```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)
```

```
{
    OriginalImg = src;
    Convert(convertFrom, convertTo, chain);
    dst = ProcessedImg;
}

/*! Convert the source image from one colorspace to a destination colorspace posibilities are:
- 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 convertFrom the starting colorspace
\param convertTo the destination colorspace
\param chain use the results from the previous operation default value = false;
*/
void Conversion::Convert(ColorSpace convertFrom, ColorSpace convertTo, bool chain)
{
    OriginalColorSpace = convertFrom;
    ProcessedColorSpace = convertTo;

    // Exception handling
    EMPTY_CHECK(OriginalImg);

    int nData = OriginalImg.rows * OriginalImg.cols;
    uint32_t i, j;

    if (convertFrom == RGB && convertTo == Intensity) // RGB 2 Intensity
    {
        ProcessedImg.create(OriginalImg.size(), CV_8UC1);
        uchar *P = ProcessedImg.data;
        uchar *O;
        CHAIN_PROCESS(chain, O, uchar);

        RGB2Intensity(O, P, nData);
    }
    else if (convertFrom == RGB && convertTo == CIE_XYZ) // RGB 2 XYZ
    {
        ProcessedImg.create(OriginalImg.size(), CV_32FC3);
```

```cpp
            float *P = (float *)ProcessedImg.data;
            uchar *O;
            CHAIN_PROCESS(chain, O, uchar);

            RGB2XYZ(O, P, nData);
        }
        else if (convertFrom == RGB && convertTo == CIE_lab) // RGB 2 Lab
        {
            ProcessedImg.create(OriginalImg.size(), CV_32FC3);
            float *P = (float *)ProcessedImg.data;
            uchar *O;
            CHAIN_PROCESS(chain, O, uchar);

            RGB2XYZ(O, P, nData);
            Convert(CIE_XYZ, CIE_lab, true);
        }
        else if (convertFrom == RGB && convertTo == RI) // RGB 2 RI
        {
            ProcessedImg.create(OriginalImg.size(), CV_32FC3);
            float *P = (float *)ProcessedImg.data;
            uchar *O;
            CHAIN_PROCESS(chain, O, uchar);

            RGB2XYZ(O, P, nData);
            Convert(CIE_XYZ, CIE_lab, true);
            Convert(CIE_lab, RI, true);
        }
        else if (convertFrom == CIE_XYZ && convertTo == CIE_lab) // XYZ 2 Lab
        {
            ProcessedImg.create(OriginalImg.size(), CV_32FC3);
            float *P = (float *)ProcessedImg.data;
            float *O;
            CHAIN_PROCESS(chain, O, float);

            XYZ2Lab(O, P, nData);
        }
        else if (convertFrom == CIE_XYZ && convertTo == RI) // XYZ 2 RI
        {
            ProcessedImg.create(OriginalImg.size(), CV_32FC3);
            float *P = (float *)ProcessedImg.data;
            float *O;
        CHAIN_PROCESS(chain, O, float);

        XYZ2Lab(O, P, nData);
        Convert(CIE_lab, RI, true);
    }
    else if (convertFrom == CIE_lab && convertTo == RI) // Lab 2 RI
    {
        ProcessedImg.create(OriginalImg.size(), CV_32FC1);
        float *P = (float *)ProcessedImg.data;
        float *O;
        CHAIN_PROCESS(chain, O, float);

        Lab2RI(O, P, nData);
    }
    else { throw Exception::ConversionNotSupportedException(); }
}

/*! Conversion from RGB to Intensity
\param O a uchar pointer to the source image
\param P a uchar pointer to the destination image
\param nData an int indicating the total number of pixels
*/
void Conversion::RGB2Intensity(uchar *O, uchar *P, int nData)
{
    uint32_t i;
    uint32_t j;
    i = 0;
    j = 0;
    while (j < nData)
    {
        P[j++] = (*(O + i + 2) * 0.2126 + *(O + i + 1) * 0.7152 + *(O + i) * 0.0722); // Grey value
        i += 3;
    }
}

/*! Conversion from RGB to CIE XYZ
\param O a uchar pointer to the source image
\param P a uchar pointer to the destination image
\param nData an int indicating the total number of pixels
*/
void Conversion::RGB2XYZ(uchar *O, float *P, int nData)
{
    uint32_t i = 0;
    uint32_t endData = nData * OriginalImg.step.buf[1];
    float R, G, B;
    for (uint32_t i = 0; i < endData; i += OriginalImg.step.buf[1])
    {
        R = static_cast<float>(*(O + i + 2) / 255.0f);
        B = static_cast<float>(*(O + i + 1) / 255.0f);
        G = static_cast<float>(*(O + i)     / 255.0f);
        P[i] =      (XYZmat[0][0] * R) + (XYZmat[0][1] * B) + (XYZmat[0][2] * G);   //X
        P[i + 1] =  (XYZmat[1][0] * R) + (XYZmat[1][1] * B) + (XYZmat[1][2] * G);   //Y
        P[i + 2] =  (XYZmat[2][0] * R) + (XYZmat[2][1] * B) + (XYZmat[2][2] * G);   //Z
    }
}

/*! Conversion from CIE XYZ to CIE La*b*
\param O a uchar pointer to the source image
\param P a uchar pointer to the destination image
\param nData an int indicating the total number of pixels
*/
void Conversion::XYZ2Lab(float *O, float *P, int nData)
{
    uint32_t i = 0;
    uint32_t endData = nData * 3;
    float yy0, xx0, zz0;
    for (size_t i = 0; i < endData; i += 3)
    {
        xx0 = *(O + i) / whitePoint[0];
        yy0 = *(O + i + 1) / whitePoint[1];
        zz0 = *(O + i + 2) / whitePoint[2];

        if (yy0 > 0.008856)
        {
            P[i] = (116 * pow(yy0, 0.333f)) - 116; // L
        }
        else
        {
            P[i] = 903.3 * yy0; // L
        }

        P[i + 1] = 500 * (f_xyz2lab(xx0) - f_xyz2lab(yy0));
        P[i + 2] = 200 * (f_xyz2lab(yy0) - f_xyz2lab(zz0));
        }
    }

    inline float Conversion::f_xyz2lab(float t)
    {
        if (t > 0.008856) { return pow(t, 0.3333333333f); }
        return 7.787 * t + 0.137931034482759f;
    }

    /*! Conversion from CIE La*b* to Redness Index
    \param O a uchar pointer to the source image
    \param P a uchar pointer to the destination image
    \param nData an int indicating the total number of pixels
    */
    void Conversion::Lab2RI(float *O, float *P, int nData)
    {
        uint32_t i = 0;
        uint32_t j = 0;
        float L, a, b;
        while (j < nData)
        {
            L = *(O + i);
            a = *(O + i + 1);
            b = *(O + i + 2);
            P[j++] = (L * (pow((pow(a, 2.0f) + pow(b, 2.0f)), 0.5f) * (pow(10, 8.2f)))) / (b * pow(L, 6.0f));
            i += 3;
        }
    }
}
```