```
/*
 * 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;
```



```cpp
            size_t searchComment;
            int readValues[COMPSIZE];

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

                lineCount++;

                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();
```

```
165         }
166         else // inputFile is not opened
167         {
168             cerr << "Palette::Palette(" << filename << ") : unable to open file"
169                  << endl;
170             exit(EXIT_FAILURE);
171         }
172     }
173     else // filename is NULL
174     {
175         cerr << "Palette::Palette(NULL filename) : empty file name" << endl;
176         exit(EXIT_FAILURE);
177     }
178 }
179
180 /*
181  * Palette destructor.
182  * Relese all images and clear vectors
183  */
184 Palette::~Palette()
185 {
186     // Release matrices
187     for (size_t i=0; i < colormap.size(); i++)
188     {
189         colormap[i].release();
190         BGRChannels[i].release();
191     }
192
193     // Clear vectors
194     colormap.clear();
195     BGRChannels.clear();
196 }
197
198 /*
199  * Apply the colormap on the single channel source image to build
200  * a destination 3 channels color image.
201  * @param src source mono-channel image
202  * @param dst destination BGR-BGRChannels image
203  */
204 void Palette::applyPalette(const Mat & src, Mat & dst)
205 {
206     const size_t BGR2RGB[CMAPSIZE] = {2,1,0};
207
208     // checks if source has only one channel
209     if (src.channels() ≡ 1)
210     {
211         if (¬channelsAllocated) // BGRChannels should be allocated first
212         {
213             for (size_t i=0; i < BGRChannels.size(); i++)
214             {
215                 BGRChannels[i].create(src.size(),CV_8UC1);
216             }
217             channelsAllocated = true;
218         }
219
220         if (src.size() ≠ BGRChannels[0].size()) // BGRChannels should be reallocated
221         {
222             for (size_t i=0; i < BGRChannels.size(); i++)
223             {
224                 BGRChannels[i].release();
225                 BGRChannels[i].create(src.size(),CV_8UC1);
226             }
227         }
228
229         // Apply Look Up Table on each channel
230         for (size_t i=0; i < COMPSIZE; i++)
231         {
232             LUT(src,colormap[BGR2RGB[i]],BGRChannels[i]);
233         }
234
235         // then merge all the BGRChannels into a BGR image
236         merge(BGRChannels, dst);
237     }
238     else // source has multiple channels
239     {
240         cerr << "Palette::applyColormap(...) : source has " << src.channels()
241              << " channels" << endl;
242     }
243 }
```



```
1  /*
2   * CvProcessor.h
3   *
4   *  Created on: 21 fÃ©vr. 2012
5   *      Author: davidroussel
6   */
7
8  #ifndef CVPROCESSOR_H_
9  #define CVPROCESSOR_H_
10
11 #include <string>
12 #include <map>
13 #include <ctime>    // for clock
14 using namespace std;
15
16 #include <opencv2/core/core.hpp>    // for Mat
17 using namespace cv;
18
19 #include "CvProcessorException.h"
20
21 /**
22  * Class to process a source image with OpenCV 2+
23  */
24 class CvProcessor
25 {
26     public:
27
28         /**
29          * Verbose level for error / warnings / notification messages
30          */
31         typedef enum
32         {
33             VERBOSE_NONE = 0,   //!< no messages are displayed
34             VERBOSE_ERRORS,   //!< only error messages are displayed
35             VERBOSE_WARNINGS,   //!< error & warning messages are displayed
36             VERBOSE_NOTIFICATIONS, //!< error, warning and notifications messages are displayed
37             VERBOSE_ACTIVITY, //!< all previouses + log messages
38             NBVERBOSELEVEL
39         } VerboseLevel;
40
41
42         /**
43          * Index of channels in OpenCV BGR or Gray images
44          */
45         typedef enum
46         {
47             BLUE = 0,//!< Blue component is first in BGR images
48             GRAY = 0,//!< Gray component is first in gray images
49             GREEN,   //!< Green component is second in BGR images
50             RED,     //!< Red component is last in BGR images
51             NBCHANNELS
52         } Channels;
53
54     protected:
55         /**
56          * The source image: CV_8UC<nbChannels>
57          */
58         Mat * sourceImage;
59
60         /**
61          * Source image number of channels (generally 1 or 3)
62          */
63         int nbChannels;
64
65         /**
66          * Source image size (cols, rows)
67          */
68         Size size;
69
70         /**
71          * The source image type (generally CV_8UC<nbChannels>)
72          */
73         int type;
74
75         /**
76          * Map to store aditionnal images pointers by name
77          */
78         map<string, Mat*> images;
79
80         /**
81          * The verbose level for printed messages
82          */
```

```
83          VerboseLevel verboseLevel;
84
85          /**
86           * Process time in ticks (~1e6 ticks/second)
87           * @see clock_t for details on ticks
88           */
89          clock_t processTime;
90
91          /**
92           * Indicates if processing time is absolute or measured in ticks/feature
93           * processed by this processor.
94           * A feature can be any kind of things the processor has to detect or
95           * create while processing an image.
96           */
97          bool timePerFeature;
98
99      public:
100         /**
101          * OpenCV image processor constructor
102          * @param sourceImage the source image
103          * @param verbose level for printed messages
104          * @pre source image is not NULL
105          */
106         CvProcessor(Mat * sourceImage,
107                     const VerboseLevel level = VERBOSE_NONE);
108
109         /**
110          * OpenCV image Processor destructor
111          */
112         virtual ~CvProcessor();
113
114         /**
115          * OpenCV image Processor abstract Update
116          * @note this method should be implemented in sub classes
117          */
118         virtual void update() = 0;
119
120         // ------------------------------------------------------------------
121         // Images accessors
122         // ------------------------------------------------------------------
123         /**
124          * Changes source image
125          * @param sourceImage the new source image
126          * @throw CvProcessorException#NULL_IMAGE when new source image is NULL
127          * @note this method should NOT be directly reimplemented in sub classes
128          * unless it is transformed into a QT slot
129          */
130         virtual void setSourceImage(Mat * sourceImage)
131             throw (CvProcessorException);
132
133         /**
134          * Adds a named image to additionnal images
135          * @param name the name of the image
136          * @param image the image reference
137          * @return true if image has been added to additionnal images map, false
138          * if image key (the name) already exists in the addtitionnal images map.
139          */
140         bool addImage(const char * name, Mat * image);
141
142         /**
143          * Adds a named image to additionnal images
144          * @param name the name of the image
145          * @param image the image reference
146          * @return true if image has been added to additionnal images map, false
147          * if image key (the name) already exists in the addtitionnal images map.
148          */
149         bool addImage(const string & name, Mat * image);
150
151 //      /*
152 //       * Update named image in additionnal images.
153 //       * @param name the name of the image
154 //       * @param image the image reference
155 //       * @post the image located at key name is updated.
156 //       */
157 //      virtual void updateImage(const char * name, const Mat & image);
158 //
159 //      /*
160 //       * Update named image in additionnal images.
161 //       * @param name the name of the image
162 //       * @param image the image reference
163 //       * @post the image located at key name is updated.
164 //       */
```



```
165 //      virtual void updateImage(const string & name, const Mat & image);
166
167         /**
168          * Get image by name
169          * @param name the name of the image we're looking for
170          * @return the image registered by this name in the additionnal images
171          * map
172          * @throw CvProcessorException#INVALID_NAME is used name is not already
173          * registerd in the images
174          */
175         const Mat & getImage(const char * name) const
176             throw (CvProcessorException);
177
178         /**
179          * Get image by name
180          * @param name the name of the image we're looking for
181          * @return the image registered by this name in the additionnal images
182          * map
183          * @throw CvProcessorException#INVALID_NAME is used name is not already
184          * registerd in the images
185          */
186         const Mat & getImage(const string & name) const
187             throw (CvProcessorException);
188
189         /**
190          * Get image pointer by name
191          * @param name the name of the image we're looking for
192          * @return the image pointer registered by this name in the additionnal
193          * images map
194          * @throw CvProcessorException#INVALID_NAME is used name is not already
195          * registerd in the images
196          */
197         Mat * getImagePtr(const char * name)
198             throw (CvProcessorException);
199
200         /**
201          * Get image pointer by name
202          * @param name the name of the image we're looking for
203          * @return the image registered by this name in the additionnal images
204          * map
205          * @throw CvProcessorException#INVALID_NAME is used name is not already
206          * registerd in the images
207          */
208         Mat * getImagePtr(const string & name)
209             throw (CvProcessorException);
210         // ------------------------------------------------------------------
211         // Options settings and gettings
212         // ------------------------------------------------------------------
213         /**
214          * Number of channels in source image
215          * @return the number of channels of source image
216          */
217         int getNbChannels() const;
218
219         /**
220          * Type of the source image
221          * @return the openCV type of the source image
222          */
223         int getType() const;
224
225         /**
226          * Get the current verbose level
227          * @return the current verbose level
228          */
229         VerboseLevel getVerboseLevel() const;
230
231         /**
232          * Set new verbose level
233          * @param level the new verobse level
234          */
235         virtual void setVerboseLevel(const VerboseLevel level);
236
237         /**
238          * Return processor processing time of step index [default implementation
239          * returning only processTime, should be reimplemented in subclasses]
240          * @param index index of the step which processing time is required,
241          * 0 indicates all steps, and values above 0 indicates step #. If
242          * required index is bigger than number of steps than all steps value
243          * should be returned.
244          * @return the processing time of step index.
245          * @note should be reimplemented in subclasses in order to define
246          * time/feature behaviour
```

```
247             */
248         virtual double getProcessTime(const size_t index = 0) const;
249
250         /**
251          * Indicates if processing time is per feature processed in the current
252          * image or absolute
253          * @return
254          */
255         bool isTimePerFeature() const;
256
257         /**
258          * Sets Time per feature processing time unit
259          * @param value the time per feature value (true or false)
260          */
261         virtual void setTimePerFeature(const bool value);
262
263     protected:
264         // --------------------------------------------------------------------
265         // Setup and cleanup attributes
266         // --------------------------------------------------------------------
267         /**
268          * Setup internal attributes according to source image
269          * @param sourceImage a new source image
270          * @param fullSetup full setup is needed when source image is changed
271          * @pre sourceimage is not NULL
272          * @note this method should be reimplemented in sub classes
273          */
274         virtual void setup(Mat * sourceImage, const bool fullSetup = true);
275
276         /**
277          * Clean up internal attributes before changing source image or
278          * cleaning up class before destruction
279          * @note this method should be reimplemented in sub classes
280          */
281         virtual void cleanup();
282 };
283
284 #endif /* CVPROCESSOR_H_ */
```



```
1   /*
2    * CvProcessor.cpp
3    *
4    *  Created on: 21 fÃ©vr. 2012
5    *      Author: davidroussel
6    */
7
8
9   #include "CvProcessor.h"
10
11  /*
12   * OpenCV image processor constructor
13   * @param sourceImage the source image
14   * @pre source image is not NULL
15   */
16  CvProcessor::CvProcessor(Mat *sourceImage, const VerboseLevel level) :
17      sourceImage(sourceImage),
18      nbChannels(sourceImage->channels()),
19      size(sourceImage->size()),
20      type(sourceImage->type()),
21      verboseLevel(level),
22      processTime(0),
23      timePerFeature(false)
24  {
25      // No dynamic links in constructors, so this setup will always be
26      // CvProcessor::setup
27      setup(sourceImage, false);
28  }
29
30  /*
31   * OpenCV image Processor destructor
32   */
33  CvProcessor::~CvProcessor()
34  {
35      // No Dynamic link in destructors ?
36      cleanup();
37
38      map<string, Mat*>::const_iterator cit;
39      for (cit = images.begin(); cit ≠ images.end(); ++cit)
40      {
41          // Release handle to evt deallocate data
42          /*
43           * Since this is a pointer it should be necessary to release data
44           */
45          cit->second->release();
46      }
47      // Calls destructors on all elements
48      images.clear();
49  }
50
51  /*
52   * Setup internal attributes according to source image
53   * @param sourceImage a new source image
54   * @param fullSetup full setup is needed when source image is changed
55   * @pre sourceimage is not NULL
56   * @note this method should be reimplemented in sub classes
57   */
58  void CvProcessor::setup(Mat *sourceImage, const bool fullSetup)
59  {
60      if (verboseLevel ≥ VERBOSE_ACTIVITY)
61      {
62          clog << "CvProcessor::"<< (fullSetup ? "full " : "") <<"setup" << endl;
63      }
64
65      // Full setup starting point (==> previous cleanup)
66      if (fullSetup)
67      {
68          this->sourceImage = sourceImage;
69          nbChannels = sourceImage->channels();
70          size = sourceImage->size();
71          type = sourceImage->type();
72      }
73
74      // Partial setup starting point  (==> in any cases)
75      processTime = (clock_t) 0;
76      addImage("source", this->sourceImage);
77  }
78
79  /*
80   * Clean up internal atrtibutes before changing source image or
81   * cleaning up class before destruction
82   * @note this method should be reimplemented in sub classes
```

```
 */
void CvProcessor::cleanup()
{
    if (verboseLevel ≥ VERBOSE_ACTIVITY)
    {
        clog << "CvProcessor::cleanup()" << endl;
    }

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

/*
 * Changes source image
 * @param sourceImage the new source image
 * @throw CvProcessorException#NULL_IMAGE when new source image is NULL
 */
void CvProcessor::setSourceImage(Mat *sourceImage)
    throw (CvProcessorException)
{
    // clean up current attributes
    cleanup();

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

    // setup attributes again
    setup(sourceImage);
}

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

    return addImage(sname, image);
}

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

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

    if (ret.second ≡ false)
    {
        if (verboseLevel ≥ VERBOSE_WARNINGS)
```



```
        {
            cerr << "CvProcessor::addImage(\"" << name
                 << "\",...) : already added" << endl;
        }

        retValue = false;
    }
    else
    {
        retValue = true;
    }

    return retValue;
}
/*
 * 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.
 */
//void CvProcessor::updateImage(const char * name, Mat * image)
//{
//  // Search for this name in the map
//  map<string, Mat*>::iterator it;
//  for (it = images.begin(); it != images.end(); ++it)
//  {
//      if (it->first == name)
//      {
//          (it->second->release();
//          images.erase(it);
//      }
//  }
//
//  string sname(name);
//
//  updateImage(sname, 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.
 */
//void CvProcessor::updateImage(const string & name, const Mat & image)
//{
//  clog << "update image " << name << " with " << (long) &image << endl;
//  images.erase(name);
//
//  addImage(name, image);
//}

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

    return getImage(sname);
}

/*
 * Get image pointer by name
 * @param name the name of the image we're looking for
 * @return the image pointer registered by this name in the additionnal
 * images map
 * @throw CvProcessorException#INVALID_NAME is used name is not already
 * registerd in the images
 */
const Mat & CvProcessor::getImage(const string & name) const
    throw (CvProcessorException)
{
    // Search for this name
    map<string, Mat*>::const_iterator cit;
    for (cit = images.begin(); cit ≠ images.end(); ++cit)
```

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



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

411


```
1  #ifndef CVPROCESSOREXCEPTION_H_
2  #define CVPROCESSOREXCEPTION_H_
3
4  #include <iostream>     // for ostream
5  #include <string>       // for string
6  #include <exception>    // for std::exception base class
7  using namespace std;
8
9  /**
10  * Exception class for CvProcessor.
11  * Contains mainly exception reasons why an CvProcessor operation could not be
12  * performed.
13  */
14 class CvProcessorException : public exception
15 {
16     public:
17         /**
18          * Matrices operation exception cases
19          */
20         typedef enum
21         {
22             /**
23              * Null image.
24              * Used when trying to add null image as source image of the
25              * processor
26              */
27             NULL_IMAGE,
28             /**
29              * Null image data.
30              * Used when trying to use image with NULL data
31              */
32             NULL_DATA,
33             /**
34              * Invalid name in image acces by name.
35              * Used when searching for images by name which is not contained
36              * in the already registered names
37              */
38             INVALID_NAME,
39             /**
40              * Invalid image type.
41              * Some Processors needs specific images types
42              */
43             INVALID_IMAGE_TYPE,
44             /**
45              * Illegal data access (i.e. read/write access on read only data)
46              */
47             ILLEGAL_ACCESS,
48             /**
49              * Allocation failure on dynamically allocated elements
50              */
51             ALLOC_FAILURE,
52             /**
53              * Unable to read a file
54              */
55             FILE_READ_FAIL,
56             /**
57              * File parse error
58              */
59             FILE_PARSE_FAIL,
60             /**
61              * Unable to write file
62              */
63             FILE_WRITE_FAIL,
64             /**
65              * OpenCV exception
66              */
67             OPENCV_EXCEPTION
68         } ExceptionCause;
69
70         /**
71          * CvProcessor exception constructor
72          * @param e the chosen error case for this error
73          * @see ExceptionCause
74          */
75         CvProcessorException(const CvProcessorException::ExceptionCause e);
76
77         /**
78          * CvProcessor exception constructor with exception message descriptor
79          * @param e the chosen error case for this error
80          * @param descr character string describing the message
81          * @see ExceptionCause
82          */
```

```
83          CvProcessorException(const CvProcessorException::ExceptionCause e,
84                               const char * descr);
85
86          /**
87           * CvProcessor exception from regular (typically OpenCV) exception
88           * @param e the exception to relay
89           */
90          CvProcessorException(const exception & e, const char * descr = "");
91
92          /**
93           * CvProcessor exception destructor
94           * @post message cleared
95           */
96          virtual ~CvProcessorException() throw ();
97
98          /**
99           * Explanation message of the exception
100          * @return a C-style character string describing the general cause
101          * of the current error.
102          */
103         virtual const char* what() const throw();
104
105         /**
106          * CvProcessorException cause
107          * @return the cause enum of the exception
108          */
109         CvProcessorException::ExceptionCause getCause();
110
111         /**
112          * Source message of the exception
113          * @return the message string of the exception
114          */
115         string getMessage();
116
117         /*
118          * Note output operators are not necessary since what() method is used
119          * to explain the reason of the exception.
120          * Example :
121          * try
122          * {
123          *  ... do something which throws an std::exception
124          * }
125          * catch (exception & e)
126          * {
127          *  cerr << e.what() << endl;
128          * }
129          */
130
131     protected:
132         /**
133          * The current error case
134          */
135         CvProcessorException::ExceptionCause cause;
136
137         /**
138          * description message of the exception
139          */
140         string message;
141 };
142
143 #endif /*CVPROCESSOREXCEPTION_H_*/
```



```
1   #include "CvProcessorException.h"
2   #include <iostream>     // for cerr et endl;
3   #include <string>       // for string
4   #include <sstream>      // for ostringstream
5   using namespace std;
6
7   /*
8    * CvProcessor exception constructor
9    * @param e the chosen error case for this error
10   * @see ExceptionCause
11   */
12  CvProcessorException::CvProcessorException(
13      const CvProcessorException::ExceptionCause e) :
14      exception(),
15      cause(e),
16      message("")
17  {
18  }
19
20  /*
21   * CvProcessor exception constructor with message descriptor
22   * @param e the chosen error case for this error
23   * @param descr character string describing the message
24   * @see ExceptionCause
25   */
26  CvProcessorException::CvProcessorException(
27      const CvProcessorException::ExceptionCause e, const char * descr) :
28      exception(),
29      cause(e),
30      message(descr)
31  {
32  }
33
34  /*
35   * CvProcessor exception from regular (typically OpenCV) exception
36   * @param e the exception to relay
37   */
38  CvProcessorException::CvProcessorException(const exception & e, const char * descr) :
39      exception(e),
40      cause(OPENCV_EXCEPTION),
41      message(descr)
42  {
43  }
44
45
46  /*
47   * CvProcessor exception destructor
48   * @post message cleared
49   */
50  CvProcessorException::~CvProcessorException() throw ()
51  {
52      message.clear();
53  }
54
55  /*
56   * Explanation message of the exception
57   * @return a C-style character string describing the general cause
58   * of the current error.
59   */
60  const char * CvProcessorException::what() const throw()
61  {
62      const char * initialWhat = exception::what();
63
64      ostringstream output;
65
66      output << initialWhat << ":";
67
68      output << "CvProcessorException :";
69
70      if (message.length() > 0)
71      {
72          output << message << ":";
73      }
74
75      switch (cause) {
76          case CvProcessorException::NULL_IMAGE:
77              output << "NULL image" << endl ;
78              break;
79          case CvProcessorException::NULL_DATA:
80              output << "NULL image data" << endl ;
81              break;
82          case CvProcessorException::INVALID_NAME:
```

```
            output << "Invalid name" << endl ;
            break;
        case CvProcessorException::INVALID_IMAGE_TYPE:
            output << "Invalid image type" << endl;
            break;
        case CvProcessorException::ILLEGAL_ACCESS:
            output << "Illegal access" << endl;
            break;
        case CvProcessorException::ALLOC_FAILURE:
            output << "New element allocation failure" << endl;
            break;
        case CvProcessorException::FILE_READ_FAIL:
            output << "Unable to read file" << endl;
            break;
        case CvProcessorException::FILE_PARSE_FAIL:
            output << "File parse error" << endl;
            break;
        case CvProcessorException::FILE_WRITE_FAIL:
            output << "Unable to write file" << endl;
            break;
        default:
            output << "Unknown exception" << endl;
            break;
    }

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


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

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



```
/*
 * CvColorSpaces.h
 *
 *  Created on: 25 fÃ©vr. 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;

        /**
         * 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;

        /**
         * Cb colored image
         */
        Mat cbColoredImage;

        /**
         * Palette to build colored red component image
         */
        Palette redMap;

        /**
         * Palette to build colored green component image
         */
        Palette greenMap;

        /**
         * Palette to build colored blue component image
         */
        Palette blueMap;

        /**
         * Pointers to RGB palettes
         * pointing respectively to
         *  - blueMap
         *  - greenMap
         *  - redMap
         */
        Palette * bgrMap[3];

        /**
         * Palette for hue component
         */
        Palette hMap;

        /**
         * Palette for Cb component
         */
        Palette cbMap;

        /**
         * Palette for Cr component
         */
        Palette crMap;

        /**
         * Booleans to choose to display channels as grayscale
         * or colored images
         */
        bool showColorChannel[NbShows];

        /**
         * Selected image type to display
         */
        Display imageDisplayIndex;

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

        /**
         * Number of frames used to compute mean processTime
         * @see CvProcessor::processTime
         */
        size_t nbFrames;

```

```
247     public :
248         /**
249          * Color spaces constructor
250          * @param inFrame input image
251          */
252         CvColorSpaces(Mat * inFrame);
253 
254         /**
255          * Color spaces destructor
256          */
257         virtual ~CvColorSpaces();
258 
259         /**
260          * Update compute selected image for display according to
261          * selected parameters such as imageDisplayIndex, showColorChannel,
262          * and eventually hueDisplay
263          */
264         virtual void update();
265 
266         /**
267          * Get currently selected image index
268          * @return the currently selected image for display index
269          */
270         Display getDisplayImageIndex();
271 
272         /**
273          * Select image to set in displayImage :
274          *  - INPUT selects input image for display
275          *  - GRAY selects gray converted input image for display
276          *  - RED selects BGR red component image for display
277          *  - GREEN selects BGR green component image for display
278          *  - BLUE selects BGR blue component image for display
279          *  - HUE selects HSV hue component image for display
280          *  - SATURATION selects HSV saturation component image for display
281          *  - VALUE selects HSV value component image for display
282          *  - Y selects YCrCb Y component image for display
283          *  - Cr selects YCrCb Cr component image for display
284          *  - Cb selects YCrCb Cb component image for display
285          * @param index select the index to select display image
286          */
287         virtual void setDisplayImageIndex(const Display index);
288 
289         /**
290          * Get the color display status for specific channels (such as red,
291          * green, blue, hue ...)
292          * @param c the channel to get color display status:
293          *  - BINDEX color display status for blue component
294          *  - GINDEX color display status for green component
295          *  - RINDEX color display status for red component
296          *  - HINDEX color display status for hue component
297          *  - CbNDEX color display status for Cb component
298          *  - CrNDEX color display status for Cr component
299          * @return the color display status of selected component
300          */
301         bool getColorChannel(const ShowColor c);
302 
303         /**
304          * Sets the color display status of selected component
305          * @param c the selected component:
306          *  - BINDEX color display status for blue component
307          *  - GINDEX color display status for green component
308          *  - RINDEX color display status for red component
309          *  - HINDEX color display status for hue component
310          *  - CbNDEX color display status for Cb component
311          *  - CrNDEX color display status for Cr component
312          * @param value the value to set on the selected component
313          */
314         virtual void setColorChannel(const ShowColor c, const bool value);
315 
316         /**
317          * Get currently selected hue display mode
318          * @return the currenlty selected hue display mode
319          */
320         HueDisplay getHueDisplaymode();
321 
322         /**
323          * Select hue display mode :
324          *  - HUECOLOR Normal Hue mode
325          *  - HUESATURATE Hue*Saturatin mode
326          *  - HUEVALUE Hue*Value mode
327          *  - HUEGRAY Gray mode
328          * @param mode the mode so select
```



```
329          */
330         virtual void setHueDisplayMode(const HueDisplay mode);
331 
332         /**
333          * Gets the image selected for display
334          * @return the display image
335          */
336         Mat & getDisplayImage();
337 
338         /**
339          * Return processor MEAN processing time of step index [default
340          * implementation returning only processTime, should be reimplemented
341          * in subclasses]
342          * @param index not used here
343          * @return the MEAN processing time between two frames.
344          */
345         double getProcessTime(const size_t index = 0) const;
346 
347 
348     protected:
349         // ----------------------------------------------------------------------
350         // Setup and cleanup attributes
351         // ----------------------------------------------------------------------
352         /**
353          * Setup internal attributes according to source image
354          * @param sourceImage a new source image
355          * @param fullSetup full setup is needed when source image is changed
356          * @pre sourceimage is not NULL
357          * @note this method should be reimplemented in sub classes
358          */
359         virtual void setup(Mat * sourceImage, bool fullSetup = true);
360 
361         /**
362          * Clean up internal attributes before changing source image or
363          * cleaning up class before destruction
364          * @note this method should be reimplemented in sub classes
365          */
366         virtual void cleanup();
367 
368         /**
369          * Show Min and Max values and locations for a matrix
370          * @param m the matrix to consider
371          */
372         static void showMinMaxLoc(const Mat & m);
373 
374         /**
375          * Compute Maximum of color images by comparing pixel norm
376          * rather than a per channel max like the openCV max function
377          * @param src1 the first color (or gray) image
378          * @param src2 the second color (or gray) image
379          * @param dst the color (or gray) destination
380          * @pre the norm max is only computed if arguments are of type CV_8UC3,
381          * otherwise ordinary max is performed
382          */
383         static void normMax(const Mat& src1, const Mat& src2, Mat& dst);
384 };
385 
386 #endif /* CVCOLORSPACES_H_ */
```

```
/*
 * CvColorSpaces.cpp
 *
 *  Created on: 8 fÃ©vr. 2012
 *      Author: davidroussel
 */
#include <cassert>  // for assert
#include <iostream> // for cerr
using namespace std;

#include <opencv2/imgproc/imgproc.hpp> // for cvtColor

#include "mapRed.h"
#include "mapGreen.h"
#include "mapBlue.h"
#include "mapHSV.h"
#include "mapCb.h"
#include "mapCr.h"

#include "CvColorSpaces.h"

/*
 * Color spaces constructor
 * @param sourceImage input image
 */
CvColorSpaces::CvColorSpaces(Mat * sourceImage) :
    CvProcessor(sourceImage),
    inFrameGray(sourceImage→size(), CV_8UC1),
    maxBGChannels(sourceImage→size(), CV_8UC1),
    maxBGRChannels(sourceImage→size(), CV_8UC1),
    maxBGChannelsColor(sourceImage→size(), CV_8UC3),
    maxBGRChannelsColor(sourceImage→size(), CV_8UC3),
    inFrameXYZ(sourceImage→size(), CV_64FC3),
    inFrameHSV(sourceImage→size(), CV_8UC3),
    hueColorImage(sourceImage→size(), CV_8UC3),
    hueMixImage(sourceImage→size(), CV_8UC3),
    hueMixedColorImage(sourceImage→size(), CV_8UC3),
    hueDisplay(HUECOLOR),
    inFrameYCrCb(sourceImage→size(), CV_8UC3),
    crColoredImage(sourceImage→size(), CV_8UC3),
    cbColoredImage(sourceImage→size(), CV_8UC3),
    redMap(mapRed),
    greenMap(mapGreen),
    blueMap(mapBlue),
    hMap(mapHSV),
    cbMap(mapCb),
    crMap(mapCr),
    imageDisplayIndex(INPUT),
    displayImageChanged(false),
    nbFrames(0)
{
    setup(sourceImage, false);

    addImage("display", &displayImage);
}

/*
 * Color spaces destructor
 */
CvColorSpaces::~CvColorSpaces()
{
    cleanup();
}

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

    assert(sourceImage ≠ NULL);

    CvProcessor::setup(sourceImage, fullSetup);

    // Full setup starting point
    if (fullSetup) // only when sourceImage changes
    {
```



```
        inFrameGray.create(sourceImage→size(), CV_8UC1);
        maxBGChannels.create(sourceImage→size(), CV_8UC1);
        maxBGRChannels.create(sourceImage→size(), CV_8UC1);
        maxBGChannelsColor.create(sourceImage→size(), CV_8UC3);
        maxBGRChannelsColor.create(sourceImage→size(), CV_8UC3);
        inFrameXYZ.create(sourceImage→size(), CV_64FC3),
        inFrameHSV.create(sourceImage→size(), CV_8UC3);
        hueColorImage.create(sourceImage→size(), CV_8UC3);
        hueMixImage.create(sourceImage→size(), CV_8UC3);
        hueMixedColorImage.create(sourceImage→size(), CV_8UC3);
        inFrameYCrCb.create(sourceImage→size(), CV_8UC3);
        crColoredImage.create(sourceImage→size(), CV_8UC3);
        cbColoredImage.create(sourceImage→size(), CV_8UC3);
        processTime = 0;
        nbFrames = 0;
    }
    else // only at construction
    {
        bgrMap[0] = &blueMap;
        bgrMap[1] = &greenMap;
        bgrMap[2] = &redMap;

        for (size_t i = 0; i < (size_t) NbShows; i++)
        {
            showColorChannel[i] = true;
        }
    }

    // Partial setup starting point (in both cases)
    for (int i=0; i < 3; i++)
    {
        bgrChannels.push_back(Mat(sourceImage→size(), CV_8UC1));
        bgrColoredChannels[i].create(sourceImage→size(), CV_8UC3);
        xyzGrayChannels[i].create(sourceImage→size(), CV_64FC1);
        xyzDisplayChannels[i].create(sourceImage→size(), CV_8UC1);
        hsvChannels.push_back(Mat(sourceImage→size(), CV_8UC1));
        hueMixChannels[i].create(sourceImage→size(), CV_8UC1);
        yCrCbChannels.push_back(Mat(sourceImage→size(), CV_8UC1));
    }
}

/*
 * Clean up images before changing source image or terminating
 * CvColorSpaces
 */
void CvColorSpaces::cleanup()
{
//  clog << "CvColorSpaces::cleanup()" << endl;

    cbColoredImage.release();
    crColoredImage.release();
    for (size_t i = 0; i < yCrCbChannels.size(); i++)
    {
        yCrCbChannels[i].release();
    }
    yCrCbChannels.clear();
    inFrameYCrCb.release();

    hueMixedColorImage.release();
    hueMixImage.release();
    for (size_t i = 0; i < 3; i++)
    {
        hueMixChannels[i].release();
    }
    hueColorImage.release();
    for (size_t i = 0; i < hsvChannels.size(); i++)
    {
        hsvChannels[i].release();
    }
    hsvChannels.clear();
    inFrameHSV.release();

    for (size_t i = 0; i < bgrChannels.size(); i++)
    {
        bgrChannels[i].release();
        bgrColoredChannels[i].release();
        xyzGrayChannels[i].release();
        xyzDisplayChannels[i].release();
    }
    bgrChannels.clear();

    inFrameXYZ.release();
```

```
165
166     maxBGRChannelsColor.release();
167
168     maxBGChannelsColor.release();
169
170     maxBGRChannels.release();
171
172     maxBGChannels.release();
173
174     inFrameGray.release();
175
176     displayImage.release();
177
178     CvProcessor::cleanup();
179 }
180
181
182 /*
183  * Update compute selected image for display according to
184  * selected parameters such as imageDisplayIndex, showColorChannel,
185  * and eventually hueDisplay
186  * @return true if display image has changed, false otherwise
187  */
188 void CvColorSpaces::update()
189 {
190     clock_t start, end;
191     start = clock();
192     // ----------------------------------------------------------------------
193     // Compute needed images
194     // ----------------------------------------------------------------------
195     switch (imageDisplayIndex)
196     {
197         case INPUT:
198             // Ain't got nothin to do here : input image doesn't need to be processed
199             break;
200
201         // ----------------------------------------------------------------------
202         // Gray level conversion
203         // ----------------------------------------------------------------------
204         case GRAY:
205             // Converts to gray
206             // sourceImage -> inFramegray
207             // TODO Ã  complÃ©ter ...
208             break;
209
210         // ----------------------------------------------------------------------
211         // RGB Decomposition
212         // ----------------------------------------------------------------------
213         case RED:
214         case GREEN:
215         case BLUE:
216         case MAX_BGR:
217             // Split BGR channels : sourceImage -> bgrChannels
218             // TODO Ã  complÃ©ter ...
219
220             // Build colored image from channels : red channel leads to a
221             // red colored image, and so on ...
222             // by applying bgrMap[x] on bgrChannels[x] to produce
223             // bgrColoredChannels[x]
224             // bgrChannels[i] -> bgrColoredChannels[i]
225             for (size_t i = 0; i < bgrChannels.size(); i++)
226             {
227                 // TODO Ã  complÃ©ter ...
228             }
229
230             if (¬showColorChannel[MAXINDEX])
231             {
232                 // Compute maximum of BGR channels
233                 // bgrChannels[0 & 1] -> maxBGChannels
234                 // bgrChannels[2] & maxBGChannels -> maxBGRChannels
235                 // TODO Ã  complÃ©ter ...
236             }
237             else
238             {
239                 // Compute colored maximum of BGR channels
240                 // bgrColoredChannels[0 & 1] -> maxBGChannelsColor
241                 // bgrColoredChannels[2] & maxBGChannelsColor -> maxBGRChannelsColor
242                 // TODO Ã  complÃ©ter ...
243             }
244
245             /*
246              * TODO What are the characteristics of blue component vs
```



```
247              * green or red ? Answer below:
248              *
249              */
250
251             break;
252
253         // ----------------------------------------------------------------------
254         // XYZ conversion
255         // ----------------------------------------------------------------------
256         case XYZ_X:
257         case XYZ_Y:
258         case XYZ_Z:
259             // Converts to XYZ : sourceImage -> inFrameXYZ
260             // TODO Ã  complÃ©ter ...
261
262             // Splits inFrameXYZ to channels xyzGrayChannels
263             // TODO Ã  complÃ©ter ...
264
265             // Converts floating point channels to display channels
266             // xyzGrayChannels[...] -> xyzDisplayChannels[...]
267             for( size_t i=0; i < 3; i++)
268             {
269                 // TODO Ã  complÃ©ter ...
270             }
271
272             /*
273              * TODO What component X, Y or Z looks more like luminance to you ?
274              * Answer below:
275              *
276              */
277
278             break;
279         // ----------------------------------------------------------------------
280         // HSV conversion
281         // ----------------------------------------------------------------------
282         case HUE:
283         case SATURATION:
284         case VALUE:
285             // Converts to HSV : sourceImage -> inFrameHSV
286             // TODO Ã  complÃ©ter ...
287
288             // Split HSV channels : inFrameHSV -> hsvChannels
289             // TODO Ã  complÃ©ter ...
290
291             // evt show min/max of H component : should be [0...179]Â°
292             // showMinMaxLoc(hsvChannels[0]);
293
294             // Normalize hue from 0 to 255 because hsv colormap (hMap)
295             // applied below expects value within 0 to 255 range
296             // hsvChannels[0] -> hsvChannels[0]
297             // TODO Ã  complÃ©ter ...
298
299             // Build colored Hue image : hsvChannels[0] -> hueColorImage
300             // TODO Ã  complÃ©ter ...
301
302             // Build Mixed Hue Color and (Saturation or Value) image
303             if ((hueDisplay > HUECOLOR) ∧ (hueDisplay < HUEGRAY))
304             {
305                 // Creates a 3 channel image from saturation or value channel
306                 // depending on huDisplay value
307                 // hsvChannels -> hueMixChannels
308                 // TODO Ã  complÃ©ter ...
309
310                 // merge mix channels into color image
311                 // hueMixChannels --> hueMixImage
312                 // TODO Ã  complÃ©ter ...
313
314                 // Build colored Hue image \times Saturation or Value
315                 // hueColorImage x hueMixImage -> hueMixedColorImage
316                 // TODO Ã  complÃ©ter ...
317             }
318             break;
319
320         // ----------------------------------------------------------------------
321         // YCbCr conversion
322         // ----------------------------------------------------------------------
323         case Y:
324         case Cr:
325         case Cb:
326             // Converts to YCrCb : sourceImage -> inFrameYCrCb
327             // TODO Ã  complÃ©ter ...
328
```

```
329             // Split YCrCb channels : inFrameYCrCb -> yCrCbChannels
330             // TODO Ã  complÃ©ter ...
331
332             // Apply palette on cr & cb components
333             // crmap, yCrCbChannels[1] -> crColoredImage
334             // TODO Ã  complÃ©ter ...
335             // cbmap, yCrCbChannels[2] -> cbColoredImage
336             // TODO Ã  complÃ©ter ...
337             break;
338         default:
339             cerr << "unknown image display index" << imageDisplayIndex << endl;
340             break;
341         /*
342          * TODO How does the Y component compares to the gray component ?
343          * Answer below :
344          */
345
346         /*
347          * TODO What can you tell about the details in Cr or Cb components vs
348          * the details in the Y component ?
349          * Answer below :
350          */
351     }
352
353     // -----------------------------------------------------------------------
354     // select image to display ...
355     // -----------------------------------------------------------------------
356
357     uchar * previousImageData = displayImage.data;
358
359     switch (imageDisplayIndex)
360     {
361         case INPUT:
362             displayImage = *sourceImage;
363             break;
364         case GRAY:
365             displayImage = inFrameGray;
366             break;
367         case RED:
368             if (showColorChannel[RINDEX])
369             {
370                 displayImage = bgrColoredChannels[RINDEX];
371             }
372             else
373             {
374                 displayImage = bgrChannels[RINDEX];
375             }
376             break;
377         case GREEN:
378             if (showColorChannel[GINDEX])
379             {
380                 displayImage = bgrColoredChannels[GINDEX];
381             }
382             else
383             {
384                 displayImage = bgrChannels[GINDEX];
385             }
386             break;
387         case BLUE:
388             if (showColorChannel[BINDEX])
389             {
390                 displayImage = bgrColoredChannels[BINDEX];
391             }
392             else
393             {
394                 displayImage = bgrChannels[BINDEX];
395             }
396             break;
397         case MAX_BGR:
398             if (showColorChannel[MAXINDEX])
399             {
400                 displayImage = maxBGRChannelsColor;
401             }
402             else
403             {
404                 displayImage = maxBGRChannels;
405             }
406             break;
407         case XYZ_X:
408             displayImage = xyzDisplayChannels[0];
409             break;
410         case XYZ_Y:
```



```
411             displayImage = xyzDisplayChannels[1];
412             break;
413         case XYZ_Z:
414             displayImage = xyzDisplayChannels[2];
415             break;
416         case HUE:
417             switch (hueDisplay)
418             {
419                 case HUECOLOR:
420                     displayImage = hueColorImage;
421                     break;
422                 case HUESATURATE:
423                 case HUEVALUE:
424                     displayImage = hueMixedColorImage;
425                     break;
426                 case HUEGRAY:
427                     displayImage = hsvChannels[0];
428                     break;
429                 case NBHUES:
430                 default:
431                     cerr << "unknown Hue display mode "<< hueDisplay
432                          << endl;
433                     break;
434             }
435             break;
436         case SATURATION:
437             displayImage = hsvChannels[1];
438             break;
439         case VALUE:
440             displayImage = hsvChannels[2];
441             break;
442         case Y:
443             displayImage = yCrCbChannels[0];
444             break;
445         case Cr:
446             if (showColorChannel[CrINDEX])
447             {
448                 displayImage = crColoredImage;
449             }
450             else
451             {
452                 displayImage = yCrCbChannels[1];
453             }
454             break;
455         case Cb:
456             if (showColorChannel[CbINDEX])
457             {
458                 displayImage = cbColoredImage;
459             }
460             else
461             {
462                 displayImage = yCrCbChannels[2];
463             }
464             break;
465         default:
466             cerr << "unknown display image index " << imageDisplayIndex << endl;
467             displayImage = *sourceImage;
468             break;
469     }
470
471     if (previousImageData ≠ displayImage.data)
472     {
473         displayImageChanged = true;
474     }
475     else
476     {
477         displayImageChanged = false;
478     }
479     end = clock();
480     processTime += (end - start);
481     nbFrames++;
482 }
483
484 /*
485  * Gets the image selected for display
486  * @return the display image
487  */
488 Mat & CvColorSpaces::getDisplayImage()
489 {
490     return displayImage;
491 }
492
```

```
493 /*
494  * Get currently selected image index
495  * @return the currently selected image for display index
496  */
497 CvColorSpaces::Display CvColorSpaces::getDisplayImageIndex()
498 {
499     return imageDisplayIndex;
500 }
501
502 /*
503  * Select image to set in displayImage :
504  *  - INPUT selects input image for display
505  *  - GRAY selects gray converted input image for display
506  *  - RED selects BGR red component image for display
507  *  - GREEN selects BGR green component image for display
508  *  - BLUE selects BGR blue component image for display
509  *  - HUE selects HSV hue component image for display
510  *  - SATURATION selects HSV saturation component image for display
511  *  - VALUE selects HSV value component image for display
512  *  - Y selects YCrCb Y component image for display
513  *  - Cr selects YCrCb Cr component image for display
514  *  - Cb selects YCrCb Cb component image for display
515  * @param select the index to select display image
516  */
517 void CvColorSpaces::setDisplayImageIndex(const Display index)
518 {
519     if (index < NbSelected)
520     {
521         imageDisplayIndex = index;
522         processTime = 0;
523         nbFrames = 0;
524     }
525     else
526     {
527         cerr << "CvColorSpaces::setDisplayImageIndex : index " << index
528              << " out of bounds" << endl;
529     }
530 }
531
532 /*
533  * Get the color display status for specific channels (such as red,
534  * green, blue, hue ...)
535  * @param c the channel to get color display status:
536  *  - BINDEX color display status for blue component
537  *  - GINDEX color display status for green component
538  *  - RINDEX color display status for red component
539  *  - MAXINDEX color display for max of RGB
540  *  - HINDEX color display status for hue component
541  *  - CbNDEX color display status for Cb component
542  *  - CrNDEX color display status for Cr component
543  * @return the color display status of selected component
544  */
545 bool CvColorSpaces::getColorChannel(const ShowColor c)
546 {
547     return showColorChannel[c];
548 }
549
550 /*
551  * Sets the color display status of selected component
552  * @param c the selected component:
553  *  - BINDEX color display status for blue component
554  *  - GINDEX color display status for green component
555  *  - RINDEX color display status for red component
556  *  - MAXINDEX color display for max of RGB
557  *  - HINDEX color display status for hue component
558  *  - CbNDEX color display status for Cb component
559  *  - CrNDEX color display status for Cr component
560  * @param value the value to set on the selected component
561  */
562 void CvColorSpaces::setColorChannel(const ShowColor c, const bool value)
563 {
564     if ( c < NbShows)
565     {
566         showColorChannel[c] = value;
567         processTime = 0;
568         nbFrames = 0;
569     }
570     else
571     {
572         cerr << "CvColorSpaces::setColorChannel : index " << c
573              << " out of bounds" << endl;
574     }
```



```
575 }
576
577 /*
578  * Get currently selected hue display mode
579  * @return the currenlty selected hue display mode
580  */
581 CvColorSpaces::HueDisplay CvColorSpaces::getHueDisplaymode()
582 {
583     return hueDisplay;
584 }
585
586 /*
587  * Select hue display mode :
588  *  - HUECOLOR Normal Hue mode
589  *  - HUESATURATE Hue*Saturatin mode
590  *  - HUEVALUE Hue*Value mode
591  *  - HUEGRAY Gray mode
592  * @param mode the mode so select
593  */
594 void CvColorSpaces::setHueDisplayMode(const HueDisplay mode)
595 {
596     if (mode < NBHUES)
597     {
598         hueDisplay = mode;
599         processTime = 0;
600         nbFrames = 0;
601     }
602     else
603     {
604         cerr << "CvColorSpaces::setHueDisplayMode : index " << mode
605              << " out of bounds" << endl;
606     }
607 }
608
609 /*
610  * Return processor MEAN processing time of step index [default
611  * implementation returning only processTime, should be reimplemented
612  * in subclasses]
613  * @param index not used here
614  * @return the MEAN processing time between two frames.
615  */
616 double CvColorSpaces::getProcessTime(const size_t) const
617 {
618     return (double) processTime / (double) MAX(1, nbFrames);
619 }
620
621
622 /*
623  * Show Min and Max values and locations for a matrix
624  * @param m the matrix to consider
625  */
626 void CvColorSpaces::showMinMaxLoc(const Mat & m)
627 {
628     // search for min & max value locations
629     double minVal, maxVal;
630     Point minLoc;
631     Point maxLoc;
632     minMaxLoc(m,&minVal, &maxVal, &minLoc, &maxLoc);
633     clog << "sat values : min = " << minVal << " at (" << minLoc.x
634          << "," << minLoc.y << ") max = " << maxVal << " at ("
635          << maxLoc.x << "," << maxLoc.y << ")" << endl;
636 }
637
638 /*
639  * Compute Maximum of color images by computing a channel wide norm
640  * to find which is the greatest rather than mixing channels
641  * @param src1 the first color (or gray) image
642  * @param src2 the second color (or gray) image
643  * @param dst the color (or gray) destination
644  */
645 void CvColorSpaces::normMax(const Mat& src1, const Mat& src2, Mat& dst)
646 {
647     // first check if src1, src2 && dst have the same sizes and type
648     if ( (src1.rows ≡ src2.rows) ∧
649          (src1.rows ≡ dst.rows) ∧
650          (src1.cols ≡ src2.cols) ∧
651          (src1.cols ≡ dst.cols) ∧
652          (src1.type() ≡ src2.type()) ∧
653          (src1.type() ≡ dst.type()) )
654     {
655         if (src1.type() ≡ CV_8UC3)
656         {
```

```
            // Compute max by pixel norm rather than mixing pixels
            Mat_<Vec3b>::const_iterator src1It = src1.begin<Vec3b>();
            Mat_<Vec3b>::const_iterator src1ItEnd = src1.end<Vec3b>();
            Mat_<Vec3b>::const_iterator src2It = src2.begin<Vec3b>();
            Mat_<Vec3b>::const_iterator src2ItEnd = src2.end<Vec3b>();
            Mat_<Vec3b>::iterator dstIt = dst.begin<Vec3b>();
            Mat_<Vec3b>::iterator dstItEnd = dst.end<Vec3b>();
            for (; src1It ≠ src1ItEnd ∧
                   src2It ≠ src2ItEnd ∧
                   dstIt ≠ dstItEnd ;
                   ++src1It, ++src2It, ++dstIt)
            {
                // compute pixel norm by using self dot product : aVector.ddot
                // TODO Ã  complÃ©ter ...
            }
        }
        else
        {
            // compute max the regular way with max function
            max(src1, src2, dst);
        }
    }
    else
    {
        // Do nothing
        cerr << "CvColorSpaces::normMax : incompatible images" << endl;
    }
}
```



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

#ifndef QCVPROCESSOR_H_
#define QCVPROCESSOR_H_

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

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

    protected:

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

        /**
         * Number format used to format numbers into QStrings
         */
        static char numberFormat[10];

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

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

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

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

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

    public:

        /**
         * QcvProcessor constructor
         * @param image the source image
         * @param imageLock the mutex for concurrent access to the source image.
         * In order to avoid concurrent access to the same image
         * @param updateThread the thread in which this processor should run
         * @param parent parent QObject
         */
        QcvProcessor(Mat * image,
                     QMutex * imageLock = NULL,
                     QThread * updateThread = NULL,
                     QObject * parent = NULL);

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

```
83
84          /**
85           * Sets new number format
86           * @param format the new number format
87           * @pre format string should look like "%8.1f" or at least not be longer
88           * than 10 chars since format is a 10 chars array.
89           * @post id format string is valid and shorter than 10 chars
90           * it has been applied as the new format string.
91           */
92          static void setNumberFormat(const char * format);
93
94      public slots:
95          /**
96           * Update computed images slot and sends updated signal
97           */
98          virtual void update();
99
100         /**
101          * Changes source image slot.
102          * Attributes needs to be cleaned up then set up again
103          * @param image the new source Image
104          * @throw CvProcessorException#NULL_IMAGE when new source image is NULL
105          * @post Various signals are emitted:
106          *  - imageChanged(sourceImage)
107          *  - imageCchanged()
108          *  - if image size changed then imageSizeChanged() is emitted
109          *  - if image color space changed then imageColorsChanged() is emitted
110          */
111         virtual void setSourceImage(Mat * image) throw (CvProcessorException);
112
113         /**
114          * Sets Time per feature processing time unit slot.
115          * @param value the time per feature value (true or false)
116          */
117         virtual void setTimePerFeature(const bool value);
118
119     signals:
120         /**
121          * Signal emitted when update is complete
122          */
123         void updated();
124
125         /**
126          * Signal emitted when processor has finished.
127          * Used to tell helper threads to quit
128          */
129         void finished();
130
131         /**
132          * Signal emitted when source image is reallocated
133          */
134         void imageChanged();
135
136         /**
137          * Signal emitted when source image is reallocated
138          * @param image the new source image pointer or none if just
139          * image changed notification is required
140          */
141         void imageChanged(Mat * image);
142
143         /**
144          * Signal emitted when source image colors changes from color to gray
145          * or from gray to color
146          */
147         void imageColorsChanged();
148
149         /**
150          * Signal emitted when source image size changes
151          */
152         void imageSizeChanged();
153
154         /**
155          * Signal emited when processing time has channged
156          * @param value the new value of the processing time
157          */
158         void processTimeUpdated(const QString & formattedValue);
159
160         /**
161          * Signal to set text somewhere
162          * @param message the message
163          */
164         void sendText(const QString & message);
```



```
165
166         /**
167          * Signal to send update message when something changes
168          * @param message the message
169          * @param timeout number of ms the message should be displayed
170          */
171         void sendMessage(const QString & message, int timeout = defaultTimeOut);
172
173 };
174
175 #endif /* QCVPROCESSOR_H_ */
```

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

#include <QRegExpValidator>
#include <QDebug>
#include <cstring>      // for strcpy
#include "QcvProcessor.h"

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

/*
 * Number format used to format numbers into QStrings
 */
char QcvProcessor::numberFormat[10] = {"%8.1f ms"};

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

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

/*
 * Sets new number format
 * @param format the new number format
 */
```



```cpp
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) ∧
        (strlen(format) ≤ 10))
    {
        strcpy(numberFormat, format);
    }
    else
    {
        qWarning("QcvProcessor::setNumberFormat(%s) : invalid format", format);
    }
}


/*
 * 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)
     */
    emit updated();
    processTimeString.sprintf(numberFormat, getProcessTime(0) / 1000.0);
    emit processTimeUpdated(processTimeString);
}

/*
 * 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)
{
    if (verboseLevel ≥ VERBOSE_NOTIFICATIONS)
    {
        clog << "QcvProcessor::setSourceImage(" << (ulong) image << ")" << endl;
    }

    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))
```

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



```
1  /*
2   * QcvColorSpaces.h
3   *
4   *  Created on: 25 fÃ©vr. 2012
5   *      Author: davidroussel
6   */
7  
8  #ifndef QCVCOLORSPACES_H_
9  #define QCVCOLORSPACES_H_
10 
11 #include "QcvProcessor.h"
12 #include "CvColorSpaces.h"
13 
14 /**
15  * Qt oriented Colorspaces
16  */
17 class QcvColorSpaces : public QcvProcessor, public CvColorSpaces
18 {
19     Q_OBJECT
20 
21     public:
22         /**
23          * QcvColorSpaces constructor
24          * @param inFrame the input frame from capture
25          * @param imageLock the mutex on source image
26          * @param updateThread the thread in which this processor runs
27          * @param parent object
28          */
29         QcvColorSpaces(Mat * inFrame,
30                        QMutex * imageLock = NULL,
31                        QThread * updateThread = NULL,
32                        QObject * parent = NULL);
33 
34         /**
35          * QcvColorSpaces destructor
36          */
37         virtual ~QcvColorSpaces();
38 
39         /**
40          * Select image to set in displayImage and sends notification message
41          * @param index select the index to select display image
42          */
43         void setDisplayImageIndex(const Display index);
44 
45         /**
46          * Sets the color display status of selected component and sends
47          * notification message
48          * @param c the selected component:
49          * @param value the value to set on the selected component
50          */
51         void setColorChannel(const ShowColor c, const bool value);
52 
53         /**
54          * Select hue display mode and sends notification message
55          * @param mode the mode so select
56          */
57         void setHueDisplayMode(const HueDisplay mode);
58 
59     public slots:
60         /**
61          * Update computed images and sends displayImageChanged signal if
62          * required
63          */
64         void update();
65 };
66 
67 #endif /* QCVCOLORSPACES_H_ */
```

```
/*
 * QcvColorSpaces.cpp
 *
 *  Created on: 25 fÃ©vr. 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)
{
}

/*
 * QcvColorSpaces destructor
 */
QcvColorSpaces::~QcvColorSpaces()
{
}

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

    CvColorSpaces::update();

    if (sourceLock ≠ NULL)
    {
        // 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)
{
    CvColorSpaces::setDisplayImageIndex(index);

    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)
{
    CvColorSpaces::setColorChannel(c, value);

    message.clear();
    message.append(tr("Setting "));
    switch (c)
    {
        case BINDEX:
            message.append(tr("blue"));
            break;
        case GINDEX:
            message.append(tr("green"));
            break;
        case RINDEX:
            message.append(tr("red"));
            break;
        case HINDEX:
            message.append(tr("hue"));
            break;
        case CbINDEX:
            message.append(tr("Cb"));
            break;
        case CrINDEX:
            message.append(tr("Cr"));
            break;
        case NbShows:
        default:
            message.append(tr("unknown"));
            break;
```

```
    }
    message.append(tr("component show as colored to: "));
    if (value)
    {
        message.append(tr("on"));
    }
    else
    {
        message.append(tr("off"));
    }

    emit sendMessage(message, defaultTimeOut);

}

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

    message.clear();
    message.append(tr("Setting hue color display as: "));
    switch (mode)
    {
        case HUECOLOR:
            message.append(tr("hue only"));
            break;
        case HUESATURATE:
            message.append(tr("hue x saturation"));
            break;
        case HUEVALUE:
            message.append(tr("hue x value"));
            break;
        case HUEGRAY:
            message.append(tr("hue as gray"));
            break;
        case NBHUES:
        default:
            message.append(tr("unknown"));
            break;
    }

    emit sendMessage(message, defaultTimeOut);
}
```



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

#ifndef QCVMATWIDGET_H_
#define QCVMATWIDGET_H_

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

#include <cv.h>
using namespace cv;

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

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

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

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

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

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

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

        /**
         * Default aspect ratio when image is not set yet
         */
```

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



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

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



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

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



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

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



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

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



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

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

using namespace std;

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

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

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

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

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

    QcvMatWidget::paintEvent(event);

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



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

#ifndef QOPENCVWIDGETQGL_H_
#define QOPENCVWIDGETQGL_H_

#include <QGLWidget>

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

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

//      size_t glCount;

    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 fÃ©vr. 2011
 *      Author: davidroussel
 */
#include <QDebug>

#include "QcvMatWidgetGL.h"

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

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

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

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

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

    convertImage();

    gl = new QGLImageRender(displayImage, this);

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

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



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

#ifndef QCVMATWIDGETIMAGE_H_
#define QCVMATWIDGETIMAGE_H_

#include <QImage>
#include <QPainter>

#include "QcvMatWidget.h"

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

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

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

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

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

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

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

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

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

    protected:
        /**
```

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



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

```
        delete qImage;
    }
    // setup and convert image
    QcvMatWidget::setSourceImage(sourceImage);
    convertImage();
    qImage = new QImage((uchar *) displayImage.data, displayImage.cols,
        displayImage.rows, displayImage.step,
        QImage::Format_RGB888);

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

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

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

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


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

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

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



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

#ifndef QGLIMAGERENDER_H_
#define QGLIMAGERENDER_H_

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

using namespace cv;

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

//      size_t fCount;

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

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

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

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

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

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

#endif /* QGLIMAGERENDER_H_ */
```

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

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

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

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

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

    glClear(GL_COLOR_BUFFER_BIT);

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

}

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

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

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



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

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

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

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

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

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


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

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

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

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

#ifndef QCVVIDEOCAPTURE_H_
#define QCVVIDEOCAPTURE_H_

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

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

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

    protected:

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

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

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

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

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

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

        /**
```



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
        static size_t defaultFrameNumberTest;

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

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

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

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

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

```



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

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

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

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

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

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

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

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

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

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

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

        /**
         * Open new video file
         * @param fileName video file to open
         * @param width desired width or 0 to keep capture width
         * @param height desired height or 0 to keep capture height
```

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



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

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

#include <QElapsedTimer>
#include <QMutexLocker>
#include <QDebug>

#include "QcvVideoCapture.h"

#include <opencv2/imgproc/imgproc.hpp>

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

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

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

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

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



```
    timer(new QTimer(updateThread ≡ NULL ? this : NULL)),
    updateThread(updateThread),
    mutex(QMutex::NonRecursive),
    lockLevel(0),
    liveVideo(true),
    flipVideo(flipVideo),
    resize(false),
    directResize(false),
    gray(gray),
    skip(skip),
    size(0, 0),
    originalSize(0, 0),
    frameRate(0.0),
    statusMessage()
{
    if (updateThread ≠ NULL)
    {
        moveToThread(this→updateThread);
        connect(this, SIGNAL(finished()), updateThread, SLOT(quit()),
                Qt::DirectConnection);
    }

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

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

/*
 * QcvVideoCapture constructor from file name
 * @param fileName video file to open
 * @param flipVideo mirror image
 * @param gray convert image to gray
 * @param skip indicates capture can skip an image. When the capture
 * result has not been processed yet, or when false that capture should
 * wait for the result to be processed before grabbing a new image.
 * This only applies when #updateThread is not NULL.
 * @param width desired width or 0 to keep capture width
 * @param height desired height or 0 to keep capture height
 * @param updateThread the thread used to run this capture
 * @param parent the parent QObject
 */
QcvVideoCapture::QcvVideoCapture(const QString & fileName,
                                 const bool flipVideo,
                                 const bool gray,
                                 const bool skip,
                                 const unsigned int width,
                                 const unsigned int height,
                                 QThread * updateThread,
                                 QObject * parent) :
    QObject(parent),
    filename(fileName),
    capture(fileName.toStdString()),
    timer(new QTimer(updateThread ≡ NULL ? this : NULL)),
    updateThread(updateThread),
    mutex(QMutex::NonRecursive),
    lockLevel(0),
    liveVideo(false),
    flipVideo(flipVideo),
    resize(false),
    directResize(false),
    gray(gray),
    skip(skip),
    size(0, 0),
```

```
    originalSize(0, 0),
    frameRate(0.0),
    statusMessage()
{
    if (updateThread ≠ NULL)
    {
        moveToThread(this→updateThread);
        connect(this, SIGNAL(finished()), updateThread, SLOT(quit()),
                Qt::DirectConnection);
    }

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

    if (grabTest())
    {
        setSize(width, height);
        QString message("File ");
        message.append(fileName);
        message.append(" ");

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

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

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

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

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

        emit finished();

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

        delete timer; // delete unparented timer
    }

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



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

/*
 * Open new device Id
 * @param deviceId device number to open
 * @param width desired width or 0 to keep capture width
 * @param height desired height or 0 to keep capture height
 * @return true if device has been opened and checked and timer launched
 */
bool QcvVideoCapture::open(const int deviceId,
                           const unsigned int width,
                           const unsigned int height)
{
    if (updateThread ≠ NULL)
    {
        if (lockLevel ≡ 0)
        {
            mutex.lock();
            // qDebug() << "QcvVideoCapture::open(" << deviceId << "...): lock";
        }
        lockLevel++;
    }

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

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

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

    capture.open(deviceId);

    bool grabbed = grabTest();

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

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

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

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

    return grabbed;
}

/*
 * Open new video file
 * @param fileName video file to open
 * @param width desired width or 0 to keep capture width
 * @param height desired height or 0 to keep capture height
```

```
329  * @return true if video has been opened and timer launched
330  */
331 bool QcvVideoCapture::open(const QString & fileName,
332                            const unsigned int width,
333                            const unsigned int height)
334 {
335     filename = fileName;
336 
337     if (timer→isActive())
338     {
339         timer→stop();
340         qDebug("timer stopped");
341     }
342 
343     if (updateThread ≠ NULL)
344     {
345         if (lockLevel ≡ 0)
346         {
347             mutex.lock();
348             // qDebug() << "QcvVideoCapture::open(" << fileName << "...): lock";
349         }
350         lockLevel++;
351     }
352 
353     if (capture.isOpened())
354     {
355         capture.release();
356     }
357 
358     if (¬image.empty())
359     {
360         image.release();
361     }
362 
363     capture.open(fileName.toStdString());
364 
365     bool grabbed = grabTest();
366 
367     if (grabbed)
368     {
369         setSize(width, height);
370         qDebug() << "open setSize done";
371         statusMessage.clear();
372         statusMessage.append("file ");
373         statusMessage.append(fileName);
374         statusMessage.append(" opened");
375 
376         int delay = grabInterval(statusMessage);
377         timer→start(delay);
378         liveVideo = false;
379         qDebug("timer started with %d ms delay", delay);
380 
381         // emit changes
382         // messageChanged already emitted by grabInterval
383         emit imageChanged(&imageDisplay);
384 
385     }
386 
387     if (updateThread ≠ NULL)
388     {
389         lockLevel−−;
390         if(lockLevel ≡ 0)
391         {
392             // qDebug() << "QcvVideoCapture::open(" << filename << "...): unlock";
393             mutex.unlock();
394         }
395     }
396 
397     return grabbed;
398 }
399 
400 /*
401  * Size accessor
402  * @return the image size
403  */
404 const QSize & QcvVideoCapture::getSize() const
405 {
406     return size;
407 }
408 
409 /*
410  * Sets #imageDisplay size according to preferred width and height
```



```
411  * @param width desired width
412  * @param height desired height
413  * @pre a first image have been grabbed
414  */
415 void QcvVideoCapture::setSize(const unsigned int width,
416                               const unsigned int height)
417 {
418     if ((updateThread ≠ NULL))
419     {
420         if (lockLevel ≡ 0)
421         {
422             mutex.lock();
423             // qDebug("QcvVideoCapture::setSize(%d, %d) locked", width, height);
424         }
425         lockLevel++;
426     }
427 
428     unsigned int preferredWidth;
429     unsigned int preferredHeight;
430 
431     // qDebug("QcvVideoCapture::setSize(%d, %d)", width, height);
432 
433     // if not empty then release it
434     if (¬imageResized.empty())
435     {
436         imageResized.release();
437     }
438 
439     if ((width ≡ 0) ∧ (height ≡ 0)) // reset to original size
440     {
441         if (directResize) // direct set size to original size
442         {
443             setDirectSize((unsigned int)originalSize.width(),
444                           (unsigned int)originalSize.height());
445             // image is updated into setDirectSize
446         }
447         preferredWidth = image.cols;
448         preferredHeight = image.rows;
449 
450         resize = false;
451         imageResized = image;
452     }
453     else // width != 0 or height != 0
454     {
455         if ((width ≡ (unsigned int)image.cols) ∧
456             (height ≡ (unsigned int)image.rows)) // unchanged
457         {
458             preferredWidth = image.cols;
459             preferredHeight = image.rows;
460             imageResized = image;
461 
462             if (((int)preferredWidth ≡ originalSize.width()) ∧
463                 ((int)preferredHeight ≡ originalSize.height()))
464             {
465                 resize = false;
466             }
467             else
468             {
469                 resize = true;
470             }
471         }
472         else // width or height have changed
473         {
474             /*
475              * Resize needed
476              */
477             preferredWidth = width;
478             preferredHeight = height;
479 
480             resize = true;
481 
482             if (directResize)
483             {
484                 setDirectSize(preferredWidth, preferredHeight);
485                 imageResized = image;
486             }
487             else
488             {
489                 imageResized = Mat(preferredHeight, preferredWidth, image.type());
490             }
491         }
492     }
```

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



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

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



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

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



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

```
                }
            }
        }
        else // capture image has data
        {
            /*
             * CAUTION
             * image->imageResized->imageFlipped->imageDisplay
             * constitute an image chain, so when size is changed with
             * setSize it should call setFlipVideo which should call
             * setGray
             */

            // resize image
            if (resize ∧ ¬directResize)
            {
                cv::resize(image, imageResized, imageResized.size(), 0, 0,
                    INTER_AREA);
            }
            /*
             * else imageResized.data is already == image.data
             */

            // flip image horizontally if required
            if (flipVideo)
            {
                flip(imageResized, imageFlipped, 1);
            }
            /*
             * else imageFlipped.data is already == imageResized.data
             */

            // convert image to gray if required
            if (gray)
            {
                cvtColor(imageFlipped, imageDisplay, CV_BGR2GRAY);
            }
            /*
             * else imageDisplay.data is already == imageFlipped.data
             */
            image_updated = true;
        }

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

        if (image_updated)
        {
            emit updated();
        }
    }
    else
    {
        // mutex hasn't been locked, so we skipped one capture
        // qDebug() << "Capture skipped an image";
    }
}
```



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

#ifndef CAPTUREFACTORY_H_
#define CAPTUREFACTORY_H_

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

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

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

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

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

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

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

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

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

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

    public:
        /**
         * Capture Factory constructor.
         * Arguments can be
         *  - [-d | --device] <device number> : camera number
         *  - [-f | --file] <filename> : video file name
         *  - [-m | --mirror] : flip image horizontally
```

```
         *  - [-g | --gray] : convert to gray level
         *  - [-s | --size] <width>x<height>: preferred width and height
         * @param argList program the argument list provided as a list of
         * strings
         */
        CaptureFactory(const QStringList & argList);

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

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

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

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

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

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

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

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

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

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

#endif /* CAPTUREFACTORY_H_ */
```



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

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

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

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

```
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;
```

```
        capture = new QcvVideoCapture(deviceNumber,
                                      flippedVideo,
                                      grayVideo,
                                      skipImages,
                                      preferredWidth,
                                      preferredHeight,
                                      updateThread);
    }
    else
    {
        // Video file or stream
        qDebug() << videoPath << "…";
        capture = new QcvVideoCapture(videoPath,
                                      flippedVideo,
                                      grayVideo,
                                      skipImages,
                                      preferredWidth,
                                      preferredHeight,
                                      updateThread);
    }

    return capture;
}

```



```
#ifndef 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);

```

```
83          /**
84           * Signal to send when requesting video flip
85           * @param flip video flip
86           */
87          void flipChanged(const bool flip);
88
89          /**
90           * Signal to send when gray source image request changes
91           * @param gray gray status
92           */
93          void grayChanged(const bool gray);
94
95      private:
96          /**
97           * The UI built in QtDesigner or QtCreator
98           */
99          Ui::MainWindow *ui;
100
101         /**
102          * The Capture object grabs frame using OpenCV HiGui
103          */
104         QcvVideoCapture * capture;
105
106         /**
107          * The Color space object to compute color components
108          */
109         QcvColorSpaces * processor;
110
111         /**
112          * Image preferred width
113          */
114         int preferredWidth;
115
116         /**
117          * Image preferred height
118          */
119         int preferredHeight;
120
121         /**
122          * Message to send to statusBar
123          */
124         QString message;
125
126         /**
127          * Changes widgetImage nature according to desired rendering mode.
128          * Possible values for mode are:
129          *  - IMAGE: widgetImage is assigned to a QcvMatWidgetImage instance
130          *  - PIXMAP: widgetImage is assigned to a QcvMatWidgetLabel instance
131          *  - GL: widgetImage is assigned to a QcvMatWidgetGL instance
132          * @param mode
133          */
134         void setupImageWidget(const RenderMode mode);
135
136         /**
137          * Setup UI according to capture settings when app launches
138          */
139         void setupUIfromCapture();
140
141         /**
142          * Setup UI according to processor settings when app launches
143          */
144         void setupUIfromProcessor();
145
146     private slots:
147
148         /**
149          * Setup processor from current UI settings when processor source image
150          * changes
151          */
152         void setupProcessorfromUI();
153
154         /**
155          * Menu action when Sources->camera 0 is selected
156          * Sets capture to open device 0. If device is not available
157          * menu item is set to inactive.
158          */
159         void on_actionCamera_0_triggered();
160
161         /**
162          * Menu action when Sources->camera 1 is selected
163          * Sets capture to open device 0. If device is not available
164          * menu item is set to inactive
```



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

```
247             * Flip capture image horizontally.
248             * changes capture flip status
249             */
250            void on_checkBoxFlip_clicked();
251
252            /**
253             * Select input image for display
254             */
255            void on_radioButtonInput_clicked();
256
257            /**
258             * Select Gray image for display
259             */
260            void on_radioButtonGray_clicked();
261
262            /**
263             * Select red component of RGB space for display
264             */
265            void on_radioButtonRed_clicked();
266
267            /**
268             * Select green component of RGB space for display
269             */
270            void on_radioButtonGreen_clicked();
271
272            /**
273             * Select blue component of RGB space for display
274             */
275            void on_radioButtonBlue_clicked();
276
277            /**
278             * Select hue component of HSV space for display
279             */
280            void on_radioButtonHue_clicked();
281
282            /**
283             * Select saturation component of HSV space for display
284             */
285            void on_radioButtonSaturation_clicked();
286
287            /**
288             * Select value component of HSV space for display
289             */
290            void on_radioButtonValue_clicked();
291
292            /**
293             * Select Y component of YCbCr space for display
294             */
295            void on_radioButtonY_clicked();
296
297            /**
298             * Select Cr component of YCbCr space for display
299             */
300            void on_radioButtonCr_clicked();
301
302            /**
303             * Select Cb component of YCbCr space for display
304             */
305            void on_radioButtonCb_clicked();
306
307            /**
308             * Select component display as colored image
309             */
310            void on_radioButtonChColor_clicked();
311
312            /**
313             * Select componet display as gray image
314             */
315            void on_radioButtonChGray_clicked();
316
317            /**
318             * Select hue component display as hue alone
319             */
320            void on_radioButtonMixHue_clicked();
321
322            /**
323             * Select hue component display as hue x saturation value
324             */
325            void on_radioButtonMixHueSat_clicked();
326
327            /**
328             * Select hue component display as hue x value value
```



```
329             */
330            void on_radioButtonMixHueVal_clicked();
331
332            /**
333             * Select X component for display
334             */
335            void on_radioButtonXYZ_X_clicked();
336
337            /**
338             * Select Y component for display
339             */
340            void on_radioButtonXYZ_Y_clicked();
341
342            /**
343             * Select Z component for display
344             */
345            void on_radioButtonXYZ_Z_clicked();
346
347            /**
348             * Select Maximum of RGB as display
349             */
350            void on_radioButtonMaxBGR_clicked();
351    };
352
353    #endif // MAINWINDOW_H
```

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

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

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

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

    // ----------------------------------------------------------------------
    // Assertions
    // ----------------------------------------------------------------------
    assert(capture ≠ NULL);

    assert(processor ≠ NULL);

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

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

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

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

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

    // Connects processor time to UI time label
    connect(processor, SIGNAL(processTimeUpdated(QString)),
            ui→labelProcessTimeValue, SLOT(setText(QString)));

    // Connects UI requests to capture
    connect(this, SIGNAL(sizeChanged(const QSize &)),
            capture, SLOT(setSize(const QSize &)));
    connect(this, SIGNAL(deviceChanged(int,uint,uint)),
            capture, SLOT(open(int,uint,uint)));
    connect(this, SIGNAL(fileChanged(QString,uint,uint)),
            capture, SLOT(open(QString,uint,uint)));
    connect(this, SIGNAL(flipChanged(bool)), capture, SLOT(setFlipVideo(bool)));
    // ----------------------------------------------------------------------
    // UI setup according to capture and processor options
    // ----------------------------------------------------------------------
    setupUIfromCapture();

    setupUIfromProcessor();
}

```



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

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

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

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

    if (w ≡ ui→widgetImage)
    {
        // delete removed widget
        delete ui→widgetImage;

        // create new widget
        Mat * image = processor→getImagePtr("display");
        switch (mode)
        {
            case RENDER_PIXMAP:
                ui→widgetImage = new QcvMatWidgetLabel(image);
                break;
            case RENDER_GL:
                ui→widgetImage = new QcvMatWidgetGL(image);
                break;
            case RENDER_IMAGE:
            default:
                ui→widgetImage = new QcvMatWidgetImage(image);
                break;
        }

        if (ui→widgetImage ≠ NULL)
        {
            ui→widgetImage→setObjectName(QString::fromUtf8("widgetImage"));

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

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

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

            // Sends message to status bar and sets menu checks
            message.clear();
            message.append(tr("Render mode set to "));
            switch (mode)
            {
                case RENDER_IMAGE:
                    ui→actionRenderPixmap→setChecked(false);
                    ui→actionRenderOpenGL→setChecked(false);
                    message.append(tr("QImage"));
                    break;
                case RENDER_PIXMAP:
                    ui→actionRenderImage→setChecked(false);
                    ui→actionRenderOpenGL→setChecked(false);
                    message.append(tr("QPixmap in QLabel"));
                    break;
                case RENDER_GL:
                    ui→actionRenderImage→setChecked(false);
                    ui→actionRenderPixmap→setChecked(false);
                    message.append("QGLWidget");
                    break;
```

```
                default:
                break;
            }
            emit sendMessage(message, 5000);
        }
        else
        {
            qDebug("MainWindow::on_actionRenderXXX new widget is null");
        }
    }
    else
    {
        qDebug("MainWindow::on_actionRenderXXX removed widget is not imageWidget");
    }
}

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

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

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

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

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

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

/*
 * Setup UI according to processor settings when app launches
 */
void MainWindow::setupUIfromProcessor()
{
    // Sets selected image for display
    switch (processor->getDisplayImageIndex())
    {
        case CvColorSpaces::INPUT:
            ui->radioButtonInput->setChecked(true);
            break;
        case CvColorSpaces::GRAY:
            ui->radioButtonGray->setChecked(true);
```



```
            break;
        case CvColorSpaces::RED:
            ui->radioButtonRed->setChecked(true);
            break;
        case CvColorSpaces::GREEN:
            ui->radioButtonGreen->setChecked(true);
            break;
        case CvColorSpaces::BLUE:
            ui->radioButtonBlue->setChecked(true);
            break;
        case CvColorSpaces::HUE:
            ui->radioButtonHue->setChecked(true);
            break;
        case CvColorSpaces::SATURATION:
            ui->radioButtonSaturation->setChecked(true);
            break;
        case CvColorSpaces::VALUE:
            ui->radioButtonValue->setChecked(true);
            break;
        case CvColorSpaces::Y:
            ui->radioButtonY->setChecked(true);
            break;
        case CvColorSpaces::Cr:
            ui->radioButtonCr->setChecked(true);
            break;
        case CvColorSpaces::Cb:
            ui->radioButtonCb->setChecked(true);
            break;
        case CvColorSpaces::NbSelected:
        default:
            // Do nothing
            break;
    }

    // By default set radio button gray channel to checked
    ui->radioButtonChGray->setChecked(true);

    // if at least one showColor index is true then set radiobutton color
    // channel to true
    for (size_t i = 0; i < CvColorSpaces::NbShows; i++)
    {
        if (processor->getColorChannel((CvColorSpaces::ShowColor)i))
        {
            ui->radioButtonChColor->setChecked(true);
            break;
        }
    }

    // Sets Hue mix mode
    switch (processor->getHueDisplaymode())
    {
        case CvColorSpaces::HUECOLOR:
            ui->radioButtonMixHue->setChecked(true);
            break;
        case CvColorSpaces::HUESATURATE:
            ui->radioButtonMixHueSat->setChecked(true);
            break;
        case CvColorSpaces::HUEVALUE:
            ui->radioButtonMixHueVal->setChecked(true);
            break;
        case CvColorSpaces::HUEGRAY:
            ui->radioButtonChGray->setChecked(true);
            break;
        default:
            break;
    }
}

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

    if (ui->radioButtonGray->isChecked())
    {
        processor->setDisplayImageIndex(CvColorSpaces::GRAY);
```

```
    }

    if (ui→radioButtonRed→isChecked())
    {
        processor→setDisplayImageIndex(CvColorSpaces::RED);
    }

    if (ui→radioButtonGreen→isChecked())
    {
        processor→setDisplayImageIndex(CvColorSpaces::GREEN);
    }

    if (ui→radioButtonBlue→isChecked())
    {
        processor→setDisplayImageIndex(CvColorSpaces::BLUE);
    }

    if (ui→radioButtonHue→isChecked())
    {
        processor→setDisplayImageIndex(CvColorSpaces::HUE);
    }

    if (ui→radioButtonSaturation→isChecked())
    {
        processor→setDisplayImageIndex(CvColorSpaces::SATURATION);
    }

    if (ui→radioButtonValue→isChecked())
    {
        processor→setDisplayImageIndex(CvColorSpaces::VALUE);
    }

    if (ui→radioButtonY→isChecked())
    {
        processor→setDisplayImageIndex(CvColorSpaces::Y);
    }

    if (ui→radioButtonCr→isChecked())
    {
        processor→setDisplayImageIndex(CvColorSpaces::Cr);
    }

    if (ui→radioButtonCb→isChecked())
    {
        processor→setDisplayImageIndex(CvColorSpaces::Cb);
    }

    if (ui→radioButtonChColor→isChecked())
    {
        for (size_t i = 0; i < CvColorSpaces::NbShows; i++)
        {
            processor→setColorChannel((CvColorSpaces::ShowColor)i, true);
        }
        if (ui→radioButtonMixHue→isChecked())
        {
            processor→setHueDisplayMode(CvColorSpaces::HUECOLOR);
        }
        else if (ui→radioButtonMixHueSat→isChecked())
        {
            processor→setHueDisplayMode(CvColorSpaces::HUESATURATE);
        }
        else
        {
            processor→setHueDisplayMode(CvColorSpaces::HUEVALUE);
        }
    }

    if (ui→radioButtonChGray→isChecked())
    {
        for (size_t i = 0; i < CvColorSpaces::NbShows; i++)
        {
            processor→setColorChannel((CvColorSpaces::ShowColor)i, false);
        }
        processor→setHueDisplayMode(CvColorSpaces::HUEGRAY);
    }
}

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



```
void MainWindow::on_actionCamera_0_triggered()
{
    int width = 0;
    int height = 0;

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

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

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

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

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

    emit deviceChanged(1, width, height);
}

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

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

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

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

    if (fileName.length() > 0)
    {
//      if (!capture->open(fileName, width, height))
//      {
//          qWarning("Unable to open device file : %s",
//                   fileName.toStdString().c_str());
//      }

```

```
493             emit fileChanged(fileName, width, height);
494         }
495         else
496         {
497             qWarning("empty file name");
498         }
499     }
500
501     /*
502      * Menu action to qui application
503      */
504     void MainWindow::on_actionQuit_triggered()
505     {
506         this->close();
507     }
508
509     /*
510      * Menu action when flip image is selected.
511      * Sets capture to change flip status which leads to reverse
512      * image horizontally
513      */
514     void MainWindow::on_actionFlip_triggered()
515     {
516         emit flipChanged(¬capture->isFlipVideo());
517
518         /*
519          * There is no need to update ui->checkBoxFlip since it is connected
520          * to ui->actionFlip through signals/slots
521          */
522     }
523
524     /*
525      * Menu action when original image size is selected.
526      * Sets capture not to resize image
527      */
528     void MainWindow::on_actionOriginalSize_triggered()
529     {
530
531         ui->actionConstrainedSize->setChecked(false);
532         emit sizeChanged(QSize(0, 0));
533     }
534
535     /*
536      * Menu action when constrained image size is selected.
537      * Sets capture resize to preferred width and height
538      */
539     void MainWindow::on_actionConstrainedSize_triggered()
540     {
541         ui->actionOriginalSize->setChecked(false);
542         emit sizeChanged(QSize(preferredWidth, preferredHeight));
543     }
544
545     /*
546      * Menu action to replace current image rendering widget by a
547      * QcvMatWidgetImage instance.
548      */
549     void MainWindow::on_actionRenderImage_triggered()
550     {
551         setupImageWidget(RENDER_IMAGE);
552     }
553
554     /*
555      * Menu action to replace current image rendering widget by a
556      * QcvMatWidgetLabel with pixmap instance.
557      */
558     void MainWindow::on_actionRenderPixmap_triggered()
559     {
560         setupImageWidget(RENDER_PIXMAP);
561     }
562
563     /*
564      * Menu action to replace current image rendering widget by a
565      * QcvMatWidgetGL instance.
566      */
567     void MainWindow::on_actionRenderOpenGL_triggered()
568     {
569         setupImageWidget(RENDER_GL);
570     }
571
572     /*
573      * Original size radioButton action.
574      * Sets capture resize to off
```



```
575      */
576     void MainWindow::on_radioButtonOrigSize_clicked()
577     {
578         ui->actionConstrainedSize->setChecked(false);
579         emit sizeChanged(QSize(0, 0));
580     }
581
582     /*
583      * Custom size radioButton action.
584      * Sets capture resize to preferred width and height
585      */
586     void MainWindow::on_radioButtonCustomSize_clicked()
587     {
588         ui->actionOriginalSize->setChecked(false);
589         emit sizeChanged(QSize(preferredWidth, preferredHeight));
590     }
591
592     /*
593      * Width spinbox value change.
594      * Changes the preferred width and if custom size is selected apply
595      * this custom width
596      * @param value the desired width
597      */
598     void MainWindow::on_spinBoxWidth_valueChanged(int value)
599     {
600         preferredWidth = value;
601         if (ui->radioButtonCustomSize->isChecked())
602         {
603             emit sizeChanged(QSize(preferredWidth, preferredHeight));
604         }
605     }
606
607     /*
608      * Height spinbox value change.
609      * Changes the preferred height and if custom size is selected apply
610      * this custom height
611      * @param value the desired height
612      */
613     void MainWindow::on_spinBoxHeight_valueChanged(int value)
614     {
615         preferredHeight = value;
616         if (ui->radioButtonCustomSize->isChecked())
617         {
618             emit sizeChanged(QSize(preferredWidth, preferredHeight));
619         }
620     }
621
622     /*
623      * Flip capture image horizontally.
624      * changes capture flip status
625      */
626     void MainWindow::on_checkBoxFlip_clicked()
627     {
628         /*
629          * There is no need to update ui->actionFlip since it is connected
630          * to ui->checkBoxFlip through signals/slots
631          */
632         emit flipChanged(ui->checkBoxFlip->isChecked());
633     }
634
635     /*
636      * Select input image for display
637      */
638     void MainWindow::on_radioButtonInput_clicked()
639     {
640         processor->setDisplayImageIndex(CvColorSpaces::INPUT);
641     }
642
643     /*
644      * Select Gray image for display
645      */
646     void MainWindow::on_radioButtonGray_clicked()
647     {
648         processor->setDisplayImageIndex(CvColorSpaces::GRAY);
649     }
650
651     /*
652      * Select red component of RGB space for display
653      */
654     void MainWindow::on_radioButtonRed_clicked()
655     {
656         processor->setDisplayImageIndex(CvColorSpaces::RED);
```

```
}

/*
 * Select green component of RGB space for display
 */
void MainWindow::on_radioButtonGreen_clicked()
{
    processor->setDisplayImageIndex(CvColorSpaces::GREEN);
}

/*
 * Select blue component of RGB space for display
 */
void MainWindow::on_radioButtonBlue_clicked()
{
    processor->setDisplayImageIndex(CvColorSpaces::BLUE);
}

/*
 * Select hue component of HSV space for display
 */
void MainWindow::on_radioButtonHue_clicked()
{
    processor->setDisplayImageIndex(CvColorSpaces::HUE);
}

/*
 * Select saturation component of HSV space for display
 */
void MainWindow::on_radioButtonSaturation_clicked()
{
    processor->setDisplayImageIndex(CvColorSpaces::SATURATION);
}

/*
 * Select value component of HSV space for display
 */
void MainWindow::on_radioButtonValue_clicked()
{
    processor->setDisplayImageIndex(CvColorSpaces::VALUE);
}

/*
 * Select Y component of YCbCr space for display
 */
void MainWindow::on_radioButtonY_clicked()
{
    processor->setDisplayImageIndex(CvColorSpaces::Y);
}

/*
 * Select Cr component of YCbCr space for display
 */
void MainWindow::on_radioButtonCr_clicked()
{
    processor->setDisplayImageIndex(CvColorSpaces::Cr);
}

/*
 * Select Cb component of YCbCr space for display
 */
void MainWindow::on_radioButtonCb_clicked()
{
    processor->setDisplayImageIndex(CvColorSpaces::Cb);
}

/*
 * Select component display as colored image
 */
void MainWindow::on_radioButtonChColor_clicked()
{
    for (size_t i = 0; i < CvColorSpaces::NbShows; i++)
    {
        processor->setColorChannel((CvColorSpaces::ShowColor)i, true);
    }
    if (ui->radioButtonMixHue->isChecked())
    {
        processor->setHueDisplayMode(CvColorSpaces::HUECOLOR);
    }
    else if (ui->radioButtonMixHueSat->isChecked())
    {
        processor->setHueDisplayMode(CvColorSpaces::HUESATURATE);
```



```
    }
    else
    {
        processor->setHueDisplayMode(CvColorSpaces::HUEVALUE);
    }
}

/*
 * Select componet display as gray image
 */
void MainWindow::on_radioButtonChGray_clicked()
{
    for (size_t i = 0; i < CvColorSpaces::NbShows; i++)
    {
        processor->setColorChannel((CvColorSpaces::ShowColor)i, false);
    }
    processor->setHueDisplayMode(CvColorSpaces::HUEGRAY);
}

/*
 * Select hue component display as hue alone
 */
void MainWindow::on_radioButtonMixHue_clicked()
{
    processor->setHueDisplayMode(CvColorSpaces::HUECOLOR);
}

/*
 * Select hue component display as hue x saturation value
 */
void MainWindow::on_radioButtonMixHueSat_clicked()
{
    processor->setHueDisplayMode(CvColorSpaces::HUESATURATE);
}

/*
 * Select hue component display as hue x value value
 */
void MainWindow::on_radioButtonMixHueVal_clicked()
{
    processor->setHueDisplayMode(CvColorSpaces::HUEVALUE);
}

void MainWindow::on_radioButtonXYZ_X_clicked()
{
    processor->setDisplayImageIndex(CvColorSpaces::XYZ_X);
}

void MainWindow::on_radioButtonXYZ_Y_clicked()
{
    processor->setDisplayImageIndex(CvColorSpaces::XYZ_Y);
}

void MainWindow::on_radioButtonXYZ_Z_clicked()
{
    processor->setDisplayImageIndex(CvColorSpaces::XYZ_Z);
}

void MainWindow::on_radioButtonMaxBGR_clicked()
{
    processor->setDisplayImageIndex(CvColorSpaces::MAX_BGR);
}
```

```
#ifndef RED_MAP_
#define RED_MAP_

/**
 * Color map for RGB red component color image
 */
unsigned char mapRed[256][3] =
{
    {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, 0}
};

#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},
    {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}
};

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

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

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

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

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



```
    {
        capThread = new QThread();
    }

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

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

    // Processsor
    QcvColorSpaces * colorSpace = NULL;
    if (procThread ≡ NULL)
    {
        colorSpace = new QcvColorSpaces(capture→getImage());
    }
    else
    {
        if (procThread ≠ capThread)
        {
            colorSpace = new QcvColorSpaces(capture→getImage(),
                                            capture→getMutex(),
                                            procThread);
        }
        else // procThread == capThread
        {
            colorSpace = new QcvColorSpaces(capture→getImage(),
                                            NULL,
                                            procThread);
        }
    }
    colorSpace→setVerboseLevel(CvProcessor::VERBOSE_WARNINGS);

    // ----------------------------------------------------------------------
    // Connects capture to colorSpaces
    // ----------------------------------------------------------------------
    // Connects capture update to ColorSpace update
    QObject::connect(capture, SIGNAL(updated()),
                     colorSpace, SLOT(update()));

    // connect capture changed image to colorSpace set input
    QObject::connect(capture, SIGNAL(imageChanged(Mat*)),
                     colorSpace, SLOT(setSourceImage(Mat*)));

    // ----------------------------------------------------------------------
    // Now that Capture & colorSpace are on then
    // add our MainWindow as toplevel
    // and launches app
    // ----------------------------------------------------------------------
    MainWindow w(capture, colorSpace);
    w.show();

    usage(argv[0]);

    int retVal = app.exec();

    // ----------------------------------------------------------------------
    // Cleanup & return
    // ----------------------------------------------------------------------
    delete capture; // Should quit the capThread if any
    delete colorSpace; // Should quit the procThread if any

    bool sameThread = capThread ≡ procThread;

    if (capThread ≠ NULL)
    {
        delete capThread;
    }
```

```
165
166     if (procThread ≠ NULL ∧ ¬sameThread)
167     {
168         delete procThread;
169     }
170
171     return retVal;
172 }
173
174 /*
175  * Usage function shown just before launching QApp
176  * @param name the name of the program (argv[0])
177  */
178 void usage(char * name)
179 {
180     cout << "usage :" << basename(name) << " "
181          << "[-d | --device] <device number> "
182          << "[-v | --video] <video file> "
183          << "[-s | --size] <width>x<height> "
184          << "[-m | --mirror] " << endl
185          << "\t if no argument provided try to open first webcam" << endl
186          << "Key help : components multiple keystrokes switches from colored "
187          << "to B&W component display" << endl
188          << "\ti : Show color input image" << endl
189          << "\tk : Show lightness image" << endl
190          << "\tr : Show red component image from RGB color model" << endl
191          << "\tg : Show green component image from RGB color model" << endl
192          << "\tb : Show blue component image from RGB color model" << endl
193          << "\tx : Show X component image from XYZ color model" << endl
194          << "\tw : Show Y component image from XYZ color model" << endl
195          << "\tz : Show Z component image from XYZ color model" << endl
196          << "\th : Show hue component image from HSV color model" << endl
197          << "\ts : Show saturation component image from HSV color model" << endl
198          << "\tv : Show value component image from HSV color model" << endl
199          << "\ty : Show lightness image from YCbCr color model" << endl
200          << "\tu : Show Cr component image from YCbCr color model" << endl
201          << "\tt : Show Cb component image from YCbCr color model" << endl
202          << "\te : prints this help" << endl;
203 }
```