```cpp
#include "Microscope.h"

//Custom exceptions
#include "MicroscopeNotFoundException.h"
#include "CouldNotGrabImageException.h"

using namespace cv;

namespace Hardware
{
    Microscope::Microscope()
    {
        FrameDelayTrigger = 3;
        Dimensions = Resolution{ 2592, 1944 };

        try { openCam(); }
        catch (Exception::MicroscopeNotFoundException& e)
        {
            // Tries to soft reset the USB port. Haven't got this working yet
            USB usbdev;
            usbdev.ResetUSB();
            captureDevice.open(0);
            if (!captureDevice.isOpened())
            {
                throw Exception::MicroscopeNotFoundException("Soft reset of microscope didn't work. Try turning the soil analyzer on and off again!");
            }
        }
    }

    /*! Constructor of the class which initializes the USB microscope
    \param frameDelayTrigger the delay between the first initialization of the microscope and the retrivial of the image expressed in seconds. Default value is 3 seconds
    \param dimension A resolution Struct indicating which resolution the webcam should use. Default is 2592 x 1944
    */
    Microscope::Microscope(uint8_t frameDelayTrigger, Resolution dimensions)
    {
        FrameDelayTrigger = frameDelayTrigger;
        Dimensions = dimensions;

        try {   openCam();  }
        catch (Exception::MicroscopeNotFoundException& e)
```

```
41          {
42              // Tries to soft reset the USB port. Haven't got this working yet
43              USB usbdev;
44              usbdev.ResetUSB();
45              captureDevice.open(0);
46              if (!captureDevice.isOpened())
47              {
48                  throw Exception::MicroscopeNotFoundException("Soft reset of microscope didn't work. Try turning the soil analyzer on
                      and off again!");
49              }
50          }
51      }
52
53      /*!< De-constructor*/
54      Microscope::~Microscope()
55      {
56          captureDevice.~VideoCapture();
57      }
58
59      /*! Get the frame after the set initialization period
60      \param dst a cv::Mat construct which stores the retrieved image
61      */
62      void Microscope::GetFrame(cv::Mat &dst)
63      {
64          if (!captureDevice.grab())  { throw Exception::CouldNotGrabImageException(); }
65          sleep(FrameDelayTrigger); // Needed otherwise scrambled picture
66          if (!captureDevice.grab())  { throw Exception::CouldNotGrabImageException(); }
67          captureDevice.retrieve(dst);
68      }
69
70      /*! Get an HDR capture of the cam using a user defined number of frames differently lit frames. Due to hardware limitations each
          frames take roughly 3 seconds to grab. This function is based upon the tutorial from openCV http://docs.opencv.org/trunk/doc/
          tutorials/photo/hdr_imaging/hdr_imaging.html
71      \param dst a cv::Mat construct with the retrieved HDR result
72      \param noframes is the number of frames that create the HDR image - default = 5
73      */
74      void Microscope::GetHDRFrame(cv::Mat &dst, uint32_t noframes)
75      {
76          //create the brightness steps
77          int8_t brightnessStep = static_cast<int8_t>((MAX_BRIGHTNESS - MIN_BRIGHTNESS)/ noframes);
78          int8_t currentBrightness = captureDevice.get(CV_CAP_PROP_BRIGHTNESS);
79          int8_t currentContrast = captureDevice.get(CV_CAP_PROP_CONTRAST);
```

```
80          captureDevice.set(CV_CAP_PROP_CONTRAST, MAX_CONTRAST);
81
82          Mat currentImg;
83
84          // take the shots at different brightness levels
85          for (uint32_t i = 1; i <= noframes; i++)
86          {
87              captureDevice.set(CV_CAP_PROP_BRIGHTNESS, (MIN_BRIGHTNESS + (i * brightnessStep)));
88              GetFrame(currentImg);
89              HDRframes.push_back(currentImg);
90          }
91
92          // Set the brightness and back to the previous used level
93          captureDevice.set(CV_CAP_PROP_BRIGHTNESS, currentBrightness);
94          captureDevice.set(CV_CAP_PROP_CONTRAST, currentContrast);
95
96          // Perform the exposure fusion
97          Mat fusion;
98          Ptr<MergeMertens> merge_mertens = createMergeMertens();
99          merge_mertens->process(HDRframes, fusion);
100         fusion *= 255;
101         fusion.convertTo(dst, CV_8UC1);
102     }
103
104     /*!< Checks if the capture device is open and returns the status as a bool
105     /return Status of the capture device expressed as a bool
106     */
107     bool Microscope::IsOpened()
108     {
109         return captureDevice.isOpened();
110     }
111
112     /*!< Safely release the capture device*/
113     void Microscope::Release()
114     {
115         captureDevice.release();
116     }
117
118     /*!< Opens the webcam*/
119     void Microscope::openCam()
120     {
121         captureDevice.open(0);
```

```
122
123         if (!captureDevice.isOpened()) { throw Exception::MicroscopeNotFoundException(); }
124         captureDevice.set(CV_CAP_PROP_FRAME_WIDTH, Dimensions.Width);
125         captureDevice.set(CV_CAP_PROP_FRAME_HEIGHT, Dimensions.Height);
126     }
127 }
```