    {0, 0, 46},
    {0, 0, 47},
    {0, 0, 48},
    {0, 0, 49},
    {0, 0, 50},
    {0, 0, 51},
    {0, 0, 52},
    {0, 0, 53},
    {0, 0, 54},
    {0, 0, 55},
    {0, 0, 56},
    {0, 0, 57},
    {0, 0, 58},
    {0, 0, 59},
    {0, 0, 60},
    {0, 0, 61},
    {0, 0, 62},
    {0, 0, 63},
    {0, 0, 64},
    {0, 0, 65},
    {0, 0, 66},
    {0, 0, 67},
    {0, 0, 68},
    {0, 0, 69},
    {0, 0, 70},
    {0, 0, 71},
    {0, 0, 72},
    {0, 0, 73},
    {0, 0, 74},
    {0, 0, 75},
    {0, 0, 76},
    {0, 0, 77},
    {0, 0, 78},
    {0, 0, 79},
    {0, 0, 80},
    {0, 0, 81},
```

```
91      {0, 0, 82},
92      {0, 0, 83},
93      {0, 0, 84},
94      {0, 0, 85},
95      {0, 0, 86},
96      {0, 0, 87},
97      {0, 0, 88},
98      {0, 0, 89},
99      {0, 0, 90},
100     {0, 0, 91},
101     {0, 0, 92},
102     {0, 0, 93},
103     {0, 0, 94},
104     {0, 0, 95},
105     {0, 0, 96},
106     {0, 0, 97},
107     {0, 0, 98},
108     {0, 0, 99},
109     {0, 0, 100},
110     {0, 0, 101},
111     {0, 0, 102},
112     {0, 0, 103},
113     {0, 0, 104},
114     {0, 0, 105},
115     {0, 0, 106},
116     {0, 0, 107},
117     {0, 0, 108},
118     {0, 0, 109},
119     {0, 0, 110},
120     {0, 0, 111},
121     {0, 0, 112},
122     {0, 0, 113},
123     {0, 0, 114},
124     {0, 0, 115},
125     {0, 0, 116},
126     {0, 0, 117},
127     {0, 0, 118},
128     {0, 0, 119},
129     {0, 0, 120},
130     {0, 0, 121},
131     {0, 0, 122},
132     {0, 0, 123},
133     {0, 0, 124},
134     {0, 0, 125},
135     {0, 0, 126},
136     {0, 0, 127},
137     {0, 0, 128},
138     {0, 0, 129},
139     {0, 0, 130},
140     {0, 0, 131},
141     {0, 0, 132},
142     {0, 0, 133},
143     {0, 0, 134},
144     {0, 0, 135},
145     {0, 0, 136},
146     {0, 0, 137},
147     {0, 0, 138},
148     {0, 0, 139},
149     {0, 0, 140},
150     {0, 0, 141},
151     {0, 0, 142},
152     {0, 0, 143},
153     {0, 0, 144},
154     {0, 0, 145},
155     {0, 0, 146},
156     {0, 0, 147},
157     {0, 0, 148},
158     {0, 0, 149},
159     {0, 0, 150},
160     {0, 0, 151},
161     {0, 0, 152},
162     {0, 0, 153},
163     {0, 0, 154},
164     {0, 0, 155},
165     {0, 0, 156},
166     {0, 0, 157},
167     {0, 0, 158},
168     {0, 0, 159},
169     {0, 0, 160},
170     {0, 0, 161},
171     {0, 0, 162},
172     {0, 0, 163},
173     {0, 0, 164},
174     {0, 0, 165},
175     {0, 0, 166},
176     {0, 0, 167},
177     {0, 0, 168},
178     {0, 0, 169},
179     {0, 0, 170},
180     {0, 0, 171},
```



```
181     {0, 0, 172},
182     {0, 0, 173},
183     {0, 0, 174},
184     {0, 0, 175},
185     {0, 0, 176},
186     {0, 0, 177},
187     {0, 0, 178},
188     {0, 0, 179},
189     {0, 0, 180},
190     {0, 0, 181},
191     {0, 0, 182},
192     {0, 0, 183},
193     {0, 0, 184},
194     {0, 0, 185},
195     {0, 0, 186},
196     {0, 0, 187},
197     {0, 0, 188},
198     {0, 0, 189},
199     {0, 0, 190},
200     {0, 0, 191},
201     {0, 0, 192},
202     {0, 0, 193},
203     {0, 0, 194},
204     {0, 0, 195},
205     {0, 0, 196},
206     {0, 0, 197},
207     {0, 0, 198},
208     {0, 0, 199},
209     {0, 0, 200},
210     {0, 0, 201},
211     {0, 0, 202},
212     {0, 0, 203},
213     {0, 0, 204},
214     {0, 0, 205},
215     {0, 0, 206},
216     {0, 0, 207},
217     {0, 0, 208},
218     {0, 0, 209},
219     {0, 0, 210},
220     {0, 0, 211},
221     {0, 0, 212},
222     {0, 0, 213},
223     {0, 0, 214},
224     {0, 0, 215},
225     {0, 0, 216},
226     {0, 0, 217},
227     {0, 0, 218},
228     {0, 0, 219},
229     {0, 0, 220},
230     {0, 0, 221},
231     {0, 0, 222},
232     {0, 0, 223},
233     {0, 0, 224},
234     {0, 0, 225},
235     {0, 0, 226},
236     {0, 0, 227},
237     {0, 0, 228},
238     {0, 0, 229},
239     {0, 0, 230},
240     {0, 0, 231},
241     {0, 0, 232},
242     {0, 0, 233},
243     {0, 0, 234},
244     {0, 0, 235},
245     {0, 0, 236},
246     {0, 0, 237},
247     {0, 0, 238},
248     {0, 0, 239},
249     {0, 0, 240},
250     {0, 0, 241},
251     {0, 0, 242},
252     {0, 0, 243},
253     {0, 0, 244},
254     {0, 0, 245},
255     {0, 0, 246},
256     {0, 0, 247},
257     {0, 0, 248},
258     {0, 0, 249},
259     {0, 0, 250},
260     {0, 0, 251},
261     {0, 0, 252},
262     {0, 0, 253},
263     {0, 0, 254},
264     {0, 0, 255}
265 };
266
267 #endif // BLUE_MAP_
```

```
#ifndef HSV_MAP_
#define HSV_MAP_

/**
 * Color map for HSV hue component color image.
 * Color circle colormap starting with red, yellow, green, cyan, blue, magenta,
 * and red again.
 */
unsigned char mapHSV[256][3] =
{
    {255, 0, 0},
    {255, 6, 0},
    {255, 12, 0},
    {255, 18, 0},
    {255, 24, 0},
    {255, 30, 0},
    {255, 36, 0},
    {255, 42, 0},
    {255, 48, 0},
    {255, 54, 0},
    {255, 60, 0},
    {255, 66, 0},
    {255, 72, 0},
    {255, 78, 0},
    {255, 84, 0},
    {255, 90, 0},
    {255, 96, 0},
    {255, 102, 0},
    {255, 108, 0},
    {255, 114, 0},
    {255, 120, 0},
    {255, 126, 0},
    {255, 131, 0},
    {255, 137, 0},
    {255, 143, 0},
    {255, 149, 0},
    {255, 155, 0},
    {255, 161, 0},
    {255, 167, 0},
    {255, 173, 0},
    {255, 179, 0},
    {255, 185, 0},
    {255, 191, 0},
    {255, 197, 0},
    {255, 203, 0},
    {255, 209, 0},
    {255, 215, 0},
    {255, 221, 0},
    {255, 227, 0},
    {255, 233, 0},
    {255, 239, 0},
    {255, 245, 0},
    {255, 251, 0},
    {253, 255, 0},
    {247, 255, 0},
    {241, 255, 0},
    {235, 255, 0},
    {229, 255, 0},
    {223, 255, 0},
    {217, 255, 0},
    {211, 255, 0},
    {205, 255, 0},
    {199, 255, 0},
    {193, 255, 0},
    {187, 255, 0},
    {181, 255, 0},
    {175, 255, 0},
    {169, 255, 0},
    {163, 255, 0},
    {157, 255, 0},
    {151, 255, 0},
    {145, 255, 0},
    {139, 255, 0},
    {133, 255, 0},
    {128, 255, 0},
    {122, 255, 0},
    {116, 255, 0},
    {110, 255, 0},
    {104, 255, 0},
    {98, 255, 0},
    {92, 255, 0},
    {86, 255, 0},
    {80, 255, 0},
    {74, 255, 0},
    {68, 255, 0},
    {62, 255, 0},
    {56, 255, 0},
    {50, 255, 0},
    {44, 255, 0},
    {38, 255, 0},
```



```
    {32, 255, 0},
    {26, 255, 0},
    {20, 255, 0},
    {14, 255, 0},
    {8, 255, 0},
    {2, 255, 0},
    {0, 255, 4},
    {0, 255, 10},
    {0, 255, 16},
    {0, 255, 22},
    {0, 255, 28},
    {0, 255, 34},
    {0, 255, 40},
    {0, 255, 46},
    {0, 255, 52},
    {0, 255, 58},
    {0, 255, 64},
    {0, 255, 70},
    {0, 255, 76},
    {0, 255, 82},
    {0, 255, 88},
    {0, 255, 94},
    {0, 255, 100},
    {0, 255, 106},
    {0, 255, 112},
    {0, 255, 118},
    {0, 255, 124},
    {0, 255, 129},
    {0, 255, 135},
    {0, 255, 141},
    {0, 255, 147},
    {0, 255, 153},
    {0, 255, 159},
    {0, 255, 165},
    {0, 255, 171},
    {0, 255, 177},
    {0, 255, 183},
    {0, 255, 189},
    {0, 255, 195},
    {0, 255, 201},
    {0, 255, 207},
    {0, 255, 213},
    {0, 255, 219},
    {0, 255, 225},
    {0, 255, 231},
    {0, 255, 237},
    {0, 255, 243},
    {0, 255, 249},
    {0, 255, 255},
    {0, 249, 255},
    {0, 243, 255},
    {0, 237, 255},
    {0, 231, 255},
    {0, 225, 255},
    {0, 219, 255},
    {0, 213, 255},
    {0, 207, 255},
    {0, 201, 255},
    {0, 195, 255},
    {0, 189, 255},
    {0, 183, 255},
    {0, 177, 255},
    {0, 171, 255},
    {0, 165, 255},
    {0, 159, 255},
    {0, 153, 255},
    {0, 147, 255},
    {0, 141, 255},
    {0, 135, 255},
    {0, 129, 255},
    {0, 124, 255},
    {0, 118, 255},
    {0, 112, 255},
    {0, 106, 255},
    {0, 100, 255},
    {0, 94, 255},
    {0, 88, 255},
    {0, 82, 255},
    {0, 76, 255},
    {0, 70, 255},
    {0, 64, 255},
    {0, 58, 255},
    {0, 52, 255},
    {0, 46, 255},
    {0, 40, 255},
    {0, 34, 255},
    {0, 28, 255},
    {0, 22, 255},
    {0, 16, 255},
    {0, 10, 255},
```

```
    {0, 4, 255},
    {2, 0, 255},
    {8, 0, 255},
    {14, 0, 255},
    {20, 0, 255},
    {26, 0, 255},
    {32, 0, 255},
    {38, 0, 255},
    {44, 0, 255},
    {50, 0, 255},
    {56, 0, 255},
    {62, 0, 255},
    {68, 0, 255},
    {74, 0, 255},
    {80, 0, 255},
    {86, 0, 255},
    {92, 0, 255},
    {98, 0, 255},
    {104, 0, 255},
    {110, 0, 255},
    {116, 0, 255},
    {122, 0, 255},
    {128, 0, 255},
    {133, 0, 255},
    {139, 0, 255},
    {145, 0, 255},
    {151, 0, 255},
    {157, 0, 255},
    {163, 0, 255},
    {169, 0, 255},
    {175, 0, 255},
    {181, 0, 255},
    {187, 0, 255},
    {193, 0, 255},
    {199, 0, 255},
    {205, 0, 255},
    {211, 0, 255},
    {217, 0, 255},
    {223, 0, 255},
    {229, 0, 255},
    {235, 0, 255},
    {241, 0, 255},
    {247, 0, 255},
    {253, 0, 255},
    {255, 0, 251},
    {255, 0, 245},
    {255, 0, 239},
    {255, 0, 233},
    {255, 0, 227},
    {255, 0, 221},
    {255, 0, 215},
    {255, 0, 209},
    {255, 0, 203},
    {255, 0, 197},
    {255, 0, 191},
    {255, 0, 185},
    {255, 0, 179},
    {255, 0, 173},
    {255, 0, 167},
    {255, 0, 161},
    {255, 0, 155},
    {255, 0, 149},
    {255, 0, 143},
    {255, 0, 137},
    {255, 0, 131},
    {255, 0, 126},
    {255, 0, 120},
    {255, 0, 114},
    {255, 0, 108},
    {255, 0, 102},
    {255, 0, 96},
    {255, 0, 90},
    {255, 0, 84},
    {255, 0, 78},
    {255, 0, 72},
    {255, 0, 66},
    {255, 0, 60},
    {255, 0, 54},
    {255, 0, 48},
    {255, 0, 42},
    {255, 0, 36},
    {255, 0, 30},
    {255, 0, 24},
    {255, 0, 18},
    {255, 0, 12},
    {255, 0, 6}
};

#endif // HSV_MAP_
```



```
#ifndef CR_MAP_
#define CR_MAP_

/**
 * Color map for YCbCr Cr component color image.
 * Green to Magenta colormap
 */
unsigned char mapCr[256][3] =
{
    {0, 255, 0},
    {1, 254, 1},
    {2, 253, 2},
    {3, 252, 3},
    {4, 251, 4},
    {5, 250, 5},
    {6, 249, 6},
    {7, 248, 7},
    {8, 247, 8},
    {9, 246, 9},
    {10, 245, 10},
    {11, 244, 11},
    {12, 243, 12},
    {13, 242, 13},
    {14, 241, 14},
    {15, 240, 15},
    {16, 239, 16},
    {17, 238, 17},
    {18, 237, 18},
    {19, 236, 19},
    {20, 235, 20},
    {21, 234, 21},
    {22, 233, 22},
    {23, 232, 23},
    {24, 231, 24},
    {25, 230, 25},
    {26, 229, 26},
    {27, 228, 27},
    {28, 227, 28},
    {29, 226, 29},
    {30, 225, 30},
    {31, 224, 31},
    {32, 223, 32},
    {33, 222, 33},
    {34, 221, 34},
    {35, 220, 35},
    {36, 219, 36},
    {37, 218, 37},
    {38, 217, 38},
    {39, 216, 39},
    {40, 215, 40},
    {41, 214, 41},
    {42, 213, 42},
    {43, 212, 43},
    {44, 211, 44},
    {45, 210, 45},
    {46, 209, 46},
    {47, 208, 47},
    {48, 207, 48},
    {49, 206, 49},
    {50, 205, 50},
    {51, 204, 51},
    {52, 203, 52},
    {53, 202, 53},
    {54, 201, 54},
    {55, 200, 55},
    {56, 199, 56},
    {57, 198, 57},
    {58, 197, 58},
    {59, 196, 59},
    {60, 195, 60},
    {61, 194, 61},
    {62, 193, 62},
    {63, 192, 63},
    {64, 191, 64},
    {65, 190, 65},
    {66, 189, 66},
    {67, 188, 67},
    {68, 187, 68},
    {69, 186, 69},
    {70, 185, 70},
    {71, 184, 71},
    {72, 183, 72},
    {73, 182, 73},
    {74, 181, 74},
    {75, 180, 75},
    {76, 179, 76},
    {77, 178, 77},
    {78, 177, 78},
    {79, 176, 79},
    {80, 175, 80},
```

```
91      {81, 174, 81},
92      {82, 173, 82},
93      {83, 172, 83},
94      {84, 171, 84},
95      {85, 170, 85},
96      {86, 169, 86},
97      {87, 168, 87},
98      {88, 167, 88},
99      {89, 166, 89},
100     {90, 165, 90},
101     {91, 164, 91},
102     {92, 163, 92},
103     {93, 162, 93},
104     {94, 161, 94},
105     {95, 160, 95},
106     {96, 159, 96},
107     {97, 158, 97},
108     {98, 157, 98},
109     {99, 156, 99},
110     {100, 155, 100},
111     {101, 154, 101},
112     {102, 153, 102},
113     {103, 152, 103},
114     {104, 151, 104},
115     {105, 150, 105},
116     {106, 149, 106},
117     {107, 148, 107},
118     {108, 147, 108},
119     {109, 146, 109},
120     {110, 145, 110},
121     {111, 144, 111},
122     {112, 143, 112},
123     {113, 142, 113},
124     {114, 141, 114},
125     {115, 140, 115},
126     {116, 139, 116},
127     {117, 138, 117},
128     {118, 137, 118},
129     {119, 136, 119},
130     {120, 135, 120},
131     {121, 134, 121},
132     {122, 133, 122},
133     {123, 132, 123},
134     {124, 131, 124},
135     {125, 130, 125},
136     {126, 129, 126},
137     {127, 128, 127},
138     {128, 127, 128},
139     {129, 126, 129},
140     {130, 125, 130},
141     {131, 124, 131},
142     {132, 123, 132},
143     {133, 122, 133},
144     {134, 121, 134},
145     {135, 120, 135},
146     {136, 119, 136},
147     {137, 118, 137},
148     {138, 117, 138},
149     {139, 116, 139},
150     {140, 115, 140},
151     {141, 114, 141},
152     {142, 113, 142},
153     {143, 112, 143},
154     {144, 111, 144},
155     {145, 110, 145},
156     {146, 109, 146},
157     {147, 108, 147},
158     {148, 107, 148},
159     {149, 106, 149},
160     {150, 105, 150},
161     {151, 104, 151},
162     {152, 103, 152},
163     {153, 102, 153},
164     {154, 101, 154},
165     {155, 100, 155},
166     {156, 99, 156},
167     {157, 98, 157},
168     {158, 97, 158},
169     {159, 96, 159},
170     {160, 95, 160},
171     {161, 94, 161},
172     {162, 93, 162},
173     {163, 92, 163},
174     {164, 91, 164},
175     {165, 90, 165},
176     {166, 89, 166},
177     {167, 88, 167},
178     {168, 87, 168},
179     {169, 86, 169},
180     {170, 85, 170},
```



```
181     {171, 84, 171},
182     {172, 83, 172},
183     {173, 82, 173},
184     {174, 81, 174},
185     {175, 80, 175},
186     {176, 79, 176},
187     {177, 78, 177},
188     {178, 77, 178},
189     {179, 76, 179},
190     {180, 75, 180},
191     {181, 74, 181},
192     {182, 73, 182},
193     {183, 72, 183},
194     {184, 71, 184},
195     {185, 70, 185},
196     {186, 69, 186},
197     {187, 68, 187},
198     {188, 67, 188},
199     {189, 66, 189},
200     {190, 65, 190},
201     {191, 64, 191},
202     {192, 63, 192},
203     {193, 62, 193},
204     {194, 61, 194},
205     {195, 60, 195},
206     {196, 59, 196},
207     {197, 58, 197},
208     {198, 57, 198},
209     {199, 56, 199},
210     {200, 55, 200},
211     {201, 54, 201},
212     {202, 53, 202},
213     {203, 52, 203},
214     {204, 51, 204},
215     {205, 50, 205},
216     {206, 49, 206},
217     {207, 48, 207},
218     {208, 47, 208},
219     {209, 46, 209},
220     {210, 45, 210},
221     {211, 44, 211},
222     {212, 43, 212},
223     {213, 42, 213},
224     {214, 41, 214},
225     {215, 40, 215},
226     {216, 39, 216},
227     {217, 38, 217},
228     {218, 37, 218},
229     {219, 36, 219},
230     {220, 35, 220},
231     {221, 34, 221},
232     {222, 33, 222},
233     {223, 32, 223},
234     {224, 31, 224},
235     {225, 30, 225},
236     {226, 29, 226},
237     {227, 28, 227},
238     {228, 27, 228},
239     {229, 26, 229},
240     {230, 25, 230},
241     {231, 24, 231},
242     {232, 23, 232},
243     {233, 22, 233},
244     {234, 21, 234},
245     {235, 20, 235},
246     {236, 19, 236},
247     {237, 18, 237},
248     {238, 17, 238},
249     {239, 16, 239},
250     {240, 15, 240},
251     {241, 14, 241},
252     {242, 13, 242},
253     {243, 12, 243},
254     {244, 11, 244},
255     {245, 10, 245},
256     {246, 9, 246},
257     {247, 8, 247},
258     {248, 7, 248},
259     {249, 6, 249},
260     {250, 5, 250},
261     {251, 4, 251},
262     {252, 3, 252},
263     {253, 2, 253},
264     {254, 1, 254},
265     {255, 0, 255}
266 };
267
268 #endif // CR_MAP_
```

```
#ifndef CB_MAP_
#define CB_MAP_

/**
 * Color map for YCbCr Cb component color image.
 * Yellow to Blue colormap
 */
unsigned char mapCb[256][3] =
{
    {255, 255, 0},
    {254, 254, 1},
    {253, 253, 2},
    {252, 252, 3},
    {251, 251, 4},
    {250, 250, 5},
    {249, 249, 6},
    {248, 248, 7},
    {247, 247, 8},
    {246, 246, 9},
    {245, 245, 10},
    {244, 244, 11},
    {243, 243, 12},
    {242, 242, 13},
    {241, 241, 14},
    {240, 240, 15},
    {239, 239, 16},
    {238, 238, 17},
    {237, 237, 18},
    {236, 236, 19},
    {235, 235, 20},
    {234, 234, 21},
    {233, 233, 22},
    {232, 232, 23},
    {231, 231, 24},
    {230, 230, 25},
    {229, 229, 26},
    {228, 228, 27},
    {227, 227, 28},
    {226, 226, 29},
    {225, 225, 30},
    {224, 224, 31},
    {223, 223, 32},
    {222, 222, 33},
    {221, 221, 34},
    {220, 220, 35},
    {219, 219, 36},
    {218, 218, 37},
    {217, 217, 38},
    {216, 216, 39},
    {215, 215, 40},
    {214, 214, 41},
    {213, 213, 42},
    {212, 212, 43},
    {211, 211, 44},
    {210, 210, 45},
    {209, 209, 46},
    {208, 208, 47},
    {207, 207, 48},
    {206, 206, 49},
    {205, 205, 50},
    {204, 204, 51},
    {203, 203, 52},
    {202, 202, 53},
    {201, 201, 54},
    {200, 200, 55},
    {199, 199, 56},
    {198, 198, 57},
    {197, 197, 58},
    {196, 196, 59},
    {195, 195, 60},
    {194, 194, 61},
    {193, 193, 62},
    {192, 192, 63},
    {191, 191, 64},
    {190, 190, 65},
    {189, 189, 66},
    {188, 188, 67},
    {187, 187, 68},
    {186, 186, 69},
    {185, 185, 70},
    {184, 184, 71},
    {183, 183, 72},
    {182, 182, 73},
    {181, 181, 74},
    {180, 180, 75},
    {179, 179, 76},
    {178, 178, 77},
    {177, 177, 78},
    {176, 176, 79},
    {175, 175, 80},
```



```
    {174, 174, 81},
    {173, 173, 82},
    {172, 172, 83},
    {171, 171, 84},
    {170, 170, 85},
    {169, 169, 86},
    {168, 168, 87},
    {167, 167, 88},
    {166, 166, 89},
    {165, 165, 90},
    {164, 164, 91},
    {163, 163, 92},
    {162, 162, 93},
    {161, 161, 94},
    {160, 160, 95},
    {159, 159, 96},
    {158, 158, 97},
    {157, 157, 98},
    {156, 156, 99},
    {155, 155, 100},
    {154, 154, 101},
    {153, 153, 102},
    {152, 152, 103},
    {151, 151, 104},
    {150, 150, 105},
    {149, 149, 106},
    {148, 148, 107},
    {147, 147, 108},
    {146, 146, 109},
    {145, 145, 110},
    {144, 144, 111},
    {143, 143, 112},
    {142, 142, 113},
    {141, 141, 114},
    {140, 140, 115},
    {139, 139, 116},
    {138, 138, 117},
    {137, 137, 118},
    {136, 136, 119},
    {135, 135, 120},
    {134, 134, 121},
    {133, 133, 122},
    {132, 132, 123},
    {131, 131, 124},
    {130, 130, 125},
    {129, 129, 126},
    {128, 128, 127},
    {127, 127, 128},
    {126, 126, 129},
    {125, 125, 130},
    {124, 124, 131},
    {123, 123, 132},
    {122, 122, 133},
    {121, 121, 134},
    {120, 120, 135},
    {119, 119, 136},
    {118, 118, 137},
    {117, 117, 138},
    {116, 116, 139},
    {115, 115, 140},
    {114, 114, 141},
    {113, 113, 142},
    {112, 112, 143},
    {111, 111, 144},
    {110, 110, 145},
    {109, 109, 146},
    {108, 108, 147},
    {107, 107, 148},
    {106, 106, 149},
    {105, 105, 150},
    {104, 104, 151},
    {103, 103, 152},
    {102, 102, 153},
    {101, 101, 154},
    {100, 100, 155},
    {99, 99, 156},
    {98, 98, 157},
    {97, 97, 158},
    {96, 96, 159},
    {95, 95, 160},
    {94, 94, 161},
    {93, 93, 162},
    {92, 92, 163},
    {91, 91, 164},
    {90, 90, 165},
    {89, 89, 166},
    {88, 88, 167},
    {87, 87, 168},
    {86, 86, 169},
    {85, 85, 170},
```

```
    {84, 84, 171},
    {83, 83, 172},
    {82, 82, 173},
    {81, 81, 174},
    {80, 80, 175},
    {79, 79, 176},
    {78, 78, 177},
    {77, 77, 178},
    {76, 76, 179},
    {75, 75, 180},
    {74, 74, 181},
    {73, 73, 182},
    {72, 72, 183},
    {71, 71, 184},
    {70, 70, 185},
    {69, 69, 186},
    {68, 68, 187},
    {67, 67, 188},
    {66, 66, 189},
    {65, 65, 190},
    {64, 64, 191},
    {63, 63, 192},
    {62, 62, 193},
    {61, 61, 194},
    {60, 60, 195},
    {59, 59, 196},
    {58, 58, 197},
    {57, 57, 198},
    {56, 56, 199},
    {55, 55, 200},
    {54, 54, 201},
    {53, 53, 202},
    {52, 52, 203},
    {51, 51, 204},
    {50, 50, 205},
    {49, 49, 206},
    {48, 48, 207},
    {47, 47, 208},
    {46, 46, 209},
    {45, 45, 210},
    {44, 44, 211},
    {43, 43, 212},
    {42, 42, 213},
    {41, 41, 214},
    {40, 40, 215},
    {39, 39, 216},
    {38, 38, 217},
    {37, 37, 218},
    {36, 36, 219},
    {35, 35, 220},
    {34, 34, 221},
    {33, 33, 222},
    {32, 32, 223},
    {31, 31, 224},
    {30, 30, 225},
    {29, 29, 226},
    {28, 28, 227},
    {27, 27, 228},
    {26, 26, 229},
    {25, 25, 230},
    {24, 24, 231},
    {23, 23, 232},
    {22, 22, 233},
    {21, 21, 234},
    {20, 20, 235},
    {19, 19, 236},
    {18, 18, 237},
    {17, 17, 238},
    {16, 16, 239},
    {15, 15, 240},
    {14, 14, 241},
    {13, 13, 242},
    {12, 12, 243},
    {11, 11, 244},
    {10, 10, 245},
    {9, 9, 246},
    {8, 8, 247},
    {7, 7, 248},
    {6, 6, 249},
    {5, 5, 250},
    {4, 4, 251},
    {3, 3, 252},
    {2, 2, 253},
    {1, 1, 254},
    {0, 0, 255}
};

#endif // CB_MAP_
```



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

using namespace std;

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

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

/**
 * Test program OpenCV2 + QT5
 * @param argc argument count
 * @param argv argument values
 * @return QTApp return value
 * @par usage : <Progname> [--device | -d] <#> | [--file | -f ] <filename>
 * [--mirror | -m] [--size | -s] <width>x<height>
 *  - device : [--device | -d] <device #> (0, 1, ...) Opens capture device #
 *  - filename : [--file | -f ] <filename> Opens a video file or URL (including rtsp)
 *  - mirror : mirrors image horizontally before display
 *  - render : use QImage and QLabel or QGLWidget for image rendering in QtWidget
 *      [-r | --render] [IM | LBL | GL]
 *      - IM for image rendering with painter
 *      - LBL for image in Label rendering
 *      - GL for OpenGL rendering
 *  - size : [--size | -s] <width>x<height> resize capture to fit desired <width>
 *  and <height>
 */
int main(int argc, char *argv[])
{
    CvProcessor::VerboseLevel verboseLevel = CvProcessor::VERBOSE_WARNINGS; // verbose up to notif
    // CvProcessor::VerboseLevel verboseLevel = CvProcessor::VERBOSE_ACTIVITY; // verbose all

    // ----------------------------------------------------------------------
    // Instanciate QApplication to receive special QT args
    // ----------------------------------------------------------------------
    QApplication app(argc, argv);

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

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

        if (currentArg == "-t" || currentArg =="--threads")
        {
            // Next argument should be thread number integer
            if (it.hasNext())
            {
                QString threadString(it.next());
                bool convertOk;
                threadNumber = threadString.toInt(&convertOk,10);
                if (!convertOk || threadNumber < 1 || threadNumber > 3)
                {
                    qWarning("Warning: Invalid thread number %d",threadNumber);
                    threadNumber = 3;
                }
            }
            else
            {
                qWarning("Warning: thread tag found with no following thread number");
            }
        }
        else if (currentArg == "-v" || currentArg == "--verbose")
        {
            // next argument should be a verbose level (from 0 to 4)
            if (it.hasNext())
            {
                QString verboseLevelString(it.next());
                bool convertOk;
                int newVerboseLevel = verboseLevelString.toUInt(&convertOk, 10);
                if (!convertOk ||
                    newVerboseLevel < 0 ||
                    newVerboseLevel > (int)CvProcessor::NBVERBOSELEVEL)
                {
                    qWarning("Invalid verbose level %d", newVerboseLevel);
                }
                else
                {
```

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



```
181
182     usage(argv[0]);
183
184     int retVal = app.exec();
185
186     // ----------------------------------------------------------------------
187     // Cleanup & return
188     // ----------------------------------------------------------------------
189     delete capture; // Should quit the capThread if any
190     delete colorSpace; // Should quit the procThread if any
191
192     bool sameThread = capThread ≡ procThread;
193
194     if (capThread ≠ NULL)
195     {
196         delete capThread;
197     }
198
199     if (procThread ≠ NULL ∧ ¬sameThread)
200     {
201         delete procThread;
202     }
203
204     return retVal;
205 }
206
207 /*
208  * Usage function shown just before launching QApp
209  * @param name the name of the program (argv[0])
210  */
211 void usage(char * name)
212 {
213     cout << "usage : " << basename(name) << " "
214          << "[-d | --device] <device number> "
215          << "[-v | --video] <video file> "
216          << "[-s | --size] <width>x<height> "
217          << "[-m | --mirror] " << endl
218          << "\t if no argument provided try to open first webcam" << endl
219          << "Key help : components multiple keystrokes switches from colored "
220          << "to B&W component display" << endl
221          << "\ti : Show color input image" << endl
222          << "\tk : Show lightness image" << endl
223          << "\tr : Show red component image from RGB color model" << endl
224          << "\tg : Show green component image from RGB color model" << endl
225          << "\tb : Show blue component image from RGB color model" << endl
226          << "\tx : Show X component image from XYZ color model" << endl
227          << "\tw : Show Y component image from XYZ color model" << endl
228          << "\tz : Show Z component image from XYZ color model" << endl
229          << "\th : Show hue component image from HSV color model" << endl
230          << "\ts : Show saturation component image from HSV color model" << endl
231          << "\tv : Show value component image from HSV color model" << endl
232          << "\ty : Show lightness image from YCbCr color model" << endl
233          << "\tu : Show Cr component image from YCbCr color model" << endl
234          << "\tt : Show Cb component image from YCbCr color model" << endl
235          << "\te : prints this help" << endl;
236 }
```