```cpp
/*! \class Conversion
class which converts a cv::Mat image from one colorspace to the next colorspace
*/
#include "Conversion.h"
namespace Vision
{
    /*! Constructor of the class    */
    Conversion::Conversion()
    {
        OriginalColorSpace = None;
        ProcessedColorSpace = None;
    }

    /*! Constructor of the class
    \param src a cv::Mat object which is the source image
    */
    Conversion::Conversion(const Mat &src)
    {
        OriginalColorSpace = None;
        ProcessedColorSpace = None;
        OriginalImg = src;
    }

    /*! De-constructor of the class*/
    Conversion::~Conversion() { }

    /*! Convert the source image from one colorspace to a destination colorspace
    - RGB 2 Intensity
    - RGB 2 XYZ
    - RGB 2 Lab
    - RGB 2 Redness Index
    - XYZ 2 Lab
    - XYZ 2 Redness Index
    - Lab 2 Redness Index
    \param src a cv::Mat object which is the source image
    \param dst a cv::Mat object which is the destination image
    \param convertFrom the starting colorspace
    \param convertTo the destination colorspace
    \param chain use the results from the previous operation default value = false;
    */
    void Conversion::Convert(const Mat &src, Mat &dst, ColorSpace convertFrom, ColorSpace convertTo, bool chain)
```

```
42      {
43          OriginalImg = src;
44          Convert(convertFrom, convertTo, chain);
45          dst = ProcessedImg;
46      }
47
48      /*! Convert the source image from one colorspace to a destination colorspace posibilities are:
49      - RGB 2 Intensity
50      - RGB 2 XYZ
51      - RGB 2 Lab
52      - RGB 2 Redness Index
53      - XYZ 2 Lab
54      - XYZ 2 Redness Index
55      - Lab 2 Redness Index
56      \param convertFrom the starting colorspace
57      \param convertTo the destination colorspace
58      \param chain use the results from the previous operation default value = false;
59      */
60      void Conversion::Convert(ColorSpace convertFrom, ColorSpace convertTo, bool chain)
61      {
62          OriginalColorSpace = convertFrom;
63          ProcessedColorSpace = convertTo;
64
65          // Exception handling
66          EMPTY_CHECK(OriginalImg);
67
68          int nData = OriginalImg.rows * OriginalImg.cols;
69          uint32_t i, j;
70
71          if (convertFrom == RGB && convertTo == Intensity) // RGB 2 Intensity
72          {
73              ProcessedImg.create(OriginalImg.size(), CV_8UC1);
74              uchar *P = ProcessedImg.data;
75              uchar *O;
76              CHAIN_PROCESS(chain, O, uchar);
77
78              RGB2Intensity(O, P, nData);
79          }
80          else if (convertFrom == RGB && convertTo == CIE_XYZ) // RGB 2 XYZ
81          {
82              ProcessedImg.create(OriginalImg.size(), CV_32FC3);
```

```
83              float *P = (float *)ProcessedImg.data;
84              uchar *O;
85              CHAIN_PROCESS(chain, O, uchar);
86
87              RGB2XYZ(O, P, nData);
88          }
89          else if (convertFrom == RGB && convertTo == CIE_lab) // RGB 2 Lab
90          {
91              ProcessedImg.create(OriginalImg.size(), CV_32FC3);
92              float *P = (float *)ProcessedImg.data;
93              uchar *O;
94              CHAIN_PROCESS(chain, O, uchar);
95
96              RGB2XYZ(O, P, nData);
97              Convert(CIE_XYZ, CIE_lab, true);
98          }
99          else if (convertFrom == RGB && convertTo == RI) // RGB 2 RI
100         {
101             ProcessedImg.create(OriginalImg.size(), CV_32FC3);
102             float *P = (float *)ProcessedImg.data;
103             uchar *O;
104             CHAIN_PROCESS(chain, O, uchar);
105
106             RGB2XYZ(O, P, nData);
107             Convert(CIE_XYZ, CIE_lab, true);
108             Convert(CIE_lab, RI, true);
109         }
110         else if (convertFrom == CIE_XYZ && convertTo == CIE_lab) // XYZ 2 Lab
111         {
112             ProcessedImg.create(OriginalImg.size(), CV_32FC3);
113             float *P = (float *)ProcessedImg.data;
114             float *O;
115             CHAIN_PROCESS(chain, O, float);
116
117             XYZ2Lab(O, P, nData);
118         }
119         else if (convertFrom == CIE_XYZ && convertTo == RI) // XYZ 2 RI
120         {
121             ProcessedImg.create(OriginalImg.size(), CV_32FC3);
122             float *P = (float *)ProcessedImg.data;
123             float *O;
```

```
124             CHAIN_PROCESS(chain, O, float);
125
126             XYZ2Lab(O, P, nData);
127             Convert(CIE_lab, RI, true);
128         }
129         else if (convertFrom == CIE_lab && convertTo == RI) // Lab 2 RI
130         {
131             ProcessedImg.create(OriginalImg.size(), CV_32FC1);
132             float *P = (float *)ProcessedImg.data;
133             float *O;
134             CHAIN_PROCESS(chain, O, float);
135
136             Lab2RI(O, P, nData);
137         }
138         else { throw Exception::ConversionNotSupportedException(); }
139     }
140
141     /*! Conversion from RGB to Intensity
142     \param O a uchar pointer to the source image
143     \param P a uchar pointer to the destination image
144     \param nData an int indicating the total number of pixels
145     */
146     void Conversion::RGB2Intensity(uchar *O, uchar *P, int nData)
147     {
148         uint32_t i;
149         uint32_t j;
150         i = 0;
151         j = 0;
152         while (j < nData)
153         {
154             P[j++] = (*(O + i + 2) * 0.2126 + *(O + i + 1) * 0.7152 + *(O + i) * 0.0722); // Grey value
155             i += 3;
156         }
157     }
158
159     /*! Conversion from RGB to CIE XYZ
160     \param O a uchar pointer to the source image
161     \param P a uchar pointer to the destination image
162     \param nData an int indicating the total number of pixels
163     */
164     void Conversion::RGB2XYZ(uchar *O, float *P, int nData)
165     {
```

```
166         uint32_t i = 0;
167         uint32_t endData = nData * OriginalImg.step.buf[1];
168         float R, G, B;
169         for (uint32_t i = 0; i < endData; i += OriginalImg.step.buf[1])
170         {
171             R = static_cast<float>(*(O + i + 2) / 255.0f);
172             B = static_cast<float>(*(O + i + 1) / 255.0f);
173             G = static_cast<float>(*(O + i)     / 255.0f);
174             P[i] =      (XYZmat[0][0] * R) + (XYZmat[0][1] * B) + (XYZmat[0][2] * G);   //X
175             P[i + 1] =  (XYZmat[1][0] * R) + (XYZmat[1][1] * B) + (XYZmat[1][2] * G);   //Y
176             P[i + 2] =  (XYZmat[2][0] * R) + (XYZmat[2][1] * B) + (XYZmat[2][2] * G);   //Z
177         }
178     }
179
180     /*! Conversion from CIE XYZ to CIE La*b*
181     \param O a uchar pointer to the source image
182     \param P a uchar pointer to the destination image
183     \param nData an int indicating the total number of pixels
184     */
185     void Conversion::XYZ2Lab(float *O, float *P, int nData)
186     {
187         uint32_t i = 0;
188         uint32_t endData = nData * 3;
189         float yy0, xx0, zz0;
190         for (size_t i = 0; i < endData; i += 3)
191         {
192             xx0 = *(O + i) / whitePoint[0];
193             yy0 = *(O + i + 1) / whitePoint[1];
194             zz0 = *(O + i + 2) / whitePoint[2];
195
196             if (yy0 > 0.008856)
197             {
198                 P[i] = (116 * pow(yy0, 0.333f)) - 116; // L
199             }
200             else
201             {
202                 P[i] = 903.3 * yy0; // L
203             }
204
205             P[i + 1] = 500 * (f_xyz2lab(xx0) - f_xyz2lab(yy0));
206             P[i + 2] = 200 * (f_xyz2lab(yy0) - f_xyz2lab(zz0));
207         }
```

```
208     }
209
210     inline float Conversion::f_xyz2lab(float t)
211     {
212         if (t > 0.008856) { return pow(t, 0.3333333333f); }
213         return 7.787 * t + 0.137931034482759f;
214     }
215
216     /*! Conversion from CIE La*b* to Redness Index
217     \param O a uchar pointer to the source image
218     \param P a uchar pointer to the destination image
219     \param nData an int indicating the total number of pixels
220     */
221     void Conversion::Lab2RI(float *O, float *P, int nData)
222     {
223         uint32_t i = 0;
224         uint32_t j = 0;
225         float L, a, b;
226         while (j < nData)
227         {
228             L = *(O + i);
229             a = *(O + i + 1);
230             b = *(O + i + 2);
231             P[j++] = (L * (pow((pow(a, 2.0f) + pow(b, 2.0f)), 0.5f) * (pow(10, 8.2f)))) / (b * pow(L, 6.0f));
232             i += 3;
233         }
234     }
235 }
```