```
1: /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8: #pragma once
9:
10: #include "Particle.h"
11: #include "Sample.h"
12: #include "AnalysisResults.h"
13: #include "AnalyseType.h"
14: #include "soilsettings.h"
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "SampleAnalysisResult.h"

namespace SoilAnalyzer {
SampleAnalysisResult::SampleAnalysisResult() {}

SampleAnalysisResult::~SampleAnalysisResult() {}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

// Source:
// http://stackoverflow.com/questions/16125574/how-to-serialize-opencv-mat-with-boost-xml-archive
#pragma once

#include <boost/archive/binary_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/serialization/access.hpp>
#include <opencv/cv.h>
#include <opencv2/core.hpp>

namespace boost {
namespace serialization {
/*!
 * \brief serialize Serialize the openCV mat to disk
 */
template <class Archive>
inline void serialize(Archive &ar, cv::Mat &m, const unsigned int version __attribute__((unused))) {
  int cols = m.cols;
  int rows = m.rows;
  int elemSize = m.elemSize();
  int elemType = m.type();

  ar &cols;
  ar &rows;
  ar &elemSize;
  ar &elemType; // element type.

  if (m.type() != elemType || m.rows != rows || m.cols != cols) {
    m = cv::Mat(rows, cols, elemType, cv::Scalar(0));
  }

  size_t dataSize = cols * rows * elemSize;

  for (size_t dc = 0; dc < dataSize; dc++) {
    ar &m.data[dc];
  }
}
}
}
```

```
 1: /* Copyright (C) Jelle Spijker - All Rights Reserved
 2:  * Unauthorized copying of this file, via any medium is strictly prohibited
 3:  * and only allowed with the written consent of the author (Jelle Spijker)
 4:  * This software is proprietary and confidential
 5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 6:  */
 7:
 8: #pragma once
 9: #include <stdint.h>
10: #include <utility>
11: #include <vector>
12:
13: #include <boost/serialization/base_object.hpp>
14: #include <boost/serialization/utility.hpp>
15: #include <boost/serialization/vector.hpp>
16:
17: #include "AnalyseType.h"
18: #include "../SoilMath/SoilMath.h"
19:
20: namespace SoilAnalyzer {
21: /*!
22:  * \brief The AnalysisResults class
23:  * \details the analysis results this is the base class for particle and soil
24:  * analysis results
25:  */
26: class AnalysisResults {
27: public:
28:   /*!
29:  * \brief AnalysisResults Constructor
30:  */
31:   AnalysisResults();
32:
33:   /*!
34:     * \brief AnalysisResult de-constructor
35:     */
36:   ~AnalysisResults();
37:
38:   std::vector<
39:       ucharStat_t> RGB_Stat; /**< A Vector with the Stats class for each color
40:                                 channel in the RGB*/
41:   std::vector<floatStat_t> LAB_Stat;
42:   floatStat_t RI_Stat;
43:
44: private:
45:   friend class boost::serialization::access;
46:   template <class Archive>
47:   void serialize(Archive &ar, const unsigned int version __attribute__((unused))) {
48:     ar &RGB_Stat;
49:     ar &LAB_Stat;
50:     ar &RI_Stat;
51:   }
52: };
53: }
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#define MIN_EDGE_PIXELS                                                    \
  128 /**< the minimum amount of pixels needed as an edge before it can be     \
         analysed*/
#define FFT_DESCRIPTORS 12 /**< the minimum amount of FFT descriptors*/

#include "AnalysisResults.h"
#include "ShapeClassification.h"
#include "../SoilVision/Vision.h"

#include <boost/serialization/base_object.hpp>

namespace SoilAnalyzer {
/*!
 * \brief The ParticleAnalysisResults class
 * \details The Analysis results of an individual particle, it inherents form
 * the class AnalysisResults
 */
class ParticleAnalysisResults : public AnalysisResults {
public:
  bool Analyzed = false;      /**< Indicates whether the results are analyzed*/
  bool SmallParticle = false; /**< Indicates if the particle is considered to
                                 small to analyze the voor shape*/
  uint32_t Area;              /**< The total area of the particle as a pixel*/
  ShapeClassification Shape;  /**< The Shape indicator*/

  /*!
   * \brief ParticleAnalysisResults The constructor
   */
  ParticleAnalysisResults();

  /*!
    * \brief Deconstructor
    */
  ~ParticleAnalysisResults();

private:
  friend class boost::serialization::access;
  template <class Archive>
  void serialize(Archive &ar, const unsigned int version __attribute__((unused))) {
    ar &Analyzed;
    ar &SmallParticle;
    ar &Area;
    ar &Shape;
    ar &BOOST_SERIALIZATION_BASE_OBJECT_NVP(AnalysisResults);
  }
};
}
```

```
 1: /* Copyright (C) Jelle Spijker - All Rights Reserved
 2:  * Unauthorized copying of this file, via any medium is strictly prohibited
 3:  * and only allowed with the written consent of the author (Jelle Spijker)
 4:  * This software is proprietary and confidential
 5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 6:  */
 7:
 8: #pragma once
 9: #include <boost/serialization/base_object.hpp>
10: #include <boost/serialization/utility.hpp>
11: #include <boost/serialization/complex.hpp>
12: #include <boost/serialization/vector.hpp>
13:
14: #include "../SoilMath/SoilMath.h"
15:
16: /*!
17:  * \brief The ShapeClassification class the class which describes the shape as
18:  * category and with FFT descriptor
19:  */
20: class ShapeClassification {
21: public:
22:   unsigned char Category;        /**< The category class*/
23:   ComplexVect_t FFT_descriptors; /**< The Fast Fourier Descriptors*/
24:
25:   /*!
26:    * \brief ShapeClassification the constructor
27:    */
28:   ShapeClassification();
29:
30:   /*!
31:    * \brief ShapeClassification the constructor
32:    * \param fft_descriptors teh fast fourier descriptors
33:    */
34:   ShapeClassification(ComplexVect_t fft_descriptors)
35:       : FFT_descriptors(fft_descriptors) {}
36:
37:   /*!
38:     * \brief The descontructor
39:     */
40:   ~ShapeClassification();
41:
42: private:
43:   friend class boost::serialization::access;
44:   template <class Archive>
45:   void serialize(Archive &ar, const unsigned int version __attribute__((unused))) {
46:     ar &Category;
47:     ar &FFT_descriptors;
48:   }
49: };
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <string>
#include <fstream>
#include <boost/archive/xml_iarchive.hpp>
#include <boost/archive/xml_oarchive.hpp>
#include "../SoilVision/Vision.h"

namespace SoilAnalyzer {
/*!
 * \brief The SoilSettings class
 * \details A class with which the used settings can easily transfered to setup
 * the Sample class in one go. This class is also used in the GUI. and has a
 * possibility to saved to disk as a serialized object
 */
class SoilSettings {
public:
  SoilSettings();

  /*!
   * \brief SaveSettings a function to save the settings to disk
   * \param filename a string with the filename
   */
  void SaveSettings(std::string filename);

  /*!
   * \brief LoadSettings a function to load the settings from disk
   * \param filename a string with the filename
   */
  void LoadSettings(std::string filename);

  bool useAdaptiveContrast =
      true; /**< Should adaptive contrast stretch be used default is true*/
  uint32_t adaptContrastKernelSize =
      9; /**< The size of the adaptive contrast kernelsize*/
  float adaptContrastKernelFactor = 1.; /**< the factor with which to multiply
                                           the effect of the adaptive contrast
                                           stretch*/

  bool useBlur = true; /**< Should the mediaan blur be used during analsyis*/
  uint32_t blurKernelSize = 5; /**< the median blurkernel*/

  Vision::Segment::TypeOfObjects typeOfObjectsSegmented =
      Vision::Segment::Dark; /**< Which type of object should be segmented*/
  bool ignorePartialBorderParticles =
      true; /**< Indication of partial border particles should be used*/
  bool fillHoles = true; /**< should the holes be filled*/
  float
      sigmaFactor = 2; /**< The sigma factor or the bandwidth indicating which
                          pixel intensity values count belong to an object*/
  int thresholdOffsetValue = 0; /**< an tweaking offset value*/

  Vision::MorphologicalFilter::FilterType morphFilterType =
      Vision::MorphologicalFilter::OPEN; /**< Indicating which type of
                                            morhpological filter should be
                                            used*/
  uint32_t filterMaskSize = 5;           /**< the filter mask*/

  uint32_t HDRframes =
      5; /**< The number of frames which should be used for the HDR image*/
  float lightLevel = 0.5; /**< The light level of the environmental case*/
  bool encInv = false;    /**< invert the values gained form the encoder*/
  bool enableRainbow =
      true; /**< run a rainbow loop on the RGB encoder during analysis*/

private:
  friend class boost::serialization::access;
  template <class Archive>
  void serialize(Archive &ar, const unsigned int version __attribute__((unused))) {
    ar &BOOST_SERIALIZATION_NVP(useAdaptiveContrast);
    ar &BOOST_SERIALIZATION_NVP(adaptContrastKernelFactor);
    ar &BOOST_SERIALIZATION_NVP(adaptContrastKernelSize);
    ar &BOOST_SERIALIZATION_NVP(useBlur);
    ar &BOOST_SERIALIZATION_NVP(blurKernelSize);
    ar &BOOST_SERIALIZATION_NVP(typeOfObjectsSegmented);
    ar &BOOST_SERIALIZATION_NVP(ignorePartialBorderParticles);
```

```
   84:     ar &BOOST_SERIALIZATION_NVP(fillHoles);
   85:     ar &BOOST_SERIALIZATION_NVP(sigmaFactor);
   86:     ar &BOOST_SERIALIZATION_NVP(morphFilterType);
   87:     ar &BOOST_SERIALIZATION_NVP(filterMaskSize);
   88:     ar &BOOST_SERIALIZATION_NVP(thresholdOffsetValue);
   89:     ar &BOOST_SERIALIZATION_NVP(HDRframes);
   90:     ar &BOOST_SERIALIZATION_NVP(lightLevel);
   91:     ar &BOOST_SERIALIZATION_NVP(encInv);
   92:     ar &BOOST_SERIALIZATION_NVP(enableRainbow);
   93:   }
   94: };
   95: }
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "ShapeClassification.h"

ShapeClassification::ShapeClassification() {}

ShapeClassification::~ShapeClassification() {}
```

```
 1: /* Copyright (C) Jelle Spijker - All Rights Reserved
 2:  * Unauthorized copying of this file, via any medium is strictly prohibited
 3:  * and only allowed with the written consent of the author (Jelle Spijker)
 4:  * This software is proprietary and confidential
 5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 6:  */
 7:
 8: #pragma once
 9:
10: #include <boost/serialization/base_object.hpp>
11:
12: #include "Soil.h"
13: #include "../SoilMath/SoilMath.h"
14: #include "ParticleAnalysisResults.h"
15:
16: namespace SoilAnalyzer {
17: /*!
18:  * \brief The Particle class
19:  * \details Representing an individual particle
20:  */
21: class Particle : public Soil {
22: public:
23:   /*!
24:  * \brief Particle the constructor
25:  */
26:   Particle();
27:
28:   /*!
29:     * \brief the deconstructor
30:     */
31:   ~Particle();
32:
33:   /*!
34:    * \brief Save the particle to disk
35:    * \param filename string indicating the filename
36:    */
37:   void Save(std::string &filename);
38:
39:   /*!
40:    * \brief Load load the particle from disk
41:    * \param filename string indicating the filename
42:    */
43:   void Load(std::string &filename);
44:
45:   ParticleAnalysisResults Analysis; /**< The Analysis results*/
46:
47:   /*!
48:    * \brief Analyze the function which analyses the particle
49:    * \param nn the neural network to be used pased as a reference
50:    * \return thes analysis results
51:    */
52:   SoilAnalyzer::AnalysisResults Analyze(SoilMath::NN &nn);
53:
54: private:
55:   friend class boost::serialization::access;
56:   template <class Archive>
57:   void serialize(Archive &ar, const unsigned int version __attribute__((unused))) {
58:     ar &BOOST_SERIALIZATION_BASE_OBJECT_NVP(Soil);
59:     ar &Analysis;
60:   }
61: };
62: }
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "ParticleAnalysisResults.h"

namespace SoilAnalyzer {
ParticleAnalysisResults::ParticleAnalysisResults() {}

ParticleAnalysisResults::~ParticleAnalysisResults() {}
}
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "Soil.h"

namespace SoilAnalyzer {
Soil::Soil() {}

Soil::~Soil() {}
}
```

```
 1: /* Copyright (C) Jelle Spijker - All Rights Reserved
 2:  * Unauthorized copying of this file, via any medium is strictly prohibited
 3:  * and only allowed with the written consent of the author (Jelle Spijker)
 4:  * This software is proprietary and confidential
 5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 6:  */
 7:
 8: #pragma once
 9: #include "AnalysisResults.h"
10: #include "ParticleAnalysisResults.h"
11: #include <boost/serialization/base_object.hpp>
12: #include <boost/serialization/vector.hpp>
13:
14: namespace SoilAnalyzer {
15: /*!
16:  * \brief The SampleAnalysisResult class
17:  * \details The class where the results are stored
18:  */
19: class SampleAnalysisResult : public AnalysisResults {
20: public:
21:
22:    /*!
23:    * \brief SampleAnalysisResult the constructor
24:    */
25:   SampleAnalysisResult();
26:
27:   /*!
28:     * \brief The deconstructor
29:     */
30:   ~SampleAnalysisResult();
31:
32: private:
33:   friend class boost::serialization::access;
34:   template <class Archive>
35:   void serialize(Archive &ar, const unsigned int version __attribute__((unused))) {
36:     ar &BOOST_SERIALIZATION_BASE_OBJECT_NVP(AnalysisResults);
37:   }
38: };
39: }
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <fstream>
#include <boost/archive/binary_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/serialization/string.hpp>
#include "Mat_archive.h"
#include <opencv2/core/core.hpp>
#include <stdint.h>
#include <string>
#include "../SoilVision/VisionDebug.h"

namespace SoilAnalyzer {
/*!
 * \brief The Soil class
 * \details The parent object of the Soil related objects
 */
class Soil {
private:
  friend class boost::serialization::access;
  template <class Archive>
  void serialize(Archive &ar, const unsigned int version __attribute__((unused))) {
    ar &ID;
    ar &Location;
    ar &TimeTaken;
    ar &TimeAnalyzed;
    ar &BW;
    ar &Intensity;
    ar &LAB;
    ar &RI;
    ar &RGB;
    ar &OptimizedInt;
  }

protected:
  cv::Mat OptimizedInt; /**< The enhanced int image*/

public:
  /*!
   * \brief Soil the constructor
   */
  Soil();

  /*!
    * \brief Soil deconstructor
    */
  ~Soil();
  cv::Mat BW; /**< The black and white image consisting of values of 0 and 1
                 where 0 is the bacground*/
  cv::Mat
      Intensity;   /**< The intensity image after it is converted from the RGB
                      color model*/
  cv::Mat LAB;     /**< The CIE Lab color image*/
  cv::Mat RGB;     /**< The RGB color image*/
  cv::Mat RI;      /**< The individual RI image*/
  cv::Mat Edge;    /**< The black and white image consiting of values of 0 and 1
                      where 0 is background and 1 is the edge of the blob*/
  uint8_t version; /**< the version of the object*/
  std::string
      TimeTaken; /**< a string indicating which time the sample was taken*/
  std::string
      TimeAnalyzed;     /**< a string indicating which time it was analyzed*/
  std::string Location; /**< a string with the location of the soilsample*/
  uint32_t ID;          /**< the sample ID*/
};
}
```

```
 1: /* Copyright (C) Jelle Spijker - All Rights Reserved
 2:  * Unauthorized copying of this file, via any medium is strictly prohibited
 3:  * and only allowed with the written consent of the author (Jelle Spijker)
 4:  * This software is proprietary and confidential
 5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 6:  */
 7:
 8: #pragma once
 9: #include <exception>
10: #include <string>
11: #include <stdint.h>
12:
13: using namespace std;
14:
15: namespace SoilAnalyzer {
16: namespace Exception {
17: class AnalysisException : public std::exception {
18: public:
19:   AnalysisException(string m = "Analysis Failed!") : msg(m){}
20:   AnalysisException(string m = "Analysis Failed!", uint8_t id = 0) : msg(m) {
21:     exid = id;
22:   }
23:   ~AnalysisException() _GLIBCXX_USE_NOEXCEPT{}
24:   const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); }
25:   const uint8_t ID() const _GLIBCXX_USE_NOEXCEPT { return exid; }
26:
27: private:
28:   string msg;
29:   uint8_t exid;
30: };
31: }
32: }
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "Particle.h"

namespace SoilAnalyzer {
Particle::Particle() {}

Particle::~Particle() {}

void Particle::Save(std::string &filename) {
  std::ofstream ofs(filename.c_str());
  boost::archive::binary_oarchive oa(ofs);
  oa << BOOST_SERIALIZATION_NVP(*this);
}

void Particle::Load(std::string &filename) {
  std::ifstream ifs(filename.c_str());
  boost::archive::binary_iarchive ia(ifs);
  ia >> BOOST_SERIALIZATION_NVP(*this);
}

SoilAnalyzer::AnalysisResults Particle::Analyze(SoilMath::NN &nn) {
  if (Analysis.Analyzed) {
    return Analysis;
  }

  // Calc the LAB stats
  std::vector<cv::Mat> lab = Vision::ImageProcessing::extractChannel(LAB);
  for_each(lab.begin(), lab.end(), [&](cv::Mat &M) {
    Analysis.LAB_Stat.push_back(floatStat_t((float *)M.data, M.rows, M.cols));
  });

  // Calc the RGB stats
  std::vector<cv::Mat> rgb = Vision::ImageProcessing::extractChannel(RGB);
  for_each(lab.begin(), lab.end(), [&](cv::Mat &M) {
    Analysis.RGB_Stat.push_back(ucharStat_t((uchar *)M.data, M.rows, M.cols));
  });

  // Calc the Redness Index stats
  Analysis.RI_Stat = floatStat_t((float *)RI.data, RI.rows, RI.cols);

  // Calc the area
  for_each(BW.data, BW.data + (BW.rows * BW.cols),
           [&](uchar P) { Analysis.Area += P; });

  // Calc the edge area
  uint32_t edgeArea = 0;
  for_each(Edge.data, Edge.data + (Edge.rows * Edge.cols),
           [&](uchar P) { edgeArea += P; });

  if (edgeArea >= MIN_EDGE_PIXELS) {
    Analysis.SmallParticle = false;
  } else {
    Analysis.SmallParticle = true;
  }

  // Determine the shape Classifiation, but only for pixels that are big enough
  if (!Analysis.SmallParticle) {
    // Calculate the FFT Descriptors
    SoilMath::FFT fft;
    Analysis.Shape.FFT_descriptors = fft.GetDescriptors(Edge);
    if (Analysis.Shape.FFT_descriptors.size() >= FFT_DESCRIPTORS) {
      Analysis.Shape.FFT_descriptors.erase(
          Analysis.Shape.FFT_descriptors.begin() + FFT_DESCRIPTORS,
          Analysis.Shape.FFT_descriptors.end());
    } else {
      while (Analysis.Shape.FFT_descriptors.size() < FFT_DESCRIPTORS) {
        Analysis.Shape.FFT_descriptors.push_back(Complex_t(0, 0));
      }
    }

    Predict_t sp = nn.Predict(Analysis.Shape.FFT_descriptors);
    uchar i = 0;
    float maxValue = 0.0;
    for_each(sp.OutputNeurons.begin(), sp.OutputNeurons.end(), [&](float &n) {
      if (n > maxValue) {
        Analysis.Shape.Category = i;
        maxValue = n;
```

```
84:       }
85:       i++;
86:     });
87:   }
88:   Analysis.Analyzed = true;
89:   return Analysis;
90: }
91: }
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "AnalysisResults.h"

namespace SoilAnalyzer {
AnalysisResults::AnalysisResults() {}

AnalysisResults::~AnalysisResults() {}
}
```

```
  1: /* Copyright (C) Jelle Spijker - All Rights Reserved
  2:  * Unauthorized copying of this file, via any medium is strictly prohibited
  3:  * and only allowed with the written consent of the author (Jelle Spijker)
  4:  * This software is proprietary and confidential
  5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
  6:  */
  7:
  8: #pragma once
  9: //#define DEBUG
 10: #define PROG_INCR(status)                                                \
 11:   currentProg += progstep;                                               \
 12:   prog_sig(currentProg, status)
 13: #include "Soil.h"
 14: #include "Particle.h"
 15: #include "AnalysisResults.h"
 16: #include "AnalysisException.h"
 17:
 18: #include <boost/serialization/base_object.hpp>
 19: #include <boost/serialization/vector.hpp>
 20: #include "Mat_archive.h"
 21:
 22: #include <boost/signals2.hpp>
 23: #include <boost/bind.hpp>
 24:
 25: #include <opencv2/core.hpp>
 26: #include <opencv2/highgui.hpp>
 27:
 28: #include <vector>
 29: #include <string>
 30:
 31: #include "../SoilVision/Vision.h"
 32: #include "../SoilMath/SoilMath.h"
 33: #include "soilsettings.h"
 34:
 35: using namespace std;
 36: using namespace cv;
 37:
 38: namespace SoilAnalyzer {
 39: /*!
 40:  * \brief The Sample class
 41:  * \details This class represent a single soilsample snapshot it inherents from
 42:  * the class Soil
 43:  */
 44: class Sample : public Soil {
 45: public:
 46:   typedef boost::signals2::signal<void(
 47:       float, std::string)> Progress_t; /**< A boost signal, used for progress
 48:                                           update indicating the progres so far
 49:                                           as a float between 0. and 1. and the
 50:                                           current progress step as string*/
 51:
 52:   /*!
 53:    * \brief connect_Progress used to connect the progress signal
 54:    * \param subscriber a reference to the subscriber
 55:    * \return returns the signal
 56:    */
 57:   boost::signals2::connection
 58:   connect_Progress(const Progress_t::slot_type &subscriber);
 59:
 60:   /*!
 61:    * \brief Sample the constructor
 62:    * \param settings a pointer to the an object of the type SoilSettings used to
 63:    * initalize the analyzing settings
 64:    */
 65:   Sample(SoilSettings *settings = nullptr);
 66:
 67:   /*!
 68:    * \brief Sample the constructor
 69:    * \param src the source rgb image
 70:    * \param settings a pointer to the an object of the type SoilSettings used to
 71:    * initalize the analyzing settings
 72:    */
 73:   Sample(const Mat &src, SoilSettings *settings = nullptr);
 74:
 75:   /*!
 76:     * \brief The Deconstructor
 77:     */
 78:   ~Sample();
 79:
 80:   cv::Mat OriginalImage;       /**< The original image*/
 81:   vector<Particle> Population; /**< a Vector with original particles*/
 82:   AnalysisResults Results;     /**< The analysis results*/
 83:
```

```
  84:   SoilSettings *Settings =
  85:       nullptr; /**< The Settings used to initialize the analyzis*/
  86:
  87:   /*!
  88:    * \brief PrepImg Prep te image for analysis
  89:    * \param settings a pointer to the an object of the type SoilSettings used to
  90:    * initalize the analyzing settings
  91:    */
  92:   void PrepImg(SoilSettings *settings = nullptr);
  93:   bool imgPrepped = false; /**< */
  94:
  95:   /*!
  96:    * \brief Analyse the RGB image
  97:    * \param nn a reference to the neural network used during analyzing
  98:    */
  99:   void Analyse(SoilMath::NN &nn);
 100:
 101:   /*!
 102:    * \brief Analyse the the RGB image
 103:    * \param src the RGB image with needs to be analysed
 104:    * \param nn a reference to the neural network used during analyzing
 105:    */
 106:   void Analyse(const Mat &src, SoilMath::NN &nn);
 107:
 108:   /*!
 109:    * \brief Save the SoilSample to disk
 110:    * \param filename the where to save
 111:    */
 112:   void Save(string &filename);
 113:
 114:   /*!
 115:    * \brief Load the SoilSample from disk
 116:    * \param filename where the saved Soilsample can be found
 117:    */
 118:   void Load(string &filename);
 119:
 120: private:
 121:   Progress_t prog_sig; /**< The progress signal*/
 122:
 123:   friend class boost::serialization::access;
 124:   template <class Archive>
 125:   void serialize(Archive &ar, const unsigned int version __attribute__((unused))) {
 126:     ar &BOOST_SERIALIZATION_BASE_OBJECT_NVP(Soil);
 127:     ar &OriginalImage;
 128:     ar &Population;
 129:     ar &Results;
 130:     ar &Settings;
 131:     ar &imgPrepped;
 132:   }
 133:
 134:   /*!
 135:    * \brief AnalysePopVect analyse the particle population vector
 136:    * \param population a reference to individual particles
 137:    * \param results
 138:    * \return
 139:    */
 140:   bool AnalysePopVect(const vector<Particle> &population,
 141:                       AnalysisResults &results);
 142:
 143:   /*!
 144:    * \brief SegmentParticles
 145:    * \param segType
 146:    */
 147:   void SegmentParticles(
 148:       Vision::Segment::SegmentationType segType = Vision::Segment::Normal);
 149: };
 150: }
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "Sample.h"

namespace SoilAnalyzer {
Sample::Sample(SoilSettings *settings) { Settings = settings; }

Sample::Sample(const Mat &src, SoilSettings *settings) {
  Settings = settings;
  OriginalImage = src.clone();
}

Sample::~Sample() {}

void Sample::Save(string &filename) {
  std::ofstream ofs(filename.c_str());
  boost::archive::binary_oarchive oa(ofs);
  oa << boost::serialization::make_nvp("SoilSample", *this);
}

void Sample::Load(string &filename) {
  std::ifstream ifs(filename.c_str());
  boost::archive::binary_iarchive ia(ifs);
  ia >> boost::serialization::make_nvp("SoilSample", *this);
}

void Sample::PrepImg(SoilSettings *settings) {
  // setup the settings
  if (settings == nullptr && Settings == nullptr) {
    Settings = new SoilSettings;
  } else if (Settings != nullptr) {
    Settings = settings;
  }

  // Determine the biggest kernelsize. These border pixels are discarded with
  // the optimized int
  uint32_t kBorder = ((Settings->adaptContrastKernelSize > Settings->blurKernelSize)
                      ? Settings->adaptContrastKernelSize : Settings->blurKernelSize);
  uint32_t khBorder = kBorder / 2;

  // set up the progress signal
  float currentProg = 0.;
  prog_sig(currentProg, "Starting segmentation");

  uint32_t totalsteps = 5;
  if (Settings->useAdaptiveContrast) {
    totalsteps++;
  }
  if (Settings->useBlur) {
    totalsteps++;
  }
  if (Settings->fillHoles) {
    totalsteps++;
  }
  if (Settings->ignorePartialBorderParticles) {
    totalsteps++;
  }
  if (Settings->morphFilterType != Vision::MorphologicalFilter::NONE) {
    totalsteps++;
  }
  float progstep = 1. / static_cast<float>(totalsteps);

  if (OriginalImage.empty()) {
    throw Exception::AnalysisException("No Image found to analyze!", 1);
  }
  SHOW_DEBUG_IMG(OriginalImage, uchar, 255, "RGB", false);

  // Convert the image to an intensity image and an enhanced Intinsity for
  // better segmentation
  Vision::Conversion RGBConvertor(OriginalImage);
  RGBConvertor.Convert(Vision::Conversion::RGB, Vision::Conversion::Intensity);
  PROG_INCR("Converted to intensity");
  Intensity = RGBConvertor.ProcessedImg.clone();
  SHOW_DEBUG_IMG(Intensity, uchar, 255, "Intensity", false);

  // Enhance the image with an Adaptive contrast stretch and/or followed by a
  // blur
  Vision::Enhance IntEnchance(Intensity);
```

```
  84:   if (Settings->useAdaptiveContrast) {
  85:     IntEnchance.AdaptiveContrastStretch(Settings->adaptContrastKernelSize,
  86:                                         Settings->adaptContrastKernelSize);
  87:     PROG_INCR("Adaptive contrast stretch applied");
  88:     if (Settings->useBlur) {
  89:       IntEnchance.Blur(Settings->blurKernelSize, true);
  90:       PROG_INCR("Blur applied");
  91:     }
  92:   } else if (Settings->useBlur) {
  93:     IntEnchance.Blur(Settings->blurKernelSize, false);
  94:     PROG_INCR("Blur applied");
  95:   } else {
  96:     IntEnchance.ProcessedImg = IntEnchance.OriginalImg;
  97:   }
  98:   OptimizedInt =
  99:       IntEnchance.ProcessedImg(cv::Rect(khBorder, khBorder,
 100:                                         OriginalImage.cols - kBorder,
 101:                                         OriginalImage.rows - kBorder)).clone();
 102:   SHOW_DEBUG_IMG(OptimizedInt, uchar, 255, "IntEnchance", false);
 103:
 104:   // Segment the Dark Objects en fill the holes
 105:   Vision::Segment Segmenter(OptimizedInt);
 106:   Segmenter.sigma = Settings->sigmaFactor;
 107:   Segmenter.thresholdOffset = Settings->thresholdOffsetValue;
 108:   Segmenter.ConvertToBW(Settings->typeOfObjectsSegmented);
 109:   PROG_INCR("Threshold applied");
 110:   if (Settings->fillHoles) {
 111:     Segmenter.FillHoles(true);
 112:     PROG_INCR("Holes filled");
 113:   }
 114:   if (Settings->ignorePartialBorderParticles) {
 115:     Segmenter.RemoveBorderBlobs(1, true);
 116:     PROG_INCR("Border Blobs removed");
 117:   }
 118:   SHOW_DEBUG_IMG(Segmenter.ProcessedImg, uchar, 255, "Segmenter", true);
 119:
 120:   // Erode the segmented image and sets the BW image use it to create the NO
 121:   // background RGB
 122:   Vision::MorphologicalFilter Filter(Segmenter.ProcessedImg);
 123:   uint kSize = Settings->filterMaskSize;
 124:   Mat mask = cv::Mat::zeros(kSize, kSize, CV_8UC1);
 125:   circle(mask, Point(kSize / 2, kSize / 2), (kSize / 2) + 1, 1, -1);
 126:   switch (Settings->morphFilterType) {
 127:   case Vision::MorphologicalFilter::CLOSE:
 128:     Filter.Close(mask);
 129:     PROG_INCR("Morphological filer - close applied");
 130:     break;
 131:   case Vision::MorphologicalFilter::DILATE:
 132:     Filter.Dilation(mask);
 133:     PROG_INCR("Morphological filer - dilate applied");
 134:     break;
 135:   case Vision::MorphologicalFilter::ERODE:
 136:     Filter.Erosion(mask);
 137:     PROG_INCR("Morphological filer - erode applied");
 138:     break;
 139:   case Vision::MorphologicalFilter::OPEN:
 140:     Filter.Open(mask);
 141:     PROG_INCR("Morphological filer - open applied");
 142:     break;
 143:   case Vision::MorphologicalFilter::NONE:
 144:     Filter.ProcessedImg = Filter.OriginalImg;
 145:   }
 146:   BW = Filter.ProcessedImg.clone();
 147:   SHOW_DEBUG_IMG(BW, uchar, 255,
 148:                  "BW after segmentation, fill holes and erosion", true);
 149:   RGB = Vision::ImageProcessing::CopyMat<uchar>(
 150:       OriginalImage(cv::Rect(khBorder, khBorder, OriginalImage.cols - kBorder,
 151:                              OriginalImage.rows - kBorder)).clone(),
 152:       BW, CV_8UC1);
 153:   PROG_INCR("RGB masked image generated");
 154:   SHOW_DEBUG_IMG(RGB, uchar, 255, "RGB no Background", false);
 155:
 156:   // Create the Edge image
 157:   Vision::Segment Edger(BW);
 158:   Edger.GetEdgesEroding();
 159:   Edge = Edger.ProcessedImg;
 160:   PROG_INCR("Edge filter applied");
 161:   SHOW_DEBUG_IMG(Edge, uchar, 255, "Edge", true);
 162:
 163:   // Make the CIE La*b* conversion
 164:   Vision::Conversion RGBnewConvertor(RGB);
 165:   RGBnewConvertor.Convert(Vision::Conversion::RGB, Vision::Conversion::CIE_lab);
 166:   LAB = RGBnewConvertor.CopyMat<float>(RGB, BW, CV_32F);
```

```
167:   PROG_INCR("CIE La*b* conversion calculated");
168:   SHOW_DEBUG_IMG(LAB, float, 1.0, "LAB", true);
169:
170:   // Create the Redness Index
171:   Vision::Conversion LABConvertor(LAB);
172:   LABConvertor.Convert(Vision::Conversion::CIE_lab, Vision::Conversion::RI);
173:   RI = LABConvertor.ProcessedImg;
174:   PROG_INCR("Redness conversion calculated");
175:   SHOW_DEBUG_IMG(RI, float, 1.0, "RI", true);
176:
177:   imgPrepped = true;
178: }
179:
180: void Sample::Analyse(SoilMath::NN &nn) {
181:   if (!imgPrepped) {
182:     PrepImg();
183:   }
184:   // Calculate the statistics CIE La*b*
185:   // vector<Mat> LABextract = Vision::ImageProcessing::extractChannel(LAB);
186:   // for_each(LABextract.begin(), LABextract.end(), [&](Mat &lab) {
187:   // Results.LAB_Stat.push_back(floatStat_t((float *)lab.data, lab.rows,
188:   // lab.cols)); });
189:
190:   // Calculate the statistics RI
191:   // Results.RI_Stat = floatStat_t((float *)RI.data, RI.rows, RI.cols);
192:
193:   // Segment and analyze the particles
194:   SegmentParticles(Vision::Segment::SegmentationType::Normal);
195:   // for_each(Population.begin(), Population.end(), [&](Particle &P)
196:   //{
197:   //    P.Analyze(nn);
198:   //});
199:
200:   // Analyze the image
201: }
202:
203: void Sample::Analyse(const Mat &src, SoilMath::NN &nn) {
204:   OriginalImage = src;
205:   Analyse(nn);
206: }
207:
208: bool Sample::AnalysePopVect(const vector<Particle> &population,
209:                             AnalysisResults &results) {
210:   return true;
211: }
212:
213: void Sample::SegmentParticles(Vision::Segment::SegmentationType segType) {
214:   Vision::Segment Segmenter(BW);
215:
216:   // Get the Particlelist
217:   Segmenter.GetBlobList();
218:   Population.resize(Segmenter.BlobList.size());
219:   uint32_t i = 0;
220:   // Analyze each particle
221:   for_each(Population.begin(), Population.end(), [&](Particle &P) {
222:     P.ID = Segmenter.BlobList[i].Label;
223:     P.Analysis.Analyzed = false;
224:     P.BW = Segmenter.BlobList[i].Img.clone();
225:     cv::Rect ROI = Segmenter.BlobList[i].ROI;
226:     P.Intensity =
227:         Vision::Segment::CopyMat<uchar>(Intensity(ROI).clone(), P.BW, CV_8UC1);
228:     P.LAB = Vision::Segment::CopyMat<float>(LAB(ROI).clone(), P.BW, CV_32FC3);
229:     P.RGB = Vision::Segment::CopyMat<uchar>(LAB(ROI).clone(), P.BW, CV_8UC3);
230:     P.RI = Vision::Segment::CopyMat<float>(LAB(ROI).clone(), P.BW, CV_32FC1);
231:     P.Edge = Vision::Segment::CopyMat<uchar>(Edge(ROI).clone(), P.BW, CV_8UC1);
232:     i++;
233:   });
234: }
235:
236: boost::signals2::connection
237: Sample::connect_Progress(const Progress_t::slot_type &subscriber) {
238:   return prog_sig.connect(subscriber);
239: }
240: }
```

```
 1: /* Copyright (C) Jelle Spijker - All Rights Reserved
 2:  * Unauthorized copying of this file, via any medium is strictly prohibited
 3:  * and only allowed with the written consent of the author (Jelle Spijker)
 4:  * This software is proprietary and confidential
 5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 6:  */
 7:
 8: #pragma once
 9: namespace SoilAnalyzer {
10: /*!
11:  * \brief The AnalyseType enum
12:  */
13: enum AnalyseType { S_LAB, S_PSD, S_RI, S_ROUNDNESS };
14: }
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "soilsettings.h"

namespace SoilAnalyzer {
SoilSettings::SoilSettings() {}

void SoilSettings::LoadSettings(string filename) {
  std::ifstream ifs(filename.c_str());
  boost::archive::xml_iarchive ia(ifs);
  ia >> boost::serialization::make_nvp("SoilSettings", *this);
}

void SoilSettings::SaveSettings(string filename) {
  std::ofstream ofs(filename.c_str());
  boost::archive::xml_oarchive oa(ofs);
  oa << boost::serialization::make_nvp("SoilSettings", *this);
}
}
```